Lecture Notes in Computer Science 2748

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Frank Dehne Jörg-Rüdiger Sack
Michiel Smid (Eds.)

Algorithms and Data Structures

8th International Workshop, WADS 2003
Ottawa, Ontario, Canada, July 30 – August 1, 2003
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Frank Dehne
Jörg-Rüdiger Sack
Michiel Smid
Carleton University, School of Computer Science
Ottawa, Canada K1S 5B6
E-mail: frank@dehne.net
{sack,michiel}@scs.carleton.ca

Cataloging-in-Publication Data applied for

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): F.2, E.1, G.2, I.3.5, G.1

ISSN 0302-9743
ISBN 3-540-40545-3 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

© Springer-Verlag Berlin Heidelberg 2003
Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP-Berlin GmbH
Printed on acid-free paper SPIN: 10929292 06/3142 5 4 3 2 1 0

Preface

The papers in this volume were presented at the 8th Workshop on Algorithms and Data Structures (WADS 2003). The workshop took place July 30–August 1, 2003, at Carleton University in Ottawa, Canada. The workshop alternates with the Scandinavian Workshop on Algorithm Theory (SWAT), continuing the tradition of SWAT and WADS starting with SWAT'88 and WADS'89.

In response to the call for papers, 126 papers were submitted. From these submissions, the program committee selected 40 papers for presentation at the workshop. In addition, invited lectures were given by the following distinguished researchers: Gilles Brassard, Dorothea Wagner, Daniel Spielman, and Michael Fellows.

At this year's workshop, Wing T. Yan (Nelligan O'Brien Payne LLP, Ottawa) gave a special presentation on "Protecting Your Intellectual Property." On July 29, Hans-Georg Zimmermann (Siemens AG, München) gave a seminar on "Neural Networks in System Identification and Forecasting: Principles, Techniques, and Applications," and on August 2 there was a workshop on "Fixed Parameter Tractability" organized by Frank Dehne, Michael Fellows, Mike Langston, and Fran Rosamond.

On behalf of the program committee, we would like to express our appreciation to the invited speakers and to all authors who submitted papers.

Ottawa, May 2003

Frank Dehne
Jörg-Rüdiger Sack
Michiel Smid

WADS Steering Committee

Frank Dehne (Carleton)
Ian Munro (Waterloo)
Jörg-Rüdiger Sack (Carleton)
Nicola Santoro (Carleton)
Roberto Tamassia (Brown)

Program Committee

Frank Dehne (Carleton), co-chair
Jörg-Rüdiger Sack (Carleton), co-chair
Michiel Smid (Carleton), co-chair
Lars Arge (Duke)
Susanne Albers (Freiburg)
Michael Atkinson (Dunedin)
Hans Bodlaender (Utrecht)
Gerth Brodal (Aarhus)
Tom Cormen (Dartmouth)
Timothy Chan (Waterloo)
Erik Demaine (MIT)
Michael Fellows (Newcastle)
Pierre Freigniaud (Paris-Sud)
Naveen Garg (Delhi)
Andrew Goldberg (Microsoft)
Giuseppe Italiano (Rome)
Ravi Janardan (Minneapolis)
Rolf Klein (Bonn)
Giri Narasimhan (Florida International University)
Rolf Niedermeier (Tübingen)
Viktor Prasanna (Southern California)
Andrew Rau-Chaplin (Halifax)
R. Ravi (Carnegie Mellon)
Paul Spirakis (Patras)
Roberto Tamassia (Brown)
Jeff Vitter (Purdue)
Dorothea Wagner (Konstanz)
Peter Widmayer (Zürich)

Referees

Faisal Abu-Khazm
Pankaj Agarwal
Jochen Alber
Lyudmil Aleksandrov
Stephen Alstrup
Helmut Alt
Luzi Anderegg
Franz Aurenhammer
David A. Bader
Mihai Bădoiu
Evripides Bampis
Nikhil Bansal
Dirk Bartz
Prosenjit Bose
Jesper Makholm Byskov
Chandra Chekuri
Danny Z. Chen
Mark de Berg
Camil Demetrescu
Joerg Derungs
Luc Devroye
Kedar Dhamdhere
Walter Didimo
Emilio Di Giacomo
Herbert Edelsbrunner
Stephan Eidenbenz
Jeff Erickson
Vladimir Estivill-Castro
Rolf Fagerberg
Irene Finocchi
Gudmund Frandsen
Olaf Delgado Friedrichs
Michael Gatto
Jens Gramm
Roberto Grossi
Joachim Gudmundsson
Jiong Guo
Prosenjit Gupta
Sariel Har-Peled
Herman Haverkort
Fabian Hennecke
Edward A. Hirsch
Bo Hong
Han Hoogeveen
Riko Jacob
Jyrki Katajainen
Rohit Khandekar
Jochen Konemann
Jan Korst
Alexander Kulikov
Keshav Kunal
Klaus-Jörn Lange
Mike Langston
Thierry Lecroq
Stefano Leonardi
David Liben-Nowell
Giuseppe Liotta
Hsueh-I Lu
Bolette A. Madsen
Christos Makris
Madhav Marathe
Joe Mitchell
Anders Moller
Pat Morin
Ian Munro
Moni Naor
Marc Nunkesser
Gianpaolo Oriolo
Andrea Pacifici
Rasmus Pagh
Ojas Parekh
Joon-Sang Park
Neungsoo Park
Mihai Patrascu
Christian N.S. Pedersen
Benny Pinkas
M.Z. Rahman
Venkatesh Raman
Theis Rauhe
Peter Rossmanith
Konrad Schlude
Michael Segal
Raimund Seidel
Rahul Shah
Mitali Singh
Amitabh Sinha
Jeremy Spinrad
Renzo Sprugnoli
Gabor Szabo
Sergei Vorobyov
Anil Vullikanti
Tandy Warnow
Birgitta Weber
Yang Yu
Norbert Zeh
Afra Zomorodian

Table of Contents

Multi-party Pseudo-Telepathy 1
Gilles Brassard, Anne Broadbent, Alain Tapp

Adapting (Pseudo)-Triangulations with a Near-Linear Number of Edge Flips 12
Oswin Aichholzer, Franz Aurenhammer, Hannes Krasser

Shape Segmentation and Matching with Flow Discretization 25
Tamal K. Dey, Joachim Giesen, Samrat Goswami

Phylogenetic Reconstruction from Gene-Rearrangement Data with Unequal Gene Content 37
Jijun Tang, Bernard M.E. Moret

Toward Optimal Motif Enumeration 47
Patricia A. Evans, Andrew D. Smith

Common-Deadline Lazy Bureaucrat Scheduling Problems 59
Behdad Esfahbod, Mohammad Ghodsi, Ali Sharifi

Bandwidth-Constrained Allocation in Grid Computing 67
Anshul Kothari, Subhash Suri, Yunhong Zhou

Algorithms and Approximation Schemes for Minimum Lateness/Tardiness Scheduling with Rejection 79
Sudipta Sengupta

Fast Algorithms for a Class of Temporal Range Queries 91
Qingmin Shi, Joseph JaJa

Distribution-Sensitive Binomial Queues 103
Amr Elmasry

Optimal Worst-Case Operations for Implicit Cache-Oblivious Search Trees 114
Gianni Franceschini, Roberto Grossi

Extremal Configurations and Levels in Pseudoline Arrangements 127
Micha Sharir, Shakhar Smorodinsky

Fast Relative Approximation of Potential Fields 140
Martin Ziegler

The One-Round Voronoi Game Replayed 150
Sándor P. Fekete, Henk Meijer

Integrated Prefetching and Caching with Read and Write Requests 162
Susanne Albers, Markus Büttner

Online Seat Reservations via Offline Seating Arrangements 174
Jens S. Frederiksen, Kim S.Larsen

Routing and Call Control Algorithms for Ring Networks 186
R. Sai Anand, Thomas Erlebach

Algorithms and Models for Railway Optimization 198
Dorothea Wagner

Approximation of Rectilinear Steiner Trees with Length Restrictions on Obstacles 207
Matthias Müller-Hannemann, Sven Peyer

Multi-way Space Partitioning Trees 219
Christian A. Duncan

Cropping-Resilient Segmented Multiple Watermarking 231
Keith Frikken, Mikhail Atallah

On Simultaneous Planar Graph Embeddings 243
P. Brass, E. Cenek, Christian A. Duncan, A. Efrat, C. Erten, D. Ismailescu, S.G. Kobourov, A. Lubiw, J.S.B. Mitchell

Smoothed Analysis (Motivation and Discrete Models) 256
Daniel A. Spielman, Shang-Hua Teng

Approximation Algorithm for Hotlink Assignments in Web Directories ... 271
Rachel Matichin, David Peleg

Drawing Graphs with Large Vertices and Thick Edges 281
Gill Barequet, Michael T. Goodrich, Chris Riley

Semi-matchings for Bipartite Graphs and Load Balancing 294
Nicholas J.A. Harvey, Richard E. Ladner, László Lovász, Tami Tamir

The Traveling Salesman Problem for Cubic Graphs 307
David Eppstein

Sorting Circular Permutations by Reversal 319
Andrew Solomon, Paul Sutcliffe, Raymond Lister

An Improved Bound on Boolean Matrix Multiplication for Highly Clustered Data 329
Leszek Gąsieniec, Andrzej Lingas

Dynamic Text and Static Pattern Matching 340
Amihood Amir, Gad M. Landau, Moshe Lewenstein, Dina Sokol

Real Two Dimensional Scaled Matching 353
Amihood Amir, Ayelet Butman, Moshe Lewenstein, Ely Porat

Proximity Structures for Geometric Graphs 365
Sanjiv Kapoor, Xiang-Yang Li

The Zigzag Path of a Pseudo-Triangulation 377
Oswin Aichholzer, Günter Rote, Bettina Speckmann, Ileana Streinu

Alternating Paths along Orthogonal Segments 389
Csaba D. Tóth

Improved Approximation Algorithms for the Quality of Service Steiner Tree Problem 401
Marek Karpinski, Ion I. Măndoiu, Alexander Olshevsky, Alexander Zelikovsky

Chips on Wafers 412
Mattias Andersson, Joachim Gudmundsson, Christos Levcopoulos

A Model for Analyzing Black-Box Optimization 424
Vinhthuy Phan, Steven Skiena, Pavel Sumazin

On the Hausdorff Voronoi Diagram of Point Clusters in the Plane 439
Evanthia Papadopoulou

Output-Sensitive Algorithms for Computing Nearest-Neighbour Decision Boundaries 451
David Bremner, Erik Demaine, Jeff Erickson, John Iacono, Stefan Langerman, Pat Morin, Godfried Toussaint

Significant-Presence Range Queries in Categorical Data 462
Mark de Berg, Herman J. Haverkort

Either/Or: Using Vertex Cover Structure in Designing FPT-Algorithms – The Case of k-Internal Spanning Tree 474
Elena Prieto, Christian Sloper

Parameterized Complexity of Directed Feedback Set Problems in Tournaments 484
Venkatesh Raman, Saket Saurabh

Compact Visibility Representation and Straight-Line Grid Embedding of Plane Graphs 493
Huaming Zhang, Xin He

New Directions and New Challenges in Algorithm Design and Complexity, Parameterized 505
Michael R. Fellows

Author Index 521

Multi-party Pseudo-Telepathy

Gilles Brassard*, Anne Broadbent**, and Alain Tapp***

Département IRO, Université de Montréal, C.P. 6128, succursale centre-ville,
Montréal (Québec), Canada H3C 3J7
{brassard,broadbea,tappa}@iro.umontreal.ca

Abstract. Quantum entanglement, perhaps the most non-classical manifestation of quantum information theory, cannot be used to transmit information between remote parties. Yet, it can be used to reduce the amount of communication required to process a variety of distributed computational tasks. We speak of *pseudo-telepathy* when quantum entanglement serves to *eliminate* the classical need to communicate. In earlier examples of pseudo-telepathy, classical protocols could succeed with high probability unless the inputs were very large. Here we present a simple multi-party distributed problem for which the inputs and outputs consist of a single bit per player, and we present a perfect quantum protocol for it. We prove that no classical protocol can succeed with a probability that differs from $^1/_2$ by more than a fraction that is exponentially small in the number of players. This could be used to circumvent the detection loophole in experimental tests of nonlocality.

1 Introduction

It is well-known that quantum mechanics can be harnessed to reduce the amount of communication required to perform a variety of distributed tasks [3], through the use of either quantum communication [13] or quantum entanglement [6]. Consider for example the case of Alice and Bob, who are very busy and would like to find a time when they are simultaneously free for lunch. They each have an engagement calendar, which we may think of as n–bit strings a and b, where $a_i = 1$ (resp. $b_i = 1$) means that Alice (resp. Bob) is free for lunch on day i. Mathematically, they want to find an index i such that $a_i = b_i = 1$ or establish that such an index does not exist. The obvious solution is for Alice, say, to communicate her entire calendar to Bob, so that he can decide on the date: this requires roughly n bits of communication. It turns out that this is optimal in the worst case, up to a constant factor, according to classical information theory [8], even when the answer is only required to be correct with probability at least $^2/_3$. Yet, this problem can be solved with arbitrarily high success probability

* Supported in part by Canada's NSERC, Québec's FCAR, the Canada Research Chair Programme, and the Canadian Institute for Advanced Research.
** Supported in part by a scholarship from Canada's NSERC.
*** Supported in part by Canada's NSERC and Québec's FCAR.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 1–11, 2003.
© Springer-Verlag Berlin Heidelberg 2003

with the exchange of a number of *quantum* bits—known as *qubits*—in the order of $\sqrt{n}$ [1]. Alternatively, a number of *classical* bits in the order of $\sqrt{n}$ suffices for this task if Alice and Bob share prior entanglement, because they can make use of quantum teleportation [2]. Other (less natural) problems demonstrate an *exponential* advantage of quantum communication, both in the error-free [5] and bounded-error [11] models.

Given that prior entanglement allows for a dramatic *reduction* in the need for classical communication in order to perform some distributed computational tasks, it is natural to wonder if it can be used to *eliminate* the need for communication altogether. In other words, are there distributed tasks that would be impossible to achieve in a classical world if the participants were not allowed to communicate, yet those tasks could be performed without *any* form of communication provided they share prior entanglement? The answer is negative if the result of the computation must become known to at least one party, but it is positive if we are satisfied with the establishment of nonlocal correlations between the parties' inputs and outputs [4].

Mathematically, consider n parties $A_1, A_2, \ldots, A_n$ and two n-ary functions f and g. In an *initialization phase*, the parties are allowed to discuss strategy and share random variables (in the classical setting) and entanglement (in the quantum setting). Then the parties move apart and they are no longer allowed any form of communication. After the parties are physically separated, each A_i is given some input x_i and is requested to produce output y_i. We say that the parties *win* this instance of the game if $g(y_1, y_2, \ldots y_n) = f(x_1, x_2, \ldots x_n)$. Given an n-ary predicate P, known as the *promise*, a protocol is *perfect* if it wins the game with certainty on all inputs that satisfy the promise, i.e. whenever $P(x_1, x_2, \ldots x_n)$ holds. A protocol is *successful with probability* p if it wins *any* instance that satisfies the promise with probability at least p; it is successful in *proportion* p if it wins the game with probability at least p when the instance is chosen at random according to the uniform distribution on the set of instances that satisfy the promise. Any protocol that succeeds with probability p automatically succeeds in proportion p, but not necessarily vice versa. In particular, it is possible for a protocol that succeeds in proportion $p > 0$ to fail systematically on some inputs, whereas this would not be allowed for protocols that succeed with probability $p > 0$. Therefore, the notion of succeeding "in proportion" is meaningful for *deterministic* protocols but not the notion of succeeding "with probability".

We say of a quantum protocol that it exhibits *pseudo-telepathy* if it is perfect provided the parties share prior entanglement, whereas no perfect classical protocol can exist. The study of pseudo-telepathy was initiated in [4], but all examples known so far allowed for classical protocols that succeed with rather high probability, unless the inputs are very long. This made the prospect of experimental demonstration of pseudo-telepathy unappealing for two reasons.

- ◇ It would not be surprising for several runs of an imperfect classical protocol to succeed, so that mounting evidence of a convincingly quantum behaviour would require a large number of consecutive successful runs.
- ◇ Even a slight imperfection in the quantum implementation would be likely to result in an error probability higher than what can easily be achieved with simple classical protocols!

In Section 2, we introduce a simple multi-party distributed computational problem for which the inputs and outputs consist of a single bit per player, and we present a perfect quantum protocol for it. We prove in Sections 3 and 4 that no classical protocol can succeed with a probability that differs from $^1/_2$ by more than a fraction that is exponentially small in the number of players. More precisely, no classical protocol can succeed with a probability better than $\frac{1}{2} + 2^{-\lceil n/2 \rceil}$, where n is the number of players. Furthermore, we show in Section 5 that the success probability of our quantum protocol would remain better than anything classically achievable, when n is sufficiently large, even if each player had imperfect apparatus that would produce the wrong answer with probability nearly 15% or no answer at all with probability 29%. This could be used to circumvent the infamous *detection loophole* in experimental proofs of the nonlocality of the world in which we live [9].

2 A Simple Game and Its Perfect Quantum Protocol

For any $n \geq 3$, game G_n consists of n players. Each player A_i receives a single input bit x_i and is requested to produce a single output bit y_i. The players are promised that there is an even number of 1s among their inputs. Without being allowed to communicate after receiving their inputs, the players are challenged to produce a collective output that contains an even number of 1s if and only if the number of 1s in the input is divisible by 4. More formally, we require that

$$\sum_i^n y_i \;\equiv\; \tfrac{1}{2}\sum_i^n x_i \pmod 2 \tag{1}$$

provided $\sum_i^n x_i \equiv 0 \pmod 2$. We say that $x = x_1 x_2 \ldots x_n$ is the *question* and $y = y_1 y_2 \ldots y_n$ is the *answer*.

Theorem 1. *If the n players are allowed to share prior entanglement, then they can always win game G_n.*

Proof. (In this proof, we assume that the reader is familiar with basic concepts of quantum information processing [10].) Define the following n-qubit entangled quantum states $|\Phi_n^+\rangle$ and $|\Phi_n^-\rangle$.

$$|\Phi_n^+\rangle = \tfrac{1}{\sqrt{2}}|0^n\rangle + \tfrac{1}{\sqrt{2}}|1^n\rangle$$
$$|\Phi_n^-\rangle = \tfrac{1}{\sqrt{2}}|0^n\rangle - \tfrac{1}{\sqrt{2}}|1^n\rangle\,.$$

Let H denote the Walsh-Hadamard transform, defined as usual by

$$H|0\rangle \mapsto \tfrac{1}{\sqrt{2}}|0\rangle + \tfrac{1}{\sqrt{2}}|1\rangle$$
$$H|1\rangle \mapsto \tfrac{1}{\sqrt{2}}|0\rangle - \tfrac{1}{\sqrt{2}}|1\rangle$$

and let S denote the unitary transformation defined by

$$S|0\rangle \mapsto |0\rangle$$
$$S|1\rangle \mapsto i|1\rangle\,.$$

It is easy to see that if S is applied to any two qubits of $|\Phi_n^+\rangle$, while the other qubits are left undisturbed, then the resulting state is $|\Phi_n^-\rangle$, and if S is applied to any two qubits of $|\Phi_n^-\rangle$, then the resulting state is $|\Phi_n^+\rangle$. Therefore, if the qubits of $|\Phi_n^+\rangle$ are distributed among the n players, and if exactly m of them apply S to their qubit, the resulting global state will be $|\Phi_n^+\rangle$ if $m \equiv 0 \pmod 4$ and $|\Phi_n^-\rangle$ if $m \equiv 2 \pmod 4$.

Moreover, the effect of applying the Walsh-Hadamard transform to each qubit in $|\Phi_n^+\rangle$ is to produce an equal superposition of all classical n-bit strings that contain an even number of 1s, whereas the effect of applying the Walsh-Hadamard transform to each qubit in $|\Phi_n^-\rangle$ is to produce an equal superposition of all classical n-bit strings that contain an odd number of 1s. More formally,

$$(H^{\otimes n})|\Phi_n^+\rangle = \tfrac{1}{\sqrt{2^{n-1}}} \sum_{\substack{\Delta(y)\equiv 0 \\ (\mathrm{mod}\ 2)}} |y\rangle$$
$$(H^{\otimes n})|\Phi_n^-\rangle = \tfrac{1}{\sqrt{2^{n-1}}} \sum_{\substack{\Delta(y)\equiv 1 \\ (\mathrm{mod}\ 2)}} |y\rangle\,,$$

where $\Delta(y) = \sum_i y_i$ denotes the Hamming weight of y.

The quantum winning strategy should now be obvious. In the initialization phase, state $|\Phi_n^+\rangle$ is produced and its n qubits are distributed among the n players. After they have moved apart, each player A_i receives input bit x_i and does the following.

1. If $x_i = 1$, A_i applies transformation S to his qubit; otherwise he does nothing.
2. He applies H to his qubit.
3. He measures his qubit in order to obtain y_i.
4. He produces y_i as his output.

We know by the promise that an even number of players will apply S to their qubit. If that number is divisible by 4, which means that $\frac{1}{2}\sum_i^n x_i$ is even, then the global state reverts to $|\Phi_n^+\rangle$ after step 1 and therefore to a superposition of all $|y\rangle$ such that $\Delta(y) \equiv 0 \pmod 2$ after step 2. It follows that $\sum_i^n y_i$, the number of players who measure and output 1, is even. On the other hand, if the number of players who apply S to their qubit is congruent to 2 modulo 4, which

means that $\frac{1}{2}\sum_i^n x_i$ is odd, then the global state evolves to $|\Phi_n^-\rangle$ after step 1 and therefore to a superposition of all $|y\rangle$ such that $\Delta(y) \equiv 1 \pmod 2$ after step 2. It follows in this case that $\sum_i^n y_i$ is odd. In either case, Equation (1) is fulfilled at the end of the protocol, as required. □

3 Optimal Proportion for Deterministic Protocols

In this section, we study the case of *deterministic* classical protocols to play game G_n. We show that no such protocol can succeed on a proportion of the allowed inputs that is significantly better than $1/2$.

Theorem 2. *The best possible deterministic strategy for game G_n is successful in proportion $\frac{1}{2} + 2^{-\lceil n/2 \rceil}$.*

Proof. Since no information may be communicated between players during the game, the best they can do is to agree on a strategy before the game starts. Any such deterministic strategy will be such that player A_i's answer y_i depends only on his input bit x_i. Therefore, each player has an individual strategy $s_i \in \{01, 10, 00, 11\}$, where the first bit of the pair denotes the strategy's output y_i if the input bit is $x_i = 0$ and the second bit of the strategy denotes its output if the input is $x_i = 1$. In other words, 00 and 11 denote the two constant strategies $y_i = 0$ and $y_i = 1$, respectively, 01 denotes the strategy that sets $y_i = x_i$, and 10 denotes the complementary strategy $y_i = \overline{x_i}$.

Let $s = s_1, s_2, \ldots, s_n$ be the global deterministic strategy chosen by the players. The order of the players is not important, so that we may assume without loss of generality that strategy s has the following form.

$$s = \overbrace{01, 01, \ldots, 01}^{k-\ell}, \overbrace{10, 10, \ldots, 10}^{\ell}, \overbrace{00, 00, \ldots, 00}^{n-k-m}, \overbrace{11, 11, \ldots, 11}^{m}$$

Assuming strategy s is being used, the Hamming weight $\Delta(y)$ of the answer is given by

$$\begin{aligned}\Delta(y) &= \Delta(x_1 \ldots, x_{k-\ell}) + \Delta(\overline{x_{k-\ell+1}}, \ldots, \overline{x_k}) + \Delta(\overbrace{00\ldots0}^{n-k-m}) + \Delta(\overbrace{11\ldots1}^{m}) \\ &\equiv \Delta(x_1, \ldots, x_k) + \ell + m \pmod 2 .\end{aligned}$$

Consider the following four sets, for $a, b \in \{0, 1\}$.

$$S_{a,b}^k = \{x \mid \Delta(x_1, \ldots, x_k) \equiv a \pmod 2 \text{ and } \Delta(x_1, \ldots, x_n) \equiv 2b \pmod 4\}$$

If $\ell + m$ is even then there are exactly $|S_{0,0}^k| + |S_{1,1}^k|$ questions that yield a winning answer, and otherwise if $\ell+m$ is odd then there are exactly $|S_{1,0}^k| + |S_{0,1}^k|$ questions that yield a winning answer. We also have that the four sets account for all possible questions and therefore

$$|S_{0,0}^k| + |S_{1,1}^k| \;=\; 2^{n-1} - (|S_{1,0}^k| + |S_{0,1}^k|) .$$

From here, the proof of the Theorem follows directly from Lemma 2 below. □

First we need to state a standard Lemma.

Lemma 1. [7, Eqn. 1.54]

$$\sum_{\substack{i\equiv a \\ (\mathrm{mod}\ 4)}} \binom{n}{i} = \begin{cases} 2^{n-2} + 2^{\frac{n}{2}-1} & \text{if } n-2a \equiv 0 \pmod 8 \\ 2^{n-2} - 2^{\frac{n}{2}-1} & \text{if } n-2a \equiv 4 \pmod 8 \\ 2^{n-2} & \text{if } n-2a \equiv 2,6 \pmod 8 \\ 2^{n-2} + 2^{\frac{n-3}{2}} & \text{if } n-2a \equiv 1,7 \pmod 8 \\ 2^{n-2} - 2^{\frac{n-3}{2}} & \text{if } n-2a \equiv 3,5 \pmod 8 \end{cases} \tag{2}$$

Lemma 2. *If n is odd, then*

$$|S^k_{0,0}| + |S^k_{1,1}| = \begin{cases} 2^{n-2} + 2^{\frac{n-3}{2}} & \text{if } (n-1)/2 + 3(n-k) \equiv 0,3 \pmod 4 \\ 2^{n-2} - 2^{\frac{n-3}{2}} & \text{if } (n-1)/2 + 3(n-k) \equiv 1,2 \pmod 4 \end{cases}$$

On the other hand, if n is even, then

$$|S^k_{0,0}| + |S^k_{1,1}| = \begin{cases} 2^{n-2} & \text{if } n/2 + 3(n-k) \equiv 1,3 \pmod 4 \\ 2^{n-2} + 2^{\frac{n}{2}-1} & \text{if } n/2 + 3(n-k) \equiv 0 \pmod 4 \\ 2^{n-2} - 2^{\frac{n}{2}-1} & \text{if } n/2 + 3(n-k) \equiv 2 \pmod 4 \end{cases}$$

Proof. From the definition of $S^k_{a,b}$, provided we consider that $\binom{0}{a} = 0$ whenever $a \neq 0$ and $\binom{0}{0} = 1$, we get

$$|S^k_{0,0}| = \sum_{\substack{i\equiv 0 \\ (\mathrm{mod}\ 4)}} \binom{k}{i} \sum_{\substack{j\equiv 0 \\ (\mathrm{mod}\ 4)}} \binom{n-k}{j} + \sum_{\substack{i\equiv 2 \\ (\mathrm{mod}\ 4)}} \binom{k}{i} \sum_{\substack{j\equiv 2 \\ (\mathrm{mod}\ 4)}} \binom{n-k}{j} \tag{3}$$

$$|S^k_{1,1}| = \sum_{\substack{i\equiv 1 \\ (\mathrm{mod}\ 4)}} \binom{k}{i} \sum_{\substack{j\equiv 1 \\ (\mathrm{mod}\ 4)}} \binom{n-k}{j} + \sum_{\substack{i\equiv 3 \\ (\mathrm{mod}\ 4)}} \binom{k}{i} \sum_{\substack{j\equiv 3 \\ (\mathrm{mod}\ 4)}} \binom{n-k}{j}. \tag{4}$$

Using Lemma 1, we compute (3) and (4). Since n and k are parameters for the equations, and since Lemma 1 depends on the values of n and k modulo 8, we have 8 cases to verify for n and 8 cases for k, hence 64 cases in total. These straightforward, albeit tedious, calculations are left to the reader. □

Theorem 3. *Very simple deterministic protocols achieve the bound given in Theorem 2. In particular, the players do not even have to look at their input when $n \not\equiv 2 \pmod 4$!*

Proof. The following simple strategies, which depend on $n \pmod 8$, are easily seen to succeed in proportion exactly $\frac{1}{2} + 2^{-\lceil n/2 \rceil}$. They are therefore optimal among all possible deterministic classical strategies. □

Table 1. Simple optimal strategies.

n (mod 8)	player 1	players 2 to n
0	00	00
1	00	00
2	01	00
3	11	11
4	11	00
5	00	00
6	10	00
7	11	11

4 Optimal Probability for Classical Protocols

In this section, we consider all possible *classical* protocols to play game G_n, including probabilistic protocols. We give as much power as possible to the classical model by allowing the playing parties unlimited sharing of random variables. Despite this, we prove that no classical protocol can succeed with a probability that is significantly better than $1/2$ on the worst-case input.

Definition 1. *A probabilistic strategy is a probability distribution over a set of deterministic strategies.*

The random variable shared by the players during the initialization phase corresponds to deciding which deterministic strategy will be used for any given run of the protocol.

Lemma 3. *Consider any multi-party game of the sort formalized in Section 1. For any probabilistic protocol that is successful with probability p, there exists a deterministic protocol that is successful in proportion at least p.*

Proof. This Lemma is a special case of a theorem proven by Andrew Yao [12], but its proof is so simple that we include it here for completeness. Consider any probabilistic strategy that is successful with probability p. Recall that this means that the protocol wins the game with probability at least p on any instance of the problem that satisfies the promise. By the pigeon hole principle, the same strategy wins the game with probability at least p if the input is chosen uniformly at random among all possible inputs that satisfy the promise. In other words, it is successful *in proportion* at least p. Consider now the deterministic strategies that enter the definition of our probabilistic strategy, according to Definition 1. Assume for a contradiction that the best among them succeeds in proportion $q < p$. Then, again by the pigeon hole principle, any probabilistic mixture of those deterministic strategies (not only the uniform mixture) would succeed in proportion no better than q. But this includes the probabilistic strategy whose existence we assumed, which does succeed in proportion at least p. This implies that $p \leq q$, a contradiction, and therefore at least one deterministic strategy must succeed in proportion at least p. □

Theorem 4. *No classical strategy for game G_n can be successful with a probability better than $\frac{1}{2} + 2^{-\lceil n/2 \rceil}$.*

Proof. Any classical strategy for game G_n that would be successful with probability $p > \frac{1}{2} + 2^{-\lceil n/2 \rceil}$ would imply by Lemma 3 the existence of a deterministic strategy that would succeed in proportion at least p. This would contradict Theorem 2. □

Theorem 4 gives an upper bound on the best probability that can be achieved by any classical strategy in winning game G_n. However, it is still unknown if there exists a classical strategy capable of succeeding with probability $\frac{1}{2} + 2^{-\lceil n/2 \rceil}$. We conjecture that this is the case. Consider the probabilistic strategy that chooses uniformly at random among all the deterministic strategies that are optimal according to Theorem 2. We have been able to prove with the help of Mathematica that this probabilistic strategy is successful with probability $\frac{1}{2} + 2^{-\lceil n/2 \rceil}$ for all $3 \leq n \leq 14$. We have also proved that this probabilistic strategy is successful with probability $\frac{1}{2} + 2^{-\lceil n/2 \rceil}$ for any odd number n of players, but only when the players all receive $x_i = 0$ as input. The general case is still open.

Conjecture 1. There is a classical strategy for game G_n that is successful with a probability that is *exactly* $\frac{1}{2} + 2^{-\lceil n/2 \rceil}$ on all inputs.

5 Imperfect Apparatus

Quantum devices are often unreliable and thus we cannot expect to witness the perfect result predicted by quantum mechanics in Theorem 1. However, the following analysis shows that a reasonably large error probability can be tolerated if we are satisfied with making experiments in which a quantum-mechanical strategy will succeed with a probability that is still better than anything classically achievable. This would be sufficient to rule out classical theories of the universe.

First consider the following model of imperfect apparatus. Assume that the classical bit y_i that is output by each player A_i corresponds to the predictions of quantum mechanics (if the apparatus were perfect) with some probability p. With complementary probability $1 - p$, the player would output the complement of that bit. Assume furthermore that the errors are independent between players. In other words, we model this imperfection by saying that each player flips his (perfect) output bit with probability $1 - p$.

Theorem 5. *For all $p > \frac{1}{2} + \frac{\sqrt{2}}{4} \approx 85\%$ and for all sufficiently large number n of players, provided each player outputs what is predicted by quantum mechanics (according to the protocol given in the proof of Theorem 1) with probability at least p, the quantum success probability in game G_n remains strictly greater than anything classically achievable.*

Proof. In the n-player imperfect quantum protocol, the probability p_n that the game is won is given by the probability of having an even number of errors:

$$p_n \;=\; \sum_{\substack{i\equiv 0 \\ (\mathrm{mod}\ 2)}} \binom{n}{i} p^{n-i}(1-p)^i \,.$$

It is easy to prove by mathematical induction that

$$p_n \;=\; \frac{1}{2} + \frac{(2p-1)^n}{2} \,.$$

Let's concentrate for now on the case where n is odd. By Theorem 4, the success probability of any classical protocol is upper-bounded by

$$p'_n = \frac{1}{2} + \frac{1}{2^{(n+1)/2}} \,.$$

For any fixed n, define

$$e_n = \frac{1}{2} + \frac{(\sqrt{2}\,)^{1+1/n}}{4} \,.$$

It follows from elementary algebra that

$$p > e_n \;\Rightarrow\; p_n > p'_n \,.$$

In other words, the imperfect quantum protocol on n players surpasses anything classically achievable provided $p > e_n$. For example, $e_3 \approx 89.7\%$ and $e_5 \approx 87.9\%$. Thus we see that even the game with as few as 3 players is sufficient to exhibit genuine quantum behaviour if the apparatus is at least 90% reliable. As n increases, the threshold e_n decreases. In the limit of large n, we have

$$\lim_{n\to\infty} e_n = \frac{1}{2} + \frac{\sqrt{2}}{4} \approx 85\% \,.$$

The same limit is obtained for the case when n is even. □

Another way of modelling the imperfect apparatus is to assume that it gives the correct answer most of the time, but sometimes it fails to give any answer at all. This is the type of behaviour that gives rise to the infamous *detection loophole* in experimental tests of the fact that the world is not classical [9]. When the detectors fail to give an answer, the corresponding player knows that all information is lost. In this case, he has nothing better to do than output a random bit. With this strategy, either every player is lucky enough to register an answer, in which case the game is won with certainty, or at least one player outputs a random answer, in which case the game is won with probability $^1/_2$ regardless of what the other players do.

Corollary 1. *For all $q > \frac{1}{\sqrt{2}} \approx 71\%$ and for all sufficiently large number n of players, provided each player outputs what is predicted by quantum mechanics (according to the protocol given in the proof of Theorem 1) when he receives an answer from his apparatus with probability at least q, but otherwise the player outputs a random answer, the data collected in playing game G_n cannot be explained by any classical local realistic theory.*

Proof. If a player obtains the correct answer with probability q and otherwise outputs a random answer, the probability that the resulting output be correct is $p = q + \frac{1}{2}(1-q) = (1+q)/2$. Therefore, this scenario reduces to the previous one with this simple change of variables. We know from Theorem 5 that the imperfect quantum protocol is more reliable than any possible classical protocol, provided n is large enough, when $p > \frac{1}{2} + \frac{\sqrt{2}}{4}$. This translates directly to $q > \frac{1}{\sqrt{2}}$. □

6 Conclusions and Open Problems

We have demonstrated that quantum pseudo-telepathy can arise for simple multi-party problems that cannot be handled by classical protocols much better than by the toss of a coin. This could serve to design new tests for the nonlocality of the physical world in which we live.

In closing, we propose two open problems. First, can Conjecture 1 be proven or are the best possible classical probabilistic protocols for our game even worse than hinted at by Theorem 4? Second, it would be nice to find a *two*-party pseudo-telepathy problem that admits a perfect quantum solution, yet any classical protocol would have a small probability of success even for inputs of small or moderate size.

References

1. Aaronson, S., Ambainis, A.: Quantum search of spatial regions. Available as `arXiv:quant-ph/0303041` (2003).
2. Bennett, C. H., Brassard, G., Crépeau, C., Jozsa, R., Peres, A., Wootters, W. K.: Teleporting an unknown quantum state via dual classical and Einstein–Podolsky–Rosen channels. *Physical Review Letters* **70** (1993) 1895–1899.
3. Brassard, G.: Quantum communication complexity. *Foundations of Physics* (to appear, 2003).
4. Brassard, G., Cleve, R., Tapp, A.: Cost of exactly simulating quantum entanglement with classical communication. *Physical Review Letter* **83** (1878) 1874–1878.
5. Buhrman, H., Cleve, R., Wigderson, A.: Quantum vs. classical communication and computation. *Proceedings of 30th Annual ACM Symposium on Theory of Computing* (1998) 63–68.
6. Cleve, R., Buhrman, H.: Substituting quantum entanglement for communication. *Physical Review A* **56** (1997) 1201–1204.
7. Gould, H. W.: *Combinatorial Identities*. Morgantown (1972).

8. Kalyanasundaram, B., Schnitger, G.: The probabilistic communication complexity of set intersection. *Proceedings of 2nd Annual IEEE Conference on Structure in Complexity Theory* (1987) 41–47.
9. Massar, S.: Non locality, closing the detection loophole, and communication complexity. *Physical Review A* **65** (2002) 032121-1–032121-5.
10. Nielsen, M. A., Chuang, I. L.: *Quantum Computation and Quantum Information*. Cambridge University Press (2000).
11. Raz, R.: Exponential separation of quantum and classical communication complexity. *Proceedings of 31st Annual ACM Symposium on Theory of Computing* (1999) 358–367.
12. Yao, A. C.–C.: Probabilistic computations: Toward a unified measure of complexity. *Proceedings of 18th IEEE Symposium on Foundations of Computer Science* (1977) 222–227.
13. Yao, A. C.-C.: Quantum circuit complexity. *Proceedings of 34th Annual IEEE Symposium on Foundations of Computer Science* (1993) 352–361.

Adapting (Pseudo)-Triangulations with a Near-Linear Number of Edge Flips

Oswin Aichholzer[1]*, Franz Aurenhammer[1], and Hannes Krasser[2]**

[1] Institute for Software Technology, Graz University of Technology, Graz, Austria.
[2] Institute for Theoretical Computer Science, Graz University of Technology, Graz, Austria.

1 Introduction

In geometric data processing, structures that partition the geometric input, as well as connectivity structures for geometric objects, play an important role. A versatile tool in this context are triangular meshes, often called triangulations; see e.g., the survey articles [6,12,5]. A *triangulation* of a finite set S of points in the plane is a maximal planar straight-line graph that uses all and only the points in S as its vertices. Each face in a triangulation is a triangle spanned by S. In the last few years, a relaxation of triangulations, called *pseudo-triangulations* (or geodesic triangulations), has received considerable attention. Here, faces bounded by three concave chains, rather than by three line segments, are allowed. The scope of applications of pseudo-triangulations as a geometric data stucture ranges from ray shooting [10,14] and visibility [25,26] to kinetic collision detection [1,21,22], rigidity [32,29,15], and guarding [31]. Still, only very recently, results on the combinatorial properties of pseudo-triangulations have been obtained. These include bounds on the minimal vertex and face degree [20] and on the number of possible pseudo-triangulations [27,3]. The usefulness of (pseudo-)triangulations partially stems from the fact that these structures can be modified by constant-size combinatorial changes, commonly called *flip operations*. Flip operations allow for an adaption to local requirements, or even for generating globally optimal structures [6,12]. A classical result states that any two triangulations of a given planar point set can be made to coincide by applying a quadratic number of edge flips; see e.g. [16,19]. A similar result has been proved recently for the class of *minimum* pseudo-triangulations [8,29].

Results and outline. The present paper demonstrates that the quadratic bound for the number of required flip operations can be beaten drastically. We will provide two main results – for minimum pseudo-triangulations when using traditional flips operations, as well as for triangulations when a novel and natural edge flip operation is included into the repertoire of admissible flips. Extending the set

* Work done while this author was with the Institute for Theoretical Computer Science, Graz University of Technology, Graz, Austria. Research partially supported by APART [Austrian Programme for Advanced Research and Technology] of the Austrian Academy of Sciences.

** Research supported by the FWF [Austrian Fonds zur Förderung der Wissenschaftlichen Forschung]

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 12–24, 2003.
© Springer-Verlag Berlin Heidelberg 2003

of flips allows to transform pseudo-triangulations of arbitrary edge rank into each other, using a near-linear number of flips, and without changing the underlying set of vertices. A tool for rapidly adapting pseudo-triangulations (and in particular, triangulations) becomes available, using constant-size combinatorial changes. Various applications may be expected, in areas where (pseudo-)triangular meshes are of importance. In particular, the new flip type is indispensable for a proper treatment of spatial embeddings of pseudo-triangulations; see [2]. Section 2 starts with revisiting edge flip operations in pseudo-triangulations. The *edge-removing flip* is introduced, and is shown to have a geodesics interpretation consistent with classical flip types. Section 3 demonstrates that when edge-removing flips are admitted, the flip distance between any two pseudo-triangulations (and especially, triangulations) of a set of n points is reduced to $O(n \log n)$. In particular, any two triangulations of a simple polygon with n vertices can be transformed into each other with at most $2n - 6$ flips. This substantially improves over the situation without using the new type, where an $\Omega(n^2)$ lower bound for triangulations holds [16,19]. We also show that every given triangulation can be made minimum (i.e. pointed, see below) using $O(n)$ flips. In Section 4, we derive an $O(n \log^2 n)$ bound on the flip distance *within* the class of minimum pseudo-triangulations, that is, without applying the new flip type. This improves previous bounds of $O(n^2)$ in [8,29], and shows that the diameter of the high-dimensional polytope in [29] is $O(n \log^2 n)$. Our results partially rely on new partitioning results for pseudo-triangulations, in Section 5, which may be of separate interest. Section 6 discusses relations of the edge-removing flip to known existing types. In view of the lack of non-trivial lower bounds, a reduction of flip distances to $O(n)$ is left as an open problem.

Basic properties of pseudo-triangulations. This is a brief review of basic notions and properties concerning pseudo-triangulations. For more details, see e.g. [20,8,29]. For a (simple) polygon P in the plane, let $vert(P)$ denote the set of vertices of P. A *corner* of P is a vertex with internal angle less than π. The other vertices of P are called *non-corners*. The chain of edges between two consecutive corners of P is called a *side chain* of P. The *geodesic* between two points $x, y \in P$ is the shortest curve that connects x and y and lies inside P. A *pseudo-triangle* is a polygon with exactly three corners. Let S be a finite set of points in the plane. We will assume, throughout this paper, that S is in general position, i.e., no three points in S are collinear. Let $conv(S)$ denote the convex hull of S. A *pseudo-triangulation* of S is a partition of $conv(S)$ into pseudo-triangles whose vertex set is exactly S. A pseudo-triangulation is a face-to-face two-dimensional cell complex. The intersection of two faces (pseudo-triangles) may consist of up to two edges, however. In case of such *double-adjacencies*, the union of the two adjacent pseudo-triangles is a pseudo-triangle itself. Let $\mathcal{PT}$ be some pseudo-triangulation of S. A vertex of $\mathcal{PT}$ is called *pointed* if its incident edges lie in an angle smaller than π. Note that all vertices of $conv(S)$ are pointed. The more pointed vertices there are in $\mathcal{PT}$, the less edges and faces it has. In particular, $\mathcal{PT}$ contains exactly $3n - p - 3$ edges and $2n - p - 2$ pseudo-triangles, if $|S| = n$ and there are $p \leq n$ pointed vertices in $\mathcal{PT}$. We define the *edge rank* of $\mathcal{PT}$ as $n - p$. The minimum edge rank is zero, where $\mathcal{PT}$ is commonly called

a *minimum* (or a *pointed*) pseudo-triangulation. $\mathcal{PT}$ then is a maximal planar straight-line graph on S where all vertices are pointed; see e.g. [32]. It contains exactly $2n-3$ edges and $n-2$ pseudo-triangles. The edge rank expresses the excess in edges, compared to a minimum pseudo-triangulation. Its value is at most $n - |vert(conv(S))|$, which is attained if and only if $\mathcal{PT}$ is a triangulation.

2 Flips in Pseudo-Triangulations Revisited

2.1 Classical Flips

So-called *flips* are operations of constant combinatorial complexity which are commonly used to modify triangulations. The standard edge flip, also called *Lawson flip* [23], takes two triangles Δ_1 and Δ_2 whose union is a convex quadrilateral and exchanges its diagonals e and e'. To generalize to pseudo-triangulations, a different view of this edge flip is of advantage: Take the vertex of Δ_1 and Δ_2, respectively, that lies opposite to e and replace e by the geodesic between these two vertices. The geodesic is just a line segment e' in this case. The geodesics interpretation above has been used in [25,32] to define flips in minimum pseudo-triangulations. Let ∇_1 and ∇_2 be two adjacent pseudo-triangles, and let e be an edge they have in common. A flip replaces e by the part contributed by the geodesic inside $\nabla_1 \cup \nabla_2$ that connects the two corners of ∇_1 and ∇_2 opposite to e. In a minimum pseudo-triangulation each vertex is pointed, so the geodesic indeed contributes a line segment e' which is no edge of ∇_1 or ∇_2. See Figure 1(a) and (b), where the edge e to be flipped is shown in bold. Note that the flipping partners e and e' may cross or not. In either case, the flip creates two valid pseudo-triangles. We refer to such flips as *exchanging* flips. Each internal edge in a minimum pseudo-triangulation is flippable in this way. In a pseudo-triangulation of non-zero edge rank, however, edges incident to non-pointed vertices may be *non-flippable* in this sense. In particular, in a full triangulation, an internal edge is non-flippable if and only if its two incident triangles form a non-convex quadrilateral; see Figure 1(c). Non-flippable edges have been the source for the theoretically poor behavior of certain flipping algorithms, concerning the flip distance [16,19] as well as the non-existence of flip sequences [11].

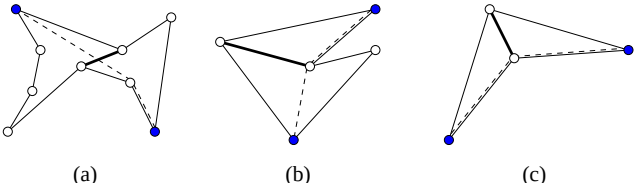


Fig. 1. Exchanging flips and non-flippable edge

2.2 A Novel Flip Type

We wish to generalize the edge flip so as to cover the situation in Figure 1(c) as well. In fact, when being consistent with the geodesics rule above, flipping a non-flippable edge $e = \nabla_1 \cap \nabla_2$ means removing e because its substitute is empty. A pseudo-triangle $\nabla_1 \cup \nabla_2$ is obtained. We include this *edge-removing* flip (and its inverse, the *edge-inserting* flip) into the repertoire of admissible flips. By definition, an edge-removing flip is applicable only if a valid pseudo-triangle is created. That is, a single non-pointed vertex of the pseudo-triangulation is made pointed by the flip. This simple modification makes each internal edge in every pseudo-triangulation (and in particular, in every triangulation) flippable. Note that edge-removing flips decrement the edge rank, whereas edge-inserting flips increment it. This allows for 'surfing' between pseudo-triangulations of different edge ranks. Several interesting consequences will be discussed, including a reduction of flip distances in Section 3, and relations to other flip types in Section 6.

Remarks. Edge-removing flips arise implicitly in a greedy flip algorithm for pseudo-triangulations of *convex objects*, in [26]. Certain flips that exchange bitangents of such objects cause an edge removal (or insertion) in the corresponding pseudo-triangulation for the object centers.[1]

3 Reducing the Flip Distance

Let S be a set of n points in the plane. It is well known that $\Theta(n^2)$ Lawson flips may be necessary, and are also sufficient, to transform two given triangulations of S into each other; see e.g. [16,19]. The upper bound also applies to exchanging flips in minimum pseudo-triangulations, see [8,29], but no non-trivial lower bounds are known in this case. For our admissible set of flip operations, several results will be shown in this section.

3.1 Simple Polygons

We start with proving that flip distances become linear between pseudo-triangulations of (simple) polygons, when edge-removing flips and their inverses are allowed. Consider a polygon P in the plane. The *shortest-path tree* of P with root $v \in vert(P)$ is the union of all geodesics in P from $vert(P)$ to v. Let $\pi_v(P)$ denote this structure. It is well known [17] that $\pi_v(P)$ is a tree that partitions P into pseudo-triangles in a unique way.

Lemma 1. *Let P be a polygon with n vertices, and let $v \in vert(P)$. The shortest-path tree $\pi_v(P)$ can be constructed by triangulating P arbitrarily, and applying at most $n - 3$ exchanging or edge-removing flips.*

[1] We recently learned that Orden and Santos [24] also considered this type of flip, to obtain a polytope representation of all possible pseudo-triangulations of a given point set.

Proof. Fix some triangulation $\mathcal{T}$ of P. We prove the assertion by induction on the number of triangles of $\mathcal{T}$. As an induction base, let Q be the union of all triangles of $\mathcal{T}$ incident to the vertex v. Clearly, the restriction of $\mathcal{T}$ to Q just gives $\pi_v(Q)$. We show that this invariant can be maintained by flipping, when an adjacent triangle Δ of $\mathcal{T}$ is added to Q. Let u be the vertex of Δ that does not belong to Q. Consider the unique edge $e = Q \cap \Delta$ (which is a diagonal of P). If e belongs to $\pi_v(Q \cup \Delta)$ then an edge of Δ connects u to $\pi_v(Q)$, and $\pi_v(Q \cup \Delta)$ is already complete. No flip is performed. Else let ∇ denote the unique pseudo-triangle in $\pi_v(Q)$ that is adjacent to Δ at e. There are two cases. If $\nabla \cup \Delta$ is a pseudo-triangle then, again, u is connected to $\pi_v(Q)$ by Δ. Perform a flip that removes e, which restores $\pi_v(Q \cup \Delta)$. Otherwise, let w be the corner of ∇ opposite to e. Apply an exchanging flip to e. The new edge e' lies on the geodesic between u and w. Thus e' connects u to $\pi_v(Q)$, which constructs $\pi_v(Q \cup \Delta)$ in this case. The total number of flips is at most $n - 3$, because each flip can be charged to the triangle of $\mathcal{T}$ that is added. □

Corollary 1. *Any two triangulations of a polygon P with n vertices can be flipped into each other by at most $2n - 6$ exchanging, edge-removing, or edge-inserting flips.*

Proof. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two triangulations of P. Choose some $v \in vert(P)$ and flip $\mathcal{T}_1$ to $\pi_v(P)$. Then flip $\pi_v(P)$ to $\mathcal{T}_2$ by reversing the sequence of flips that transforms $\mathcal{T}_2$ to $\pi_v(P)$. This is possible and takes at most $2n - 6$ flips, by Lemma 1. □

Corollary 1 implies a flip distance of $O(n)$ between *any* two pseudo-triangulations $\mathcal{PT}_1$ and $\mathcal{PT}_2$ of a given polygon P, because $\mathcal{PT}_1$ and $\mathcal{PT}_2$ can be completed to triangulations of P with $O(n)$ edge-inserting flips.

3.2 Planar Point Sets

We continue with pseudo-triangulations of planar point sets. In fact, we choose a slightly more general scenario, namely a point set enclosed by an arbitrary simple polygon (a so-called augmented polygon). This setting will turn out to be more appropriate for our developments, as it arises naturally from constraining the pseudo-triangulated domain. We will show how to flip any given pseudo-triangulation into a canonical one, by splitting the underlying augmented polygon in a balanced way, until empty polygons are obtained and Corollary 1 applies. Let P be a polygon, and consider a finite point set $S \subset P$ with $vert(P) \subseteq S$. We call the pair (P, S) an *augmented* polygon. A pseudo-triangulation $\mathcal{PT}$ of (P, S) is a partition of P into pseudo-triangles whose vertex set is exactly S. It contains exactly $3n - m + k - p - 3$ edges and $2n - m + k - p - 2$ pseudo-triangles if $|S| = n$, P is an m-gon with k corners, and p counts the pointed vertices of $\mathcal{PT}$. The maximum edge rank of $\mathcal{PT}$ is $n - k$. In the special case $P = conv(S)$, we have $m = k$ and deal with pseudo-triangulations of the point set S. Below is another corollary to Lemma 1.

Corollary 2. *Let $\mathcal{T}$ be a (full) triangulation of an augmented polygon (P, S). Let e be some line segment spanned by S, which lies inside P and crosses $\mathcal{T}$ at $j \geq 1$ edges. Then $\mathcal{T}$ can be modified to a triangulation that contains e by applying $O(j)$ exchanging, edge-removing, or edge-inserting flips.*

Proof. Let Q be the union of the triangles of $\mathcal{T}$ that are crossed by e. Note that Q may contain points of S in its interior, or even may contain holes, namely if Q contains internal edges which do not cross e. In this case, we cut Q along these edges, and move Q apart infinitesimally at the cuts, to obtain a simple polygon empty of points in S. This is possible because general position is assumed for S. Now choose any triangulation $\mathcal{T}_e$ of Q which includes the edge e to be integrated. By Corollary 1, the part of $\mathcal{T}$ inside Q can be flipped to $\mathcal{T}_e$ by $O(j)$ flips. □

We are now prepared to prove the following general assertion on flip distances.

Theorem 1. *Any two pseudo-triangulations of a given planar point set S (or more generally, of a given augmented polygon (P, S)) can be transformed into each other by applying $O(n \log n)$ flips of the types exchanging, edge-removing, and edge-inserting, for $n = |S|$.*

Proof. The two pseudo-triangulations of the augmented polygon (P, S) in question can be completed to triangulations by applying $O(n)$ edge-inserting flips. We show how to transform two arbitrary triangulations $\mathcal{T}_1$ and $\mathcal{T}_2$ of (P, S) into the same, using $O(n \log n)$ flips. Let P be an m-gon. If $m = n$ then $O(n)$ flips suffice by Corollary 1. Else we partition P into subpolygons, each containing at most $\frac{2}{3}(n - m)$ points of $S \setminus vert(P)$. A constant number of line segments spanned by S suffice for this purpose, by Theorem 4(1) in Section 5. Incorporate these segments into $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively, in $O(n)$ flips, which is possible by Corollary 2. Treat the obtained $O(1)$ augmented polygons recursively. This yields a polygonal partition of P whose vertex set is exactly S, and two triangulations thereof, in $O(n \log n)$ flips. By Corollary 1, another $O(n)$ flips let these two triangulations coincide. □

Remarks. Theorem 1 demonstrates that flip distances are substantially reduced when using the new flip type. 'Shortcuts' via pseudo-triangulations with varying edge rank become possible. The interested reader may check that the constant involved in the $O(n \log n)$ term is small (less than 6). We conjecture that Theorem 1 can be improved to $O(n)$ flips, because the $\Omega(n^2)$ worst-case examples for Lawson flips in triangulations are based on (non-convex) polygons without internal points [19], an instance covered by Corollary 1 in $O(n)$ flips. All flips used in Theorem 1 are constant-size combinatorial operations, which can be carried out in $O(\log m)$ time each, if the size of the two pseudo-triangles involved is at most m; see e.g. [13]. This implies that any two (pseudo-)triangulations of a given set of n points can be adapted by local operations in $O(n \log^2 n)$ time – a result we expect to have various applications. It is well known that not every pseudo-triangulation can be made minimum by removing edges. It can only be made *minimal* in edge rank, and is termed thereafter in [1,20,30]. In particular, a minimal pseudo-triangulation may be a full triangulation, even when its vertices are not in convex position [30]. We can show the following:

Lemma 2. *Let $\mathcal{PT}$ be any pseudo-triangulation of a planar n-point set S. Then $\mathcal{PT}$ can be transformed into a minimum pseudo-triangulation of S with $O(n)$ exchanging, edge-removing, or edge-inserting flips.*

4 Minimum Pseudo-Triangulations

Our next aim is to provide a stronger version of Theorem 1, namely for minimum pseudo-triangulations and *without* using the new flip type. That is, we restrict ourselves to staying within the class of minimum pseudo-triangulations, and use exchanging flips exclusively. By extending and modifying the arguments in Subsections 3.1 and 3.2, we will arrive at a flip distance bound of $O(n \log^2 n)$.

4.1 Two Basic Tools

Let P be a polygon. The *minimum shortest-path tree* $\mu_c(P)$, for a fixed corner c of P, is the union of all geodesics inside P that lead from c to the *corners* of P. Observe that $\mu_c(P)$ defines a minimum pseudo-triangulation for P, which is a subset of the shortest-path tree $\pi_c(P)$. The proof of Lemma 1 now can be adapted easily, to show that every minimum pseudo-triangulation of P can be transformed into $\mu_c(P)$ by at most $n-3$ *exchanging* flips. The new flip type is not used here, because each edge is flippable in the classical sense. We obtain:

Lemma 3. *Let P be a polygon with k corners. Any two minimum pseudo-triangulations of P are transformable into each other by applying at most $2k-6$ exchanging flips.*

The following assertion (which we state here without proof) is a variant of Corollary 2, for minimum pseudo-triangulations.

Lemma 4. *Let $\mathcal{MPT}$ be a minimum pseudo-triangulation of an augmented polygon (P, S), and let G be some pointed planar straight-line graph on S and in P. Then $\mathcal{MPT}$ can be made to contain G by applying $O(nj)$ exchanging flips, if S has n vertices and $G \setminus P$ has j edges.*

4.2 Exchanging-Flip Distance Bound

Lemma 4 implies an $O(n^2)$ bound on the exchanging-flip distance in minimum pseudo-triangulations. The following theorem shows that this can be improved.

Theorem 2. *Let S be a set of n points in the plane, and let $\mathcal{MPT}_1$ and $\mathcal{MPT}_2$ be two minimum pseudo-triangulations of S. Then $\mathcal{MPT}_1$ can be transformed into $\mathcal{MPT}_2$ by applying $O(n \log^2 n)$ exchanging flips. No other flip types are used. The same result holds for augmented polygons (P, S).*

Proof. Consider an augmented polygon (P, S). We recursively split (P, S) in a balanced way, by applying Theorem 4(2) from Section 5. This constructs a polygonal partition Π of P whose vertex set is S, and where all vertices are pointed. Π is obtained by introducing $O(\log n)$ edges internal to P in each

recursive step, and the number of recursive steps is $O(\log n)$ as well. By Lemma 4, $O(n \log^2 n)$ exchanging flips are sufficient to make $\mathcal{MPT}_1$ and $\mathcal{MPT}_2$ contain all the edges of Π. Finally, Lemma 3 allows for adapting pseudo-triangles within the polygons of Π in $O(n)$ such flips. □

Remarks. Theorem 2 improves the recent bound of $O(n^2)$ in [8] for minimum pseudo-triangulations of point sets. Again, we conjecture that the truth is $O(n)$ flips.[2] In [29], the polytope, $\mathcal{M}(S)$, of minimum pseudo-triangulations of a point set S has been introduced. $\mathcal{M}(S)$ is a high-dimensional convex polytope. Its vertices correspond to all the minimum pseudo-triangulations of S, and its edges represent all possible exchanging flips. By Theorem 2, the diameter of $\mathcal{M}(S)$ is bounded by $O(n \log^2 n)$. There are examples where the transformation between two given minimum pseudo-triangulations is speeded up by using intermediate edge-inserting and edge-removing flips; see [18]. This indicates that flexibility of pseudo-triangulations does not only come from low edge rank, but also stems from the ability to change this parameter – using the new flip type.

5 Partitioning Results

This section presents some partitioning results concerning pseudo-triangulations, which have been referred to in Sections 3 and 4. The theorems in Subsections 5.1 and 5.2 might be of separate interest.

5.1 Pseudo-Triangulations with Small Cut

Let P be a simple polygon. Consider a pseudo-triangle $\nabla \subset P$ with vertices from $vert(P)$. ∇ is called *nice* if its three corners are corners of P. We define the *cut* of ∇ as the number of diagonals of P on ∇'s boundary. A polygon P is *pseudo-convex* if every geodesic inside P is a convex chain. A *corner tangent* is an inner tangent of P incident to at least one corner of P. A pseudo-convex polygon P is *strongly pseudo-convex* if no corner tangent exists for P. We state the following fact without proof.

Lemma 5. *Let P be a strongly pseudo-convex polygon with k corners. There exists a nice pseudo-triangle for P with cut $O(\log k)$.*

We are now ready to prove the following structural result for minimum pseudo-triangulations of simple polygons.

Theorem 3. *For every polygon P with n vertices, there exists a minimum pseudo-triangulation of P where each face has cut $O(\log n)$.*

[2] By a very recent result in [7], a flip distance of $O(n \log n)$ for minimum pseudo-triangulations of point sets is obtainable, using a different divide-and-conquer approach. This approach does not carry over to the more general case of augmented polygons, however.

Proof. We first partition P into strongly pseudo-convex polygons. Diagonals on non-convex geodesics and corner tangents are used, such that each introduced diagonal is incident to some corner in both polygons it bounds. (These diagonals will contribute to the cut of the final faces in the minimum pseudo-triangulation to be constructed, but their number is at most 6 per face.) Each strongly pseudo-convex polygon Q with more than 3 corners is partitioned further as follows. Integrate a nice pseudo-triangle ∇ with small cut for Q, whose existence is guaranteed by Lemma 5. Because ∇ is nice, it does not violate the pointedness of any vertex. Moreover, each diagonal of Q on ∇'s boundary is incident to two corners of the polygon it cuts off from Q. These polygons are partitioned recursively. A minimum pseudo-triangulation $\mathcal{MPT}$ of P results. Each face f of $\mathcal{MPT}$ has cut $O(\log n)$: A diagonal on f's boundary comes from Lemma 5 or is among the at most 6 edges incident to some corner of f. ◻

Remarks. Theorem 3 is asymptotically optimal. There exist polygons with n vertices where every minimum pseudo-triangulation contains some face with cut $\Omega(\log n)$; see [4]. The theorem is related to a result in [20] which shows, for every point set S, the existence of a minimum pseudo-triangulation with constant face complexity. Another related result, in [10], shows that every n-gon P admits a minimum pseudo-triangulation $\mathcal{MPT}$ such that each line segment interior to P crosses only $O(\log n)$ edges of $\mathcal{MPT}$.

5.2 Partition Theorem

We continue with a ham-sandwich type result for pseudo-triangles.

Lemma 6. *Let ∇ be a pseudo-triangle that contains a set M of i points in its interior. There exists a point $p \in M$ whose geodesics to two corners of ∇ divide M into two subsets of cardinality $\frac{2i}{3}$ or less.*

Proof. For each point $p \in M$, the geodesics from p to the three corners of ∇ partition ∇ into three pseudo-triangles (faces). Such a face f is called *sparse* if f encloses at most $\frac{2i}{3}$ points of M. We claim that, for each pair c, c' of corners of ∇, there exist at least $\frac{2i}{3} + 1$ sparse faces: Consider the sorted order of M, as given by the shortest-path tree with root c for M. The j-th point of M in this order spans a face that contains strictly less than j points. We conclude that there are at least $2\,i + 3$ sparse faces in total. So the mean number of sparse faces per point in M exceeds two, which implies that there exists a point $p \in M$ incident to three sparse faces. Among them, let f be the face that contains the most points, which are at least $\frac{i}{3}$. We take the two geodesics that span f to partition ∇. This yields two parts with at most $\frac{2i}{3}$ points each. ◻

Lemma 6 combines with Theorem 3 to the following partition theorem for augmented polygons.

Theorem 4. *Let (P, S) be an augmented polygon, and let $I = S \setminus vert(P)$. There exist polygonal partitions Π_1 and Π_2 of (P, S) such that (1) Π_1 uses $O(1)$*

line segments spanned by S, and assigns at most $\frac{2}{3} \cdot |I|$ points of I to each polygon (2) Π_2 uses $O(\log n)$ line segments spanned by S, assigns at most $\frac{2}{3} \cdot |I|$ points of I to each polygon, and guarantees the pointedness of each vertex of S.

Proof. To construct Π_1, let $\mathcal{T}$ be some triangulation of the polygon P. Call a polygon $Q \subset P$ *sparse* if Q contains at most $\frac{2}{3} \cdot |I|$ points of I. Let ∇ be any face of $\mathcal{T}$. If each part of $P \setminus \nabla$ is sparse then we are done, because we can partition P with ∇, and ∇ with two line segments as in Lemma 6 if ∇ is non-sparse. Otherwise, we continue with the face of $\mathcal{T}$ adjacent to ∇ in the (unique) non-sparse part of $P \setminus \nabla$, until the first condition is met. To construct Π_2 we proceed analogously, but use a minimum pseudo-triangulation $\mathcal{MPT}$ of P with face cuts bounded by $O(\log n)$. The existence of $\mathcal{MPT}$ is given by Theorem 3. The $O(\log n)$ edges of ∇ that are used to partition (P, S) retain the pointedness of all vertices, as do the two segments from Lemma 6 that may have to be used to split ∇. □

Remarks. The fraction $\frac{2}{3}$ in Lemma 6 is optimal, even if ∇ is a triangle. The set M may consist of three groups of $\frac{i}{3}$ points such that, for each choice of $p \in M$, the two groups not containing p end up in the same subset. Theorem 4 is similar in flavor to a result in [9], which asserts that any simple n-gon can be split by a diagonal into two subpolygons with at most $\frac{2}{3}n$ vertices.

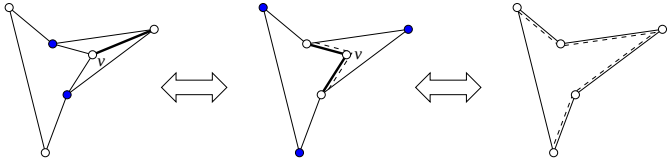


Fig. 2. Edge-removing and vertex-removing flips

6 Relation between Flip Types

The new flip type introduced in Section 2 can be used to simulate certain other flip types. Let us briefly comment on this fact. As an example, the exchanging flip in Figure 1(b) can be simulated by an edge-inserting flip followed by an edge-removing flip. Interestingly, this is not possible for the exchanging flip in Figure 1(a). A more important example concerns a flip that arises in the context of Delaunay and regular triangulations; see [28,11]. This flip inserts a new vertex v in the interior of a triangle Δ, and connects v by edges to the three vertices of Δ. Vertex insertion is meaningful for pseudo-triangulations as well [32, 3]. Connect v by geodesics to (at least two) corners of the pseudo-triangle ∇ that v lies in. Each geodesic contributes one edge incident to v, and ∇ is partitioned into (two or three) pseudo-triangles. The inverse operation, namely the removal of a degree-3 vertex v, can be simulated using edge-removing flips. See

Figure 2. Apply an edge-removing flip to one of v's edges first, which leaves a partition of ∇ into two pseudo-triangles in double-adjacency. Then, carry out two edge-removing flips *simultaneously*. This deletes v and leaves ∇ empty, because no edges are created by the geodesics rule. This simultaneous operation can be considered a single flip – the *vertex-removing* flip. By definition, such a flip is applicable to vertices of degree 2 only. Vertex-removing flips (as well as edge-removing flips) play an important role for surface realizations of pseudo-triangulations in three-space [2].

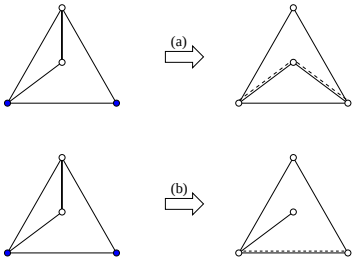


Fig. 3. Ambiguous geodesics interpretation

Remarks. Instead of the vertex-removing flip, a different version – namely the exchanging flip in Figure 3(a) – has been commonly used. It also leads to a valid pseudo-triangulation (which now does contain the vertex v). However, care has to be taken not to misinterpret this version as in Figure 3(b), where the geodesic still lies inside the union of the two pseudo-triangles involved. Also, this version conflicts with a three-dimensional interpretation of flips in surfaces [2]. When change in edge rank is conceded, we may circumvent the flip in Figure 3(a) by performing two consecutive flips of the new type, namely an edge-inserting flip followed by an edge-removing flip. Vertex-removing and vertex-inserting flips are not used in Theorems 1 and 2. Dropping this restriction makes things easy, because every pseudo-triangulation contains some vertex of constant degree, which can be removed with $O(1)$ flips. A flip distance of $O(n)$ is obvious in this setting. However, removing a vertex does not only change the vertex set S, but rather changes the underlying domain (the polygon P) if removal was for a boundary vertex. In contrast, in the setting for the theorems above, both S and P remain unchanged. The situation where S but not P is allowed to change is of some interest, because no internal vertex of constant degree might exist in a pseudo-triangulation. We are able to show the following:

Lemma 7. *Let S be a planar n-point set. Any two pseudo-triangulations of S can be flipped into each other in $O(n)$ flips, and without changing the underlying domain conv(S), if the entire repertoire of flips from Section 2 is used.*

Acknowledgements. We gratefully acknowledge discussions on the presented topic with Michel Pocchiola, Günter Rote, and Francisco Santos.

References

[1] P.K. Agarwal, J. Basch, L.J. Guibas, J. Hershberger, L. Zhang. Deformable free space tilings for kinetic collision detection. In B.R. Donald, K. Lynch, D. Rus (eds.), Algorithmic and Computational Robotics: New Directions (Proc. 5th Workshop Algorithmic Found. Robotics), 2001, 83–96.

[2] O. Aichholzer, F. Aurenhammer, P. Brass, H. Krasser. Spatial embedding of pseudo-triangulations. Proc. 19^{th} Ann. ACM Sympos. Computational Geometry 2003, to appear.

[3] O. Aichholzer, F. Aurenhammer, H. Krasser, B. Speckmann. Convexity minimizes pseudo-triangulations. Proc. 14^{th} Canadian Conf. Computational Geometry 2002, 158–161.

[4] O. Aichholzer, M. Hoffmann, B. Speckmann, C.D. Tóth. Degree bounds for constrained pseudo-triangulations. Manuscript, Institute for Theoretical Computer Science, Graz University of Technology, Austria, 2003.

[5] F. Aurenhammer, Y.-F. Xu. Optimal triangulations. In: P.M. Pardalos, C.A. Floudas (eds), Encyclopedia of Optimization 4, Kluwer Academic Publishing, 2000, 160–166.

[6] M. Bern, D. Eppstein. Mesh generation and optimal triangulation. In: D.-Z. Du, F. Hwang (eds), Computing in Euclidean Geometry, Lecture Notes Series on Computing 4, World Scientific, 1995, 47–123.

[7] S. Bespamyatnikh. Transforming pseudo-triangulations. Manuscript, Dept. Comput. Sci., University of Texas at Dallas, 2003.

[8] H. Brönnimann, L. Kettner, M. Pocchiola, J. Snoeyink. Counting and enumerating pseudo-triangulations with the greedy flip algorithm. Manuscript, 2001.

[9] B. Chazelle. A theorem on polygon cutting with applications. Proc. 23^{rd} IEEE Symp. FOCS, 1982, 339–349.

[10] B. Chazelle, H. Edelsbrunner, M. Grigni, L.J. Guibas, J. Hershberger, M. Sharir, J. Snoeyink. Ray shooting in polygons using geodesic triangulations. Algorithmica 12 (1994), 54–68.

[11] H. Edelsbrunner, N.R. Shah. Incremental topological flipping works for regular triangulations. Algorithmica 15 (1996), 223–241.

[12] S. Fortune. Voronoi diagrams and Delaunay triangulations. In: D.-Z. Du, F. Hwang (eds), Computing in Euclidean Geometry, Lecture Notes Series on Computing 4, World Scientific, 1995, 225–265.

[13] J. Friedman, J. Hershberger, J. Snoeyink. Efficiently planning compliant motion in the plane. SIAM J. Computing 25 (1996), 562–599.

[14] M.T. Goodrich, R. Tamassia. Dynamic ray shooting and shortest paths in planar subdivisions via balanced geodesic triangulations. J. Algorithms 23 (1997), 51–73.

[15] R. Haas, D. Orden, G. Rote, F. Santos, B. Servatius, H. Servatius, D. Souvaine, I. Streinu, W. Whiteley. Planar minimally rigid graphs and pseudo-triangulations. Proc. 19^{th} Ann. ACM Sympos. Computational Geometry, to appear.

[16] S. Hanke, T. Ottmann, S. Schuierer. The edge-flipping distance of triangulations. Journal of Universal Computer Science 2 (1996), 570–579.

[17] J. Hershberger. An optimal visibility graph algorithm for triangulated simple polygons. Algorithmica 4 (1989), 141–155.

[18] C. Huemer. Master Thesis, Institute for Theoretical Computer Science, Graz University of Technology, Austria, 2003.

[19] F. Hurtado, M. Noy, J. Urrutia. Flipping edges in triangulations. Discrete & Computational Geometry 22 (1999), 333–346.

[20] L. Kettner, D. Kirkpatrick, A. Mantler, J. Snoeyink, B. Speckmann, F. Takeuchi. Tight degree bounds for pseudo-triangulations of points. Computational Geometry: Theory and Applications 25 (2003), 3–12.
[21] D. Kirkpatrick, J. Snoeyink, B. Speckmann. Kinetic collision detection for simple polygons. Intern. J. Computational Geometry & Applications 12 (2002), 3–27.
[22] D. Kirkpatrick, B. Speckmann. Kinetic maintenance of context-sensitive hierarchical representations for disjoint simple polygons. Proc. 18^{th} Ann. ACM Sympos. Computational Geometry 2002, 179–188.
[23] C.L. Lawson. Properties of n-dimensional triangulations. Computer Aided Geometric Design 3 (1986), 231–246.
[24] D. Orden, F. Santos. The polyhedron of non-crossing graphs on a planar point set. Manuscript, Universidad de Cantabria, Santander, Spain, 2002.
[25] M. Pocchiola, G. Vegter. Minimal tangent visibility graphs. Computational Geometry: Theory and Applications 6 (1996), 303–314.
[26] M. Pocchiola, G. Vegter. Topologically sweeping visibility complexes via pseudo-triangulations. Discrete & Computational Geometry 16 (1996), 419–453.
[27] D. Randall, G. Rote, F. Santos, J. Snoeyink. Counting triangulations and pseudo-triangulations of wheels. Proc. 13^{th} Canadian Conf. Computational Geometry 2001, 117–120.
[28] V.T. Rajan, Optimality of the Delaunay triangulation in R^d. Discrete & Computational Geometry 12 (1994), 189–202.
[29] G. Rote, F. Santos, I. Streinu. Expansive motions and the polytope of pointed pseudo-triangulations. In: Discrete & Computational Geometry – The Goodman-Pollack Festschrift, B.Aronov, S.Basu, J.Pach, M.Sharir (eds.), Algorithms and Combinatorics, Springer, Berlin, 2003, 699–736.
[30] G. Rote, C.A. Wang, L.-Wang, Y.-Xu. On constrained minimum pseudo-triangulations. Manuscript, Inst. f. Informatik, FU-Berlin, 2002.
[31] B. Speckmann, C.D. Toth. Allocating vertex π-guards in simple polygons via pseudo-triangulations. Proc. 14^{th} ACM-SIAM Symposium on Discrete Algorithms, 2003, 109–118.
[32] I. Streinu. A combinatorial approach to planar non-colliding robot arm motion planning. Proc. 41^{st} IEEE Symp. FOCS, 2000, 443–453.

Shape Segmentation and Matching with Flow Discretization*

Tamal K. Dey[1], Joachim Giesen[2], and Samrat Goswami[1]

[1] The Ohio State U., Columbus, Ohio 43210, USA,
{tamaldey,goswami}@cis.ohio-state.edu
[2] ETH Zürich, CH-8092 Zürich, Switzerland
giesen@inf.ethz.ch

Abstract. Geometric shapes are identified with their features. For computational purposes a concrete mathematical definition of features is required. In this paper we use a topological approach, namely dynamical systems, to define features of shapes. To exploit this definition algorithmically we assume that a point sample of the shape is given as input from which features of the shape have to be approximated. We translate our definition of features to the discrete domain while mimicking the set-up developed for the continuous shapes. Experimental results show that our algorithms segment shapes in two and three dimensions into so-called features quite effectively. Further, we develop a shape matching algorithm that takes advantage of our robust feature segmentation step.

1 Introduction

The features of a shape are its specific identifiable subsets. Although this high level characterization of features is assumed routinely, more concrete and mathematical definitions are required for computational purposes. Many applications including object recognition, classification, matching, tracking need to solve the problem of segmenting a shape into its salient features, see for example [1,4,5,10]. Most of these applications need an appropriate definition of features that are computable. In the computational domains, often the shapes are represented with discrete means that approximate them. Consequently, a consistent definition of features in the discrete domain is needed to compute them reliably. In this paper we use a topological approach, namely dynamical systems, to define features of shapes. We assume that a point sample of the shapes is given as input from which features of the shape have to be approximated. We translate our definition of features to this discrete domain while mimicking the set-up that we develop in the continuous case. The outcome of this approach is a clean mathematical definition of features that are computable with combinatorial algorithms. For shapes in the plane we compute them exactly whereas we approximate them for shapes embedded in $\mathbb{R}^3$ mimicking the two dimensional algorithm. Our experimental results show that our algorithms segment shapes in two and three dimensions into so-called features quite effectively.

* This work is partially supported by NSF under grant DMS-0138456 with a subcontract from Stanford University and by IST(ECG) programme under contract no. IST-2000-26473.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 25–36, 2003.
© Springer-Verlag Berlin Heidelberg 2003

We apply our feature segmentation technique to the shape matching problem, where a similarity measure is sought between two shapes. Among the shape matching approaches (see e.g. [2,3,6,10,11,12,13,14]) feature based approaches depend mainly on the quality of the feature detection step. We give a shape matching algorithm that takes the advantage of our robust feature segmentation step. Each significant feature segment is represented with a weighted point where the weight is the volume of the segment. Then, the shape matching problem boils down to matching two small weighted point sets. We carry out these steps so that the entire matching process remains invariant to rotation, translation, mirroring and scaling.

2 Flow and Critical Points

In shape segmentation and shape matching we deal with continuous shapes Σ. Typically these shapes are bounded by one- or two dimensional manifolds embedded in $\mathbb{R}^2$ or $\mathbb{R}^3$ respectively. In this section we outline a theory of the flow induced by a shape. Later we will use this theory to define and compute features of shapes. Here we will develop the theory in a more general setting by considering general shapes embedded in d-dimensional Euclidean space $\mathbb{R}^d$.

Height function. In the following Σ always denotes a compact subset of $\mathbb{R}^d$. The set Σ can be used to define a distance function $h : \mathbb{R}^d \to \mathbb{R}$ as

$$h(x) \;=\; \inf_{p\in\Sigma} \|p-x\|^2 \;\text{ for all } x \in \mathbb{R}^d.$$

Anchor set. Associated with the distance function, we define an *anchor set* for each point $x \in \mathbb{R}^d$ as $A(x) \;=\; \operatorname{argmin}_{p\in\Sigma} \|p-x\|$.

Basically, $A(x)$ is the set of closest point to x in Σ; see Figure 1. Note that $A(x)$ can contain even a continuum of points.

We would like to define a unit vector field $v : \mathbb{R}^d \to \mathbb{R}^d$ that assigns to every point $x \in \mathbb{R}^d$ the direction in which the distance function increases the most. If h is smooth at x then $v(x)$ coincides with the normalized gradient $\nabla h(x)/\|\nabla h(x)\|$. In our case h is not smooth everywhere. So, we have to be careful to define $v(x)$ at any non-smooth point x. Instead of smooth and non-smooth points we will talk about regular and critical points in the following. Critical points are either local extrema or saddle points of the distance function. We use a generalized theory of critical points [9] to derive the following definition.

Regular and critical point. For every point $x \in \mathbb{R}^d$ let $H(x)$ be the convex hull of $A(x)$, i.e. the convex hull of the points on Σ that are closest to x. We call x a critical point of h if $x \in H(x)$. Otherwise we call x a regular point.

The following definition turns out to be very helpful in the subsequent discussion. It allows us to characterize the direction of steepest ascent of the distance function h at every point $x \in \mathbb{R}^d$.

Driver. For any point $x \in \mathbb{R}^d$ let $d(x)$ be the point in $H(x)$ closest to x. We call $d(x)$ the driver of x.

We leave the proof of the following lemma for the full version of this paper.

Lemma 1. *For any regular point $x \in \mathbb{R}^d$ let $d(x)$ be the driver of x. The steepest ascent of the distance function h at x is in the direction of $x - d(x)$.*

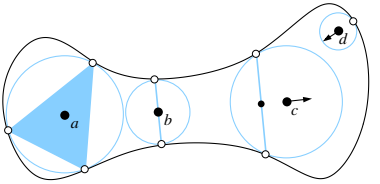


Fig. 1. In this example Σ is a curve embedded in $\mathbb{R}^2$. The sets $A(x)$ are shown with hollow circles for four points $x = a, b, c, d \in \mathbb{R}^2$. The convex hulls of $A(x)$ are light shaded. The driver of the point c is the smaller black circle. The driver of the point d is the single point in $A(d)$. The points a and b are critical since they are contained in $H(a)$ and $H(b)$, respectively. The points c and d are regular. The direction of steepest ascent of the distance function at c and d is indicated by an arrow.

We are now going to use the direction of steepest ascent to define a flow on $\mathbb{R}^d$, i.e. a dynamical system on $\mathbb{R}^d$.

Induced flow. Define a vector field $v : \mathbb{R}^d \to \mathbb{R}^d$ by setting

$$v(x) = \frac{x - d(x)}{\|x - d(x)\|} \text{ if } x \neq d(x) \text{ and } 0 \text{ otherwise.}$$

The flow induced by the vector field v is a function $\phi : [0, \infty) \times \mathbb{R}^d \to \mathbb{R}^d$ such that the right derivative at every point $x \in \mathbb{R}^d$ satisfies the following equation

$$\lim_{t \downarrow t_0} \frac{\phi(t, x) - \phi(t_0, x)}{t - t_0} = v(\phi(t_0, x)).$$

Orbits and fixpoints. Given $x \in \mathbb{R}^d$ and an induced flow ϕ, the curve $\phi_x : [0, \infty) \to \mathbb{R}^d$, $t \mapsto \phi(t, x)$ is called the *orbit* of x. A point $x \in \mathbb{R}^d$ is called a *fixpoint* of ϕ if $\phi(t, x) = x$ for all $t \geq 0$.

Basically, the orbit of a point is the curve it will follow if it were let move along the flow.

Observation 1 *The fixpoints of ϕ are the critical points of the distance function h.*

Because of this observation we refer to a fixpoint of ϕ as a minimum, saddle or maximum if the corresponding critical point of the distance function is a minimum, saddle or maximum, respectively.

Stable manifold. The stable manifold $S(x)$ of a critical point x is the set of all points that flow into x, i.e. $S(x) = \{y \in \mathbb{R}^d : \lim_{t\to\infty} \phi_y(t) = x\}$. The stable manifolds of all critical points partition $\mathbb{R}^d$, i.e. $\mathbb{R}^d = \bigcup_{\text{critical points } x} S(x)$ and $S(x) \cap S(y) = \emptyset$ for any two different critical points x and y.

3 Discretization

To deal with continuous shapes algorithmically we discretize them. Here discretization means taking a finite sample P of the shape $\Sigma \subset \mathbb{R}^d$. That is, we replace Σ by a finite subset of Σ. The sample P induces another vector field which resembles the vector field induced by Σ provided P is sufficiently dense in Σ. The vector field induced by P is intimately linked with the Voronoi- and the Delaunay diagram of P. Moreover, the stable manifolds corresponding to the flow induced by this vector field are efficiently computable in dimensions two and three. Let us first summarize the definitions of Voronoi- and Delaunay diagrams before we show how the concepts we introduced in the last section can be specialized to the case of finite point sets.

Voronoi diagram. Let P be a finite set of points in $\mathbb{R}^d$. The *Voronoi cell* of $p \in P$ is given as $V_p = \{x \in \mathbb{R}^d : \forall q \in P - \{p\},\ \|x-p\| \leq \|x-q\|)\}$. The sets V_p are convex polyhedra or empty since the set of points that have the same distance from two points in P forms a hyperplane. Closed facets shared by $k, 2 \leq k \leq d$, Voronoi cells are called $(d-k+1)$-dimensional *Voronoi facets* and points shared by $d+1$ or more Voronoi cells are called *Voronoi vertices*. The term *Voronoi object* denotes either a Voronoi cell, facet, edge or vertex. The *Voronoi diagram* V_P of P is the collection of all Voronoi objects. It defines a cell decomposition of $\mathbb{R}^d$.

Delaunay diagram. The *Delaunay diagram* D_P of a set of points P is dual to the Voronoi diagram of P. The convex hull of $d+1$ or more points in P defines a *Delaunay cell* if the intersection of the corresponding Voronoi cells is not empty and there exists no superset of points in P with the same property. Analogously, the convex hull of $k \leq d$ points defines a $(k-1)$-dimensional *Delaunay face* if the intersection of their corresponding Voronoi cells is not empty. Every point in P is called *Delaunay vertex*. The term *Delaunay object* denotes either a Delaunay cell, face, edge or vertex. The Delaunay diagram D_P defines a decomposition of the convex hull of all points in P. This decomposition is a triangulation if the points are in general position.

We always refer to the interior and to the boundary of Voronoi-/Delaunay objects with respect to their dimension, e.g. the interior of a Delaunay edge contains all points in this edge besides the endpoints. The interior of a vertex and its boundary are the vertex itself. Furthermore, we always assume general position unless stated differently.

Now consider the distance function h as in the previous section but replacing Σ with its discrete sample P. Define critical points for h as we did in the continuous case.

Lemma 2. *Let P be a finite set of points such that Voronoi- and their dual Delaunay objects intersect in their interiors if they intersect at all. Then the critical points of*

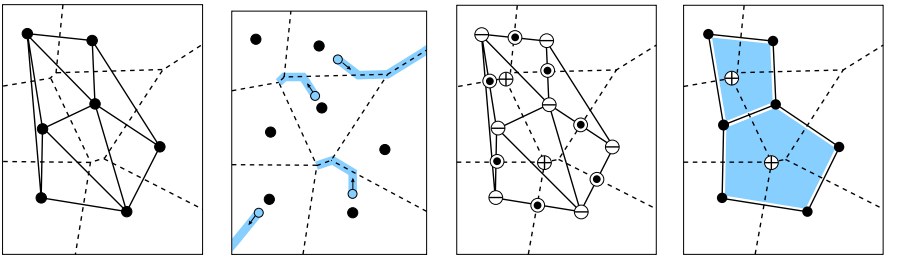


Fig. 2. Left: The Voronoi diagram (dashed lines) and the Delaunay triangulation (solid lines) of seven points in $\mathbb{R}^2$. Middle left: Some orbits of the flow induced by the points. Middle right: The critical points (maxima $\oplus$, saddle points $\odot$ and minima $\ominus$) of the distance function induced by the seven points. Right: The stable manifolds of the maxima $\oplus$ of the flow induced by the seven points.

the distance function h are the intersection points of Voronoi objects V and their dual Delaunay object σ.

This characterization of critical points can be used to assign a meaningful index to critical points, namely, the index of a critical point is the dimension of the Delaunay object used in the above characterization, see also [8]. Minima always have index 0 and maxima always have index d.

The driver of a point in $\mathbb{R}^d$ can now also be described in terms of Voronoi- and Delaunay objects.

Lemma 3. *Given $x \in \mathbb{R}^d$. Let V be the lowest dimensional Voronoi object in the Voronoi diagram of P that contains x and let σ be the dual Delaunay object of V. The driver of x is the point on σ closest to x.*

We have a much more explicit characterization of the flow induced by a finite point set than in the general case.

Observation 2 *The flow ϕ induced by a finite point set P is given as follows. For all critical points x of the distance function associated with P we set $\phi(t, x) = x\ ,\ t \in [0, \infty)$. Otherwise let $d(x)$ be the driver of x and R be the ray originating at x and shooting in the direction $v(x) = x - d(x)/\|x - d(x)\|$. Let z be the first point on R whose driver is different from $d(x)$. Note that such a z need not exist in $\mathbb{R}^d$ if x is contained in an unbounded Voronoi object. In this case let z be the point at infinity in the direction of R. We set $\phi(t, x) = x + t \cdot v(x)\ ,\ t \in [0, \|z - x\|)$. For $t \geq \|z - x\|$ the flow is given as*

$$\begin{aligned}\phi(t, x) &= \phi\left(t - \|z - x\| + \|z - x\|,\ x\right) \\ &= \phi\left(t - \|z - x\|,\ \phi\left(\|z - x\|,\ x\right)\right).\end{aligned}$$

It is not completely obvious, but it can be shown that this flow is well defined [8]. It is also easy to see that the orbits of ϕ are piecewise linear curves that are linear in Voronoi objects. See Figure 2 for some examples of orbits.

Under some mild non-degeneracy condition the stable manifolds of the critical points have a nice recursive structure. A stable manifold of index $k, 0 \leq k \leq d$, has dimension k and its boundary is made up from stable manifolds of index $k-1$ critical points.

In $\mathbb{R}^2$ the stable manifolds of index 1 critical points, i.e. saddle points, are exactly the Delaunay edges whose circumcircle is empty. They form the Gabriel graph of the point set P. The Gabriel graph is efficiently computable. The recursive structure of the stable manifolds now tells us that the stable manifolds of the maxima, i.e. index 2 critical points, are exactly the compact regions of the Gabriel graph. That is, the stable manifolds of maxima (index 2 critical points) are given as a union of Delaunay triangles.

The stable manifolds of flows induced by finite point sets in $\mathbb{R}^3$ can also be computed efficiently, see [8]. But already in $\mathbb{R}^3$ the stable manifolds of index 2 saddle points and maxima are not given as sub-complexes of the three dimensional Delaunay triangulation. Nevertheless, we will show in the next section that these stable manifolds can be approximated by sub-complexes of the Delaunay triangulation.

4 Approximating Stable Manifolds

Our goal is to decompose a two or three dimensional shape Σ into disjoint segments that respect the 'features' of the shape. In our first attempt to define features we resort to stable manifolds of maxima. So, we define a feature to be the closed stable manifold $F(x)$ of a maximum x,

$$F(x) = \text{closure}\,(\,S(x)).$$

Figure 3(a) shows the segmentation of a shape in $\mathbb{R}^2$ with this definition of features. We can translate this definition to the discrete setting immediately as we have mimicked all concepts of the continuous case in the discrete setting. Figure 3(b) shows this segmentation. From a point sample P of a shape Σ we would like to compute $F(x)$ for all maxima x. These maxima are a subset of the Voronoi vertices in V_P. For computing the feature segmentation it is sufficient to compute the boundary of all such $F(x)$. As we observed earlier this boundary is partitioned by the stable manifolds of critical points of lower index. In $\mathbb{R}^2$ this means that Gabriel edges separate the features.

We also want to separate the features in $\mathbb{R}^3$ by a subset of the Delaunay triangles. That is, we want to approximate the boundary of the stable manifolds of maxima by Delaunay triangles. These boundaries are made up from stable manifolds of critical points of index 1 and 2. The closures of the stable manifolds of index 1 critical points are again exactly the Gabriel edges. By Lemma 2 each critical point of index 2 lies in a Delaunay triangle which we call a *saddle* triangle. The stable manifolds of the index 2 critical points may not be contained only in the saddle triangles. This makes computing the boundary of the stable manifolds of maxima harder in $\mathbb{R}^3$. Although it can be computed exactly, we propose an alternative method that approximates this boundary using only Delaunay triangles. We derive this method by generalizing a simple algorithm that computes the closed stable manifolds for maxima in $\mathbb{R}^2$ exactly.

In $\mathbb{R}^2$ we can compute the closed stable manifold $F(x)$ of a maximum x by exploring out from the Delaunay triangle containing x. To explain the algorithm we define a flow relation among Delaunay triangles which was proposed by Edelsbrunner et al. [7] for computing pockets in molecules.

Flow relation in $\mathbb{R}^2$. Let σ_1, σ_2 be two triangles that share an edge e. We say $\sigma_1 < \sigma_2$ if σ_1 and its dual Voronoi vertex lie on the opposite sides of the supporting line of e.

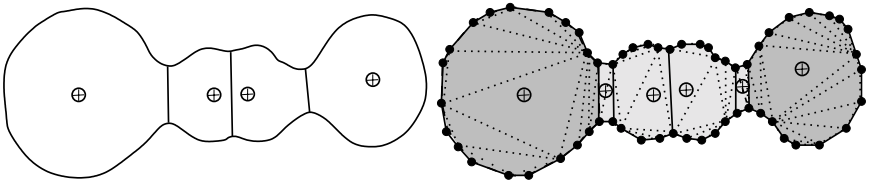
Observation 3 *Let σ_1 and σ_2 be two triangles sharing an edge e where $\sigma_1 < \sigma_2$. Then the flow on the dual Voronoi edge v_1v_2 of e is directed from v_1 to v_2 where v_i is the dual Voronoi vertex of σ_i.*

It is obvious from the definition that the transitive closure $<^*$ of $<$ is acyclic [7]. If $\sigma_1 < \sigma_2$, then the radius of the circumcircle of σ_2 is larger than the radius of the circumcircle of σ_1. So, in a chain of triangles related by $<$ relation the circumradii of the triangles can never decrease, thus making it impossible for $<^*$ to be cyclic. This means that, for each triangle σ', there is a triangle σ containing a maximum x such that $\sigma' <^* \sigma$. We will say that σ' flows into σ.

The following lemma holds in $\mathbb{R}^2$.

Lemma 4. *Let σ be a triangle containing a maximum x. We have $F(x) = \bigcup_{\sigma' <^* \sigma} \sigma'$.*

The algorithm, orginally proposed by Edelsbrunner et al. [7], for computing the closed stable manifold $F(x)$ follows immediately from the above lemma. Initially $F(x)$ is set to the triangle σ that contains x. At any generic step of this exploration, let e be a Delaunay edge that lies on the boundary of $F(x)$ computed so far. Let σ_1 and σ_2 be two triangles that share e where σ_1 is outside $F(x)$. If $\sigma_1 < \sigma_2$ we update $F(x)$ as $F(x) := F(x) \cup \sigma_1$. This process continues till we cannot include any more triangle into $F(x)$.



(a) (b)

Fig. 3. The closed stable manifolds of the maxima decompose the interior of the curve into four segments, the middle two segments are mergeable (a). The discretized version has six segments. The Gabriel edges (solid) among the Delaunay edges (dashed) form the boundaries of these segments. All four middle segments can be merged into a single segment.

Now we turn our attention to $\mathbb{R}^3$. In our attempt to compute $F(x)$ for a maximum x in $\mathbb{R}^3$, we mimic the setup that we used in $\mathbb{R}^2$.

Flow relation in $\mathbb{R}^3$. Let σ_1, σ_2 be two tetrahedra sharing a triangle t. We say $\sigma_1 < \sigma_2$ if σ_1 and its dual Voronoi vertex lie on the opposite sides of the plane of t.

It follows from the definition of $<$ that if $\sigma_1 < \sigma_2$, then the radius of the circumsphere of σ_1 is smaller than the radius of the circumsphere of σ_2. Thus, as in $\mathbb{R}^2$, the transitive closure $<^*$ is acyclic. For a maximum x let

$$\widetilde{F}(x) = \bigcup_{\sigma' < \sigma} \sigma' \text{ where } x \in \sigma.$$

So far everything seems analogous to the two dimensional case, but here we face two difficulties. First, Lemma 4 is no more valid, i.e. it may be that $F(x) \neq \widetilde{F}(x)$ for a maximum x. This is mainly because the stable manifolds of index 2 critical points may not be composed of Delaunay triangles. However, we could use $\widetilde{F}(x)$ as an approximation to $F(x)$. But, we face another difficulty. It might be that $\widetilde{F}(x)$ and $\widetilde{F}(x')$ are not disjoint for two maxima x and x'. The reason is that, for a tetrahedron σ, there may exist more than one tetrahedron σ' so that $\sigma < \sigma'$. This may lead σ to flow into two or more different maxima. However, it is interesting to notice the following.

Observation 4 *There exist no three tetrahedra $\sigma_1, \sigma_2, \sigma_3$ so that a tetrahedron σ satisfies $\sigma < \sigma_i$ for $i = 1, 2, 3$.*

In order to get pairwise disjoint sets $\widetilde{F}(x)$ we change the relation $<$ to a new relation $\widetilde{<}$ so that for a tetrahedron σ there are no two tetrahedra σ_1, σ_2 with $\sigma \widetilde{<} \sigma_1$ and $\sigma \widetilde{<} \sigma_2$. Note that the *height* of a maximum x, i.e. its least squared distance to the sample points P, is the circumradius of the tetrahedron containing x. Define the *strength* of a tetrahedron σ as the largest of the heights of all maxima that it flows into.

Strengthened flow relation. We say $\sigma_1 \widetilde{<} \sigma_2$ if

(1) $\sigma_1 < \sigma_2$ and
(2) no other tetrahedron σ_3 exists with $\sigma_1 < \sigma_3$ and the strength of σ_3 is larger than σ_2.

The transitive closure $\widetilde{<}^*$ is acyclic since $<^*$ is. Now for a maximum x we redefine $\widetilde{F}(x)$ as

$$\widetilde{F}(x) = \bigcup_{\sigma' \widetilde{<}^* \sigma} \sigma' \text{ where } x \in \sigma.$$

The sets $\widetilde{F}(x)$ are pairwise disjoint since no tetrahedron can flow into more than one maximum. We compute these sets as an initial segmentation of the shape represented by the finite sample P. Most of the segments but not all of them lie in the interior of the shape. One can obtain only inner segments after reconstructing the boundary of the shape from its sample using any of the known surface reconstruction algorithms.

We sort the maxima in decreasing order of their strengths and process them in this order. So, when we process a maximum x, all tetrahedra flowing into x and having

a strength larger than that containing x have been claimed by some other maxima processed earlier. This is what is required by the definition of $\widetilde{F}(x)$.

```
StableManifold(P)
 1  compute V_P and D_P;
 2  determine the maxima among the Voronoi vertices;
 3  sort the maxima in decreasing order of their heights;
 4  for each maximum x in this order
 5     F̃(x) := σ where x ∈ σ;
 6     mark σ and all its triangles;
 7     while ∃ an unmarked triangle t in boundary F̃(x)
 8        let t = σ1 ∩ σ2 where σ2 ∈ F̃(x);
 9        if σ1 is unmarked and σ1 < σ2
10           F̃(x) := F̃(x) ∪ σ1;
11           mark σ1 and all its triangles
12        endif
13     endwhile
14  endfor
```

Sometimes closed stable manifolds segment a shape unnecessarily into small features. For example, small perturbations in a shape can cause insignificant segmentation, see for example Figure 3(a). Also, at the discrete level, sampling artifacts may introduce even smaller segments, see Figure 3(b). We propose merging such small segments till two adjacent segments differ significantly.

For a shape $\Sigma \subseteq \mathbb{R}^d$, let $S(x)$ be a stable manifold of an index $d-1$ critical point x which belongs to the boundary of a closed stable manifold $F(y)$ for a maximum y. We say $F(y)$ is *shallow* with respect to $S(x)$ if the distances $h(x)$ and $h(y)$ are close to each other measured by a threshold $\rho < 1$ as $h(x)/h(y) \geq \rho$. Two closed stable manifolds $F(x_1)$ and $F(x_2)$ are *mergeable* if they are shallow with respect to a shared stable manifold on their boundaries.

Merging all mergeable closed stable manifolds we obtain the feature segmentation of Σ. For example, the middle two segments for the curve in Figure 3(a) are mergeable. We can translate the definitions and hence the merging algorithm to the discrete setting easily. The distance of a critical point x is measured with the circumradius of the lowest dimensional Delaunay simplex that contains x. This means, in $\mathbb{R}^2$, we merge two closed stable manifolds $F(x_1)$ and $F(x_2)$ if they share a saddle edge whose circumradius is more than $\rho < 1$ times the circumradii of the triangles containing x_1 and x_2.

In $\mathbb{R}^3$, we only compute approximations $\widetilde{F}(x)$ to a closed stable manifold $F(x)$ for a maximum x. Mimicking the definition and the algorithm in $\mathbb{R}^2$ we define mergeability of two approximated stable manifolds as follows.

ρ-mergeable stable manifolds. Let $\widetilde{F}(x_1)$ and $\widetilde{F}(x_2)$ be two approximated stable manifolds that share a triangle t. We say $\widetilde{F}(x_1)$, $\widetilde{F}(x_2)$ are ρ-mergeable if the circumradius of t is more than $\rho < 1$ times the circumradii of the tetrahedra containing x_1 and x_2.

The final algorithm to compute a feature segmentation of a shape $\Sigma \subseteq \mathbb{R}^3$ from a sample P is described below. Some examples are shown in Figure 4.

SEGMENT(P, ρ)
1 STABLEMANIFOLD(P);
2 Merge all ρ-mergeable segments and output the resulting decomposition.

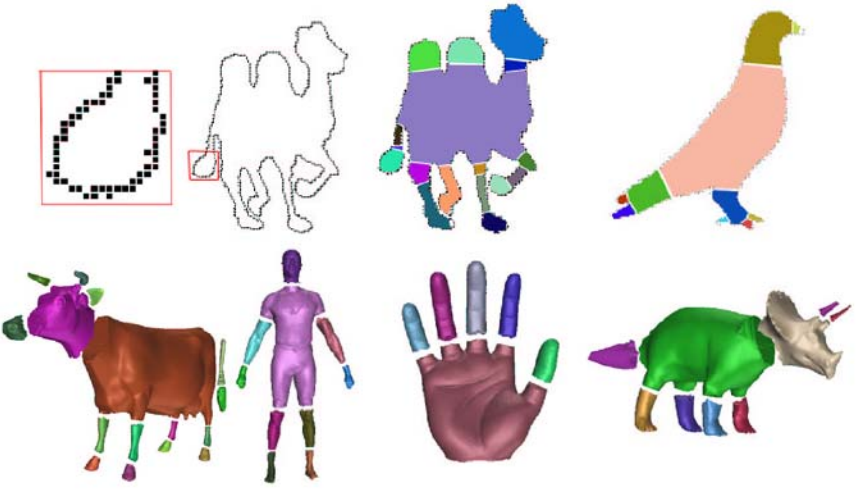


Fig. 4. Segmentation of 2D and 3D models. In the leftmost picture of the first row we zoom in the tail of the camel to show that the point sample is noisy as it is derived from the boundary extraction of a 2D image. The second row shows that the 3D models are segmented into so-called features.

5 Matching

For shape matching we take advantage of our segmentation scheme by matching two shapes with respect to their features. Given a point sample P of a shape Σ, we identify a small set of significant features from our feature segmentation. These features are then mapped to a set of weighted points called the *signature* of Σ. In order to measure the similarity of two shapes, we compare their signatures which boils down to matching two sets of small number of weighted points.

Signature. Let $R_{P,\Sigma}$ denote the set of features that the function SEGMENT computes from a point sample P of a shape Σ. To simplify notations we use R_Σ for $R_{P,\Sigma}$. By definition a feature $r \in R_\Sigma$ is a collection of Delaunay triangles if Σ is a shape in two dimensions and it is a collection of Delaunay tetrahedra if Σ is a shape in three dimensions. For a Delaunay simplex σ let c_σ and v_σ denote the centroid and volume of σ respectively. The *representative point* r^* of a feature r and its weight $\hat{r}$ are defined as

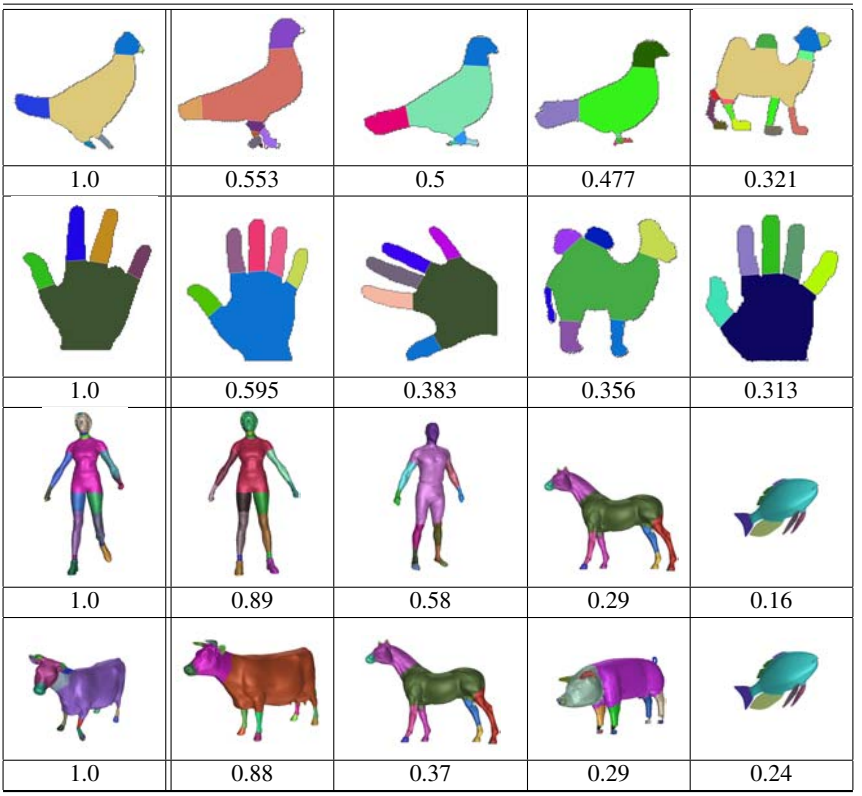


Fig. 5. Matching results in 2D and 3D: Each row contains the matching scores for the query shape in the first column with the highest score 1.0.

$$\hat{r} = \Sigma_{\sigma \in r}\, v_\sigma$$
$$r^* = \frac{\Sigma_{\sigma \in r}\, (c_\sigma \cdot v_\sigma)}{\hat{r}}.$$

That is, the weight of r is its volume and its representative point is the weighted average of the centroids of all $\sigma \in r$, the weight being the volume of each simplex. We call a feature r *significant* if its volume is more than a certain fraction of the total volume of the shape. Given a segmentation R_Σ of a shape Σ, the signature $\mathrm{sign}(\Sigma)$ is defined as the set of weighted feature representative points, i.e., $\mathrm{sign}(\Sigma) = \{(r^*, \hat{r}) \mid r \in R_\Sigma \text{ is significant}\}$.

The amount of similarity between two shapes is measured by first scaling them with bounding boxes and then scoring the similarity between their signatures. In order to score the similarity between two signatures $\mathrm{sign}(\Sigma_1)$ and $\mathrm{sign}(\Sigma_2)$, we need to align them first.

Let r^*, s^* be the representative points in $\mathrm{sign}(\Sigma_1)$ and $\mathrm{sign}(\Sigma_2)$ respectively with maximum weights. We first translate $\mathrm{sign}(\Sigma_2)$ so that r^*, s^* coincide. Then an alignment is obtained by rotating $\mathrm{sign}(\Sigma_2)$ so that a line segment between s^* and another point of

$\mathrm{sign}(\Sigma_2)$ aligns with a line segment between r^* and another point in $\mathrm{sign}(\Sigma_1)$. Certainly, there are $\Theta(mn)$ alignments possible where $|\mathrm{sign}(\Sigma_1)| = m$ and $|\mathrm{sign}(\Sigma_2)| = n$. Since m, n are typically small (less than ten), checking all alignments is not prohibitive.

For each alignment we compute a score based on the matching of weighted points. Both a similarity measure (positive) and a dissimilarity measure (negative) are taken into account while computing the score. The maximum of all the scores is taken to be the amount of similarity and corresponding transformations give the best alignment. We skip the details of the scoring and instead exhibit our results in Figure 5.

References

1. H. Alt and L. J. Guibas. Discrete geometric shapes: matching, interpolation, and approximation: a survey. Tech. report B 96-11, EVL-1996-142, Institute of Computer Science, Freie Universität Berlin, 1996.
2. S. Belongie and J. Malik. Matching with shape contexts. *IEEE Trans. PAMI* **24** (2002), 509–522.
3. P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. *IEEE Trans. PAMI* **14** (1992) 239–256.
4. T. A. Cass. Robust Affine Structure Matching for 3D Object Recognition. *IEEE Trans. PAMI* **20** (1998), 1265–1264.
5. Y. Chen and G. Medioni. Object Modelling by Registration of Multiple Range Images. *Image abd Vision Computing* **10** (1992) 145–155.
6. L. P. Chew, M. T. Goodrich, D. P. Huttelnocher, K. Kedem, J. M. Kleinberg and D. Kravets. Geometric Pattern Matching under Euclidean motion. *Computational Geometry, Theory and Applications* **7** (1997) 113–124.
7. H. Edelsbrunner, M. A. Facello and J. Liang. On the definition and the construction of pockets in macromolecules. *Discrete Apl. Math.* **88** (1998), 83–102.
8. J. Giesen and M. John. The flow complex: a data structure for geometric modeling. *Proc. 14th Annu. ACM-SIAM Sympos. Discrete Algorithms*, (2003), 285–294.
9. K. Grove. Critical point theory for distance functions. *Proc. Sympos. Pure Math.* **54** (1993), 357–385.
10. M. Hilaga, Y. Shinagawa, T. Komura and T. Kunni. Topology matching for fully automatic similarity estimation of 3D shapes. *Proc. SIGGRAPH 2001*, (2001), 203–212.
11. R. Osada, T. Funkhouser, B. Chazelle and D. Dobkin. Matching 3D Models with Shape Distribution. *Proc. Shape Modelling Int'l*, 2001.
12. T. B. Sebastian, P. N. Klein and B. Kimia. Recognition of shapes by editing shock graphs. *Proc. ICCV*, (2001), 755–762.
13. K. Siddiqi, A. Shokoufandeh, S. J. Dickinson and S. W. Zucker. Shock graphs and shape matching. *Computer Vision*, (1998), 222–229.
14. R. C. Veltkamp and M. Hagedoorn. State-of-the-art in shape matching. Technical report UU-CS-1999-27, Utrecht University, the Netherlands, 1999.

Phylogenetic Reconstruction from Gene-Rearrangement Data with Unequal Gene Content

Jijun Tang and Bernard M.E. Moret

Dept. of Computer Science, University of New Mexico, Albuquerque, NM 87131, USA
{jtang,moret}@cs.unm.edu, http://compbio.unm.edu

Abstract. Phylogenetic reconstruction from gene-rearrangement data has seen increased attention over the last five years. Existing methods are limited computationally and by the assumption (highly unrealistic in practice) that all genomes have the same gene content. We have recently shown that we can scale our reconstruction tool, GRAPPA, to instances with up to a thousand genomes with no loss of accuracy and at minimal computational cost. Computing genomic distances between two genomes with unequal gene contents has seen much progress recently, but that progress has not yet been reflected in phylogenetic reconstruction methods. In this paper, we present extensions to our GRAPPA approach that can handle limited numbers of duplications (one of the main requirements for analyzing genomic data from organelles) and a few deletions. Although GRAPPA is based on exhaustive search, we show that, in practice, our bounding functions suffice to prune away almost all of the search space (our pruning rates never fall below 99.995%), resulting in high accuracy and fast running times. The range of values within which we have tested our approach encompasses mitochondria and chloroplast organellar genomes, whose phylogenetic analysis is providing new insights on evolution.

Keywords. Computational biology, phylogenetic reconstruction, gene-order data, whole-genome data, signed permutations, lower bounds, Hannenhalli-Pevzner theory, inversion distance, reversal distance, edit distance, gene duplications, experimental assessment

1 Introduction

A phylogeny is the evolutionary history of a group of organisms; in most cases, it is represented (in obviously simplified form) by a tree where the leaves represent current organisms and the internal nodes represent ancestral organisms, and where the edges denote evolutionary relationships. Such phylogenies have long been reconstructed on the basis of morphological data and more recently on the basis of molecular data such as DNA sequence data.

Biologists can infer the ordering and strandedness of genes on a chromosome, and thus represent each chromosome by an ordering of signed genes (where the sign indicates the strand). These gene orders can be rearranged by evolutionary events such as inversions (also called reversals) and transpositions and, because they evolve slowly (much more slowly, for instance, than DNA sequences), give biologists an important new

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 37–46, 2003.
© Springer-Verlag Berlin Heidelberg 2003

source of data for phylogeny reconstruction (see, e.g., [10,20,21,23]). Appropriate tools for analyzing such data may help resolve some difficult phylogenetic reconstruction problems. Developing such tools is thus an important area of research—indeed, the recent DCAF symposium [27] was devoted to this topic.

A natural optimization problem for phylogeny reconstruction from gene-order data is to reconstruct an evolutionary scenario with a minimum number of the permitted evolutionary events on the tree. This problem is NP-hard for most criteria—even the very simple problem of computing the median[1] of *three* genomes with identical gene content under such models is NP-hard [7,22]—although the algorithms of Caprara [8] and of Siepel and Moret [28] have done well in practice (see, e.g., [18]). Indeed, even the problem of computing the edit distance between two genomes is difficult: for instance, even with equal gene content and with only inversions allowed, the problem is NP-hard for unsigned permutations [6].

2 Background

2.1 Genomic Distances

Hannenhalli and Pevzner [12] made a fundamental breakthrough by developing an elegant theory for signed permutations and providing a polynomial-time algorithm to compute the edit distance (and the corresponding shortest edit sequence) between two signed permutations under inversions; Bader et al. [2] later showed that this edit distance can be computed in linear time. El-Mabrouk [11] extended the results of Hannenhalli and Pevzner to the computation of edit distances for inversions and deletions and also for inversions and non-duplicating insertions; she also gave an approximation algorithm with bounded error for computing edit distances in the presence of all three operations (inversions, deletions, and non-duplicating insertion). Sankoff had proposed the so-called exemplar strategy [25] (itself an NP-hard problem [4]) to handle duplications: only one copy of each gene is retained, that which minimizes a breakpoint scoring function. Experiments we conducted suggested that too much information is lost in reducing the genomes to a single copy of each gene; working to use all duplicates in the computation, our group recently extended the work of El-Mabrouk by providing tight approximations for edit distances under arbitrary operations (including duplications) [17].

2.2 Gene-Order Reconstruction

Extending the computation of genomic distances to genomes with unequal gene contents is but the first step in a reconstruction effort. While it is possible to reconstruct the tree's topology on the basis of pairwise distances only (using standard methods such as neighbor-joining [24]), reconstructing ancestral genomes requires additional steps. Sankoff had proposed an iterative strategy which he called breakpoint analysis [26], which we subsequently improved by combining it with our fast inversion distance computation and various speedup heuristics to produce the software suite GRAPPA [1,

[1] The median of k genomes is a genome that minimizes the sum of the pairwise distances between itself and each of the k given genomes.

19]. Other approaches include classical parsimony analysis based on binary encodings of the genome data [9], fast heuristic uses of the reversal distance for ancestral genome reconstruction [3], and a recent endeavor based on likelihood maximization [16].

2.3 GRAPPA

GRAPPA is based on Sankoff's breakpoint analysis. It works by enumerating every possible tree topology for the given collection of organisms and, for each tree, by reconstructing the ancestral genomes associated with the internal nodes of the tree, thereby making it possible to score the tree. The trees of lowest score are then returned. Ancestral genomes are reconstructed through iterative refinement: on successive traversals of the tree, each ancestral genome is compared with the median of its three neighbors and replaced by that median if the tree score is thereby improved. Since computing the median is itself an NP-hard problem, scoring each tree is computationally intensive and should be avoided if at all possible. We devised and built into GRAPPA an effective bounding scheme, which runs in linear time, to prune most candidate trees without having to score them, using nothing more than the triangle inequality. The resulting speed-up (of up to a billion-fold on many datasets) enabled us to solve datasets of up to 15 genomes. Most recently, we combined the disk-covering approach of Warnow and her colleagues [13, 14,15] with GRAPPA, thereby scaling up the approach to up to one thousand genomes with no loss of accuracy [29].

3 Our Approach to Duplication

We assume a fixed set of genes $\{g_1, g_2, ..., g_k\}$. Let $d_i \geq 1$ be the number of copies of g_i, which we assume to be equal for all genomes—unequal numbers of duplicates introduce the possibility of deletions, which we address in a later section. Since the number of copies for a gene is identical for all genomes, we can define the multiset

$$\{\underbrace{g_1, \ldots, g_1}_{d1}, \underbrace{g_2, \ldots, g_2}_{d2}, \cdots, \underbrace{g_k, \ldots, g_k}_{dk}\}$$

and each genome is then an ordering (circular or linear) of this superset, with each gene copy given an orientation (sign).

We assume that copies of a gene are rearranged as if they were distinct genes; by renaming the copies, we then obtain a signed permutation of a set of $\sum_{i=1}^{k} d_i$ distinct genes. For example, given the ordering $(1, 2, -3, 4, 3, 5)$, in which gene 3 appears twice, we can relabel one of the copies as gene 6, yielding two possible new orderings: $(1, 2, -3, 4, 6, 5)$ and $(1, 2, -6, 4, 3, 5)$, both signed permutations of the set $\{1, 2, 3, 4, 5, 6\}$. We call the collection of $\Pi_{i=1}^{k} d_i$ possible new orderings obtained through the relabeling of copies as new genes a *differentiated genome family*. Each genome in the input data has a family of that size. We can then define the inversion distance between two genomes with identical duplications as the minimum pairwise inversion distance between a member of the differentiated family of one genome and a member of the family of the other genome. Because inversion distances between genomes

with equal gene content can be computed efficiently in linear time, this definition can be computed quickly for modest numbers of duplications by checking all $\Pi_{i=1}^{k} d_i^2$ pairs.

To solve the median problem—the central computational problem for GRAPPA—we can extend this simple idea and consider all triples of elements from the three differentiated families, for a total of $\Pi_{i=1}^{k} d_i^3$ possibilities. This number can quickly grow uncomfortably large since each median computation is potentially very expensive, so we need to avoid as many of these computations as possible. We can use the same bounding strategy at this stage as is used by GRAPPA in bounding the cost of individual trees: by the triangle inequality, the sum of the distances from the median to its three neighbors is at least as large as half of the sum of the three pairwise distances between the three neighbors. (Bryant [5] developed a slightly tighter bound, but it is limited to breakpoint distances and our previous experimental work [19] showed that it is too slow and gains too little to be useful in pruning the search space.) We can compute the pairwise distances in linear time and avoid computing the median of a particular triple of family members whenever their lower bound exceeds the current best median score. (We still need to examine all $\Pi_{i=1}^{k} d_i^3$ triples of family members, but our intended application, to organellar genomes with only a hundred or so genes and typically fewer than 10 duplicates in all, yields reasonable values for this product.) Clearly, we will get better bounding if we can start with some reasonable choice of median; in particular, the choice of initial family members for the genomes at the leaves has a huge impact on the pruning rate for median computations. We found the following initialization method to be very effective: for each leaf genome, we pick that member of the differentiated family which minimizes the sum of the minimum pairwise distances to the other leaf genomes.

4 Experimental Results for Duplications

We ran simulation tests on trees of 10, 11, and 12 genomes (sizes easily handled by the basic GRAPPA), under two different models of topologies (uniform random trees and birth-death trees) and three different rates of evolution (with r, the expected number of inversions per edge, set to 2, 4, and 8). For a given rate of evolution r, we generate an actual number of evolutionary events for each edge by using a random integer in the set $\{0, 1, \ldots, 2r\}$. We then start with the identity permutation on the genes at the root of the tree topology and evolve permutations down the tree by applying to the parent permutation the number of events prescribed by the edge; each event is an inversion, with its two endpoints chosen independently and uniformly at random. All genomes have a total of 100 genes; a duplication is generated by selecting two of the genes at random and calling them duplicates of each other. We used three scenarios with limited duplications: (i) one gene is duplicated once; (ii) two genes are duplicated, one once and the other twice; and (iii) three genes are duplicated, two once and one twice. Overall, then, we used 54 combinations of parameters; we generated 20 datasets for each combination and report in the tables below the average value of the 20 runs. We ran all of our experiments on a 2.4GHz desktop Pentium-4 machine with 1GB of memory running Linux; running times naturally increased with the number of genomes, but, even at 12 genomes (cases in which GRAPPA must examine half a billion trees), the running time never exceeded five minutes.

Table 1. Average numbers of edges in error for one duplication: (a) uniform trees, (b) birth-death trees.

(a)

	$r=2$		$r=4$		$r=8$	
n	FP	FN	FP	FN	FP	FN
10	0	0	0	0.05	0	0
11	0.15	0.20	0	0	0	0.10
12	0.10	0.10	0	0.10	0	0

(b)

	$r=2$		$r=4$		$r=8$	
n	FP	FN	FP	FN	FP	FN
10	0	0	0	0	0	0
11	0	0	0.10	0.10	0	0
12	0.10	0.15	0	0	0	0

Table 2. Average numbers of edges in error for two duplications $(2, 3)$: (a) uniform trees, (b) birth-death trees.

(a)

	$r=2$		$r=4$		$r=8$	
n	FP	FN	FP	FN	FP	FN
10	0.20	0.20	0	0	0	0
11	0.10	0.10	0	0	0	0.10
12	0	0	0.10	0.10	0	0

(b)

	$r=2$		$r=4$		$r=8$	
n	FP	FN	FP	FN	FP	FN
10	0.05	0.15	0	0	0	0
11	0	0	0	0	0	0
12	0	0	0	0	0	0.20

Table 3. Average numbers of edges in error for three duplications $(2, 2, 3)$: (a) uniform trees, (b) birth-death trees.

(a)

	$r=2$		$r=4$		$r=8$	
n	FP	FN	FP	FN	FP	FN
10	0.10	0.10	0	0	0	0
11	0	0	0	0	0	0.05
12	0.05	0.20	0	0	0	0.10

(b)

	$r=2$		$r=4$		$r=8$	
n	FP	FN	FP	FN	FP	FN
10	0.10	0.10	0	0	0	0
11	0	0	0	0.10	0	0
12	0	0.05	0	0	0.10	0.10

Tables 1 through 3 show the average numbers of false positive (FP) and false negative (FN) edges in the reconstructed trees when compared to the model trees generated by the simulations. A false negative arises when the reconstructed tree does not include an edge present in the model tree; conversely, a false positive arises when the reconstructed tree includes an edge not present in the model tree. The reconstructed trees are not always binary (fully resolved), because some of their edges may have length zero; edges of zero length are removed and thus may give rise to false negatives—unless the true tree itself had edges of zero length, something that does occur at lower evolutionary rates. Observe that both FN and FP remain extremely low, often even zero, indicating high accuracy in the reconstruction, a consequence, we conjecture, of matching all duplicates in the reconstruction process.

Tables 4 and 5 show the pruning rates for the median computations and for the tree enumeration. The latter shows very high pruning rates: in most cases fewer than one tree in $100,000$ remains to be scored. Pruning rates for medians are also excellent: we only rarely have to compute more than a couple of medians per node.

5 Our Approach to Deletions

For simplicity, we now consider genomes without duplications to present our approach to deletions. (The two strategies can easily be combined to handle both duplications and

Table 4. Pruning rates (percentage of eliminated problems) for uniform trees.

(a) one duplication

	$r = 2$		$r = 4$		$r = 8$	
n	*Medians*	*Overall*	*Medians*	*Overall*	*Medians*	*Overall*
10	85.4	100	85.0	100	85.5	99.999
11	85.5	100	85.1	100	85.6	100
12	82.1	100	83.6	100	82.9	100

(b) two duplications $(2, 3)$

n	*Medians*	*Overall*	*Medians*	*Overall*	*Medians*	*Overall*
10	99.4	99.999	99.4	100	99.5	99.999
11	99.0	99.999	99.1	100	98.7	100
12	98.8	100	98.8	100	99.2	100

(v) three duplications $(2, 2, 3)$

n	*Medians*	*Overall*	*Medians*	*Overall*	*Medians*	*Overall*
10	99.8	99.999	99.7	100	99.8	100
11	99.8	99.999	99.8	100	99.6	100
12	99.8	100	99.8	100	99.5	100

Table 5. Pruning rates (percentage of eliminated problems) for birth-death trees.

(a) one duplication

	$r = 2$		$r = 4$		$r = 8$	
n	*Medians*	*Overall*	*Medians*	*Overall*	*Medians*	*Overall*
10	85.1	99.999	85.6	99.999	79.8	99.999
11	85.5	99.999	80.2	99.999	74.5	100
12	85.6	100	85.5	99.999	84.0	100

(b) two duplications $(2, 3)$

n	*Medians*	*Overall*	*Medians*	*Overall*	*Medians*	*Overall*
10	98.9	99.995	99.0	99.998	99.3	99.999
11	99.3	100	98.9	100	99.0	99.999
12	99.3	100	99.2	100	98.7	100

(c) three duplications $(2, 2, 3)$

n	*Medians*	*Overall*	*Medians*	*Overall*	*Medians*	*Overall*
10	99.6	100	99.8	100	99.8	99.999
11	99.8	100	99.6	100	99.5	100
12	99.8	100	99.8	100	99.7	100

deletions, albeit at the usual multiplicative cost of cases.) We need to devise strategies for computing pairwise distances between two genomes, for computing the median of three genomes, and for initializing ancestral labels at internal nodes. In developing these strategies, we will ignore "silent" changes (such as a gene loss followed by an insertion that restores the same gene). We will also assume that the probability of a gene loss is small enough that, when faced with the choice of assigning the loss to a parent or assigning it to both children, we always choose to assign it to the parent, since the probability of that one loss is some small p, but the probability of its being lost within the same time frame by both children is an infinitesimally small p^2.

Assume G_1 has N genes and G_2 has $N - m$ genes, i.e., G_2 lost m genes. There are $N \times N - 1 \times \cdots \times N - m$, or roughly N^m different ways to equalize the two gene contents; using the same approach as for duplications, we define the distance between

G_1 and G_2 to be the smallest pairwise inversion distance between G_1 and the various "completions" of G_2. In fact, we could directly use the method of El-Mabrouk [11] to solve this problem exactly in polynomial time, but we use the brute-force paradigm because it extends easily to the computation of medians, something that El-Mabrouk's approach does not. Consider now the computation of the median. For simplicity, assume we have $m = 1$—our reasoning easily extends to arbitrary values of m. Given three genomes G_1, G_2, and G_3, each of which could have lost a given gene, we face three cases:

- All three genomes lost that gene or none did. Then the median is in the same situation and the computation proceeds as currently implemented in GRAPPA.
- One genome, say G_1, lost that gene, but the other two still have it. Then the median retained the gene and thus a single loss event took place between the median and G_1. This gives us N choices of completion for G_1—we compute the median for each choice (if needed, since we can prune some choices through the same bounding strategy used for duplications).
- Two genomes lost that gene—say that only G_1 retains it. Then the median also lost that gene, so that we again have a single loss event, between the median and G_1. We remove that gene from G_1 and compute the median in the usual manner.

Thus we can compute the median in the case of a one-gene loss with at most N regular median computations; in the case of an m-gene loss, the number of regular median computations is on the order of N^m.

Before we can apply the median computations, however, we need to initialize the internal nodes of a tree with ancestral genomes. The first step in this initialization is to determine the gene content at each node. We accomplish this task using the same principle of always preferring a single loss event to a pair of concurrent loss events. Specifically, we run the following iterative algorithm:

- Identify all sibling pairs of leaves.
- For each sibling pair of leaves, assign to their parent the larger of the two gene contents—corresponding to a single loss event from the parent to the smaller child.
- Remove all processed leaves (thus turning their parents into leaves) and repeat.

6 Experimental Results for Deletions

Our test simulations are structured in the same manner as those we used for duplications. We tested for minimal gene loss (only two of the genomes lost a gene) and widespread gene loss (half of the genomes lost that gene). Once again, we ran 20 datasets for each combination of parameters and we report the average of the runs. Tables 6 and 7 show the average numbers of false positive and false negative edges in our reconstructions. As in the case of duplications, the reconstructions are remarkably accurate. Tables 8 and 9 give the corresponding pruning rates. The rates remain extremely high for tree pruning, but the simple triangle inequality proves fairly weak for pruning median computations.

Table 6. Average numbers of edges in error for one missing gene in two genomes: (a) uniform trees, (b) birth-death trees.

(a)

	$r=2$		$r=4$		$r=8$	
n	FP	FN	FP	FN	FP	FN
10	0.10	0.10	0	0	0	0
11	0.10	0.15	0	0	0.05	0.10
12	0.20	0.20	0	0	0	0

(b)

	$r=2$		$r=4$		$r=8$	
n	FP	FN	FP	FN	FP	FN
10	0	0.10	0	0	0	0.05
11	0	0	0	0.10	0	0
12	0	0.05	0	0	0	0

Table 7. Average numbers of edges in error for one missing gene in half the genomes: (a) uniform trees, (b) birth-death trees.

(a)

	$r=2$		$r=4$		$r=8$	
n	FP	FN	FP	FN	FP	FN
10	0	0	0	0	0	0
11	0.10	0.10	0	0	0.10	0.10
12	0	0	0	0.10	0	0

(b)

	$r=2$		$r=4$		$r=8$	
n	FP	FN	FP	FN	FP	FN
10	0.10	0.10	0	0.05	0	0.10
11	0	0	0	0	0	0
12	0.10	0.15	0	0	0.10	0.10

Table 8. Pruning rates (percentage of eliminated problems) for uniform trees.

gene lost in two genomes

	$r=2$		$r=4$		$r=8$	
n	*Medians*	*Overall*	*Medians*	*Overall*	*Medians*	*Overall*
10	71.7	99.999	62.5	100	64.1	99.999
11	72.2	99.999	57.4	100	66.8	100
12	68.6	100	65.2	100	60.5	100

gene lost in half the genomes

n	*Medians*	*Overall*	*Medians*	*Overall*	*Medians*	*Overall*
10	57.6	99.999	73.4	99.999	63.3	100
11	77.9	100	52.9	100	55.8	100
12	78.5	100	62.1	100	65.6	100

7 Conclusions

We have presented a simple approach to the handling of a limited number of gene duplications and gene losses in the reconstruction of phylogenies from gene-order data.

Table 9. Pruning rates (percentage of eliminated problems) for birth-death trees.

gene lost in two genomes

	$r=2$		$r=4$		$r=8$	
n	*Medians*	*Overall*	*Medians*	*Overall*	*Medians*	*Overall*
10	70.8	99.999	53.6	100	53.8	99.999
11	58.2	100	64.8	100	62.6	100
12	72.5	100	54.5	100	55.4	100

gene lost in half the genomes

n	*Medians*	*Overall*	*Medians*	*Overall*	*Medians*	*Overall*
10	68.9	100	65.7	100	51.2	99.999
11	68.8	99.999	51.2	100	74.6	100
12	78.5	99.999	62.6	100	67.8	100

While the exhaustive nature of our approach limits its applicability to a fairly modest number of duplication and deletion events, it does allow us to analyze many datasets of organellar genomes, particularly chloroplast and mitochondria genomes, which are of special interest to evolutionary biologists. The success of our approach on such datasets also opens up the possibility that it could be scaled up through some type of divide-and-conquer paradigm, much in the manner in which we successfully scaled up GRAPPA, usually limited to 13–14 genomes, to one thousand genomes through the application of the sophisticated divide-and-conquer approach known as disk-covering.

Acknowledgments. This research is supported by the National Science Foundation under grants ACI 00-81404, DEB 01-20709, EIA 01-13095, and EIA 01-21377.

References

1. D.A. Bader, B.M.E. Moret, T. Warnow, S.K. Wyman, and M. Yan. *GRAPPA (Genome Rearrangements Analysis under Parsimony and other Phylogenetic Algorithms)*. `www.cs.unm.edu/~moret/GRAPPA/`.
2. D.A. Bader, B.M.E. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *J. Comput. Biol.*, 8(5):483–491, 2001. A preliminary version appeared in WADS'01, pp. 365–376.
3. G. Bourque and P. Pevzner. Genome-scale evolution: reconstructing gene orders in the ancestral species. *Genome Research*, 12:26–36, 2002.
4. D. Bryant. The complexity of calculating exemplar distances. In D. Sankoff and J. Nadeau, editors, *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment, and the Evolution of Gene Families*, pages 207–212. Kluwer Academic Pubs., Dordrecht, Netherlands, 2000.
5. D. Bryant. A lower bound for the breakpoint phylogeny problem. In *Proc. 11th Ann. Symp. Combin. Pattern Matching (CPM'00)*, volume 1848 of *Lecture Notes in Computer Science*, pages 235–247. Springer-Verlag, 2000.
6. A. Caprara. Sorting by reversals is difficult. In *Proc. 1st Int'l Conf. on Comput. Mol. Biol. RECOMB97*, pages 75–83. ACM Press, 1997.
7. A. Caprara. Formulations and hardness of multiple sorting by reversals. In *Proc. 3rd Int'l Conf. on Comput. Mol. Biol. RECOMB99*, pages 84–93. ACM Press, 1999.
8. A. Caprara. On the practical solution of the reversal median problem. In *Proc. 1st Workshop on Algs. in Bioinformatics WABI 2001*, volume 2149 of *Lecture Notes in Computer Science*, pages 238–251. Springer-Verlag, 2001.
9. M.E. Cosner, R. K. Jansen, B.M.E. Moret, L. A. Raubeson, L.-S. Wang, T. Warnow, and S. K. Wyman. An empirical comparison of phylogenetic methods on chloroplast gene order data in Campanulaceae. In D. Sankoff and J. Nadeau, editors, *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment, and the Evolution of Gene Families*, pages 99–121. Kluwer Academic Pubs., Dordrecht, Netherlands, 2000.
10. S.R. Downie and J.D. Palmer. Use of chloroplast DNA rearrangements in reconstructing plant phylogeny. In P. Soltis, D. Soltis, and J.J. Doyle, editors, *Plant Molecular Systematics*, pages 14–35. Chapman and Hall, 1992.
11. N. El-Mabrouk. Genome rearrangement by reversals and insertions/deletions of contiguous segments. In *Proc. 11th Ann. Symp. Combin. Pattern Matching (CPM'00)*, volume 1848 of *Lecture Notes in Computer Science*, pages 222–234. Springer-Verlag, 2000.

12. S. Hannenhalli and P.A. Pevzner. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In *Proc. 27th Ann. Symp. Theory of Computing STOC 95*, pages 178–189. ACM Press, 1995.
13. D. Huson, S. Nettles, K. Rice, T. Warnow, and S. Yooseph. The hybrid tree reconstruction method. *ACM J. Experimental Algorithmics*, 4(5), 1999. http://www.jea.acm.org/1999/HusonHybrid/.
14. D. Huson, S. Nettles, and T. Warnow. Disk-covering, a fast converging method for phylogenetic tree reconstruction. *J. Comput. Biol.*, 6(3):369–386, 1999.
15. D. Huson, L. Vawter, and T. Warnow. Solving large-scale phylogenetic problems using DCM-2. In *Proc. 7th Int'l Conf. on Intelligent Systems for Molecular Biology (ISMB99)*, pages 118–129. AAAI Press, 1999.
16. B. Larget, J.B. Kadane, and D. Simon. A Markov chain Monte Carlo approach to reconstructing ancestral genome rearrangements. Technical report, Carnegie Mellon University, Pittsburgh, PA, 2002. Available at `www.stat.cmu.edu/tr/tr765/`.
17. M. Marron, K.M. Swenson, and B.M.E. Moret. Genomic distances under deletions and insertions. In *Proc. 9th Ann. Int'l Conf. Computing and Combinatorics (COCOON'03)*, Lecture Notes in Computer Science. Springer-Verlag, 2003. Accepted, to appear.
18. B.M.E. Moret, A.C. Siepel, J. Tang, and T. Liu. Inversion medians outperform breakpoint medians in phylogeny reconstruction from gene-order data. In R. Guigo and D. Gusfield, editors, *Proc. 2nd Int'l Workshop Algorithms in Bioinformatics (WABI'02)*, volume 2452 of *Lecture Notes in Computer Science*, pages 521–536. Springer-Verlag, 2002.
19. B.M.E. Moret, J. Tang, L.-S. Wang, and T. Warnow. Steps toward accurate reconstructions of phylogenies from gene-order data. *J. Comput. Syst. Sci.*, 65(3):508–525, 2002.
20. R.G. Olmstead and J.D. Palmer. Chloroplast DNA systematics: a review of methods and data analysis. *Amer. J. Bot.*, 81:1205–1224, 1994.
21. J.D. Palmer. Chloroplast and mitochondrial genome evolution in land plants. In R. Herrmann, editor, *Cell Organelles*, pages 99–133. Springer Verlag, 1992.
22. I. Pe'er and R. Shamir. The median problems for breakpoints are NP-complete. *Elec. Colloq. on Comput. Complexity*, 71, 1998.
23. L.A. Raubeson and R.K. Jansen. Chloroplast DNA evidence on the ancient evolutionary split in vascular land plants. *Science*, 255:1697–1699, 1992.
24. N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, 4:406–425, 1987.
25. D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 15(11):909–917, 1999.
26. D. Sankoff and M. Blanchette. Multiple genome rearrangement and breakpoint phylogeny. *J. Comput. Biol.*, 5:555–570, 1998.
27. D. Sankoff and J. Nadeau, editors. *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment, and the Evolution of Gene Families*. Kluwer Academic Pubs., Dordrecht, Netherlands, 2000.
28. A.C. Siepel and B.M.E. Moret. Finding an optimal inversion median: Experimental results. In O. Gascuel and B.M.E. Moret, editors, *Proc. 1st Int'l Workshop Algorithms in Bioinformatics (WABI'01)*, volume 2149 of *Lecture Notes in Computer Science*, pages 189–203. Springer-Verlag, 2001.
29. J. Tang and B.M.E. Moret. Scaling up accurate phylogenetic reconstruction from gene-order data. In *Proc. 11th Int'l Conf. on Intelligent Systems for Molecular Biology (ISMB03)*, volume 19, Suppl. 1 of *Bioinformatics*, pages i305–i312. Oxford U. Press, 2003.

Toward Optimal Motif Enumeration

Patricia A. Evans[1] and Andrew D. Smith[1,2]

[1] University of New Brunswick, P.O. Box 4400, Fredericton N.B., E3B 5A3, Canada
[2] Ontario Cancer Institute, University Health Network, Suite 703
620 University Avenue, Toronto, Ontario, Canada, M5G 2M9 pevans@unb.ca,
asmith@uhnres.utoronto.ca

Abstract. We present algorithms that reduce the time and space needed to solve problems of finding all motifs common to a set of sequences. In particular, we give algorithms that (1) require time and space linear in the size of the input, (2) succinctly encode the output so that the time and space requirements depend on the number of motifs, not directly on motif length, and (3) efficiently parallelize the enumeration.

1 Introduction

The problem of discovering short strings occurring approximately in each member of a set of longer strings is important in computational biology. We refer to the short strings as *motifs*, and the longer strings as *sequences*[1]. By "occurring approximately", we mean that motifs must match a segment of each sequence with at most some specified number of mismatches.

The motif discovery problem abstracts many problems encountered in the analysis of biology sequence data, where the sequences are molecular sequences and motifs represent short biologically important patterns. A popular technique for finding motifs is to enumeratively test all strings over the sequence alphabet having length equal to the desired motif length. An advantage of the enumerative approach is that it does just that; enumerative algorithms produces all possible motifs for a set of sequences. This allows the discovered motifs, which posses a certain combinatorial property, to be evaluated according to other criteria. In this capacity, the enumerative algorithms can provide input to other algorithms that filter motifs based on other properties. Formally, we define the motif enumeration problem as follows:

Problem 1. The input is a set $\mathcal{F} = \{S_1, \ldots, S_m\}$ of strings over an alphabet Σ such that $|S_i| \leq n$, $1 \leq i \leq m$, and integers l and d such that $0 \leq d < l \leq n$. The solution is a set of motifs $\mathcal{M}_{\mathcal{F}} \subseteq \Sigma^l$ such that for each motif $\mathcal{C} \in \mathcal{M}_{\mathcal{F}}$ and each $S_i \in \mathcal{F}$, there exists a length l substring of S_i that is Hamming distance $\leq d$ from $\mathcal{C}$.

Hamming distance is defined, for equal length strings, as the number of mismatches between the strings. Note that it is sufficient, and often desirable, to

[1] This terminology may conflict with terminology used elsewhere.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 47–58, 2003.
© Springer-Verlag Berlin Heidelberg 2003

produce a small encoding of $\mathcal{M}_{\mathcal{F}}$, from which the motifs can be efficiently extracted.

There are two major computational challenges to enumerating motifs. The first challenge is that the problem of deciding if $\mathcal{M}_{\mathcal{F}} = \emptyset$ is NP-hard; practical solutions are thus non-trivial. The second is that we are concerned with more than simply the decision, so we may have to produce output of exponential size.

The paper is organized as follows. Section 2 describes previous work on the problem. In Section 3 we describe the first algorithm, Census, that improves on an algorithm of Sagot [9] and establishes an upper bound on the time and space complexity that is linear in both the string length and the number of strings. We also discuss parallelizations of the algorithm. In Section 4, we describe the MotifIntersection algorithm, further reducing the upper bound on the time complexity. The algorithm is based on a data structure that succinctly encodes sets of motifs, and allows efficient set operations. In Section 5, we show the problem admits an FPP algorithm, placing the corresponding decision version in the subclass of fixed parameter tractable problems that are highly parallelizable [3].

2 Background

Many algorithms have used enumerative strategies to find motifs in sets of sequences (*e.g.* [2] [7] [10] [11]). These algorithms each approach the problem differently; most attempt to eliminate as much of the search space as possible. Each, however, attempts to enumerate all strings of length l over the sequence alphabet. This most naive form of search introduces a factor of $\Omega(|\Sigma|^l)$ into the time complexity. The benefit of this type of enumeration is that it requires space bounded by a linear function of the size of the input.

New ground was broken when Sagot [9] introduced a different approach that enumerates only those strings that are potential motifs, letting information from the sequences guide the enumeration. This more intelligent search remains within the (l, d)-neighborhood of each sequence. In the following definition, d_H refers to the Hamming distance.

Definition 1. (neighborhood) *For a string $S \in \Sigma^n$, with $n \geq l$, the (l, d)-neighborhood of S is the set*

$$\{s' : s' \in \Sigma^l \wedge d_H(s, s') \leq d \text{ for some substring } s \text{ of } S \text{ with } |s| = l\}.$$

For any string S, we use $N_{l,d}(S)$ to denote the (l, d)-neighborhood of S. For a family $\mathcal{F}$ of strings, the (l, d)-neighborhood of $\mathcal{F}$ is the set

$$\{s' : s' \in \Sigma^l \wedge \forall S \in \mathcal{F}, s' \in N_{l,d}(S)\},$$

and is denoted $N_{l,d}(\mathcal{F})$.

We also define the value $N = \sum_{i=0}^{d} \binom{l}{i} (|\Sigma|-1)^i$, and note that this value appears throughout our analysis. The significance of N is that for a string s with $|s| = l$, $N = |N_{l,d}(s)|$.

The method of Sagot has a time complexity of $O(lm^2nN)$, a space complexity of $O(lm^2n)$, and has proved successful in practice [13]. We note that the algorithm in [9] was actually designed with a "quorum" parameter, so that a motif is only required to be common to some $q \leq m$ of the sequences.

3 Improved Time and Space Complexity

In this section we make an initial improvement to the time and space complexity for the enumeration problem. We eliminate a factor of m from the requirements of the algorithm of Sagot [9]. This brings the time complexity to $O(lmnN)$, and the space complexity to $O(lmn)$. In Section 4 we further reduce the time complexity to $O(mnN)$.

3.1 The Census Algorithm

The algorithm in [9] employed a generalized suffix tree [6], and required that each node indicate the subset of strings having the node's label as a prefix. We eliminate the use of generalized suffix trees, and therefore eliminate the sets stored at each node. Analysis indicates that we have also eliminated the factor of m in the time complexity without increasing the influence of the remaining parameters.

Census begins with the construction, for each $S_i \in \mathcal{F}$, of the lexicographic tree T_i encoding all length l substrings of S_i, which requires $O(lmn)$ time [1]. The potential motifs of desired length l are not searched directly. The search process iteratively searches for each prefix of a given motif in order to take advantage of the fact that prefixes are shared by many potential motifs. This eliminates redundant processing, as will be shown in the complexity analysis.

Let $\mathcal{C}$ be a (length $\leq l$) motif for $\mathcal{F}$. Define the family of sets $F = \{F_1, \ldots, F_m\}$ with respect to $\mathcal{C}$ as $F_i = \{(v, k) : v \text{ is a node in the tree } T_i, \text{ and } 0 \leq k \leq d\}$, where k counts mismatches between the label of v and $\mathcal{C}$, for each $1 \leq i \leq m$. Think of F_i as the frontier of nodes in T_i whose path labels are of Hamming distance $\leq d$ from $\mathcal{C}$. For any $(v, k) \in F_i$, the path label of node v spells out an occurrence of $\mathcal{C}$ in S_i. For any $1 \leq i \leq m$ and element $(v, k) \in F_i$, the value of k is the number of mismatches between $\mathcal{C}$ and the path label of node v in T_i. Given a character $\alpha \in \Sigma$ and a frontier set F defined with respect to a motif $\mathcal{C}$, the family of sets $F^\alpha = \{F_1^\alpha, \ldots, F_m^\alpha\}$ is the frontier set defined with respect to the length $|\mathcal{C}| + 1$ motif $\mathcal{C}\alpha$.

While searching the space of possible motifs, if any $F_i \in F$ is found to be empty, the search space is pruned. The emptiness condition implies that there exists some member of $\mathcal{F}$ containing no occurrence of the motif presently being searched. Pseudocode for the Census algorithm is provided in Algorithm 1. For the initial call to Census, $\mathcal{C}$ is the empty string and F is the set of roots of the trees T_i, each given an error value of 0.

Algorithm 1: Pseudocode for the CENSUS algorithm.
Input: A set of lexicographic trees $F = \{F_1, \ldots F_m\}$ and a string $\mathcal{C}$.
Output: All motifs for $\mathcal{F}$.

```
CENSUS(C, F)
1.      for each character α ∈ Σ
2.          for each F_i ∈ F
3.              for each (v, k) ∈ F_i
4.                  if node v has a child v' labeled with α
5.                      F_i^α ← F_i^α ∪ {(v', k)}
6.                  if k < d
7.                      for each child v' of v that is not labeled with α
8.                          F_i^α ← F_i^α ∪ {(v', k + 1)}
9.          if ∀F_i^α ∈ F^α, F_i^α ≠ ∅
10.             C' ← Cα
11.             if |C'| = l then output(C')
12.                     else make the recursive call CENSUS(C', F^α)
```

Theorem 1. *The time complexity of* CENSUS *is* $O(lmnN)$.

Proof. The time complexity of the algorithm is proportional to the number of motifs in the search space, multiplied by the size of the family of frontiers F that must be constructed for each point in the search space. In the worst case, for m strings of length n, there are $O(nN)$ potential motifs for $\mathcal{F}$. The maximum size of the (l, d)-neighborhood of a string S of length n is $(n - l + 1)N$, and this is achieved when the d-neighborhoods of all length l substrings of S are disjoint. Further observe that this upper bound on the number of motifs in the search space gives an upper bound on the search space traversed by the algorithm when all (l, d)-neighborhoods of members of $\mathcal{F}$ completely overlap. Attaining this limit on the search space requires that each $F_i \in F$ have exactly one element. For $1 \leq j \leq l$, let X_j be the space of all motifs $\mathcal{C}$ for $\mathcal{F}$ such that $|\mathcal{C}| = j$. Then under the condition of complete (l, d)-neighborhood overlap for members of $\mathcal{F}$:

$$|F| \sum_{j \leq l} |X_j| < mn \sum_{j \leq l} \binom{j}{d} (|\Sigma| - 1)^d = O(lmnN).$$

Consider how the search space is affected should any F_i have more than one element. The (l, d)-neighborhoods of substrings of S_i would no longer be disjoint and the d-neighborhood of S_i would have at least one fewer member. Since each increase in the size of a set $F_i \in F$, with respect to any motif $\mathcal{C}$, decreases the size of the search space by an equal amount, the situation of total (l, d)-neighborhood overlap is the worst case. Hence, the overall running time of the algorithm is $O(lnmN)$. □

The space complexity of CENSUS is $O(lmn)$, exactly the space required for the lexicographic trees, since the frontier sets consist of nodes from the lexicographic

trees, and no node exists simultaneously in more than one frontier set. We note that if CENSUS were modified to solve the quorum version as in [9], the time complexity would increase by a factor of $(m - q)$ for a quorum of q; the space complexity would not be altered.

The CENSUS algorithm was implemented in `C` and tested on a 1.4GHz Pentium 3 processor. Simulated data consisted of sequences generated uniformly at random from a 4 character alphabet. For (m, n, l, d) values of $(1000, 1000, 12, 3)$, CENSUS required 95 minutes and 370 megabytes of memory; 4.5 hours and 97 megabytes was required for values of $(100, 2000, 15, 4)$. The algorithm was also tested on a dataset taken from the *E. coli* genome, with values $(2645, 2000, 9, 2)$, and required 37 minutes and 991 megabytes of memory.

3.2 Parallelizations

The nature of the search in the algorithm makes it a prime candidate for distributed search. Practical aspects of such distributed searching are facilitated by the linear space requirements. The CENSUS algorithm can be parallelized to achieve $O(1)$ supersteps within the bulk-synchronous parallel (BSP) model [12]. BSP models parallelism using virtual processors that are mapped during execution to a smaller number of actual processors. An algorithm's computation is broken into *supersteps*, units of processing and communication that represent the necessary synchronization. All virtual processors must complete each specific superstep before any proceed to the next superstep. A parallel algorithm with a large superstep complexity is considered to be *fine grained*; it needs high synchronization and short time intervals. If the number of supersteps is small, the algorithm is *coarse grained* and requires little synchronization between virtual processors, which is desirable for parallel algorithms. The algorithm is modified as follows to partition the search space.

1. Each processor computes the prefixes of motifs for which it will search (these prefixes are distributed uniformly among the processors). Each processor searches for motifs, and when finished, broadcasts the number found.
2. With the information about the number of motifs each processor has found, processor P computes the starting address it will use to write the lexicographic tree into global memory. Then P writes the lexicographic tree encoding its motif set into global memory.

Theorem 2. *For all $1 \leq p \leq N$, the partitioned search space algorithm takes $O((N/p)mnl)$ time and requires $O(mn)$ space per processor, while performing $O(1)$ supersteps.*

Proof. The time complexity of the supersteps follows from the time complexity of CENSUS. This algorithm only needs to be synchronized after each step in the modification description, so the number of supersteps is constant. □

Another way to parallelize the algorithm is to assign each string in $\mathcal{F}$ to a unique processor. Since the data for each input string is indexed in a separate

lexicographic tree, this would be feasible for systems without shared memory. This type of parallelization reduces the influence of the number of sequences (m) on the time complexity of Census.

1. Each processor constructs the lexicographic trees corresponding to its assigned sequences.
2. If the present motif prefix has length l, a motif has been found (a situation that can be handled in many ways). Otherwise, each processor makes the appropriate updates to the frontier sets based on the present motif prefix. When the frontiers have been updated, processors communicate whether or not to extend the present motif prefix, or backtrack. This step is repeated until the search is completed.

This is a fine grained parallelization, as the processors need to communicate after examining each extension. As such, we also consider the time for the communication required to direct the search.

Theorem 3. *Using p processors, the problem can be solved in $O(l(mn/p + \log p)N))$ time and $O(n)$ space per node.*

Proof. The factor of N in the time complexity of Census corresponds to the search space that is traversed; this factor remains untouched by the parallelization. Similarly, the factor ln corresponds to the frontiers that must be maintained. Since disjoint sets of m/p frontiers are associated with distinct processors, they are updated independently in parallel, thus eliminating a factor of p from the time complexity. The only additional work to be accounted for is that required to determine when to backtrack. This requires communication, and essentially computing a logical "or" of a value from each processor. Done carefully, this requires $O(\log p)$ time. □

4 Near Optimal Enumeration

In this section we describe how to eliminate a factor of l from the time complexity required by the enumeration. The result is an algorithm that requires $O(mnN)$ time, and suggests an efficient parallelization that will be described in Section 5. The algorithm is based on a new data structure, the neighborhood tree, that concisely encodes the (l, d)-neighborhood for a set of strings.

Definition 2. (neighborhood tree) *For any set $\mathcal{F}$ of strings, each of length $n \geq l$, the (l, d)-neighborhood tree $T_{l,d}(\mathcal{F})$ for $\mathcal{F}$ is a rooted directed tree satisfying four conditions: 1. each edge is labeled with a string; 2. any two edges out of the same node have labels beginning with distinct characters; 3. any internal node has out-degree at least 2; and 4. every string in $z \in N_{l,d}(\mathcal{F})$ maps to some leaf u of T such that the string formed by concatenating in order the labels on the path from the root to u exactly spell out z, and every leaf of T is mapped to some $z \in N_{l,d}(\mathcal{F})$. When $\mathcal{F} = \{S\}$, we write $T_{l,d}(\mathcal{F})$ as $T_{l,d}(S)$ for convenience.*

The (l, d)-neighborhood tree has the important property that, given a query string s of length l, it is capable of answering whether s is contained in the (l, d)-neighborhood of $\mathcal{F}$, and can do so in time proportional to size of the query (*i.e.* $|s|$). Our goal is to build and represent these structures in time and space proportional to the number of leaves in the tree, which is exactly $|N_{l,d}(\mathcal{F})|$. To accomplish this we must avoid explicitly representing the edge labels, as doing so would require $\omega(1)$ space per node. Our method is inspired by the edge compression used in linear time suffix tree algorithms [8], which represent edge labels by indexing substrings of the underlying strings. The strategy does not transfer directly to neighborhood trees since not all substrings of members of $N_{l,d}(\mathcal{F})$ occur exactly as substrings of members of $\mathcal{F}$. The representation we use is based on an observation about $T_{l,d}(s)$ for a string s of length l.

Property 1. Let s be a string with $|s| = l$. For any edge $(u, v) \in T_{l,d}(s)$, if the string labeling edge (u, v) has length $x > 1$, then v is a leaf and the string labeling (u, v) may differ by at most 1 position from the suffix of s having length x. In addition, such a mismatch can only occur at the first position of the label.

Thus, in the restricted case of an (l, d)-neighborhood tree for a string s with $|s| = l$, any edge label may be represented in constant space. It is sufficient to index the beginning and end of a substring in s, and indicate the character occupying the first position of the label (which, by Property 1, is the only position where the character in the label may not match the character in the indexed substring of s).

We describe an algorithm to construct (l, d)-neighborhood trees for strings of length l. The reason we consider this restricted case is that the (l, d)-neighborhood tree for a string S of length $n > l$ may be obtained by taking the union of the (l, d)-neighborhood trees for each length l substring of S. The construction algorithm is based on the following recurrence. The symbol $\circ$ denotes concatenation and when applied to a set operates on each member of the set.

Property 2. Let s and s' be strings with $|s| = l$ and $|s'| = l - 1$. If $s = \alpha \circ s'$ for some $\alpha \in \Sigma$, then

$$N_{l,d}(s) = \Big(\alpha \circ N_{l-1,d}(s')\Big) \bigcup \Big(\cup_{\beta \in \Sigma} \beta \circ N_{l-1,d-1}(s')\Big). \tag{1}$$

The recursive characterization of a neighborhood suggests a recursive algorithm for constructing a neighborhood tree. The information stored at each node in the tree includes numbers indexing a substring in the underlying string, and a modifier character that might override the first character indexed. We note that for this restricted case, the modifier alone is sufficient since edges with labels of length > 1 are incident on leaves, and their index is completely determined by the depth of the leaf. The reason for using the indexes is that they will be necessary later when constructing (l, d)-neighborhood trees for strings of length $> l$. We also anticipate the extension to neighborhood trees for sets of strings, and therefore assume the identity of each string is encoded along with the pair of indexes. The following algorithm is based on the recursion in Property 2.

Algorithm 2: Pseudocode for the BUILDNEIGHBORHOOD algorithm.
Input: The root of an $(l, 0)$-neighborhood tree for a string s. A distance parameter d.
Output: The root of an (l, d)-neighborhood tree $T_{l,d}(s)$ for s.
BUILDNEIGHBORHOOD(v, d)

```
1.  if d > 0 and v is not a leaf
2.      for each β ∈ Σ \ {α}
3.          Create child u_β of v with index (depth(v)+1, depth(v)+1)
            and modifier character β.
4.          Create child x_β of u_β with index (depth(v) + 2, |s|) and
            no modifier character.
5.          BUILDNEIGHBORHOOD(u_β, d − 1)
6.      Let x_α be the original child of v. Create node u_α with index
        (depth(v)+1, depth(v)+1) and no modifier character. Increment
        the start index of x_α, insert x_α below u_α, and replace
        x_α with u_α as child of v.
7.      BUILDNEIGHBORHOOD(u_α, d)
8.  return v
```

Lemma 1. *For any string s, such that $|s| = l$, the (l, d)-neighborhood tree of s can be built in $O(N)$ time.*

Proof. First, notice that the total number of leaves in the tree is $O(N)$, since they are in 1-to-1 correspondence with members of $N_{l,d}(s)$. Because we use indexes instead of explicitly representing edge labels, the size of each node is constant, as is the time to create each node. Finally, since all nodes have out-degree 2 or 0, the total number of nodes is proportional to the number of leaves in the tree. □

After constructing $T_{l,d}(s_{ij})$ for each length l substring s_{ij} of each $S_i \in \mathcal{F}$, we take the union of these structures to obtain each $T_{l,d}(S_i)$. The intersection of all $T_{l,d}(S_i)$ gives $T_{l,d}(\mathcal{F})$. For two (l, d)-neighborhood trees $T_{l,d}(s)$ and $T_{l,d}(s')$, corresponding to strings s and s', the union $T_{l,d}(s) \cup T_{l,d}(s')$ is defined as the (l, d)-neighborhood tree encoding $N_{l,d}(s) \cup N_{l,d}(s')$. The intersection of neighborhood trees is defined similarly with respect to the intersection of the neighborhoods. The union and intersection operations are accomplished by recursively applying the operations to subtrees. This requires being able to determine the extent to which two nodes are identical, which requires determining the length of the longest identical prefix between two edge labels. As an example, let $s = abcd$ and $s' = abba$, and consider $T_{4,0}(s)$ and $T_{4,0}(s')$ which each have two nodes. The union $T_{4,0}(s) \cup T_{4,0}(s')$ has four nodes (a root with one child that has two children), and three edge labels (ab, cd and ba). In order to determine the length (and label) of the edge labeled with ab while taking the union, we are required to determine the length of the longest prefix of s and s'. Our goal is to do this in constant time regardless of the length of the edge labels, so simply matching the

strings does not suffice. The problem is handled using *longest common extension* queries, as explained in the proof of the next lemma.

Lemma 2. *The union and intersection operations for neighborhood trees can be performed in time bounded by a linear function of the size of the input structures.*

Proof. In a neighborhood tree resulting from a union or intersection operation, the number of nodes is bounded by a linear function of the total number of nodes in the trees being operated on. The union and intersection algorithms proceed by recursively doing unions and intersections on the appropriate subtrees of the input structures, and each node in the input structures need only be visited once. When the length of each edge label is $O(1)$, we need only spend constant time at each node in the input structures to determine the identity of the nodes to be created in the resulting structure. So for this restricted case, the set operations take linear time.

The only complication arises when two edge labels of length $\omega(1)$ must be compared to determine the length of their longest common prefix (as illustrated by the example above). Sequentially matching individual characters requires time proportional to the length of the shorter of the two edge labels. We use longest common extension queries to speed up the comparison. Given a pair of start indexes (i, j) for two substrings from (not necessarily distinct) strings S and S', the longest common extension for (i, j) is the length of the longest prefix of suffix i of S that matches a prefix of suffix j of S'. The longest common extension for the starting indexes of two edge labels is equal to the length of the longest prefix that is identical in the two labels.

After linear time preprocessing, longest common extension queries can be done in constant time. This is implemented by (1) creating a generalized suffix tree for the strings, which can be done in linear time by a number of methods, and (2) augmenting the tree for lowest common ancestor queries, which can also be done in linear time (for details see [6]). After creating and augmenting the generalized suffix tree, lowest common ancestor queries can be answered in constant time. The depth of the lowest common ancestor for two leaves gives the length of the longest common extension for the corresponding suffixes.

Therefore, even when arbitrary length edge labels are allowed, the edge labels can be compared in constant time during union and intersection operations. So linear time is sufficient for union and intersection operations in the general case. □

While enumerative algorithms must have their running times dependent on the output size, we avoid this by encoding the set of motifs in a structure instead of producing each motif. The following algorithm is the best known that solves this modified problem; it also provides the best known upper bound on the complexity of the corresponding decision problem.

Algorithm 3: Pseudocode for the MotifIntersection algorithm.
Input: A set of strings $\mathcal{F}$ and two integers $d < l$.
Output: An edge-compressed neighborhood tree $T_{l,d}(\mathcal{F})$ encoding $\mathcal{M}_{\mathcal{F}}$.
MotifIntersection$(\mathcal{F}, l, d)$
1. **for each** $S_i \in \mathcal{F}$
2. **for each** substring s_{ij} of S_i such that $|s_{ij}| = l$
3. construct $T_{l,d}(s_{ij})$ using BuildNeighborhood
4. $T_{l,d}(S_i) \leftarrow \cup_{1 \leq j \leq n-l+1} T_{l,d}(s_{ij})$
5. $T_{l,d}(\mathcal{F}) \leftarrow \cap_{1 \leq i \leq m} T_{l,d}(S_i)$
6. **return** $T_{l,d}(\mathcal{F})$

Theorem 4. *The time complexity of* MotifIntersection *is* $O(mnN)$.

Proof. By Lemma 1, each call to BuildNeighborhood requires $O(N)$ time. There are $O(m)$ union operations, which by Lemma 2, each require $O(nN)$ time. Also by Lemma 2, the intersection operation requires at most $O(mnN)$ time. □

Once the structure has been built, the complete list of motifs can be extracted in $O(lN)$ time, so the enumeration can be done in $O(mnN + lN) = O(mnN)$ time. The space requirements of the algorithm are the same as the time requirements.

5 An FPP Algorithm

Of more theoretical interest is the complexity of the problem when we are allowed a polynomial number of processors. The class of algorithms that achieve a logarithmic running time using polynomial number of processors is called NC [4]. Two analogues of NC have been defined within the context of parameterized complexity [3]. For a problem Π with parameter k, the class PNC contains problems with algorithms requiring at most $O(f(k)(\log |x|)^{g(k)})$ time, using $O(h(k)|x|^c)$ processors, on instance $x \in \Pi$, for some constant c and arbitrary functions f, g, h. The definition of the class FPP modifies the allowed time to be $O(f(k)(\log |x|)^c)$, so FPP $\subseteq$ PNC $\subseteq$ FPT (see [5] for definitions of the basic concepts of parameterized complexity).

The algorithm can be seen as an adaptation of the MotifIntersection algorithm, where the set operations proceed from the leaves to the root of a binary processor tree. For any processor p at an internal node in this tree, subscripts L and R indicate data held by the left and right children of p. Leaf processors are assigned substrings of the strings in $\mathcal{F}$, and all processors assigned a substring from the same string are arranged consecutively. For the purpose of illustration, we assume both m and $n - l + 1$ are powers of 2. Our algorithm requires that the neighborhood trees do not have compressed edges, so that construction of the suffix trees required for longest common extension queries may be avoided.

The simple lexicographic trees used instead have a number of nodes bounded by $f(|\Sigma|, l) = O(|\Sigma|^l)$.

Algorithm 4: Pseudocode for the FPPMotifs algorithm.
Input: A set $\mathcal{F}$ of strings.
Output: A lexicographic tree encoding $\mathcal{M}_{\mathcal{F}}$.

```
FPPMotifs(F)
1.    for each leaf processor p (in parallel)
2.        (p computes) T_ij ← BuildNeighborhood(s_ij)
3.    for k ← 2 to log(n − l + 1)
4.        for each processor p (in parallel)
5.            if p is at level k then (p computes) T ← T_L ∪ T_R
6.    for k ← log(n − l + 1) + 1 to log(m(n − l + 1))
7.        for each processor p (in parallel)
8.            if p is a level k then (p computes) T ← T_L ∩ T_R
9.    return T
```

The above algorithm establishes the following result concerning the parallel parameterized complexity of the motif enumeration problem.

Theorem 5. *The motif enumeration problem can be solved in* $O(f(|\Sigma|, l) \log(mn))$ *time using* $O(mn)$ *processors.*

6 Discussion

We have presented an algorithm for enumerating all motifs for a set of sequences. This algorithm, presented as Census and MotifIntersection in two stages of incremental improvement, improves over the best previously known algorithm, in that the running time has been reduced by eliminating a factor of m, the number of strings, and of l, the length of the motifs sought. The space requirements have also been reduced by eliminating a factor of m. These improvements advance the frontier of which parameter values are usable for motif enumeration.

Given the potentially exponential size of the output, this algorithm must be close to optimal in its running time for the worst case. Consider the task of any enumerative algorithm that tries to obtain all motifs for a set of sequences. In the worst case, there are $N = \Theta(n\binom{l}{d}|\Sigma|^d)$ motifs, so no algorithm can eliminate the factor $nN = n\binom{l}{d}|\Sigma|^d$ from the time complexity. Also, the factor mn represents the size of the input, so those two cannot be separated; any algorithm must thus take at least $\Omega(mn+nN) = \Omega(mn+n\binom{l}{d}|\Sigma|^d)$ time. It seems unlikely that the factor m can be separated from $N = \binom{l}{d}|\Sigma|^d$ through some sort of preprocessing without introducing a factor of $|\Sigma|^l$ into the time complexity.

This algorithm can be parallelized in a variety of ways. It admits both a coarse and a finer grained parallelism. The coarse grained parallel algorithm reduces the

time by a factor of p, for any number of processors $1 \leq p \leq N$, though it requires $O(mn)$ space for each processor. The finer grained algorithm, on the other hand, runs in $O(l(mn/p + \log p)N)$ time and $O(n)$ space for each of p processors. Our motif enumeration can also be parallelized to run in $O(f(|\Sigma|, l)\log(mn))$ time, using a linear number of processors, which positions the problem in FPP.

Acknowledgments. The authors thank Todd Wareham for helpful comments and discussions.

References

1. A. Aho. Algorithms for finding patterns in strings. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 257–300. MIT Press/Elsevier, 1990.
2. M. Blanchette, B. Schwikowski, and M. Tompa. An exact algorithm to identify motifs in orthologous sequences from multiple species. *Proceedings of the Annual International Conference on Computational Molecular Biology*, pages 37–45. ACM Press, 2000.
3. Marco Cesati and Miriam Di Ianni. Parameterized parallel complexity. Technical Report 4(6), *Electronic Colloquium on Computational Complexity (ECCC)*, 1997.
4. Stephen A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1-3):2–21, 1985.
5. R. Downey and M. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer-Verlag, New York, 1999.
6. D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.
7. J. van Helden, B. Andre, and J. Collado-Vides. Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *Journal of Molecular Biology*, 281:827–842, 1998.
8. E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the Association for Computing Machinery*, 23(2):262–272, 1976.
9. Marie-France Sagot. Spelling approximate repeated or common motifs using a suffix tree. In C. L. Lucchesi and A. V. Moura, editors, *Latin '98: Theoretical Informatics*, volume 1390 of *Lecture Notes in Computer Science*, pages 374–390. Springer, 1998.
10. S. Sinha and M. Tompa. A statistical method for finding transcription factor binding sites. *Proceedings of the Annual International Symposium on Intelligent Systems for Molecular Biology*, 2000. AAAI Press, pages 344–344.
11. Martin Tompa. An exact method for finding short motifs in sequences, with application to the ribosome binding site problem. *Proceedings of the Annual International Symposium on Intelligent Systems for Molecular Biology*, 1999. AAAI Press, pages 262–271.
12. L. G. Valiant. A Bridging Model for Parallel Computation. *Communications of the Association for Computing Machinery*, 33(8):103-111, 1990.
13. A. Vanet, L. Marsan, A. Labigne and M.-F. Sagot. Inferring regulatory elements from a whole genome. An application to the analysis of the genome of Helicobacter pylori σ^{80} family of promoter signals. *Journal of Molecular Biology*, 297:335–353, 2000.

Common-Deadline Lazy Bureaucrat Scheduling Problems

Behdad Esfahbod, Mohammad Ghodsi, and Ali Sharifi

Computer Engineering Department
Sharif University of Technology, Tehran, Iran,
{behdad,ghodsi}@sharif.edu, ali@bamdad.org

Abstract. The lazy bureaucrat scheduling is a new class of scheduling problems that was introduced in [1]. In these problems, there is one employee (or more) who should perform the assigned jobs. The objective of the employee is to minimize the amount of work he performs and to be as inefficient as possible. He is subject to a constraint, however, that he should be busy when there is some work to do.
In this paper, we focus on the cases of this problem where all jobs have the same common deadline. We show that with this constraint, the problem is still NP-hard, and prove some hardness results. We then present a tight 2-approximation algorithm for this problem under one of the defined objective functions. Moreover, we prove that this problem is weakly NP-hard under all objective functions, and present a pseudo-polynomial time algorithm for its general case.

Keywords: Scheduling Problems, Approximation Algorithms, Dynamic Programming, NP-hardness.

1 Introduction

In most scheduling problems, there is a number of jobs to be performed by some workers or employees. Studies have looked at these problems with the objective to perform the assigned jobs as efficient as possible and to maximize the number of completed jobs. This is the employer's point of view. We can also look at these problems from employee's point of view, some of whom do not have enough motivation to do their jobs efficiently. Some may even want to be as inefficient as possible while performing their duties. We call such employees *lazy*, and such scheduling problems has been classified as *Lazy Bureaucrat Scheduling Problems* (LBSP). LBSP was introduced in [1] and some results were presented there. A summary of some of them is presented here.

More specifically, we are given a set J of n jobs $j_1, \ldots, j_n$. Job j_i has *processing time* of t_i, *arrival time* of a_i , and *deadline* of d_i ($t_i \leq d_i - a_i$). It is assumed that t_i, a_i, and d_i are nonnegative integers. Jobs have hard deadlines, that is, j_i can only be executed during its allowed interval $w_i = [a_i, d_i]$, called its *window*. It is also assumed that there always exists some job which arrives at time 0. The maximum deadline time is also denoted by D.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 59–66, 2003.
© Springer-Verlag Berlin Heidelberg 2003

We study the non-preemptive case of this problem and restrict ourselves to off-line scheduling in which all jobs are known to the scheduler beforehand. We also assume that there is only one processor (employee) available to do the jobs. Some results on preemptive case and also cases with multiple bureaucrats can be found in [1,4].

Definition 1. (Executable Job) A job j_i is called *executable* at some time t, if and only if it has been arrived, its deadline has not yet passed, it is not processed yet, and if it is started now, it will be fully processed before its deadline (that is, $a_i \leq t \leq d_i - t_i$).

Definition 2. (Greedy Requirement) At any time, the bureaucrat should work on an executable job, if there is any such job.

The bureaucrat is asked to process the jobs within their windows satisfying the above greedy requirement. His goal is to be as inefficient as possible. This is captured by any of the following objective functions that is to minimize.

1.1 Objective Functions

Four objective functions have been defined for this problem [1].

1. `[min-time-spent]`: *Minimize the total amount of time spent working.* This objective naturally appeals to a *lazy* bureaucrat.
2. `[min-weighted-sum]`: *Minimize the weighted sum of completed jobs.* This objective appeals to a *spiteful* bureaucrat whose goal is to minimize the fees that the company collects on the basis of his labors, assuming that the fee is collected only for those tasks that are actually completed.
3. `[min-makespan]`: *Minimize the makespan, the maximum completion time of the jobs.* This objective appeals to an *impatient* bureaucrat, whose goal is to go home as early as possible, at the completion of the last job, when he is allowed to leave. He cares only about the number of hours spent at the office, not the number of hours spent doing work (productive or otherwise).
4. `[min-number-of-jobs]`: *Minimize the total number of completed jobs.* This can be meaningful when the overhead of performing a job is too high for the bureaucrat.

Clearly, the objective function 1 is a special case of 2, as we can define the weight of each job to be equal to its length. Objective function 4 is also a special case of 2, when the weights are all equal. Also, if all jobs have the same arrival time, the objective functions 1 and 3 are equivalent.

1.2 Previous Related Results

The main related results in [1] are summarized below:

– LBSP is strongly NP-hard [2] under all objective functions and is not approximable to within any fixed factor.

- LBSP with the same arrival times for the jobs, is weakly NP-hard, and can be solved by a pseudo-polynomial dynamic programming algorithm.
- LBSP with all the jobs having unit lengths, can be solved in polynomial time by the Latest Deadline First (LDF) scheduling policy.
- Assuming for each job i, $d_i - a_i < 2t_i$, LBSP can be solved in $O(nD \max(n, D))$ time.
- Even with a bound on δ (the ratio of the longest job to the shortest job), LBSP is strongly NP-hard. It cannot be approximated to within a factor of $\delta - \epsilon$, for any $\epsilon > 0$, unless $P = NP$.
- Given bounds on R (the maximum ratio of window length to job length) and δ, LBSP can be solved in $O(Dn^{4R \lg \delta})$.

In [3], these results have been shown:

- LBSP with all jobs having unit lengths, can be solved in polynomial time by the Latest Deadline First (LDF) scheduling policy, even with more than one bureaucrat (worker, processor).
- Assuming $d_i - a_i < 2t_i$ for each job i, LBSP can be solved in $O(nD)$ time.

Hepner and Stein have studied the preemptive case [4] where they propose pseudo-polynomial time algorithm to minimize makespan of such schedule. They have also extended this scheduling problem to the multiple-bureaucrat setting and provided pseudo-polynomial time algorithms for such problems.

1.3 Our Results

We consider a restricted case of LBSP where the deadlines for all jobs are the same (denoted by D). We call these problems common-deadline LBSP, denoted by CD-LBSP. This problem can be considered with any of the above objective functions. We denote such cases by CD-LBSP`[objective-function]` and CD-LBSP`[*]` is used to denote all these objective functions.

On the hardness of this problem, we first prove that CD-LBSP`[*]` is still NP-hard and show that CD-LBSP`[min-number-of-jobs]`is not approximable to within any fixed factor. But, for CD-LBSP`[min-makespan]`, we provide a tight 2-approximation algorithm. Finally, we prove that CD-LBSP`[*]` is weakly NP-hard; we provide a pseudo-polynomial time dynamic programming algorithm for the general case.

2 Hardness Results

In the following theorems we reduce the Subset Sum problem to prove that CD-LBSP`[*]` is NP-hard, but existence of approximation algorithms is not the same under different objective functions. We have found reasonable results for `[min-weighted-sum]`, `[min-makespan]`, and `[min-number-of-jobs]` objective functions, but not for `[min-time-spent]`.

Theorem 1. CD-LBSP`[*]` *is NP-hard.*

Proof. We reduce the SUBSET SUM problem to this problem. We are given $S = \{x_1, \ldots, x_n\}$ of n positive integers with $\Sigma_{i=1}^{n} x_i = s$, and an integer b $(0 < b < s)$. It is asked whether there is a subset T of S, satisfying $\Sigma_{x \in T}\, x = b$? Without loss of generality, we assume that $b \leq \lfloor \frac{s}{2} \rfloor$ and $x_i < b$ for all i.

We construct an instance of CD-LBSP containing $n + 1$ jobs, all having deadlines $D = 2s$. For each $x_i \in S$, we define a job j_i $(1 \leq i \leq n)$ with arrival time $a_i = 0$ and processing time $t_i = 2x_i$. The last job j_{n+1} has arrival time $a_{n+1} = 2b$ and processing time $t_{n+1} = 2s - 2b - 1$.

The bureaucrat is to avoid working up to time $2s$, and to finish by $2s - 1$. This can be done if and only if he starts j_{n+1} at time of $2b$ and finishes it at time $2s - 1$. This is the case if and only if there is a subset of $\{j_1, \ldots, j_n\}$ with total processing time $2b$, which is equivalent to a solution for the SUBSET SUM problem.

This argument is clearly correct for objective functions `[min-time-spent]`, `[min-weighted-sum]`, and `[min-makespan]`. For `[min-number-of-jobs]`, the assumptions we made for the data in the SUBSET SUM problem is used to prove this case. □

Theorem 2. CD-LBSP `[min-number-of-jobs]` *is not approximable to within any fixed factor* $\Delta > 1$*, unless* $P = NP$.

Proof. As in theorem 1, we reduce the SUBSET SUM problem to prove the hardness. Assume by contradiction that there is an approximation algorithm with fixed factor Δ for CD-LBSP `[min-number-of-jobs]`. Using this assumption, we will show that we can solve the SUBSET SUM problem leading to $P = NP$.

Let $m = \lceil \Delta \rceil$ and $D = b + m(n + 2)s$. We construct an instance of CD-LBSP `[min-number-of-jobs]` containing the following jobs, all with deadline D. For each $x_i \in S$, we define an *element job* j_i $(1 \leq i \leq n)$ having $a_i = 0$, and $t_i = x_i$. One *long job*, j_{n+1} with $a_{n+1} = b$ and $t_{n+1} = D - b$. We also define $m(n + 2) - 1$ *extra jobs*, all having arrival times b, and processing times s.

The bureaucrat wants to do as few jobs as possible. We claim that the answer to the SUBSET SUM problem is 'yes', if and only if the bureaucrat performs the long job. In this case, he should be working up to time b, which leads to the answer of the SUBSET SUM problem and he has processed at most $n + 1$ jobs (at most n element jobs and one long job), so the approximation algorithm will produce an output with at most $m(n + 1)$ jobs. On the other hand, if he does not process the long job, then he has to process $m(n + 2) - 1$ extra jobs (he has enough time to process all element and extra jobs), so the approximation algorithm will end with more than $m(n + 1)$ jobs.

Then, the answer to the SUBSET SUM problem is 'yes' if and only if there are at most $m(n + 1)$ jobs in the output of the algorithm, and 'no' otherwise. □

Corollary 1. CD-LBSP `[min-weighted-sum]` *is not approximable to within any fixed factor* $\Delta > 1$*, unless* $P = NP$.

Proof. We know that CD-LBSP `[min-number-of-jobs]` is a special case of CD-LBSP `[min-weighted-sum]`. Theorem 2 completes the proof. □

3 Approximation Algorithm

It is shown in this section that, under objective function 3 (`[min-makespan]`), the bureaucrat can reach a nearly optimal solution with a simple algorithm.

For a schedule σ, we say $j_i \in \sigma$ if job j_i is processed in σ. Also, we use $s_i(\sigma)$ and $f_i(\sigma)$ to denote the time when the processing of j_i is started and the time when it is finished in σ respectively ($f_i(\sigma) = s_i(\sigma) + t_i$).

Theorem 3. *The Shortest Job First (SJF) scheduling policy is a 2-approximation algorithm for* CD-LBSP`[min-makespan]` *and this bound is tight.*

Proof. Let σ_{OPT} be an optimal solution and σ be the schedule which the SJF policy has generated, and OPT and SJF be their makespans respectively. In the SJF policy, among the executable jobs, the one with the shortest processing time is picked first. We will show that $SJF - OPT < OPT$.

Without loss of generality, suppose that jobs $j_1 \dots j_k$ (for some k) are processed in σ in that order. Let $j_q \in \sigma$ be the job that is being processed at time OPT in σ. That is, $s_q(\sigma) < OPT \leq f_q(\sigma)$. We know that $a_i < OPT$ for all jobs j_i. The SJF policy, therefore, forces that $t_{q+1} \leq t_{q+2} \leq \dots \leq t_k$. Note that there is no gap between jobs $j_q, \dots, j_k$ in σ. From the greedy requirement, we can easily conclude that $j_i \in \sigma_{OPT}$ for all $q+1 \leq i \leq k$ (otherwise, at least one of them can be processed at time OPT).

Assume that $q < k$. The $q = k$ case is treated similarly and we omit it. We consider two different cases:

- $t_q \leq t_{q+1}$: With the same argument as above, we have $j_q \in \sigma_{OPT}$. Therefore, since $s_q(\sigma) < OPT$, we have $SJF - OPT < \Sigma_{i=q}^{k} t_i \leq OPT$.
- $t_q > t_{q+1}$: If job $j_q \in \sigma_{OPT}$, then we will have $SJF - OPT < \Sigma_{i=q}^{k} t_i \leq OPT$ and $SJF < 2OPT$; otherwise, we can show that there exists a job j_p, not shorter than j_q, such that $j_p \in \sigma_{OPT}$ and $j_p \notin \sigma$. Let $Q = \{j_i \mid q < i \leq k\}$. All jobs in Q are shorter than j_q, otherwise, there will be enough time to process j_q at time OPT, which means that σ_{OPT} is invalid. Also, from the SJF policy, we conclude that all jobs in Q have arrived after $s_q(\sigma)$.
  Let T be the set of jobs $j_i \in \sigma_{OPT}$ such that $s_i(\sigma_{OPT}) \leq s_q(\sigma)$. Obviously, T cannot be empty. Not all jobs in T can be processed in σ. To show this, consider a subset of T which has been processed continuously (without any break) and has the maximum start time among these subsets. The finish time of this subset is after time $s_q(\sigma)$. From the greedy requirement, we conclude that this set of jobs cannot be processed and finished by time $s_q(\sigma)$, and thus they cannot all be processed in σ (because job j_q has been already started at time $s_q(\sigma)$.) Thus, there is some job $j_p \notin \sigma$, with $a_p \leq s_q(\sigma)$. The SJF policy forces that $t_q \leq t_p$, and also $j_p \in \sigma_{OPT}$. Putting all together leads to

$$SJF - OPT < t_q + \Sigma_{i=q+1}^{k} t_i \leq t_p + \Sigma_{i=q+1}^{k} t_i < t_p \leq OPT.$$

To prove the tightness, for a given n and $0 < \epsilon < 1$, we construct an instance of CD-LBSP with n jobs such that the SJF policy does not do better than

$2-\epsilon$. Such instance contains n jobs with zero arrival times and deadlines D. Let $D = n-3+2L-1 = n+2L-4$ with $L = 2n/\epsilon$. The first three jobs have $t_1 = L-1$, $t_2 = L$, and $t_3 = L+1$. The next $n-3$ jobs, $j_4, \ldots, j_n$, all have $t_i = 1$.

The σ_{OPT} should process all $j_4, \ldots, j_n$ jobs and then j_3, having makespan $OPT = n-3+L+1 = n+L-2$. There is not enough time to process j_1 or j_2.

The SJF schedule processes $j_4, \ldots, j_n$, j_1, and j_2 having makespan $SJF = n-3+L-1+L = n+2L-4$. Thus, we have

$$\frac{SJF}{OPT} = \frac{n+2L-4}{n+L-2} = \frac{n\epsilon+4n-4\epsilon}{n\epsilon+2n-2\epsilon} = \frac{\epsilon+4-\frac{8}{L}}{\epsilon+2-\frac{4}{L}} > 2-\frac{\epsilon}{2(1-\frac{2}{L})} > 2-\epsilon,$$

which completes the proof. □

It looks quite likely that the same algorithm yields the same approximation bound under function `[min-time-spent]`, but we do not have a complete proof.

4 Pseudo-Polynomial Time Algorithms

We assume that the jobs are numbered in order of their arrival times (that is, $a_1 \leq a_2 \leq \ldots a_n$). Let T_i and $T_{i,k}$ denote the set of jobs $j_i, j_{i+1}, \ldots, j_n$ and $j_i, j_{i+1}, \ldots, j_k$ respectively. We will also use the following definitions:

Definition 3. The time α is called the *first rest time* of a schedule σ, if the bureaucrat has paused processing the jobs in σ for the first time at α. If there is no pause during σ, the first rest time is defined as the time when the schedule is finished.

Definition 4. For a time α, we define *critical jobs* H_α as the set of jobs $j_i \in J$ which can be processed in $[\alpha, D]$, i.e. $\max(a_i, \alpha) + t_i \leq D$.

Definition 5. For a given (T, α, U) in which $T, U \subset J$ and α is a time point, sequence E of some jobs in T is said to be a *valid sequence* if we can process these jobs in this order without any gaps in between, starting from first arrival time of the jobs in T and finishing at α such that every job in $T \cap U$ appears in E. A valid sequence E is said to be an *optimal sequence* under some objective function, if its cost is minimum among all valid sequences of (T, α, U).

Lemma 1. *For a given (T, α, U), let E be an optimal sequence and $j_m \in E$ be the job with the latest arrival time. There exists another optimal sequence F in which j_m is the last processed job.*

Proof. This can easily be done by repeated swapping of j_m with its adjacent jobs. □

Lemma 2. *There is a pseudo-polynomial time algorithm that finds the optimal sequence for any given (T_i, α, U) under any of the objective functions, if such sequence exists $(1 \leq i \leq n)$.*

Proof. Let j_f be the last job arrived before α in T_i, and $C_{x,y}$ ($i \leq x \leq f, a_i \leq y \leq \alpha$) be the cost of the optimal sequence for $(T_{i,x}, \alpha, U)$, or ∞ if no such optimal sequence exists. Our goal is to compute $C_{f,\alpha}$. We show how $C_{x,y}$ can be computed recursively from the values of $C_{x',y'}$, where $x' < x$ and $y' \leq y$.

If $j_x \in U$, then it is in any valid sequence. Hence, from lemma 1, j_x can be processed last in $[y - t_x, y]$. Based on the objective function used, we can easily compute $C_{x,y}$ from $C_{x-1,y-t_x}$. For example, $C_{x,y} = C_{x-1,y-t_x} + t_x$ under `[min-time-spent]`.

On the other hand, if $j_x \notin U$, there are two options depending on whether or not it is in the optimal sequence. If j_x is processed in the optimal sequence, it can be processed last, in which case, $C_{x,y}$ can be computed from $C_{x-1,y-t_x}$ as before. Otherwise, $C_{x,y} = C_{x-1,y}$, since we can ignore j_x. The minimum of these two values is taken for $C_{x,y}$.

The running time of this algorithm is $O(nD)$, as there are at most nD values of $C_{x,y}$ to compute. □

Theorem 4. CD-LBSP`[*]` *is weakly NP-hard.*

Proof. We present a pseudo-polynomial time algorithm which can be used for any of the objective functions.

Consider T_i for some $1 \leq i \leq n$ and temporarily assume that the jobs in T_i are the only jobs available, and that the greedy requirement is to be satisfied on only these jobs. Let P_i be this subproblem and C_i be its optimal value. Clearly, C_1 is the desired value.

Consider an optimal schedule σ for P_i. Let α be the first rest time in σ. No job in T_i arrives at α. We know that the jobs in T_i appearing in the set of critical jobs H_α should be processed before the rest time α.

Let j_k be the first job arrived after α. Because of the pause at time α, we know that no job having arrival time less than α can be processed after α. So, we can break up the schedule σ into two subschedules: those processed before α and those processed after. These subschedules are independent. We can consider the first subschedule as a valid sequence for $(T_{i,k-1}, \alpha, H_\alpha)$. From the optimality of σ, it is clear that this sequence is optimal. Similarly, the second subschedule is an optimal schedule for P_k.

We compute C_i for every $1 \leq i \leq n$ from the values of C_j ($i < j \leq n$) for all times α ($a_i < \alpha \leq D$). Note that we only consider those α's at which time there is no job arrival. It is first checked whether there exists an optimal sequence for $(T_{i,k-1}, \alpha, H_\alpha)$. If there is no such sequence, there will be no schedule for T_i having α as the first rest time; otherwise, let C be the cost of that sequence. We know that the lowest cost of a schedule for T_i having α as the first rest time can be computed easily from the values of C and C_k and the objective function used. For example, under `[min-time-spent]` this is equal to $C + P_k$. The value of P_i is the minimum cost for different values of α.

The running time of this algorithm is $O(n^2D^2)$ because it calls the subroutine of finding optimal sequence at most $O(nD)$ times. □

5 Conclusion

In this paper, we studied a new class of the Lazy Bureaucrat Scheduling Problems (LBSP), called common-deadline LBSP, where the deadlines of all jobs are the same. We proved that this problem is still NP-hard under all four pre-defined objective functions. We also showed that this problem is not approximable to within any fixed factor in cases of `[min-weighted-sum]` and `[min-number-of-jobs]` objective functions. The problem is shown to have a tight 2-approximation algorithm under `[min-makespan]`. But, it is still open whether it is approximable under `[min-time-spent]`. In the rest of the paper, we presented pseudo-polynomial time dynamic programming algorithms for this problem under all objective functions. Further work on this problem is underway.

subsubsection*Acknowledgements. The authors would like to thank the anonymous referees for their useful comments.

References

1. Arkin, E. M., Bender, M. A., Mitchell, J. S. B., Skiena, S. S.: The lazy bureaucrat scheduling problem. Workshop on Algorithms and Data Structures (WADS'99), LNCS 1663, pp. 122–133, Springer-Verlag, 1999.
2. Gary, M. R., Johnson D. S.: Computers and intractability, a guide to the theory of NP-completeness. W. H. Freeman and Company, New York, 1979.
3. Farzan, A., Ghodsi, M.: New results for lazy bureaucrat scheduling problem. 7th CSI Computer Conference (CSICC'2002), Iran Telecommunication Research Center, March 3–5, 2002, pp. 66–71.
4. Hepner, C., Stein, C.: Minimizing makespan for the lazy bureaucrat problem, SWAT 2002, LNCS 2368, pp. 40–50, Springer-Verlag, 2002.

Bandwidth-Constrained Allocation in Grid Computing*

Anshul Kothari[1], Subhash Suri[1], and Yunhong Zhou[2]

[1] Department of Computer Science, University of California, Santa Barbara, CA 93106.
{kothari,suri}@cs.ucsb.edu
[2] Hewlett-Packard Laboratories, 1501 Page Mill Road, MS 1250, Palo Alto, CA 94304.
yunhong.zhou@hp.com

Abstract. Grid computing systems pool together the resources of many workstations to create a virtual computing reservoir. Users can "draw" resources using a pay-as-you-go model, commonly used for utilities (electricity and water). We model such a system as a capacitated graph, and study a basic allocation problem: given a set of jobs, each demanding computing and bandwidth resources and yielding a profit, determine which feasible subset of jobs yields the maximum total profit.

1 Introduction

Nearly all leading computer hardware vendors (IBM, Sun, Hewlett-Packard) have recently announced major initiatives in on-demand or grid computing. These initiatives aim to deliver computing resources as utilities (electricity or water)—users "draw" computing power or disk storage from a "reservoir" and pay only for the amount they use. Despite their different names (IBM's On-Demand computing, Sun's N1 computing and HP's Adaptive Infrastructure), the motivation behind these technologies is the same: many users (scientific labs, industries) often need extremely high computing power, but only for short periods of time. Examples include software testing of new systems or applications, verification of new chip designs, scientific simulations (geological, environmental, seismic), molecular modeling etc. Building and managing dedicated infrastructure is expensive, especially if its use is sparse and bursty. In addition, a vast amount of computing and disk capacity at enterprises is idle for large fraction of the time. These new initiatives aim to harness this power by creating a virtual computing reservoir.

The current grid systems only provide the CPU or disk units; *there is no bandwidth guarantee.* Many scientific simulations, as well as real-time applications like financial services, involve sustained high data transfer rates, and thus require a guaranteed application level bandwidth. The bandwidth is a different

* Anshul Kothari and Subhash Suri are supported in part by National Science Foundation grants IIS-0121562 and CCR-9901958.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 67–78, 2003.
© Springer-Verlag Berlin Heidelberg 2003

type of resource: it's a *link* resource, whereas computing cycles and disk units are *node* resources. We consider the following natural problem in this setting: given a set of tasks, each requesting some computing and some bandwidth resources and yielding a profit if chosen, determine which subset of jobs yields the maximum profit, given the current resources of the grid. We will only consider the *offline* version of the problem, leaving the online case as a future direction.

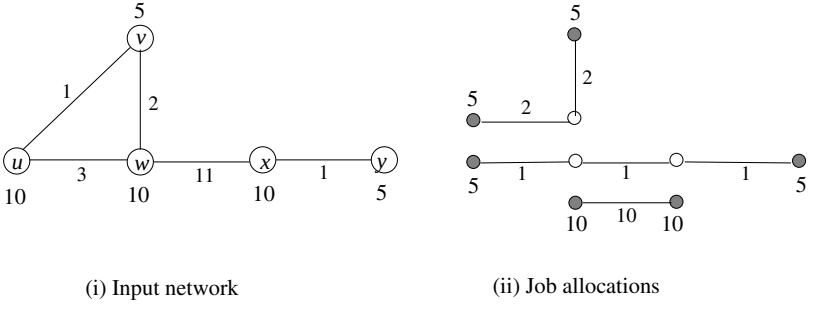


Fig. 1. An example with 3 jobs, $J_1 = \langle 20, 10, p_1 \rangle$, $J_2 = \langle 10, 1, p_2 \rangle$, $J_3 = \langle 10, 2, p_3 \rangle$. Figure (i) shows the input network. Numbers below the nodes denote the resource units available at that node; numbers next to links denote bandwidth. Figure (ii) shows an allocation where all 3 jobs are satisfied; the filled nodes contribute resource units.

We model the resource pool (grid) as an undirected graph $G = (V, E)$, with n nodes and m edges, where each node $v \in V$ has a computing resource $C(v)$, and each link (u, v) has a bandwidth $B(u, v)$. (We assume that the computing resources are expressed in a common unit, such as normalized CPU cycles.) We are given a set of k jobs, $J_1, J_2, \ldots, J_k$. The job J_i is specified by a triple $\langle c_i, b_i, p_i \rangle$, where c_i, b_i are the computing and the bandwidth resource needed by J_i, and p_i is the profit for this job if chosen. Let $C_i(v_k)$ denote the computing resource that v_k contributes to J_i, and let $B_i(u, v) \in \{0, b_i\}$ denote the bandwidth that (u, v) reserves for J_i. If job J_i is accepted, then we must have (i) $\sum_k C_i(v_k) \geq c_i$, namely, c_i units of the computing resource are allocated to J_i, and (ii) the set of edges $\{(u, v) \mid B_i(u, v) = b_i\}$ spans V_i. That is, the set of nodes that contribute computing resources for J_i must be *connected* by a subset of links with *reserved* bandwidth b_i. (Acceptance of a job is a binary decision: either it is accepted, or it is rejected; it cannot be partially accepted.) An index set of jobs $\mathcal{J}$ is *feasible* if neither the computing nor the bandwidth resource capacity is violated, that is, $\sum_{i \in \mathcal{J}} C_i(v_k) \leq C(v_k)$, for all nodes $v_k \in V$, and $\sum_{i \in \mathcal{J}} B_i(u, v) \leq B(u, v)$, for all links $(u, v) \in E$. See Figure 1 for an example. The total profit for the accepted jobs is $\sum_{i \in \mathcal{J}} p_i$. The goal of the allocation problem is to determine the feasible subset of jobs that yields the maximum profit.

Our Results

Without the bandwidth constraint, the allocation problem in the grid computing is the integer knapsack problem: the CPU pool is the knapsack, and each job is an item. Integer knapsack is (weakly) NP-complete, but one can solve it optimally in pseudo-polynomial time. (One can reasonably assume that the total number of computing units is polynomially bounded in n.)

We begin our investigation by studying *when* does the network bandwidth even become a bottleneck in grid computing. To this end, let $b_{\max}$ denote the *maximum bandwidth requested* by any job, and let $B_{\min}$ denote the *minimum capacity* of any link in G. Our first result shows that as long as no job requests more than half the minimum link bandwidth, namely, $b_{\max} \leq \frac{1}{2}B_{\min}$, the bandwidth guarantee can be provided essentially for free (Theorem 1). In this case, therefore, an optimal allocation can be computed in (pseudo) polynomial time.

We next show that $\frac{1}{2}B_{\min}$ forms a sharp boundary: if job bandwidths are even slightly larger than $\frac{1}{2}B_{\min}$, then the allocation problem becomes *strongly* NP-complete. Under the reasonable assumption that $b_{\max} \leq B_{\min}$ (i.e. no link is a bottleneck for any *single* job), we present an efficient approximation scheme that guarantees at least one-third of the maximum profit.

The allocation problem turns out to be hard if we allow $b_{\max} > B_{\min}$; that is, the jobs demand bandwidths in excess of some of the link capacities. In this case, we show that even a *path topology* network is intractably hard. We present an $O(\log B)$ approximation scheme for the path topology, where all the bandwidths requested by the jobs lie in the range $[1, B]$. As part of our path topology solution, we also develop a new algorithm for the strongly NP-complete *multiple knapsack* problem, improving the $(2+\varepsilon)$-approximation scheme of Chekuri and Khanna [3] with running time $O(nk \log \frac{1}{\varepsilon} + \frac{n}{\varepsilon^4})$. Instead, we give a simple 2-approximation algorithm with worst-case running time $O((n+k)\log(n+k))$.

2 Allocation in Grid Computing

The underlying resource pool (grid) is modeled as an undirected graph $G = (V, E)$, with n nodes and m edges, where each node $v \in V$ has a computing resource $C(v)$, and each link (u, v) has a bandwidth $B(u, v)$. A job J_i, for $i = 1, 2, \ldots, k$, is specified by a triple $\langle c_i, b_i, p_i \rangle$, where c_i, b_i are the computing and the bandwidth resource needed by J_i, and p_i is the profit. Let $C_i(v_k)$ denote the computing resource that v_k contributes to J_i, and let $B_i(u, v) \in \{0, b_i\}$ denote the bandwidth that (u, v) reserves for J_i. (Note that computing resources are aggregated across multiple nodes, but the bandwidth resource is binary. Unless a link contributes full b_i units of the bandwidth, it cannot be used for communication between nodes allocated to J_i.) See Figure 1 for an example.

If job J_i is accepted, then we must have (i) $\sum_k C_i(v_k) \geq c_i$, namely, c_i total units of the computing resource are allocated to J_i, and (ii) the set of edges $\{(u, v) \mid B_i(u, v) = b_i\}$ spans V_i. That is, the set of nodes that contribute computing resources for J_i must be *connected* by a subset of links with *reserved*

bandwidth b_i. An index set of jobs $\mathcal{J}$ is *feasible* if neither the computing nor the bandwidth resource capacity is violated, that is, $\sum_{i\in\mathcal{J}} C_i(v_k) \le C(v_k)$, for all nodes $v_k \in V$, and $\sum_{i\in\mathcal{J}} B_i(u,v) \le B(u,v)$, for all links $(u,v) \in E$. The total profit for the accepted jobs is $\sum_{i\in\mathcal{J}} p_i$. The goal of the allocation problem is to determine the feasible subset of jobs that yields the maximum profit.

We begin our investigation by asking *when* does the network bandwidth even become a bottleneck. Surprisingly, there turns out to be a rather sharp boundary. Let $b_{\max}$ be the maximum requested bandwidth of any job, and let $B_{\min}$ be the minimum bandwidth of any link in G.

Theorem 1. *Suppose that $b_{\max} \le \frac{1}{2}B_{\min}$ holds. Then, the allocation problem can be solved optimally in time $O(k|C| + n + m)$, where $|C|$ is the total number of computing units available, and n, m are the number of nodes and edges in the network. One can also achieve $(1+\varepsilon)$ approximation of the optimal in time polynomial in $k, 1/\varepsilon$ and linear in n and m.*

Proof. We take all the jobs and solve a 0/1 knapsack problem, where we simply aggregate the computing resources of all the nodes in the graph. Job i has size c_i and value p_i; the knapsack capacity is $|C|$. Let W be the set of winning jobs (solution of the knapsack), and let $p(W)$ be their total profit. Clearly, the optimal solution of the resource allocation problem cannot have profit larger than $p(W)$. In the following, we show how to allocate all the jobs of W in G.

Construct any spanning tree T of G. Each link of this tree has capacity at least $B_{\min}$. We root this tree arbitrarily at a node r, and perform a pre-order walk of T. We allocate jobs of W to the nodes encountered in the pre-order; when a node's capacity is depleted, we move to the next node. It is easy to see that no link of the tree is shared by more than 2 jobs, and all the jobs are allocated.

The running time is dominated by the knapsack problem, which takes $O(k|C|)$ time using dynamic programming. If $(1+\varepsilon)$ approximation is needed, we can use a fully polynomial approximation scheme, whose running time is polynomial in k and $1/\varepsilon$; the $O(n+m)$ time is for constructing a spanning tree and traversing it. This completes the proof.

Surprisingly, letting the job bandwidth exceed $\frac{1}{2}B_{\min}$ even slightly makes the problem strongly intractable.

Theorem 2. *The optimal allocation problem is* strongly *NP-Complete even if the job bandwidths satisfy the condition $\frac{1}{2}B_{\min} + \varepsilon \le b_{\max} \le B_{\min}$.*

Proof. We reduce the well-known 3-partition problem [7], which is strongly NP-Complete, to our allocation problem. The 3-partition problem is the following:

Instance: Integers m, d and x_i, for $i = 1, 2, \cdots, 3m$ satisfying $\sum_i x_i = md$ and $\frac{d}{4} < x_i < \frac{d}{2} \quad \forall i$.

Question: Is there a partition of x's into m disjoint (3-element) subsets A_1, A_2, $\cdots$, A_m such that $\sum_{i\in A_j} x_i = d$, for $j = 1, 2, \cdots, m$.

Given an an instance of the 3-partition problem, we construct a tree (of height one) with $3m + 1$ nodes $u_0, v_1, \cdots, v_{3m}$. The node u_0 is root and the other $3m$ nodes are its children. The node v_i has x_i units of the resource; the root node has zero unit of the resource. Each link has a bandwidth B. We create m identical jobs $\langle d, \frac{1}{2}B + \varepsilon,\ p\rangle$. One can show that all m jobs can be allocated exactly when the input 3-partition instance has a feasible solution.

In the next section, we present a constant factor approximation scheme when $b_{\max} \leq B_{\min}$. That is, no network link is a bottleneck for any *single* job. In the subsequent section, we address the general grid model without any constraint on the network link bandwidth.

3 An Approximation Scheme when $b_{\max} \leq B_{\min}$

We construct a spanning tree, T, of the input network G, rooted at an arbitrary node r. Since each link of G has bandwidth at least $B_{\min}$, all edges of T have bandwidth at least $B_{\min} \geq b_{\max}$. For a node v, we let T_v denote the subtree rooted at v. Let $C(T_v)$ denote the total (remaining) resource units at the nodes in T_v. That is, $C(T_v) = \sum_{u \in T_v} C(u)$. Similarly, for a subset of nodes $S \subseteq V$, let $C(S)$ denote the total resource units available at the nodes of S. The input set of jobs is $J_1, J_2, \ldots, J_k$. We assume that $c_i \leq \sum_{v \in V} C(v)$; otherwise job J_i clearly cannot be satisfied. Our algorithm can be described as follows.

Algorithm Approx

1. Sort the input jobs in descending order of p_i/c_i (profit per compute cycle). Process jobs in the sorted order. Let $J_a = \langle c_a, b_a, p_a\rangle$ be the next job.
2. If $c_a \leq C(T_r)$, do Step 3; else do Step 4. (Recall that r is the root of the spanning tree T.)
3. Find the *best fit* node v in the current tree; that is, among all nodes u for which $C(T_u) - c_a \geq 0$, v minimizes $C(T_u) - c_a$.
   - Among the children nodes of v, choose a set S such that $c_a \leq C(S) \leq 2c_a$. Allocate the set S (and their descendants) to job J_a, and delete these nodes from the tree.
   - If no such S exists, we allocate all the children of v plus the appropriate fraction of v's resources to the job J_a; in this case, we delete all the children of v from T, and update the remaining resource units $C(v)$ for the node v.
   - Add J_a to the set Z, which contains all the accepted jobs.
4. Let $p(Z)$ be the total profit of all the jobs accepted in Z. If $p(Z) \geq p_a$, we output Z, otherwise, we output the single job $\{J_a\}$.

end Algorithm

Theorem 3. *The algorithm* Approx *computes a feasible set of jobs whose profit is at least* 1/3 *of the optimal. The algorithm can be implemented in worst-case time* $O(m + k \log k + n(k + \log n))$.

Proof. Suppose J_a is the first job that is rejected by the algorithm. Let Z be the current set of accepted jobs when J_a is encountered. Let C_Z be the total number of resource units demanded by jobs in Z; that is, $C_Z = \sum_{i \in Z} c_i$. By the best fit rule, whenever we accept a job in Z, it wastes at most an equal amount of resource. Since J_a could not be allocated, we have the following inequality:

$$2C_Z + c_a > C(T), \tag{1}$$

where $C(T)$ is the total number of resource units initially available in the system. Let d_Z denote the *average unit price* for the jobs in Z. That is,

$$d_Z = \frac{\sum_{i \in Z} p_i}{\sum_{i \in Z} c_i}$$

Let d be the average unit price of $Z \cup J_a$, and let d^* be the average unit price of the jobs in an optimal solution. Since our algorithm considers jobs in the decreasing unit price order, we have $d_Z \geq d \geq d*$. Thus,

$$2p(Z) + p_a = d_Z C_Z + d(C_Z + c_a) \geq d^* C(T) \geq OPT$$

Since our algorithm chooses $\max\{p(Z), p_a\}$, it follows that $3\max\{p(Z), p_a\} \geq OPT$. The bound on the worst-case running time follows easily from the description of the algorithm.

The analysis of Approx is tight. The following is an example where the algorithm's output approaches one third of the optimal. Consider the tree network shown in Figure 2. Assume there are 4 jobs. Jobs J_1 and J_2 are $\langle M+\varepsilon, 1, M+2\varepsilon\rangle$, while jobs J_3 and J_4 are $\langle 2M-3, 1, 2M-3\rangle$. The bandwidth of each link in the tree is also 1. All four jobs can be feasibly allocated, by assigning nodes u, x to J_1, nodes v, y to J_2, node w and half of r to J_3, and node z and half of r to J_4. The total profit is $6M - 6 + 4\varepsilon$.

We now consider the performance of Approx. The algorithm will process jobs in the order $\{J_1, J_2\}, \{J_3, J_4\}$. The algorithm will allocate J_1 to nodes w and x and J_2 to nodes y and z, and will fail to schedule the other jobs. The total profit is $2M + 4\varepsilon$, which approaches $1/3$ of the optimal as M grows.

A natural question is whether the resource allocation problem becomes easier for tree topologies, within the cluster computing model. Unfortunately, that is not the case, as the reduction of Theorem 2 already establishes the hardness for the trees. If the topology is further restricted to a *path*, however, the problem can be solved optimally in (pseudo) polynomial time.

Theorem 4. *If the network topology is a path and the input satisfies* $b_{\max} \leq B_{\min}$*, then the allocation problem can be solved optimally in (pseudo) polynomial time.*

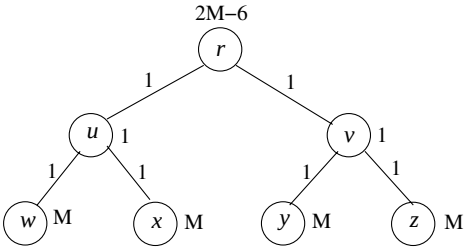


Fig. 2. Tightness of Approx. Nodes u, v have 1 unit of resource; nodes w, x, y, z have M units, and the root has $2M - 6$ units. All links have capacity 1.

4 The Global Grid Model

In the previous section, we assumed that the minimum network link bandwidth is at least as large as the maximum job bandwidth; that is, $b_{\max} \leq B_{\min}$. This is a realistic model for the grid computing at an enterprise level, where a collection of workstations are joined by high bandwidth links. However, when one envisions a larger, Internet scale grid, then this assumption no longer seems justified. In this section, we consider the allocation for this "global grid" model.

Suppose that the link bandwidths are in some arbitrary range $[B_{\min}, B_{\max}]$, and the jobs can request an arbitrary bandwidth (even $b > B_{\max}$); if a job requests bandwidth greater than $B_{\max}$, then it must be allocated to a single node. We call this the *global grid* model for ease of reference. The allocation problem in the global grid appears to be significantly harder than in the previous model. The latter is clearly a special case of the former, and so the intractability theorems of the preceding sections all apply to the global grid as well. In the global grid, however, *even the path topology* is intractable. We use a reduction from the multiple knapsack problem [3], which unlike the single knapsack problem is strongly NP-Complete.

Lemma 1. *The optimal allocation problem in the global grid model is* strongly *NP-complete even if the network topology is a* path.

The special case of the problem when the network consists of isolated nodes is equivalent to the multiple knapsack problem. We start our discussion with an approximation algorithm for this case.

4.1 Isolated Nodes: 2-Approximation of Multiple Knapsack

Suppose all jobs request bandwidth greater than the maximum link capacity in the network (or, equivalently, if all links have zero bandwidth), then the network reduces to a set of isolated nodes. Our problem is equivalent to the well-known Multiple Knapsack problem. Chekuri and Khanna [3] have given an $O(n^{O(\log(1/\varepsilon)/\varepsilon^8)})$ time approximation scheme for the multiple knapsack problem.

They also gave a $(2+\varepsilon)$-approximation scheme with running time $O(nk \log \frac{1}{\varepsilon} + \frac{n}{\varepsilon^4})$. In the following, we show that a simple greedy algorithm achieves a factor 2 approximation in time $O((n+k)\log(n+k))$.

Let $S = \{a_1, a_2, \ldots, a_k\}$ be the set of items, where item a_i has size $s(a_i)$ and profit $p(a_i)$. Given a subset $A \subseteq S$, let $s(A)$ and $p(A)$ denote the total size and total profit of the set of items in A. Let $K = \{1, 2, \ldots, n\}$ be the set of knapsacks, where the jth knapsack has capacity c_j. We assume that knapsacks are given in *non-decreasing* order of capacity; that is, $c_1 \leq c_2 \leq \cdots \leq c_n$. The items are given in *non-increasing* order of unit price; that is, $p(a_i)/s(a_i) \geq p(a_{i+1})/s(a_{i+1})$.

Algorithm MKP-Approx

1. Let L be the list of the remaining items, initialized to S.
2. Initialize greedy solution $G = \emptyset$.
3. Consider the knapsacks in sorted order. Let knapsack j be the next one.
   a) Let $L_j \subseteq L$ be the subset of items such that $s(x) \leq c_j$, for $x \in L_j$.
   b) Greedily (descending unit price) add items of L_j to the knapsack j. Let f_j be the first item to exceed the remaining capacity of knapsack j.
   c) Let $A_j \subseteq L_j$ be the set of items that have been added to the knapsack when f_j is encountered.
   d) If $p(A_j) \geq p(f_j)$, add A_j to greedy solution G; otherwise add f_j to G.
   e) Remove A_j and f_j from L.
4. Return G.

Due to limited space, we omit the proof of the following theorem. The proof can be found in the extended version of the paper [12].

Theorem 5. *The algorithm* MKP-Approx *achieves a 2-approximation of the Multiple Knapsack Problem in time $O((n+k)\log(n+k))$, where n and k are the number of knapsacks and items.*

4.2 An Approximation Scheme for Path Topology

Our main result for the global grid is an $O(\log B)$ factor approximation scheme, where all jobs have bandwidths in the range $[1, B]$. We begin with some simple observations.

Let $v_1, v_2, \ldots, v_n$ denote the nodes of the path, in the left to right order. Suppose in some allocation v_i (resp. v_j) is the leftmost (resp. rightmost) node contributing the computing resources to a job J. Then, we call $[i, j]$ the *span* of J. We say that two spans $[i, j]$ and $[i', j']$ are *partially overlapping* if they overlap but neither contains the other. In other words, $[i, j]$ and $[i', j']$ partially overlap if $i < i' < j < j'$ or $i' < i < j' < j$. We say that job $J_1 = \langle c_1, b_1, p_1 \rangle$ is *nested* inside job $J_2 = \langle c_2, b_2, p_2 \rangle$ if the span of J_1 is contained inside the span of J_2. The following two elementary lemmas will be useful in our approximation.

Lemma 2. *There always exists a maximum profit allocation in which no two jobs have partially overlapping spans.*

Lemma 3. *If job* $J_1 = \langle c_1, b_1, p_1 \rangle$ *is* nested *inside job* $J_2 = \langle c_2, b_2, p_2 \rangle$*, then* $b_1 > b_2$*, and there is some link contained in the span of* J_2 *whose bandwidth is strictly smaller than* b_1*.*

We can draw two simple conclusions from the preceding lemmas: (1) if all the jobs require the same bandwidth, then there is an optimal non-nesting solution; and (2) if the maximal bandwidth required by any job is no more than the minimum bandwidth of any link, then again there is an optimal non-nesting solution. In the more general setting, we have the following:

Lemma 4. *If each link in the path network has bandwidth capacity either* 0 *or* B*, then we can get a* $(2+\varepsilon)$*-approximation in polynomial time.*

Proof. We partition the input jobs into two classes: *big* jobs, which need bandwidth more than B, and *small* jobs, which need bandwidth at most B. Clearly, the big jobs cannot be served by multiple nodes, while the small jobs can be served by multiple nodes if they are connected with bandwidth B links. Our approximation algorithm works in the following way.

First we consider big jobs and solve it by using the multiple knapsack problem (MKP) with approximation ratio $(1+\varepsilon/2)$ [3]. We then consider small jobs. The network links with bandwidth 0 partition the path into multiple subpaths, where each subpath is joined by links of capacity B. A small job can only be satisfied by nodes within one subpath. We now consider each subpath as a *bin* with its capacity equal to the sum of capacities for all the nodes contained in it. We apply another $(1+\varepsilon/2)$-approximation MKP algorithm to this problem and get another candidate solution. Of the two solutions, we pick the one with the larger profit. The following argument shows that this algorithm achieves approximation ratio $(2+\varepsilon)$.

Consider an optimal solution; it consists of some small jobs and some big jobs. Let Π_s and Π_b, respectively, denote the total profit of the optimal solution contributed by small and big jobs. Thus $OPT = \Pi_s + \Pi_b \leq 2\max\{\Pi_s, \Pi_b\}$. If A denotes the total profit for our algorithm, then $\Pi_s \leq (1+\varepsilon/2)A$. Similarly, by considering the large jobs, we get $\Pi_b \leq (1+\varepsilon/2)A$. By combining these inequalities together, we get $OPT \leq (2+\varepsilon)A$. This completes the proof.

In order to prove our main result for the path topology in the grid model, we first partition the set of jobs into $\log B$ classes such that each job has roughly the same amount of bandwidth requirement. Let us suppose that all the jobs in the set have their bandwidth requirement between b and $2b$.

Lemma 5. *Suppose that all the jobs have bandwidth requirement in the range* $[b, 2b]$*. The maximum profit realizable by the best nesting solution is at most twice the maximum profit realizable by a non-nesting solution. Thus, limiting our search to the non-nesting solutions costs at most a factor of two in the approximation.*

Proof. Consider an optimal solution for the problem, where jobs may nest arbitrarily with each other. Consider the *containment partial order* among these jobs: $J < J'$ if the span of J is contained in the span of J'; in case of ties, the lower indexed job comes earlier in the partial order. Let s_0 be the set of *maximal* elements in this partial order—these are the jobs whose spans are not contained in any other job's span. Let s_1 denote the set of remaining jobs. Let Π_0 denote the total profit of s_0 in the optimal solution, and let Π_1 be the profit of the s_1 jobs. We argue below that either all jobs in s_0 or all jobs in s_1 can be allocated with non-nesting spans.

The spans of all the jobs in s_0 are clearly non-nesting (by definition). Next, observe that any link that lies in the span of a job in s_1 must have bandwidth at least $2b$, since this link is shared by at least two jobs, and every job has bandwidth at least b. Since the bandwidth needed by any job is at most $2b$, using arguments like the one in Lemma 2, we can re-allocate resources among the jobs of s_1 so that no two jobs nest. Thus, there exist an alternative non-nesting solution with profit at least $\max\{J_0, J_1\}$, which gives at least $1/2$ the profit of the optimal solution.

Lemma 6. *Given a set of jobs $J_1, J_2, \ldots, J_k$, and a path network $(v_1, \ldots, v_n)$, in polynomial time, we can compute a* 2*-approximation of the* best non-nesting *solution of the resource allocation problem.*

Proof. We use a single-processor job scheduling algorithm of Bar-Noy et al. [1]. The input to the job scheduling problem is a set of tuples (r_i, d_i, ℓ_i, w_i), where r_i is the release time, d_i is the deadline, ℓ_i is the length, and w_i is the weight (profit) of the job i. The job i can only be scheduled to start between r_i and $d_i - \ell_i$. The goal is to determine a maximum weight schedule. Bar-Noy [1] give a polynomial time 2-approximation scheme for polynomially bounded integral input.[1]

In order to formulate our allocation problem as job scheduling, we need a slightly stronger model: each job has multiple, non-overlapping (release time, deadline) intervals; it can be scheduled during any of them (but at most once). It turns out that the scheme of Bar-Noy et al. [1] extends to this more general setting and yields the same approximation result [12]. We now describe the scheduling formulating of job allocation problem.

A job i has length equal to its resource demand c_i, and has weight equal to the profit p_i. The time in the scheduling problem corresponds to the resource units in our path network. (Recall our assumption that these units are polynomially bounded.) If we delete from the path network all links of bandwidth strictly less than b_i, the network is decomposed into disjoint subpaths. These subpaths correspond to the non-overlapping periods of release time and deadline for the job i. Due to space limitation, we omit the remaining details, which can be found in the extended version of the paper [12].

We can summarize the main result of this section in the following theorem.

[1] Without the assumption of polynomial bound on the number of resource units, a scheme with 6-approximation can be obtained [1].

Theorem 6. *Consider the resource allocation problem in the grid model for a n-node path topology. Suppose there are k jobs, each requiring bandwidth in the range* $[1, B]$*. Then, there is a polynomial time* $O(\log B)$*-approximation algorithm.*

Proof. We first partition all the requests into $\log B$ classes such that all jobs in one class have bandwidth requirement within a factor of two. When all bandwidth requests are in the range $[b, 2b]$ for some b, by Lemma 5, we can consider only non-nesting solutions at the expense of factor two in the approximation quality. For each of these $\log B$ classes of jobs, we run the approximation algorithm described in Lemma 6, which yields a factor 2-approximation of the best non-nesting solution. By choosing the best solution from the $\log B$ classes, we guarantee an approximation ratio of $O(\log B)$.

5 Related Work

Several grid systems have been developed, such as Globus [6], Legion [2], Condor [8] and SETI@Home [11], yet many interesting resource allocation problems in these systems remain to be addressed. Resource allocation schemes for grid computing include the market-based resource *sharing* as proposed by Chun and Culler [4], where *all* the jobs receive some resource, only the amount differs based on the offered price; the *SPAWN* model of Waldspurger et al. [9] essentially run parallel auctions for the different resources; the artificial economy model of Wolski et al. [10] uses supply and demand to set the prices. None of these models have any theoretical performance guarantees, or handle resource allocation with explicit bandwidth constraints.

Our resource allocation problem superficially resembles the multiple knapsack problem, but it differs considerably from the latter because in our problem jobs can be allocated across several different nodes if the bandwidth constraint is satisfied. Indeed, the multiple knapsack problem is a rather special case of the resource allocation problem (i.e. disjoint nodes topology).

For the special case of path topology, the resource allocation problem is similar to *Job Interval scheduling problem (JISP)*, where the input for each job is its length and a set of intervals, in which it can be scheduled. The objective is to maximize the number of scheduled jobs. JISP is strongly NP-Complete [7] and Chuzhoy et al. [5] gave a 1.582 approximation algorithm for it. Our model differs from JISP because there is no notion of profit associated with jobs in JISP. A more general version of JISP called *real time scheduling (RTP)* associates a weight with each job, and the objective is to maximize the total weight. BarNoy et al. [1] gave a 2-approximation algorithm for the the case of single machine. In section 4.2, we reduced the allocation problem for the path topology to RTP. This reduction however only works when there exists an optimal solution in which no link is used by more than one job, as RTP does not allow preemption. The scheduling techniques used in RTP can be applied to only path topologies as it is not at all clear how to reduce more general topologies to RTP.

6 Concluding Remarks

We studied an allocation problem motivated by grid computing and peer-to-peer systems. These systems pool together the resources of many workstations to create a virtual computing reservoir. Users can "draw" resources using a pay-as-you-go model, commonly used for utilities (electricity and water). As these technologies mature, and more advanced applications are implemented using computational grids, we expect providing bandwidth guarantees for the applications will become important. With that motivation, we studied the bandwidth-constrained allocation problems in grid computing.

Several open problems are suggested by this work. Is it possible to obtain a polynomial time $(1+\varepsilon)$-approximation scheme when $b_{\max} \leq B_{\min}$? If not, what is the best approximation factor one can achieve in polynomial time? In the global grid model, can one achieve a constant factor approximation independent of B? Extend our results to more general topologies than the path in the global grid model? Develop competitive algorithms for the online versions of the allocation problems.

References

1. A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. In *Proc. of the 31st ACM Symp. on Theory of Computing*, pages 622–631, 1999.
2. S. Chapin, J. Karpovich, and A. Grimshaw. The legion resource management system. In *Workshop on Job Scheduling Strategies for Parallel Processing.*, 1999.
3. C. Chekuri and S. Khanna. A ptas for the multiple knapsack problem. In *Proc. of 11th Annual Symposium on Discrete Algorithms.*, pages 213–222, 2000.
4. B. Chun and D. E. Culler. Market-based proportional resource sharing clusters. Technical report, UC Berkeley, Computer Science, 2000.
5. J. Chuzhoy, R. Ostrovsky and Y. Rabani. Approximation algorithms for the job interval scheduling problem and realted scheduling problems. In *Proc. of 42nd Annual Symposium on Foundation of Computer Science.*, pages 348–356, 2001.
6. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *Workshop on Job Scheduling Strategies for Parallel Processing.*, 1998.
7. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, 1979.
8. M. Litzkow, M. Ivny, and M. Mutka. Condor—a hunter of idle workstations. In *Proc. of 8th International Conference on Distributed Computing.*, 1988
9. C. Waldspurger, T. Hogg, B. Huberman, J. Kephart, and W. Stornetta. Spawn—a distributed computational economy. *IEEE Trans. on Software Engineering.*, 1992.
10. R. Wolski, J. Plank, J. Brevik, and T. Bryan. Analyzing market-based resource allocation strategies for the computational grid. *Int. Journal of High Performance Computing Applications*, 2001.
11. http://setiathome.ssl.berkeley.edu. Seti@home.
12. http://www.cs.ucsb.edu/∼suri/pubs.html Extended Version.

Algorithms and Approximation Schemes for Minimum Lateness/Tardiness Scheduling with Rejection

Sudipta Sengupta

Bell Laboratories, Lucent Technologies
101 Crawfords Corner Road
Holmdel, NJ 07733, USA

Abstract. We consider the problem of minimum lateness/tardiness scheduling with rejection for which the objective function is the sum of the maximum lateness/tardiness of the scheduled jobs and the total rejection penalty (sum of rejection costs) of the rejected jobs. If rejection is not considered, the problems are solvable in polynomial time using the *Earliest Due Date (EDD)* rule. We show that adding the option of rejection makes the problems $\mathcal{NP}$-complete. We give pseudo-polynomial time algorithms, based on dynamic programming, for these problems. We also develop a fully polynomial time approximation scheme (FPTAS) for minimum tardiness scheduling with rejection using a geometric rounding technique on the total rejection penalty.
Observe that the usual notion of an approximation algorithm (guaranteed factor bound relative to optimal objective function value) is inappropriate when the optimal objective function value could be negative, as is the case with minimum lateness scheduling with rejection. An alternative notion of approximation, called *ϵ-optimization approximation* [7], is suitable for designing approximation algorithms for such problems. We give a polynomial time ϵ-optimization approximation scheme (PTEOS) for minimum lateness scheduling with rejection and a fully polynomial time ϵ-optimization approximation scheme (FPTEOS) for a modified problem where the total rejection penalty is the *product* (and not the sum) of the rejection costs of the rejected jobs.

1 Introduction

Most of traditional scheduling theory [1,2] begins with a set of jobs to be scheduled in a particular machine environment so as to optimize a particular optimality criterion. At times, however, a higher-level decision has to be made: given a set of tasks, and limited available capacity, choose only a subset of these tasks to be scheduled, while perhaps incurring some penalty for the jobs that are not scheduled, i.e., "rejected". We focus on techniques for scheduling a set of independent jobs with the flexibility of rejecting a subset of the jobs in order to guarantee an average good quality of service for the scheduled jobs.

The idea of scheduling with rejection is relatively new and there has been little prior research in the area. Multiprocessor scheduling with the objective

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 79–90, 2003.
© Springer-Verlag Berlin Heidelberg 2003

of trading off between schedule *makespan* and job rejection penalty is studied in [3,4,5]. The problem of minimizing sum of weighted completion times with rejection has been considered in [6]. Along with makespan and sum of weighted completion times, maximum lateness/tardiness constitutes the most basic and well-studied of all scheduling optimality criteria; therefore, it is of interest to understand the impact of the "rejection option" on scheduling to minimize maximum lateness/tardiness.

1.1 Our Work

In this paper, we consider the problem of minimum lateness/tardiness scheduling with rejection for which the (minimization) objective function is the sum of the maximum lateness/tardiness of the scheduled jobs and the total rejection penalty (sum of rejection costs) of the rejected jobs. We use the scheduling notation introduced in [1], and denote the rejection cost of each job j by e_j. For a given job j with due date d_j that completes at time C_j, the lateness of the job is defined as $L_j = C_j - d_j$ and its tardiness is defined as $T_j = \max(C_j - d_j, 0) = \max(L_j, 0)$. Thus, the one machine versions of these two problems are denoted as $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$ respectively, where S is the set of scheduled jobs. If rejection is not considered, the problems are solvable in polynomial time using the *Earliest Due Date (EDD)* rule: schedule the jobs in non-decreasing order of due dates d_j. In Section 2, we show that adding the option of rejection makes the problems $\mathcal{NP}$-complete. In Section 3, we give two pseudo-polynomial time algorithms, based on dynamic programming, for these problems. We also develop a fully polynomial time approximation scheme (FPTAS) for $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$ in Section 4. The FPTAS uses a geometric rounding technique on the total rejection penalty and works with what we call the *inflated rejection penalty*.

Observe that the usual notion of an approximation algorithm (guaranteed factor bound relative to optimal objective function value) is inappropriate for $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ because the optimal objective function value could be negative. An alternative notion of approximation, called ϵ-*optimization approximation* [7], is suitable for designing approximation algorithms for such problems. In Section 5, we discuss this notion of approximation. We give a polynomial time ϵ-optimization approximation scheme (PTEOS) for $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ in Section 5.1 and a fully polynomial time ϵ-optimization approximation scheme (FPTEOS) for the problem $1|\ |(L_{max}(S) + \prod_{\bar{S}} e_j)$ in Section 5.2. The total rejection penalty for the latter problem is the *product* (and not the sum) of the rejection costs of the rejected jobs.

2 Hardness Results

In this section, we show that the problems $Pm|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $Pm|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$ are $\mathcal{NP}$-complete for any $m \geq 1$. Both the problems are solvable on one machine in polynomial time using the Earliest Due Date First

(EDD) rule *when rejection is not considered*. We show that adding the rejection option to even the *single* machine problem in each case makes it $\mathcal{NP}$-complete.

The decision problem formulation of $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ is defined as follows:

> Given a set of n independent jobs, $N = \{J_1, \ldots, J_n\}$, with processing times p_j, due dates d_j, and rejection costs e_j, $\forall\ 1 \leq j \leq n$, a single machine, and a number K, is there a schedule of a subset of jobs $S \subseteq N$ on the machine such that $L_{max}(S) + \sum_{j \in \bar{S} = N - S} e_j \leq K$?

We reduce the Partition Problem [8] to this problem, thus proving that even on one machine, maximum lateness with rejection is $\mathcal{NP}$-complete.

Theorem 1. $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ *is* $\mathcal{NP}$*-complete.*

Proof. $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ is clearly in $\mathcal{NP}$. To prove that it is also $\mathcal{NP}$-hard, we reduce the Partition Problem [8] to $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$. The Partition Problem is defined as follows:

> Given a set $A = \{a_1, a_2, \ldots, a_n\}$ of n numbers such that $\sum_{i=1}^{n} a_i = 2b$, is there a subset A' of A such that $\sum_{a_i \in A'} a_i = b$?

Given an instance $A = \{a_1, \ldots, a_n\}$ of the partition problem, we create an instance of $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ with $n + 1$ jobs, $J_0, , J_1, \ldots, J_n$. For $i = 1, 2, \ldots, n$, each of the n elements a_i in the Partition Problem corresponds to a job J_i in $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ with processing time $p_i = a_i$, due date $d_i = b$, and rejection cost $e_i = a_i/2$, where $b = \frac{1}{2}\sum_{i=1}^{n} a_i$. The special job J_0 has processing time equal to b, due date equal to 0, and rejection cost equal to ∞.

Consider any optimal schedule for $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$. Since J_0 has rejection cost of ∞ and the smallest due date, it must be scheduled first. Let S and $\bar{S}$ be the set of indices of the scheduled and rejected jobs respectively among $J_1, J_2, \ldots, J_n$ and let $x = \sum_{i \in S} p_i = \sum_{i \in S} a_i$. Observe that the makespan of the set of jobs in S is $x + b$, and since every job in S has the same due date b, the maximum lateness of this set of jobs is x. Also, the total rejection penalty of the rejected jobs is $\sum_{i \in \bar{S}} e_i = \sum_{i \in \bar{S}} a_i/2 = (2b - x)/2 = b - x/2$. Then, the value of the objective function for this schedule is $\max(x, b) + (b - x/2)$.

This function has a unique minimum of $\frac{3}{2}b$ at $x = b$. Hence, the best possible solution has $\sum_{i \in S} p_i = b$, and is optimum if it exists. Therefore, if the optimum solution to $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ is equal to $\frac{3}{2}b$, then there exists a subset $A' = S$ of A such that $\sum_{i \in A'} a_i = b$, i.e., the answer to the Partition Problem is 'Yes', and S is a witness. If the optimum solution to $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ is greater than $\frac{3}{2}b$, then there does not exist any partition A' of A such that $\sum_{i \in A'} a_i = b$, i.e., the answer to the Partition Problem is 'No'. Conversely, if the answer to the Partition Problem is 'Yes', the optimum solution to $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ is clearly equal to $\frac{3}{2}b$. If the answer to the Partition Problem is 'No', the optimum solution to $1|\ |(\sum_{S} w_j C_j + \sum_{\bar{S}} e_j)$ is clearly greater than $\frac{3}{2}b$.

The above proof also works for $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$ since every job in our reduction has a non-negative lateness that is equal to its tardiness.

Theorem 2. $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$ *is* $\mathcal{NP}$*-complete.*

As a corollary, it follows that the multiprocessor version of these problems, $Pm|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $Pm|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$, are both $\mathcal{NP}$-complete for any $m \geq 1$.

3 Pseudo-Polynomial Time Algorithms

In this section, we give pseudo-polynomial time algorithms for solving $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$ exactly. We first give an $O(n \sum_{j=1}^{n} e_j)$ time algorithm (in section 3.1) and then an $O(n \sum_{j=1}^{n} p_j)$ time algorithm (in Section 3.2), using dynamic programming, to solve $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$. We also generalize our second dynamic program to a fixed number of unrelated parallel machines. In Section 4, we show how to modify the dynamic program of Section 3.1 to obtain an FPTAS for $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$.

3.1 Dynamic Programming on the Rejection Costs

To solve our problem, we set up a dynamic program for the following problem: to find the schedule that minimizes the maximum lateness when the total rejection penalty of the rejected jobs is given. We number the jobs in non-decreasing order of due dates d_j. This is because the Earliest Due Date (EDD) rule minimizes the maximum lateness for any given set of scheduled jobs. Let $\phi_{e,j}$ denote the minimum value of the maximum lateness when the jobs in consideration are $j, j+1, \ldots, n$, and the total rejection penalty of the rejected jobs is e. Note that the boundary conditions for the dynamic program are:

$$\phi_{e,n} = \begin{cases} -\infty & \text{if } e = e_n \\ p_n - d_n & \text{if } e = 0 \\ \infty & \text{otherwise} \end{cases}$$

Now, consider any schedule for the jobs $j, j+1, \ldots, n$ that minimizes the maximum lateness when the total rejection penalty of the rejected jobs is e. We will refer to this as the optimal schedule in the discussion below. In any such schedule, there are two possible cases — either job j is rejected or job j is scheduled.

Case 1: Job j is rejected. This is possible only if $e \geq e_j$. Otherwise, there is no feasible solution with rejection penalty e and job j rejected, in which case only Case 2 applies. Hence, assume that $e \geq e_j$. Then, the value of the maximum lateness for the optimal schedule is clearly $\phi_{e-e_j,j+1}$, since the total rejection penalty of the rejected jobs among $j+1, \ldots, n$ must be $e - e_j$.

Case 2: Job j is scheduled. In this case, the total rejection penalty of the rejected jobs among $j+1, \ldots, n$ must be e. Also, when job j is scheduled before all jobs in the optimal schedule for jobs $j+1, j+2, \ldots, n$, the lateness of every scheduled job among $j+1, j+2, \ldots, n$ is increased by p_j and the lateness of job j is exactly $p_j - d_j$. Then, the value of the maximum lateness for the optimal schedule is clearly $\max(\phi_{e,j+1} + p_j, p_j - d_j)$.

Combining the above two cases, we have:

$$\phi_{e,j} = \begin{cases} \max(\phi_{e,j+1} + p_j, p_j - d_j) & \text{if } e < e_j \\ \min[\phi_{e-e_j,j+1}, \max(\phi_{e,j+1} + p_j, p_j - d_j)] & \text{otherwise} \end{cases}$$

Now, observe that the total rejection penalty of the rejected jobs can be at most $\sum_{j=1}^{n} e_j$, and the answers to our original problems are

- $\min\{(\phi_{e,1} + e) \mid 0 \leq e \leq \sum_{j=1}^{n} e_j\}$ for $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$, and
- $\min\{(\max(0, \phi_{e,1}) + e) \mid 0 \leq e \leq \sum_{j=1}^{n} e_j\}$ for $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$.

Thus, we need to compute at most $n \sum_{j=1}^{n} e_j$ table entries $\phi_{e,j}$. Computation of each such entry takes $O(1)$ time, so that the running time of the algorithm is $O(n \sum_{j=1}^{n} e_j)$.

Theorem 3. *Dynamic programming yields an $O(n \sum_{j=1}^{n} e_j)$ time algorithm for exactly solving $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$.*

3.2 Dynamic Programming on the Lateness of the Jobs

In this section, we give another dynamic program that solves $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$ in $O(n \sum_{j=1}^{n} p_j)$ time.

We set up a dynamic program to find the schedule that minimizes the total rejection penalty of the rejected jobs when an upper bound on the maximum lateness of the scheduled jobs is given. Let $\phi_{\ell,j}$ denote the minimum value of the total rejection penalty of the rejected jobs when the jobs in consideration are $j, j+1, \ldots, n$, and the maximum lateness of the scheduled jobs is at most ℓ. The boundary conditions of this dynamic program are given by

$$\phi_{\ell,n} = \begin{cases} e_n \text{ if } \ell = -\infty \\ 0 \quad \text{if } \ell \geq p_n - d_n \\ \infty \text{ otherwise} \end{cases}$$

Now, consider any schedule for the jobs $j, j+1, \ldots, n$ that minimizes the total rejection penalty of the rejected jobs when the maximum lateness of the scheduled jobs is at most ℓ. We will refer to this as the optimal schedule in the discussion below. In any such schedule, there are two possible cases — either job j is rejected or job j is scheduled.

Case 1: Job j is rejected. Then, the value of the total rejection penalty of the rejected jobs for the optimal schedule is clearly $\phi_{\ell,j+1} + e_j$, since the maximum lateness of the scheduled jobs among $j+1, \ldots, n$ is at most ℓ.

Case 2: Job j is scheduled. In this case, the lateness of job j is $p_j - d_j$. Hence, if the value of ℓ is smaller than $p_j - d_j$, there is no feasible solution with maximum lateness ℓ and job j scheduled, in which case only Case 1 applies. Therefore, assume that $\ell \geq p_j - d_j$. Now, when job j is scheduled before all jobs in the schedule for jobs $j+1, j+2, \ldots, n$, the lateness of every scheduled job among $j+1, j+2, \ldots, n$ is increased by p_j. Thus, the maximum lateness

of the scheduled jobs among $j+1, \ldots, n$ can be at most $\ell - p_j$. Then, the value of the total rejection penalty of the rejected jobs for the optimal schedule is $\phi_{\ell-p_j,j+1}$.

Combining the above two cases, we have:

$$\phi_{\ell,j} = \begin{cases} \phi_{\ell,j+1} + e_j & \text{if } \ell < p_j - d_j \\ \min(\phi_{\ell,j+1} + e_j, \phi_{\ell-p_j,j+1}) & \text{otherwise} \end{cases}$$

Let ℓ_{min} and ℓ_{max} denote lower and upper bounds respectively on the maximum lateness of any schedule. It can be shown that $\ell_{max} \leq \sum_{j=1}^{n} p_j$, and $\ell_{min} \geq -\sum_{j=1}^{n} p_j$ (the latter assumes, without any loss of generality, that the maximum due date is at most $\sum_{j=1}^{n} p_j$). Thus, the possible number of *finite* values of the maximum lateness ℓ for any schedule is at most $\ell_{max} - \ell_{min} \leq 2\sum_{j=1}^{n} p_j$. Note that in addition to this, the value of ℓ can also be $-\infty$ (for the empty schedule).

We can now see that the answers to our original problems are

- $\min\{(\ell + \phi_{\ell,1}) \mid \ell_{min} \leq \ell \leq \ell_{max} \text{ or } \ell = -\infty\}$ for $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$, and
- $\min\{(\max(0, \ell) + \phi_{\ell,1}) \mid \ell_{min} \leq \ell \leq \ell_{max} \text{ or } \ell = -\infty\}$ for $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$.

Thus, we need to compute at most $n(2\sum_{j=1}^{n} p_j)$ table entries $\phi_{\ell,j}$. Computation of each such entry takes $O(1)$ time, so that the running time of the algorithm is $O(n\sum_{j=1}^{n} p_j)$.

Theorem 4. *Dynamic programming yields an $O(n\sum_{j=1}^{n} p_j)$ time algorithm for exactly solving $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$.*

The above dynamic program can be generalized to any fixed number m of unrelated parallel machines to solve $Rm|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $Rm|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$. Let p_{ij} denote the processing time of job i on machine j for $1 \leq i \leq n$ and $1 \leq j \leq m$ in the unrelated parallel machine model. The basic idea is to develop a dynamic programming recursion for $\phi_{\ell_1,\ell_2,\ldots,\ell_m,j}$, the minimum value of the total rejection penalty of the rejected jobs when the jobs in consideration are $j, j+1, \ldots, n$, and the maximum lateness of the jobs scheduled on machine k is at most ℓ_k for all $1 \leq k \leq m$. We will omit the details here and summarize the result in the following theorem.

Theorem 5. *Dynamic programming yields an $O(nm2^m \prod_{i=1}^{m}(\sum_{j=1}^{n} p_{ji}))$ time algorithm for exactly solving $Rm|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $Rm|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$.*

4 FPTAS for Minimum Tardiness Scheduling with Rejection

In this section, we describe a fully polynomial time approximation scheme (FPTAS) for $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$. The algorithm runs in time polynomial in n,

$\frac{1}{\epsilon}$, and the size (number of bits) of the rejection costs of the jobs. We "trim" the state space of the dynamic program of Section 3.1 by fusing states that are "close" to each other. This fusion of "close" states is achieved by considering the *inflated rejection penalty* instead of the actual rejection penalty for a set of rejected jobs.

4.1 Inflated Rejection Penalty

The actual rejection penalty for a set R of rejected jobs is $\sum_{i \in R} e_i$. The definition of *inflated rejection penalty* involves a *geometric rounding* technique which we state first. For any $\epsilon' > 0$ and $x \geq 1$, the quantities $\lceil x \rceil_{\epsilon'}$ and $\lfloor x \rfloor_{\epsilon'}$ denote x rounded up and rounded down respectively to the nearest power of $(1+\epsilon')$. Thus, if $(1+\epsilon')^{k-1} < x < (1+\epsilon')^k$, then $\lceil x \rceil_{\epsilon'} = (1+\epsilon')^k$ and $\lfloor x \rfloor_{\epsilon'} = (1+\epsilon')^{k-1}$. If x is an exact power of $(1+\epsilon')$, then $\lceil x \rceil_{\epsilon'} = \lfloor x \rfloor_{\epsilon'} = x$. Note that $\lceil x \rceil_{\epsilon'} \leq (1+\epsilon')x$ for any $x \geq 1$. We will use this property in Lemma 1.

Let $R = \{i_1, i_2, \ldots, i_k\}$, where $i_1 < i_2 < \cdots < i_k$ and $k \geq 0$. We define the *ϵ'-inflated rejection penalty* $f_{\epsilon'}(R)$ of the set R of jobs with respect to any $\epsilon' > 0$ as

$$f_{\epsilon'}(R) = \begin{cases} \lceil e_{i_1} + f_{\epsilon'}(R - \{i_1\}) \rceil_{\epsilon'} & \text{if } k \geq 1 \\ 0 & \text{if } R \text{ is empty} \end{cases}$$

As an illustrative example, let $R = \{1, 2, 5\}$. Then, $f_{\epsilon'}(R) = \lceil e_1 + \lceil e_2 + \lceil e_5 \rceil_{\epsilon'} \rceil_{\epsilon'} \rceil_{\epsilon'}$. Note how we start with the largest indexed job in the set R and consider the jobs in decreasing order of job index. At every step, we add the rejection cost of the next job and then round up. We will see later why this particular order of rounding is useful. Since we are rounding up at each stage, it is easy to see that $f_{\epsilon'}(R) \geq \sum_{j \in R} e_j$ for any set R of jobs and any $\epsilon' > 0$. Hence, the reason for the term "inflated". The following lemma establishes an upper bound on the inflated rejection penalty in terms of the actual rejection penalty.

Lemma 1. *For any set R of jobs and any $\epsilon' > 0$, $f_{\epsilon'}(R) \leq (1+\epsilon')^{|R|} \sum_{j \in R} e_j$.*

This implies that if we work with the inflated rejection penalty instead of the actual rejection penalty, we will overestimate the rejection penalty by a factor of at most $(1+\epsilon)$. Working with the inflated rejection penalty has the following advantage. Since the inflated rejection penalty for any set of jobs is of the form $(1+\epsilon')^k$, we can store the exponent k instead of the actual value in the state of the dynamic program of Section 3.1. This reduces the number of states of the dynamic program so much so that we get an FPTAS out of it.

4.2 The Algorithm

In this section, we arrive at an FPTAS for $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$ by setting up a dynamic program for the following problem: to find the schedule that minimizes the maximum lateness when the *inflated rejection penalty* of the rejected jobs is given. As before, we number the jobs in ascending order of due date d_j.

Let $\phi_{k,j}$ denote the minimum value of the maximum lateness when the jobs in consideration are $j, j+1, \ldots, n$, and the inflated rejection penalty of the rejected jobs is $\tau_k = (1+\epsilon')^k$, where $\epsilon' = \epsilon/2n$. We will accommodate the zero inflated rejection cost (for the case when all the jobs are scheduled) by having $\tau_{-1} = 0$ for this case. The boundary conditions for the dynamic program are given by

$$\phi_{k,n} = \begin{cases} -\infty & \text{if } \tau_k = \lceil e_n \rceil_{\epsilon'} \\ p_n - d_n & \text{if } k = -1 \\ \infty & \text{otherwise} \end{cases}$$

Now, consider any schedule for the jobs $j, j+1, \ldots, n$ that minimizes the maximum lateness when the inflated rejection penalty of the rejected jobs is $\tau_k = (1+\epsilon')^k$. We will refer to this as the optimal schedule in the discussion below. In any such schedule, there are two possible cases — either job j is rejected or job j is scheduled.

Case 1: Job j is rejected. This is possible only if $\tau_k \geq \lceil e_j \rceil_{\epsilon'}$. Otherwise, there is no feasible solution with inflated rejection penalty τ_k and job j rejected, in which case only Case 2 applies. Hence, assume that $\tau_k \geq \lceil e_j \rceil_{\epsilon'}$. Then, the value of the maximum lateness for the optimal schedule is $\phi_{k',j+1}$, where $(1+\epsilon')^{k'}$ is the inflated rejection penalty of the rejected jobs among $j+1, \ldots, n$. From the definition of inflated rejection penalty, the possible values of k' must be such that $\lceil e_j + (1+\epsilon')^{k'} \rceil_{\epsilon'} = (1+\epsilon')^k$. Thus, the largest value of k' (call it $\tilde{k}$) is given by $(1+\epsilon')^{\tilde{k}} = \lfloor (1+\epsilon')^k - e_j \rfloor_{\epsilon'}$. But, k' may also take values smaller than $\tilde{k}$. Hence, the value of the maximum lateness for the optimal schedule is $\min\limits_{-1 \leq k' \leq \tilde{k}} \phi_{k',j+1}$.

Case 2: Job j is scheduled. In this case, the inflated rejection penalty of the rejected jobs among $j+1, \ldots, n$ must be $(1+\epsilon')^k$. Also, when job j is scheduled before all jobs in the optimal schedule for jobs $j+1, j+2, \ldots, n$, the lateness of every scheduled job among $j+1, j+2, \ldots, n$ is increased by p_j and the lateness of job j is exactly $p_j - d_j$. Then, the value of the maximum lateness for the optimal schedule is clearly $\max(\phi_{k,j+1} + p_j, p_j - d_j)$.

Combining the above two cases, we have:

$$\phi_{k,j} = \begin{cases} \max(\phi_{k,j+1} + p_j, p_j - d_j) & \text{if } \tau_k < \lceil e_j \rceil_{\epsilon'} \\ \min[\min\limits_{-1 \leq k' \leq \tilde{k}} \phi_{k',j+1}, \max(\phi_{k,j+1} + p_j, p_j - d_j)] & \text{otherwise} \end{cases}$$

Now, observe that the inflated rejection penalty of the rejected jobs is the largest when all the jobs are rejected. Hence, the inflated rejection penalty is at most $f_{\epsilon'}(\{1, 2, \ldots, n\}) \leq (1+\epsilon')^n \sum_{j=1}^n e_j$ (using Lemma 1). Thus, the largest value of k for which we need to compute $\phi_{k,j}$ is L, where L is the smallest integer such that $(1+\epsilon')^L \geq (1+\epsilon')^n \sum_{j=1}^n e_j$. Thus, L is the smallest integer greater than or equal to $\frac{\log \sum_{j=1}^n e_j}{\log(1+\epsilon')} + n$, whence $L = O(\frac{n}{\epsilon} \log \sum_{j=1}^n e_j)$.

When we consider the inflated rejection penalty instead of the actual rejection penalty, our problem becomes $1|\ |(T_{max}(S) + f_{\epsilon'}(\bar{S}))$. The answer to this problem

is given by $\min\{\phi_{k,1} + \tau_k \mid -1 \leq k \leq L\}$. Thus, we need to compute exactly $n(L+2)$ values $\phi_{k,j}$. Computation of each such value takes $O(L)$ time, so that the overall time for the dynamic program (FPTAS) is $O(nL^2) = O(\frac{n^3}{\epsilon^2} \log^2 \sum_{j=1}^{n} e_j)$. This is polynomial in the input size, since we need $\sum_{j=1}^{n} \log e_j$ bits to represent the rejection costs.

We now relate the optimal objective function values for the problems $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$ and $1|\ |(T_{max}(S) + f_{\epsilon'}(\bar{S}))$ through the following lemma.

Lemma 2. *For $\epsilon' = \epsilon/2n$, the optimal objective function value for $1|\ |(T_{max}(S) + f_{\epsilon'}(\bar{S}))$ is at most a factor of $(1+\epsilon)$ times the optimal objective function value for $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$.*

This implies that the above dynamic program, which solves $1|\ |(T_{max}(S) + f_{\epsilon'}(\bar{S}))$ *exactly*, also gives a $(1+\epsilon)$-factor approximation for $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$.

Theorem 6. *There exists a $(1+\epsilon)$-factor FPTAS for $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$ which runs in $O(\frac{n^3}{\epsilon^2} \log^2 \sum_{j=1}^{n} e_j)$ time.*

5 ϵ-Optimization Approximation for Minimum Lateness Scheduling with Rejection

Any approximation algorithm must use some notion of distance from the optimal solution in order to measure the quality of the approximate solution that it produces. The most commonly used notion in the literature is that of *worst-case relative error* – a worst-case factor by which the objective function value of the output solution differs from the optimal objective function value. Although widely accepted, this way of measuring the quality is inappropriate when the optimal objective function value could be negative, as is the case with our problem $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$. An alternative notion of approximation, called *ϵ-optimization approximation*, that can accommodate such problems into an approximation framework was defined and its properties and advantages over the worst-case relative error notion of approximation discussed in [7]. We first define ϵ-optimization approximation below.

A feasible solution x^* for an optimization problem with input costs (parameters) c_j is said to be *ϵ-optimal* if x^* is optimal for a problem with *ϵ-perturbed* costs c'_j, i.e., costs c'_j satisfying the following conditions: $(1-\epsilon)c_j \leq c'_j \leq (1+\epsilon)c_j$ for all $c_j \geq 0$, and $(1+\epsilon)c_j \leq c'_j \leq (1-\epsilon)c_j$ for all $c_j < 0$. An *ϵ-optimization approximation scheme* returns an ϵ-optimal feasible solution for any $\epsilon > 0$. If the running time is polynomial in the input size for a fixed ϵ, then it is called a polynomial time ϵ-optimization approximation scheme (PTEOS). If the running time is polynomial in the input size and $1/\epsilon$, then it is called a fully polynomial time ϵ-optimization approximation scheme (FPTEOS). Note that this notion of approximation is properly defined even if the objective function takes on negative values.

Under this notion of approximation, we provide a PTEOS for $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ in section 5.1 and an FPTEOS for $1|\ |(L_{max}(S) + \prod_{\bar{S}} e_j)$ in section 5.2, both when the *rejection costs* are allowed to be ϵ-perturbed.

5.1 PTEOS for $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$

In this section, we give a PTEOS for $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$. Our approach consists of first rounding up the rejection costs e_j to $e'_j = \lceil e_j \rceil_\epsilon$, and then finding an optimal solution for $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ with the modified costs e'_j. Note that $e'_j \leq (1+\epsilon)e_j$ for all j. Hence, by the definition of ϵ-optimization approximation, it is clear that this solution is ϵ-optimal.

To find the optimal solution to the modified problem, we run the dynamic program of Section 3.1. Observe that due to the modified rejection costs, the total rejection penalty of any set of jobs is of the form $\sum_{i=0}^{L} a_i(1+\epsilon)^i$ with $a_i \geq 0$ for all i. Here, L is such that $(1+\epsilon)^L$ is the maximum rounded rejection cost. Thus, if e_{max} is the maximum rejection cost, then L is the smallest integer such that $(1+\epsilon)^L \geq e_{max}$, i.e., $L = O(\frac{1}{\epsilon} \log e_{max})$. Note that it is possible for a_i to be greater than 1, since two rounded rejection costs could have the same value $(1+\epsilon)^i$.

Hence, instead of storing the actual rejection penalty $e = \sum_{i=0}^{L} a_i(1+\epsilon)^i$ (which is no longer an integer) in the state of the dynamic program, we can store the $(L+1)$-tuple $(a_0, a_1, \ldots, a_L)$, which denotes the rejection penalty of $\sum_{i=0}^{L} a_i(1+\epsilon)^i$. Note that $a_i \leq n$, and hence, the total number of such tuples is $n^{L+1} = n^{O(\log e_{max}/\epsilon)}$. Thus, we need to compute at most $n * n^{O(\log e_{max}/\epsilon)}$ entries $\phi_{(a_0,a_1,\ldots,a_L),j}$. Computation of each such entry takes $O(1)$ time, so that the running time of the algorithm is $O(n^{1+O(\log e_{max}/\epsilon)}) = O(n^{O(\log e_{max}/\epsilon)})$.

Theorem 7. *There exists a PTEOS for* $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ *which runs in* $O(n^{O(\log e_{max}/\epsilon)})$ *time.*

An FPTEOS for $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ when the rejection costs are allowed to be ϵ-perturbed is given in [7].

5.2 FPTEOS for $1|\ |(L_{max}(S) + \prod_{\bar{S}} e_j)$

In this section, we describe an FPTEOS for $1|\ |(L_{max}(S) + \prod_{\bar{S}} e_j)$. The algorithm runs in time polynomial in n, $\frac{1}{\epsilon}$, and the size (number of bits) of the rejection costs of the jobs. Note that for this problem, the total rejection penalty is the *product* and not the sum of the rejection costs of the rejected jobs.

As in the previous section, we first round up the rejection costs e_j to $e'_j = \lceil e_j \rceil_\epsilon$, and then find an optimal solution for $1|\ |(L_{max}(S) + \prod_{\bar{S}} e_j)$ with the modified costs e'_j. Observe that due to the modified rejection costs, the total rejection penalty of any set of jobs is of the form $(1+\epsilon)^k$, i.e., a power of $(1+\epsilon)$. Hence, instead of storing the actual rejection penalty $e = (1+\epsilon)^k$ (which is no longer an integer) in the state of the dynamic program, we can store the exponent of the rejection penalty, i.e., the value k will denote a rejection penalty

of $\tau_k = (1+\epsilon)^k$ for $k > 0$. We explain below why $k = 0$ is a special case and how we handle it.

Since the total rejection penalty is the product of the rejection costs of the rejected jobs, jobs with a rejection cost of 1 do not increase the rejection penalty when they get rejected. In order to avoid this anomaly, we will assume that $e_j > 1$ for all j. Then, the exponent of $k = 0$ in the rejection penalty will be indicative of the fact that none of the jobs are rejected, and we will make the rejection penalty zero in this case by defining $\tau_0 = 0$.

We set up a dynamic program for the following problem: to find the schedule that minimizes the maximum lateness when the total rejection penalty (product form) of the rejected jobs is given. Let $\phi_{k,j}$ denote the minimum value of the maximum lateness when the jobs in consideration are $j, j+1, \ldots, n$, and the total rejection penalty of the rejected jobs is τ_k, where $\tau_k = (1+\epsilon)^k$ for $k > 0$ and $\tau_0 = 0$. Let L_j denote the exponent of e'_j, i.e., $e'_j = (1+\epsilon)^{L_j}$. The boundary conditions for the dynamic program are given by

$$\phi_{k,n} = \begin{cases} -\infty & \text{if } k = L_n \\ p_n - d_n & \text{if } k = 0 \\ \infty & \text{otherwise} \end{cases}$$

Now, consider any schedule for the jobs $j, j+1, \ldots, n$ that minimizes the maximum lateness when the total rejection penalty of the rejected jobs is $(1+\epsilon)^k$. We will refer to this as the optimal schedule in the discussion below. In any such schedule, there are two possible cases — either job j is rejected or job j is scheduled.

Case 1: Job j is rejected. This is possible only if $(1+\epsilon)^k \geq e'_j$, i.e., $k \geq L_j$. Otherwise, there is no feasible solution with total rejection penalty $(1+\epsilon)^k$ in which job j (with rejection cost e'_j) is rejected, in which case only Case 2 applies. Hence, assume that $k \geq L_j$. Then, the value of the maximum lateness for the optimal schedule is clearly $\phi_{k-L_j,j+1}$, since the total rejection penalty of the rejected jobs among $j+1, \ldots, n$ must be $(1+\epsilon)^k / e'_j = (1+\epsilon)^{(k-L_j)}$.

Case 2: Job j is scheduled. In this case, the total rejection penalty of the rejected jobs among $j+1, \ldots, n$ must be $(1+\epsilon)^k$. Also, when job j is scheduled before all jobs in the optimal schedule for jobs $j+1, j+2, \ldots, n$, the lateness of every scheduled job among $j+1, j+2, \ldots, n$ is increased by p_j and the lateness of job j is exactly $p_j - d_j$. Then, the value of the maximum lateness for the optimal schedule is clearly $\max(\phi_{k,j+1} + p_j, p_j - d_j)$.

Combining the above two cases, we have:

$$\phi_{k,j} = \begin{cases} \max(\phi_{k,j+1} + p_j, p_j - d_j) & \text{if } k < L_j \\ \min[\phi_{k-L_j,j+1}, \max(\phi_{k,j+1} + p_j, p_j - d_j)] & \text{otherwise} \end{cases}$$

Now, observe that the total rejection penalty of the rejected jobs is at most $\prod_{j=1}^{n} e'_j = \prod_{j=1}^{n} (1+\epsilon)^{L_j} = (1+\epsilon)^{\sum_{j=1}^{n} L_j}$. From the definition of the L_j's, it follows that L_j is the smallest integer such that $(1+\epsilon')^{L_j} \geq e_j$, i.e.,

$L_j = O(\frac{1}{\epsilon} \log e_j)$. Hence, the maximum exponent of the total rejection penalty is $\sum_{j=1}^{n} L_j = O(\frac{1}{\epsilon} \sum_{j=1}^{n} \log e_j)$.

The answer to our problem with modified rejection costs e'_j is given by $\min\{\phi_{k,1} + \tau_k \mid 0 \leq k \leq \sum_{j=1}^{n} L_j\}$. Thus, we need to compute at most $n \sum_{j=1}^{n} L_j$ values $\phi_{k,j}$. Computation of each such value takes $O(1)$ time, so that the overall running time for the dynamic program is $O(n \sum_{j=1}^{n} L_j) = O(\frac{n}{\epsilon} \sum_{j=1}^{n} \log e_j)$.

Theorem 8. *There exists an FPTEOS for* $1|\ |(L_{max}(S) + \prod_{\bar{S}} e_j)$ *which runs in* $O(\frac{n}{\epsilon} \sum_{j=1}^{n} \log e_j)$ *time.*

Acknowledgements. Thanks to Jim Orlin for suggesting the approach to the $\mathcal{NP}$-completeness proof of Section 2, for introducing and suggesting the application of ϵ-optimization approximation to minimum lateness scheduling with rejection, and for helpful discussions and numerous other suggestions.

References

1. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.
2. E. L. Lawler, J. K. Lenstra, A. H. G. Rinooy Kan, and D. B. Shmoys. Sequencing and Scheduling: Algorithms and Complexity. *Handbooks in Operations Research and Management Science, Vol. 4, Logistics of Production and Inventory*, pp. 445–522, North-Holland, 1993.
3. Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, and L. Stougie. Multiprocessor scheduling with rejection. *7th ACM-SIAM Symposium on Discrete Algorithms*, pp. 95–103, 1996.
4. H. Hoogeveen, M. Skutella, and G. J. Woeginger. Preemptive scheduling with rejection. *Algorithms – ESA 2000*, 8th Annual European Symposium on Algorithms, September 2000, *Lecture Notes in Computer Science*, vol. 1879, pp. 268 – 277.
5. S. Seiden. Preemptive multiprocessor scheduling with rejection. *Theoretical Computer Science*, vol. 262, issue 1, pp. 437–458, July 2001.
6. D. W. Engels, D. R. Karger, S. G. Kolliopoulos, S. Sengupta, R. N. Uma, and J. Wein. Techniques for Scheduling with Rejection. *Algorithms – ESA '98*, 6th Annual European Symposium on Algorithms, August 1998, *Lecture Notes in Computer Science*, vol. 1461, pp. 490 – 501.
7. J. B. Orlin, A. S. Schulz, and S. Sengupta. ϵ-Optimization Schemes and L-Bit Precision: Alternative Perspectives in Combinatorial Optimization. *32nd Annual ACM Symposium on Theory of Computing (STOC)*, 2000.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. W. H. Freeman and Company, New York, 1979.

Fast Algorithms for a Class of Temporal Range Queries*

Qingmin Shi and Joseph JaJa

Institute for Advanced Computer Studies, Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20742, USA
{qshi,joseph@umiacs.umd.edu}

Abstract. Given a set of n objects, each characterized by d attributes specified at m fixed time instances, we are interested in the problem of designing efficient indexing structures such that the following type of queries can be handled efficiently: given d value ranges and a time interval, report or count all the objects whose attributes fall within the corresponding d value ranges at each time instance lying in the specified time interval. We establish efficient data structures to handle several classes of the general problem. Our results include a linear size data structure that enables a query time of $O(\log n \log m + f)$ for one-sided queries when $d = 1$, where f is the output size. We also show that the most general problem can be solved with polylogarithmic query time using nonlinear space data structures.

1 Introduction

In this paper, we introduce a framework for exploring temporal patterns of a set of objects and discuss the design of indexing structures for handling temporal orthogonal range queries in such a framework. We assume that each object is characterized by a set of attributes, whose values are given for a sequence of time snapshots. The temporal patterns of interest can be defined as the values of certain attributes remaining within certain bounds, changing according to a given pattern (say increasing or decreasing), or satisfying certain statistical distributions. We focus here on temporal patterns characterized by orthogonal range values over the attributes. More specifically, we are aiming to design indexing structures to quickly find objects whose attributes fall within a set of ranges at each time instance within a time period, where the ranges and the time period are specified at query time.

More formally, let S be a set of n objects $\{O_1, O_2, \cdots, O_n\}$, each of which is characterized by a set of d attributes whose values change over time. We are given m snapshots of each object at time instances $t_1, t_2, \ldots, t_m$. The set of

* Supported in part by the National Science Foundation through the National Partnership for Advanced Computational Infrastructure (NPACI), DoD-MD Procurement under contract MDA90402C0428, and NASA under the ESIP Program NCC5300.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 91–102, 2003.
© Springer-Verlag Berlin Heidelberg 2003

values of the d attributes of object O_i at time instance t_j is denoted as a vector $\mathbf{v}(i,j) = [v_i^j(1), v_i^j(2), \ldots, v_i^j(d)]$.

We are interested in developing a data structure for S so that the following types of queries, called *temporal range queries*, can be handled very quickly:

> Given two vectors $\mathbf{a} = [a_1, a_2, \cdots, a_d]$ and $\mathbf{b} = [b_1, b_2, \cdots, b_d]$, and a time interval $[t_s, t_e]$, determine the set Q of objects such that for every $O_i \in Q$, $a_k \leq v_i^j(k) < b_k$ for all $1 \leq k \leq d$ and $t_s \leq t_j \leq t_e$.

Note that the general multidimensional orthogonal range search is a special case of our problem corresponding to a single time snapshot. Typically, we measure the complexity in terms of the storage cost of the data structure and the query time as functions of n, m, and the output size f.

Many applications fall in a natural way under our general framework. The following is a list of a few such examples.

- Climatologists are often interested in studying the climate change patterns for certain geographical areas, each characterized by a set of environmental variables such as temperature, precipitation, humidity, etc. Given a time series of such information for n regions, one would like to quickly explore relationships among such regions by asking queries of the following type: determine the regions where the annual precipitation is above 40 inches and the summer temperature is above 70°F between the years 1965 and 1975.
- In the stock market, each stock can be characterized by its daily opening price, closing price, and trading volume. Related interesting queries that fall under our framework are of the following type: determine the stocks, each of whose daily opening price is less than \$2 and whose daily trading volume is larger than 200 million shares during each day of the year 2000.
- As an application related to data warehousing, consider a retail chain that has stores across the country, each of which reports their sales on a monthly basis. A typical query will for example be to identify the stores whose sales exceeded \$3,000,000 for each of the past 12 months.
- Consider a set of n cities, each characterized by annual demographic and health data, for a period of 30 years. In exploring patterns among these cities, one may be interested in asking queries about the number of cities that had a high cancer rate and a high ozone level in each year between 1990 and 2000.

1.1 Background

The d-dimensional orthogonal range search problem, which is a special case of our problem, has been studied extensively in the literature. The best results are output sensitive algorithms that achieve linear space and polylogarithmic query time for three-sided reporting queries and four-sided counting queries for $d = 2$ [15,3], and for dominance reporting queries for $d = 3$. Otherwise, all fast query time algorithms require nonlinear space, sometimes coupled with matching lower bounds under certain computational models [2,5,4]. Note that we cannot

treat our problem as an orthogonal range search problem by simply treating the time snapshots as just an extra dimension appended to the d dimensions corresponding to the attributes. This is the case since the values of an object's attributes at different time instances should not be treated simply as independent of each other. Even though we can combine all the attribute values of an object together to specify that object, this will result in an (md)-dimensional range search problem, which is clearly undesirable, especially for large m. The techniques presented in [11,9] to handle the *generalized intersection searching* problem can be used to solve a variation of our problem in which we only require that the attributes fall within the specified value ranges during *some* time instances in the time interval. However, except for a special case discussed later in the paper, their techniques do not appear to shed any light on the general problem considered in this paper.

Another related class of problems studied in the literature, especially in the database literature, deals with a time series of data by appending a time stamp (or time interval) to each piece of data separately. However such an approach will be quite inefficient to capture temporal information about single objects since it will have to somehow process the values at all the time steps between t_s and t_e at query time. Examples of such techniques include those based on *persistent* data structures [6], such as the Multiversion B-tree [12] and the Multiversion Access Methods [22], and the Overlapping B^+-trees [14] and its extensions such as the Historical R-tree [16], the HR^+-tree [19], and the Overlapping Linear Quadtrees [20,21].

Another related topic involves the so-called *kinetic data structures*, which are used for indexing moving objects. Queries similar to ours involving both time periods and positions of objects have been studied, for example in the work of Agarwal et al. [1] and Saltenis et al. [17]. However, the objects are considered there to be points moving along a straight line and at a consistent speed. As a result, the positions of the objects need not be explicitly stored. In our case, such a problem will be formulated as the positions of each object at different time instances, without any assumption about expected trajectories or speeds.

1.2 Main Result

Our results include the following:

- A linear space data structure that handles temporal range queries for a single object in $O(1)$ time, assuming the number d of attributes is constant.
- Two data structures that handle temporal one-sided range reporting queries for a set of objects in $O(\log m \log n + f)$, and $O(\log m \log n / \log\log n + f)$ time respectively, the first using $O(nm)$ space, and the second using $O(mn \log^{\epsilon} n)$, where f is the number of objects satisfying the query, ϵ is an arbitrarily small positive constant, and $d = 1$.
- Two data structures that use $O(nm \log(nm))$ and $O(nm \log^{1+\epsilon}(nm))$ space respectively to answer the temporal one-sided range counting queries. The first data structure enables $O(\log^2(nm))$ query time and the second enables $O((\log(nm)/\log\log(nm))^2)$ time, under the assumption that $d = 1$.

- By a reduction to the $2d$-dimensional dominance problem, the most general problem can be solved in polylogarithmic query time using $O(nm^2\text{polylog}(n))$ space. When m is extremely large, we show that it is possible to use $o(nm^2)$ space to achieve polylogarithmic query time.

Before proceeding, we notice that the actual time instances $\{t_1, t_2, \cdots, t_m\}$ can be replaced by their subscripts $\{1, 2, \cdots, m\}$. By doing so, we introduce the additional complexity of having to convert t_s and t_e specified by the query to l_1 and l_2 respectively, where t_{l_1} is the first time instance no earlier than t_s and t_{l_2} is the last time instance no later than t_e. This conversion can be done in $O(\log m)$ time and $O(m)$ space using binary search or an asymptotically faster $O(\log m/\log\log m)$ algorithm and the same $O(m)$ space using the fusion tree of Fredman and Willard on a variation of the RAM model [7]. In the remainder of this paper, we assume that the time instances are represented by integers $\{1, 2, \cdots, m\}$ and the time interval in the query is represented by two integers l_1 and l_2. For brevity, we will use the $[i, j]$ to denote the set of integers $\{i, i+1, \cdots, j\}$ as well as a time interval.

The rest of the paper is organized as follows. The next section discusses a special version of the temporal range search problem, which involves only a single object. The data structure for the reporting case of temporal one-sided range queries is covered in Section 3, while the counting version is covered in Section 4. In Section 5, we deal with the two-sided temporal range query.

2 Preliminaries: Handling Range Queries of a Single Object

We provide a simple solution to the case of a single object O, which will then be used to handle the more general case. Let the values of the attributes of O at time instance j be $[v^j(1), v^j(2), \cdots, v^j(d)]$. Given two real vectors $\mathbf{a} = [a_1, a_2, \cdots, a_d]$ and $\mathbf{b} = [b_1, b_2, \cdots, b_d]$, and the time interval $[l_1, l_2]$, we will describe an efficient method to test whether the following predicate holds:

P: For every time instances j that satisfies $l_1 \leq j \leq l_2$, $a_k \leq v^j(k) \leq b_k$ for all k between 1 and d.

Since we are assuming that d is a fixed constant, we can restrict ourselves to the following case. Let the object O be specified by $[v^1, v^2, \cdots, v^m]$, where each v^i is a real number. We develop a data structure that can be used to test the following predicate for any given parameters l_1, l_2, and a:

P': For every time instance j satisfying $l_1 \leq j \leq l_2$, $v^j \geq a$.

We start by making the following straightforward observation.

Observation 1. *A predicate of type P' is true if and only if* $\min\{v^j | j \in [l_1, l_2]\} \geq a$.

Using this observation, our problem is reduced to finding the minimum value v^j of the object during the time period $[l_1, l_2]$ and comparing it against the value of a.

The problem of finding the minimum value in time period $[l_1, l_2]$ can be reduced to the problem of finding the nearest common ancestor of the appropriate nodes in the so called *Cartesian tree*, as described in [8].

A Cartesian tree [23] for a sequence of m real numbers is a binary tree with m nodes. In our case, a Cartesian tree for time instances $[l, r]$ with $l \leq r$ has $r - l + 1$ nodes. The root stores the smallest value v^i over the time period $[l, r]$, where i is an integer between l and r. If there are multiple v^i's with the smallest value, the earliest one is chosen to be stored at the root. The left subtree of the root is the Cartesian tree for time instances $[l, i-1]$ and the right subtree is the Cartesian tree for the time instances $[i+1, r]$. The left (resp. right) tree is null if $i = l$ (resp. $i = r$). The tree nodes are labeled l through r according to the in-order traversal of the tree (which correspond to their time instances). Figure 1 gives an example of a Cartesian tree.

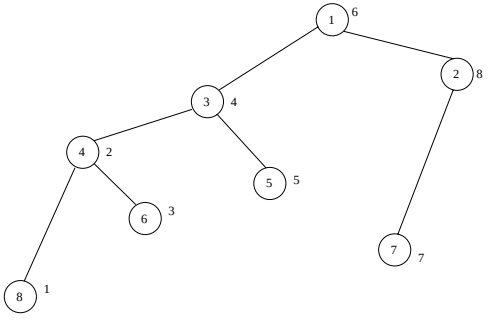


Fig. 1. A Cartesian tree for the sequence $[8, 4, 6, 3, 5, 1, 7, 8]$. The number outside each node represents the time instance of the attribute value stored at the node.

It is easy to realize that the smallest value among $\{v^i, \ldots, v^j\}$ is the one stored in the nearest common ancestor of nodes i and j. The problem of finding nearest common ancestors was addressed in [10], where the following result is shown.

Lemma 1. *Given a collection of rooted trees with n vertices, the nearest common ancestor of any two vertices can be found in $O(1)$ time, provided that pointers to these two vertices are given as input. This algorithm uses $O(n)$ space.*

Given the above lemma, we immediately have the following results.

Theorem 1. *Predicate P' can be evaluated in $O(1)$ time using an $O(m)$ space data structure.*

Corollary 1. *A P predicate can be evaluated in $O(1)$ time using an $O(m)$ space data structure.*

3 Handling One-Sided Queries for an Arbitrary Number of Objects

In this section, we deal with temporal range queries for n objects with only one attribute, that is $d = 1$. Let v_i^j denote the value of object O_i at time instance j. We want to preprocess the data and construct a linear size data structure so that queries of the following type can be answered quickly:

Q_1: Given a tuple (l_1, l_2, a), with $l_1 \leq l_2$, report all objects whose attributes are greater than or equal to a for each of the time instances between l_1 and l_2.

We call such queries *temporal one-sided reporting queries.*

Observation 1 plays an very important role in dealing with queries of type Q_1. A straightforward approach to solve our problem would be to determine for each possible time interval the set of minimal values, one for each object, and store the minima corresponding to each time interval in a sorted list. A query can then be immediately handled using the sorted list corresponding to the time interval $[l_1, l_2]$. However, the storage cost would then be $O(nm^2)$, which is quite high especially in the case when m is much larger than n. We will develop an alternative strategy that requires only linear space.

Assume that we have built a Cartesian tree C_i for object O_i. Then, each attribute v_i^j of this object can be associated with the maximum sequence of contiguous time instances $[s_i^j, e_i^j]$ during which v_i^j is the smallest. (Ties are broken by the value of j.) We call this sequence the *dominant interval* of v_i^j. In fact, the dominant interval corresponds to the set of nodes in the subtree rooted at node j in C_i. For example, consider the object O_i whose corresponding Cartesian tree is shown in Fig. 1. The dominant interval of v_i^4 is $[1, 5]$.

Consider the set of the nm tuples $(v_i^j, s_i^j, e_i^j, i, j)$, $1 \leq i \leq n$, $1 \leq j \leq m$. One way of answering a Q_1 query would be to identify those 5-tuples that satisfy $[s_i^j, e_i^j] \supseteq [l_1, l_2]$ and $v_i^j \geq a$. However an object can be reported a non-constant number of times, which does not meet our goal of achieving a query time of $O(\text{polylog}(nm) + f)$. We can use the techniques in [11] for 3-D point enclosure or those in [9] for 3-D dominance query to design an output-sensitive algorithm. The former results in an $O((nm)^{1.75})$ space structure with $O(\log(nm) + f)$ query time and the latter results in an $O(nm \log(nm))$ space structure with $O(\log^2(nm) + f)$ query time, and hence they both use non-linear space. Our strategy is different and will result in a linear space indexing scheme that is based on the following lemma.

Lemma 2. *An object O_i should be reported if and only if there exist a 5-tuple $(v_i^j, s_i^j, e_i^j, i, j)$ such that the following conditions are true: $[s_i^j, e_i^j] \supseteq [l_1, l_2]$; $j \in [l_1, l_2]$; and $v_i^j \geq a$. If such a tuple exists, it is unique.*

Proof. Suppose an object O_i satisfies the query. Then its values during the time period $[l_1, l_2]$ are no smaller than a. Let $v_i^j = \min\{v_i^l | l_1 \leq l \leq l_2\}$. It is obvious that the 5-tuple $(v_i^j, s_i^j, e_i^j, i, j)$ satisfies the three conditions in the lemma. On

the other hand, the existence of such a 5-tuple ensures that v_i^j, which is the minimum value of object O_i over $[s_i^j, e_i^j] \supseteq [l_1, l_2]$, is at least as large as a, and hence object O_i should be reported. The uniqueness of the 5-tuple is guaranteed by the definition of dominant intervals. Indeed, suppose we have another 5-tuple $(v_i^{j'}, s_i^{j'}, e_i^{j'}, i, j')$ that satisfies $[s_i^{j'}, e_i^{j'}] \supseteq [l_1, l_2]$, $j' \in [l_1, l_2]$, and $v_i^{j'} \geq a$. By definition, both v_i^j and $v_i^{j'}$ are the smallest values during the time interval $[l_1, l_2]$. Without loss of generality, assume $j < j'$. Then $s_i^{j'} > j$, which contradicts the condition that $s_i^{j'} \leq l_1 \leq j$. □

Lemma 2 reduces the problem of determining the objects satisfying the query to finding a 5-tuple for each such object, which satisfies the three stated conditions. To solve the latter problem, we first single out those attributes that were taken during the time period $[l_1, l_2]$ and then filter them using the remaining two conditions.

We first construct a balanced binary tree T based on the m time instances. The jth leaf node from the left corresponds to time instance j. Each node v of this tree is associated with a set $S(v)$ of n tuples, one from each object. If v is the jth leaf node, then $S(v) = \{(v_i^j, s_i^j, e_i^j, i, j) | i = 1, \ldots, n\}$. If v is an internal node with two children u and w and the 5-tuples of object O_i in $S(u)$ and $S(w)$ are $(v_i^{j_1}, s_i^{j_1}, e_i^{j_1}, i, j_1)$ and $(v_i^{j_2}, s_i^{j_2}, e_i^{j_2}, i, j_2)$ respectively, then the 5-tuple of object O_i in $S(v)$ is $(v_i^j, s_i^j, e_i^j, i, j)$, where j is either j_1 or j_2, depending on whether $[s_i^{j_1}, e_i^{j_1}] \supseteq [s_i^{j_2}, e_i^{j_2}]$ or $[s_i^{j_2}, e_i^{j_2}] \supseteq [s_i^{j_1}, e_i^{j_1}]$. (The reason why one and only one of the above conditions must be true should be easy to understand by recalling the definition of dominant intervals.)

Given a Q_1 query (l_1, l_2, a), we can easily find the set of $O(\log m)$ *allocation nodes* in T, using the interval $[l_1, l_2]$. An allocation node is a node whose corresponding time interval is fully contained in $[l_1, l_2]$ and that of whose parent is not. For each allocation node v, we know that all the n attributes in $S(v)$ are taken during the time period $[l_1, l_2]$. Therefore, if a 5-tuple $(v_i^j, s_i^j, e_i^j, i, j) \in S(v)$ satisfies $[s_i^j, e_i^j] \supseteq [l_1, l_2]$ and $v_i^j \geq a$, then O_i should be reported. Otherwise, object O_i should not be reported. In either case, no further search on v's descendants is needed. This is true because of the following. First, if O_i is reported at node v, then there is no need to look for O_i any more. Second, if O_i is not reported at v, this means either $[s_i^j, e_i^j] \not\supseteq [l_1, l_2]$ or $v_i^j < a$. If the former is true, then no tuple of O_i stored in the descendants of v can cover $[l_1, l_2]$ because $[s_i^j, e_i^j]$ covers the dominant intervals of all the other values of O_i stored in the subtree rooted at v. If the latter is true, then we are sure O_i should not be reported at all.

One final note is that, even though an object is represented multiples times in the form of its tuples, it will be reported at most once. This can be justified as follows. If an object is reported, then only one of its m tuples satisfies the conditions derived from the query. Note that even though a tuple may be stored in up to $\Theta(\log m)$ nodes, these nodes form a partial path from the root to a leaf node and, as a result, only the one that is an allocation node with respect to $[l_1, l_2]$ will be considered.

For each node v, looking for 5-tuples $(v_i^j, s_i^j, e_i^j, i, j) \in S(v)$ which satisfy $[s_i^j, e_i^j] \supseteq [l_1, l_2]$ and $v_i^j \geq a$ is equivalent to a three-dimensional dominance reporting problem, which can be solved in $O(\log n + f(v))$ time using the data structure of Makris and Tsakalidis [13], which we call the *dominance tree*. Here $f(v)$ is the number of objects reported when node v is visited. Note that there are $O(m)$ nodes in the tree and each node is associated with a dominance tree of size $O(n)$. The overall size of the data structure is $O(nm)$. A query process involves identifying the $O(\log m)$ allocation nodes in $O(\log m)$ time and searching the dominance trees associated with these allocation nodes. Hence $O(\log n + f(v))$ time is spent at each such node v. Therefore, the complexity of the overall algorithm is $O(\log n \log m + f)$, where f is total number of objects reported.

In [18], we provide a faster algorithm for solving the the three-dominance query problem under the RAM model of [7]. The algorithm uses $O(n \log^{\epsilon} n)$ space and $O(\log n / \log\log n + f)$ query time, where ϵ is an arbitrarily small positive constant. Using this data structure instead of the dominance tree, we can further reduce the query complexity to $O(\log m \log n / \log\log n + f)$ at the expense of increasing the storage cost to $O(mn \log^{\epsilon} n)$. We thus have the following theorem.

Theorem 2. *Given n objects, each specified by the values of its attribute at m time instances, we can build an indexing structure so that any one-sided reporting query can be answered in $O(\log n \log m + f)$ time and $O(nm)$ space, or $O(\log m \log n / \log\log n + f)$ time and $O(mn \log^{\epsilon} n)$ space, where f is the number of objects reported and ϵ is an arbitrarily small positive constant.*

We next consider the counting query counterpart.

4 Handling One-Sided Counting Queries

In this section, we consider the following temporal range counting queries.

Q_2: Given a tuple (l_1, l_2, a), with $l_1 \leq l_2$, determine the number of objects whose values are greater than or equal to a for all time instances between l_1 and l_2.

The conditions stated in Lemma 2 (Section 3) can be expressed as $s_i^j \leq l_1 \leq j$, $j \leq l_2 \leq e_i^j$, and $v_i^j \geq a$; and there is at most one such instance. Hence the answer to the query is $|A(l_1, l_2, a)|$, where $A(l_1, l_2, a) = \{(i, j) | s_i^j \leq l_1 \leq j, j \leq l_2 \leq e_i^j,\ and\ v_i^j \geq a\}$.

Let $U(l_1, l_2, a) = \{(i, j) | v_i^j \geq a\}$, $B_1(l_1, l_2, a) = \{(i, j) | l_2 < j\ and\ v_i^j \geq a\}$, $B_2(l_1, l_2, a) = \{(i, j) | l_2 > e_i^j\ and\ v_i^j \geq a\}$, $B_3(l_1, l_2, a) = \{(i, j) | l_1 < s_i^j\ and\ v_i^j \geq a\}$, $B_4(l_1, l_2, a) = \{(i, j) | l_1 > j\ and\ v_i^j \geq a\}$, $C_1(l_1, l_2, a) = \{(i, j) | l_1 < s_i^j, l_2 < j\ and\ v_i^j \geq a\}$, $C_2(l_1, l_2, a) = \{(i, j) | l_1 > j, l_2 < j\ and\ v_i^j \geq a\}$, $C_3(l_1, l_2, a) = \{(i, j) | l_1 < s_i^j, l_2 > e_i^j\ and\ v_i^j \geq a\}$, and $C_4(l_1, l_2, a) = \{(i, j) | l_1 > j, l_2 > e_i^j\ and\ v_i^j \geq a\}$. We have the following lemma:

Lemma 3. $|A| = |U| - |B_1| - |B_2| - |B_3| - |B_4| + |C_1| + |C_2| + |C_3| + |C_4|$.

Proof (sketch). It is easy to see that $\overline{A} = U - A = B_1 \cup B_2 \cup B_3 \cup B_4$. Thus, $|\overline{A}| = \sum_{i=1,2,3,4} |B_i| - \sum_{i,j \in \{1,2,3,4\}, i \neq j} |B_i \cap B_j| + \sum_{i,j,k \in \{1,2,3,4\}, i \neq j \neq k} |B_i \cap B_j \cap B_k| - |\cap_{i=1,2,3,4} B_i|$. It is clear the third and the fourth terms in the right hand side of this equation are both zero. As for the second term, the only four non-empty intersections are $B_1 \cap B_3$, $B_1 \cap B_4$, $B_2 \cap B_3$, and $B_2 \cap B_4$, which correspond to the sets C_1, C_2, C_3, C_4 respectively. □

The problem of determining the size of each of the sets U, B_i or C_i can be viewed as special versions of the three-dimensional dominance counting problem defined as follows:

Q_2'**:** *Given a set V of n three dimensional points, preprocess V so that for a given point (x, y, z), the number of points in V that are dominated by (x, y, z) can be reported efficiently.*

Unlike the reporting case, algorithms for the three-dimensional dominance counting problem that have linear space and polylogarithmic query time are not known to the authors' best knowledge. However Chazelle gives a linear space and $O(\log n)$ time algorithm [3] for the two-dimensional case. Using the scheme of the range tree, his result can easily be extended to the three-dimensional case by first building a binary search tree on the x-coordinates, and then associate with each node the data structure for answering two-dimensional dominance queries involving only the y- and z-coordinates. The resulting data structure provides an $O(n \log n)$ space and $O(\log^2 n)$ time solution.

By using the fusion tree techniques, we were able to improve the query time to $O((\log n / \log \log n)^2)$ at the expense of increasing the storage cost by a factor of $O(\log^{\epsilon} n / \log \log n)$. For details, see [18]. Since we have a total of nm tuples, Theorem 3 follows.

Theorem 3. *Given n objects, each characterized by the values of its attribute at m time instances, we can preprocess the input so that any one-sided counting query can be answered in $O(\log^2(nm))$ time using $O(nm \log(nm))$ space, or $O((\log(nm)/\log\log(nm))^2)$ time using $O(nm \log^{1+\epsilon}(nm)/\log\log(nm))$ space, where ϵ is an arbitrarily small positive constant.*

Note that the techniques described in [9] for three-sided range counting can be used to handle the one-sided temporal range counting query in $O(\log^2(nm))$ time using $O(nm \log^2(nm))$ space, and hence our algorithm achieves the same query time but uses less space.

5 Fast Algorithms for Handling Two-Sided Queries

In this section, we address the general type of queries for which the values of the objects to be reported are bounded between two values a and b during the time period $[l_1, l_2]$. More specifically,

Q_3: Given a tuple (l_1, l_2, a, b), with $l_1 \leq l_2$ and $a \leq b$, report all objects O_i, such that $a \leq v_i^j \leq b$ for all $j = l_1, \ldots, l_2$.

The following is a direct extension of Observation 1.

Observation 2. *An object O_i should be reported for a Q_3 query if and only if* $\min\{v_i^j | j \in [l_1, l_2]\} \geq a$ *and* $\max\{v_i^j | j \in [l_1, l_2]\} \leq b$.

We first show that, even for an arbitrary number d of attributes, the two-sided queries can be handled fast if we are willing to use $O(nm^2\text{polylog}(n))$ space for the indexing structure. We later show that we can achieve fast query time using $o(nm^2)$ space in the case when m is extremely large. We start by looking at the case when $d = 1$, which admits a simple solution.

To achieve a polylogarithmic query time, we compute for each pair $(t_1, t_2) \in [1, m] \times [1, m]$ with $t_1 < t_2$ the minimum value $m_i(t_1, t_2)$ and maximum value $M_i(t_1, t_2)$ for each object O_i and index the n minimum-maximum pairs in a suitable data structure $T(t_1, t_2)$ designed to efficiently handle two-dimensional dominance queries. Pointers to these $O(m^2)$ structures can be stored in a array to allow constant-time access. Given any query (l_1, l_2, a, b), we use (l_1, l_2) to locate the appropriate data structure $T(l_1, l_2)$ in constant time and use it to answer the two-dimensional dominance query: $m_i(t_1, t_2) \geq a$ and $M_i(t_1, t_2) \leq b$.

A possible data structure for $T(t_1, t_2)$ is the priority search tree [15] or the improved version of the priority search tree that appeared in [24]. The former allows $O(\log n + f)$ query time and the latter allows $O(\log n / \log\log n + f)$ query time, both using linear space.

We can handle counting queries in a similar fashion using as $T(t_1, t_2)$ Chazelle's linear space data structure to achieve $O(\log n)$ query complexity or the one in [18] with $O(n \log^{\epsilon} n)$ space and $O(\log n / \log\log n)$ query time. Since we have $m(m-1)/2$ (t_1, t_2)-pairs, Theorem 4 follows.

Theorem 4. *Given n objects, each of which is specified by the values of its attribute at m time instances, it is possible to design an indexing structure so that the reporting version of any two-sided query can be answered in $O(\log n / \log\log n + f)$ time using $O(nm^2)$ space for the indexing structure. The counting version can be handled in $O(nm^2)$ space and $O(\log n)$ query time, or $O(nm^2 \log^{\epsilon} n)$ space and $O(\log n / \log\log n)$ query time.*

The strategy described above can be extended to handle any arbitrary number d of attributes describing each object. Our general problem will be reduced to $O(m^2)$ $2d$-dimensional dominance queries. Using the results of [18], we obtain the following theorem.

Theorem 5. *The general temporal range query problem, with n objects, each with $d > 1$ attributes specified at m time instances, can be handled with a data structure of size $O(m^2 \cdot n \log^{\epsilon} n (\log n / \log\log n)^{2d-3})$ and a query time $O((\log n / \log\log n)^{2d-2} + f)$. The counting query can be handled using $O(m^2 \cdot n \log^{\epsilon} n (\log n / \log\log n)^{2d-2})$ space and in $O((\log n / \log\log n)^{2d-1})$ time.*

Clearly the space used to handle two-sided queries, even in the case when $d = 1$, is quite high. An interesting problem is whether there exists a data structure whose size is $o(nm^2)$, such that the general temporal range search problem can be solved in time that is polylogarithmic in nm and proportional to the number of objects found. We provide a partial answer to this question by showing that this is indeed the case when m is extremely large.

Theorem 6. *Given n objects, each characterized by the values of its attribute at m time instances such that $m > n!$, it is possible to design an indexing structure of size $o(nm^2)$ such that the reporting version of any two-sided query can be answered in $O(\log^2(nm) + f)$ time.*

Proof. For each pair of time instances j_1 and j_2, let $m_i(j_1, j_2) = \min\{v_i^j | j \in [j_1, j_2]\}$, and $M_i(j_1, j_2) = \max\{v_i^j | j \in [j_1, j_2]\}$. Let $r_i(j_1, j_2)$ be the rank of $m_i(j_1, j_2)$ in the set $\{m_l(j_1, j_2) | l = 1, 2, \ldots, n\}$ and $R_i(j_1, j_2)$ be the rank of $M_i(j_1, j_2)$ in the set $\{M_l(j_1, j_2) | l = 1, 2, \ldots, n\}$. Thus an object O_i is represented by the point $(r_i(j_1, j_2), R_i(j_1, j_2))$ corresponding to the time period $[j_1, j_2]$.

Note that at most $O((n!)^2)$ different point sets are possible for each pair of j_1 and j_2. During preprocessing time, we simply build a priority search tree for each possible point set and construct an array of m^2 entries that indicate for each pair (j_1, j_2) the corresponding priority search tree.

Since the query is given as (l_1, l_2, a, b), we have to map the numbers a and b to the rank space of (l_1, l_2) before the corresponding priority search tree can be searched. Let $a(j_1, j_2)$ and $b(j_1, j_2)$ be the parameters used to search the appropriate priority search tree. Then $a(j_1, j_2)$ is equal to the number of points that are always greater than or equally a during the time period $[l_1, l_2]$ and $b(j_1, j_2)$ is equal to the number of points that are always less than or equal to b in that period. These two numbers can be independently computed using the results in Section 4. Even without using the fusion tree, this step still can be done in $O(\log^2(nm))$ time using $O(nm \log(nm))$ space.

The storage cost of this scheme is $O(m^2 + n(n!)^2 + nm \log(nm)) = o(nm^2)$. After the ranks of a and b are determined, the query can be answered in $O(\log n + f)$ time. Thus the total computational time is $O(\log^2(nm) + f)$. □

Acknowledgements. The authors wish to thank the referees for providing helpful suggestions on the previous version of this paper and for pointing out references [11] and [9] to the authors.

References

[1] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *19th ACM Symp. Principles of Database Systems*, pages 175–186, 2000.

[2] B. Chazelle. Filtering search: A new approach to query-answering. *SIAM J. Computing*, 15(3):703–724, Aug. 1986.

[3] B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Computing*, 17(3):427–463, June 1988.

[4] B. Chazelle. Lower bounds for orthogonal range search I. The arithmetic model. *J. ACM*, 37(3):439–463, 1990.

[5] B. Chazelle. Lower bounds for orthogonal range search I. The reporting case. *J. ACM*, 37(2):200–212, 1990.

[6] J. R. Driscoll, N. Sarnak, D. Sleattor, and R. E. Tarjan. Make data structures persistent. *J. of Comput. and Syst. Sci.*, 38:86–124, 1989.

[7] M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. and Syst. Sci.*, 48:533–551, 1994.
[8] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th Annual ACM Symp. Theory of Computing*, pages 135–143, 1984.
[9] P. Gupta, R. Janardan, and M. Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization. *J. Algorithms*, 19:282–317, 1995.
[10] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Computing*, 13(2):338–355, 1984.
[11] R. Janardan and M. Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry & Applications*, 3(1):39–69, 1993.
[12] S. Lanka and E. Mays. Fully persistent B^+-trees. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 426–435, 1991.
[13] C. Makris and A. K. Tsakalidis. Algorithms for three-dimensional dominance searching in linear space. *Information Processing Letters*, 66(6):277–283, 1998.
[14] Y. Manolopoulos and G. Kapetanakis. Overlapping B^+-trees for temporal data. In *Proc. 5th Jerusalem Conf. on Information Technology*, pages 491–498, 1990.
[15] E. M. McCreight. Priority search trees. *SIAM J. Computing*, 14(2):257–276, 1985.
[16] M. A. Nascimento and J. R. O. Silva. Towards historical R-trees. In *Proc. ACM Symp. Applied Computing*, pages 235–240, Feb. 1998.
[17] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *Proc. 2000 ACM SIGMOD Int. Conf. on Management of Data*, pages 331–342, 2000.
[18] Q. Shi and J. JaJa. Fast algorithms for 3-d dominance reporting and counting. Technical Report CS-TR-4437, Institute of Advanced Computer Study (UMIACS), Unveristy of Maryland, 2003.
[19] Y. Tao and D. Papadias. Efficient historical R-trees. In *Proc. 13th Int. Conf. on Scientific and Statistical Database Management*, pages 223–232, 2001.
[20] T. Tzouramanis, Y. Manolopoulos, and M. Vassilakopoulos. Overlapping Linear Quadtrees: A spatio-temporal access method. In *Proc. of the 6th ACM Symp. on Advances in Geographic Information Systems (ACM-GIS)*, pages 1–7, 1998.
[21] T. Tzouramanis, M. Vassilakopoulos, and Y. Manolopoulos. Processing of spatio-temporal queries in image databases. In *Proc. 3rd East-European Conf. on Advances in Databases and Information Systems (ADBIS'99)*, pages 85–97, 1999.
[22] P. J. Varman and R. M. Verma. An efficient multiversion access structure. *IEEE Trans. Knowledge and Data Engineering*, 9(3):391–409, 1997.
[23] J. Vuillemin. A unifying look at data structures. *Comm. ACM*, 23(4):229–239, 1980.
[24] D. E. Willard. Examining computational geometry, van Emde Boas trees, and hashing from the perspective of the fusion three. *SIAM J. Computing*, 29(3):1030–1049, 2000.

Distribution-Sensitive Binomial Queues

Amr Elmasry

Computer Science Department
Alexandria University
Alexandria, Egypt

Abstract. A new priority queue structure is introduced, for which the amortized time to insert a new element is $O(1)$ while that for the minimum-extraction is $O(\log \overline{K})$. $\overline{K}$ is the average, taken over all the deleted elements x, of the number of elements that are inserted during the lifespan of x and are still in the heap when x is removed. Several applications of our structure are mentioned.

1 Introduction

A data structure is called distribution-sensitive if the asymptotic time bound taken by the structure to perform an operation varies according to the distribution of the input sequence. Though having upper bounds on the running time of different operations over all possible sequences, some structures may perform better for some sequences than others. This is analogous to a sorting algorithm running in $O(n \log n)$ for any sequence of length n, while performing better and running in $O(n)$ if the input is already sorted or inversely sorted.

In order to characterize such structures, several properties are introduced describing the behavior of these structures. These properties can be viewed as characterizations of distribution-sensitive behavior that give insights into the possibilities and limitations of these data structures. Relationships among such properties are introduced in [15], thus establishing a hierarchy of properties.

Following finger trees [13], splay trees [20] is the classical example of a distribution-sensitive structure. Most of the known distribution-sensitive properties were introduced either as theorems or conjectures characterizing the performance of splay trees. Examples are: The static optimality theorem, the static finger theorem, the working-set theorem (all in [20]), the sequential access theorem [11,21,22], the dequeue theorem [21], and the dynamic finger theorem [4]. Each of these theorems describes a natural class of sequences of operations, and shows that the amortized cost of performing any of these sequences on an n-node splay tree is $o(\log n)$ per operation. With a special interest with respect to our structure, we present the working-set property for search trees: The time spent to search item x in a search tree is $O(\log w_x)$, where w_x is the number of distinct items that have been accessed since x's last access. Informally, in a data structure with the working-set property accesses to items recently accessed are faster than accesses to items that have not been accessed in a while.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 103–113, 2003.
© Springer-Verlag Berlin Heidelberg 2003

Though originally formulated for the use of analyzing dictionaries, some of these properties have been applied to other structures, such as priority queues [14,15]. Applying these properties to priority queues is more robust since the heap size and contents are allowed to dynamically change, as opposed to only analyzing access operations for search trees. Iacono [14] proved that if the minimum item in a pairing heap [12] of maximum size n is to be removed, and k heap operations have been performed since its insertion, the minimum-extraction operation takes amortized time $O(\log \min(n, k))$. Because of the similarity between this property and the working-set property, we call this property the weak working-set property for priority queues.

Iacono and Langerman [16] introduced the queueish property. The queueish property implies the complementary idea, which states that an access to an item is fast if it is one of the least recently accessed items. Formally, a data structure is said to be queueish if the time to search item x is $O(\log (n - w_x))$. They showed that there is no search tree that can have this property. A priority queue is said to be queueish if the amortized cost of the insertion is $O(1)$, and the amortized cost of the minimum-extraction of x is $O(\log q_x)$, where q_x is the number of items that have been in the queue longer than x (the number of items that are inserted before x and are still in the heap at the time of x's removal). They introduced a priority queue, the *queap*, that has the queueish property.

We introduce a new distribution-sensitive priority queue structure based on the well-known binomial queues. Let K_x denote the number of elements that are inserted during the lifespan of x and are still in the heap when x is removed. Let $\overline{K}$ be the average of these K_x's over all the deleted elements. Our modified binomial queues have the property that the amortized cost of the insert operation is $O(1)$, while the amortized cost of the delete-minimum operation is $O(\log \overline{K})$. We call this property the strong working-set property, which implies the weak working-set property. We may also call this property the stack-like property, in analogy to the queueish property.

The paper is organized as follows. The next section reviews the operations of binomial queues that we use as a basic structure for our new implementation. Section 3 is an informal discussion to the problems and solutions that motivates the way we implement our structure. We describe the operations of our structure in Section 4. Some of the possible applications are given in Section 5. We conclude with an improvement that achieves better constants with respect to the number of comparisons.

2 Binomial Queues

A binomial tree [1,24] of rank (height) r is constructed recursively by making the root of a binomial tree of rank $r-1$ the leftmost child of the root of another binomial tree of rank $r-1$. A binomial tree of rank 0 is a single node. The following properties follow from the definition:

- The rank of an n-node (assume n is a power of 2) binomial tree is $\log_2 n$.

– The root of a binomial tree, with rank r, has r sub-trees each of which is a binomial tree, having respective ranks $0, 1, \ldots, r-1$ from right to left.

To represent a set of n elements, where n is not necessarily a power of 2, we use a forest having a tree of height i whenever the binary representation of the number n has a 1 in the i-th position. A binomial queue is such a forest with the additional constraint that every node contains a data value smaller than those stored in its children.

Each binomial tree within a binomial queue is implemented using the binary representation. In such an implementation, every node has two pointers, One pointing to its left sibling and the other to its leftmost child. The sibling pointer of the leftmost child points to the rightmost child to form a circular list. Given a pointer to a node, both its rightmost and leftmost children can be accessed in constant time. The list of its children can be sequentially accessed from right to left. To implement some operations efficiently each node may, in addition, contain a pointer to its parent. The roots of the binomial trees within a binomial queue are organized in a linked list, which is referred to as the *root-list*. The ranks of the roots strictly increase as the root list is traversed right to left.

Two binomial trees of the same height can be merged in constant time, by making the root of the tree that has the larger value the leftmost child of the other root. The following operations are defined on binomial queues:

Insert. The new element is added to the forest as a tree of rank 0, and successive merges are performed until there are no two trees of the same rank. (This is equivalent to adding 1 to the number in the binary representation.)

Delete-minimum. The root with the smallest element is found and removed, thus leaving all the sub-trees of that element as independent trees. Trees of equal ranks are then merged until no two trees of the same rank remain.

For an n-node binomial queue, the worst-case cost for the *insert* and the *delete-minimum* is $O(\log n)$. The amortized cost [23] for the *insert* is $O(1)$.

3 Discussion

Denote our queue structure by Q. We call the sequence of values obtained by a pre-order traversal of Q the corresponding sequence of Q and denote it by $Pre(Q)$. Our traversal gives precedence to the trees of Q in a right-to-left order. Also, the precedence ordering of the sub-trees of a given node proceeds from right to left. Hence, a newly inserted element is appended as the first element in $Pre(Q)$. At the moment when an element i is to be deleted from Q, let D_i be the number of elements preceding i in $Pre(Q)$. Our goal is to maintain the order in which the elements are input to the heap. What we are looking for is a set of operations that maintain the following property at any point of time: If we sum the D_i's over all the deleted elements and get the average, this number is upper

bounded by $\overline{K}$ (i.e. $\sum_i D_i \leq \sum_i K_i$). We call an operation that preserves this property an *inversion-preserving* operation. See [17] for the notion of inversions.

We build on the notion of binomial queues trying to obtain an implementation that is distribution-sensitive. When a new element is inserted, a single-node tree is added as the rightmost tree in the queue. The first problem we face as a result of this insertion is when two trees with the same rank are merged such that the root of the tree to the right is larger than the root of the tree to the left. As a result, the root of the tree to the right becomes the leftmost child of the root of the tree to the left. This case immediately affects the order in which the elements are input. To keep track of this order, we add an extra bit to each node of the binomial queue, and call it the *reverse bit*. When a node is linked to its left sibling, the *reverse bit* of this node is set to 1 indicating, what is called, a *rotation*. See [8,9] for a similar notion.

The next problem is with respect to the *delete-minimum* operation. When the root with the minimum value is deleted, its sub-trees are scattered according to their ranks and merged with other sub-trees in the heap, again affecting the order in which the elements are input. Our solution to this problem is to change the way the *delete-minimum* is implemented. When the root of the minimum value is deleted, one of the nodes of this tree is promoted to replace the deleted root. The heap property is maintained by a special implementation of a *heapify* operation. Two problems will pop-up as a result. The first problem is how to implement the *heapify* operation within a logarithmic time in the size of the tree. This leads to augmenting each node of the binomial queue with an extra pointer, as will be explained in details in the next section. The second problem occurs when several nodes are repeatedly deleted from a tree, causing such a tree to lose the structural properties of binomial trees. To overcome this problem, some restructuring is performed on such trees and a relaxation to the properties of the standard binomial trees is required.

We are not on the safe side yet. Consider the case when the root of a tree T of rank $r1$ is the minimum node that is required to be deleted from the heap, such that the rank of the tree to the right of T is $r2$, where $r1 \gg r2$. The time required by this *delete-minimum* operation can be implemented to be in $\Theta(r1)$, which is not comparable to $r2$ that represents the logarithm of the number of elements that precedes the deleted element in $Pre(Q)$. Our solution towards the claimed amortized cost is to perform several split operations on T. The split operation is in a sense the opposite of the merge operation. A binomial tree is split into two binomial trees, by cutting the leftmost sub-tree of the given tree and adding it to the *root-list* either to the left or to the right of the rest of the tree, depending on the value of the *reverse bit*. As a result, there will be, instead of T, several trees whose ranks are in the range from $r1$ to $r2$. The idea is to reduce such gaps among the ranks of adjacent nodes in the *root-list* in order to reduce this extra cost for the subsequent *delete-minimum* operations.

Having two trees of the same rank is not permitted in the standard implementation of binomial queues. In our new structure, we allow the existence of at most two trees of any rank. This is similar to using a redundant binary rep-

resentation. The redundant number system has the base two but in addition to using zeros and ones we are allowed to use twos as well. Any number can be represented using this number system. See [3,7,19]. The usage of a redundant number representation is crucial to achieve the required bounds. Consider the usage of the normal binary number representation instead, with the following nasty situation. Suppose that the size n of the entire structure is one less than a power of two, and suppose that we have a long alternating sequence of *insert* and *delete-minimum*, such that every time the inserted element is the smallest element that will be immediately deleted afterwards. Each of the *insert* operations requires $\log n$ merges. The claimed bounds for our structure imply that both operations must be implemented in constant time, which is not achievable with the normal binary number representation. It is the savings of the carry operations in the redundant binary representation that make our data structure more efficient, achieving the claimed bounds.

4 The Data Structure

We introduce the new basic structure, which we call *relaxed binomial trees*, as an alternative to binomial trees.

Relaxed binomial trees. The children of the root of a relaxed binomial tree of rank r are relaxed binomial trees. There are one or two children having each of the respective ranks $0, 1, \ldots, r-1$. The number of these children is, therefore, between r and $2r$ inclusive. The ranks of these children form a non-decreasing sequence, from right to left. A relaxed binomial tree with rank 0 is a single node.

Lemma 1. *The rank of an n-node relaxed binomial tree is at most* $\log_2 n$.

Proof. The fact that a single node tree has rank 0 establishes the base case. Let r be the rank of an n-node relaxed binomial tree. By induction, $n \geq 1 + 2^0 + 2^1 + \ldots + 2^{r-1} = 2^r$. □

We are now ready to describe our data structure. We use relaxed binomial trees in place of the traditional binomial trees. Our binomial queue may have up to two (0, 1, *or* 2) relaxed binomial trees with the same rank. The order of the roots of the trees is important within the *root-list*. The ranks of these roots form a non-decreasing sequence from right to left. The following procedures are used to perform the priority queue operations:

Heapify. Given a relaxed binomial tree T of rank r, such that the heap property is valid for all the nodes except for the root. The question is how to restore this property. Applying the standard *heapify* operation will do, while maintaining the *inversion-preserving* property. Recall that the *heapify* operation proceeds by

finding the node, say x, with the smallest value among the children of the root and swapping its value with that of the root. This step is repeated with the node x as the current root, until either a leaf or a node that has a value smaller than or equal to all the values of its children is reached. To show that the *heapify* operation is *inversion-preserving*, consider any two elements $x_i, x_j \in Pre(T)$, where $i < j$. If these two elements were swapped during the *heapify* operation, then $x_i > x_j$. Since x_i precedes x_j in $Pre(T)$, we conclude that this swap decreases the number of inversions.

It remains to investigate how the *heapify* operation is implemented. Finding the minimum value within a linked list requires linear time. This may lead to an $O(r^2)$ time for the *heapify* operation. We can do better, however, by maintaining with every node an extra pointer that points to the node with the smallest value among all its right siblings, including itself. We call this pointer, the pointer for the prefix minimum (*pm*). The *pm* pointer of the leftmost child of a node will, therefore, point to the node with the smallest value among all the children of the parent node. To maintain the correct values in the *pm* pointers, whenever the value of a node is updated all the *pm* pointers of its left siblings, including itself, have to be updated. This is accomplished by proceeding from right to left; the *pm* pointer of a given node x is updated to point to the smaller of the value of x and the value of the node pointed to by the *pm* pointer of the right sibling of x. A *heapify* at a node with rank $r1$ reduces to a *heapify* at its child with the smallest value whose rank is $r2$ after $O(r1 - r2)$ time and at most $3(r2 - r1)$ comparisons. The time spent by the *heapify* on T is, therefore, $O(r)$.

If we are concerned with constant factors, the number of comparisons can still be reduced as follows. First, the path from the root to a leaf, where every node has the smallest value among its siblings, is determined by utilizing the *pm* pointers. No comparisons are required for this step. Next, the value at the root is compared with the values of the nodes of this path bottom up, until the correct position of the root is determined. The value at the root is then inserted at this position, and all the values at the nodes above this position are shifted up. The *pm* pointers of the nodes whose values moved up and those of all their left siblings are updated. The savings are due to the fact that at each level of the queue (except possibly for the level of the final destination of the old value of the root) either a comparison with the old value of the root takes place or the *pm* pointers are updated, but not both. Then, the number of comparisons is at most $2r$. See [10] for a similar description to this procedure.

Merge. Given two relaxed binomial trees of the same rank r whose roots are adjacent in the *root-list* of the binomial queue, the two trees can be merged into one tree of rank $r + 1$ by making the root with the larger value the leftmost child of the root of the other tree. If the right sibling is linked to its left sibling its *reverse bit* is set to 1, otherwise the *reverse bit* of the linked node (the left sibling) is set to 0. The *pm* pointer of the linked node is updated. The roots of the two trees are removed from the *root-list*, and the root of the new tree is inserted in their position. The *merge* operation takes constant time.

Insert. The new element is added to the forest as the rightmost tree whose height (rank) is 0, and successive merges are performed until there are no three trees of the same rank. The merging must be done while maintaining the ordering of the elements. More specifically, if there are three trees with the same rank, the two leftmost trees are merged and the root of the resulting tree replaces the roots of these two trees in the *root-list*.

Split. A relaxed binomial tree T of rank r can be split into two trees as follows. The first tree is the sub-tree of the leftmost child of the root of T, and the second tree is the rest of T. The rank of the first tree is $r-1$, and the rank of the second tree is either r or $r-1$ (depending on the rank of its current leftmost child). The *reverse bit* of the root of the first tree is checked. If this bit was set to 1 (as a result of a previous *merge* operation), we make the root of the first tree the right sibling of the root of the second tree, otherwise we make the root of the first tree the left sibling of the root of the second tree. The two roots are inserted in place of the root of T in the *root-list*. The *split* operation takes constant time, and no comparisons are needed.

Promote. Given a relaxed binomial tree T with a deleted root of rank r, the purpose of this procedure is to promote a node to replace the root, while maintaining the structural properties of relaxed binomial trees together with the *inversion-preserving* property. The procedure starts by promoting the single node representing the rightmost child, making it the new root of the binomial tree. As a result, there may become no tree of rank 0. To maintain the properties of relaxed binomial trees, assume that before performing the following iterative step there is no child of T with rank i. We call the following iterative step *gap(i)*. The rightmost tree with rank $i+1$ is split, and three cases may take place depending on the ranks of the resulting two trees:

1. The left tree has rank $i+1$ and the right tree has rank i: This case is terminal.
2. The left tree has rank i and the right tree has rank $i+1$: The right tree is split into two trees each with rank i (this is the only possibility for this second split). Now, there are three trees each with rank i. The two leftmost trees are merged into one tree with rank $i+1$. This case is also terminal.
3. Both of the resulting two trees have rank i: If there was another tree of rank $i+1$, the iterative step terminates. If there was only one tree of rank $i+1$, there is none after the split. The iterative step is performed with no trees of rank $i+1$ (i.e. call *gap(i+1)*).

If the iterative step is repeated until there is no tree of rank $r-1$, the iterative step terminates and the promoted root is assigned a rank of $r-1$. Otherwise, the promoted root is assigned a rank of r.

To maintain the *pm* pointers of the children of the promoted root without performing extra comparisons, the following trick is made. Before the *promote*, if the value of the single node representing the rightmost child is smaller than the value of its left sibling, the two nodes are swapped. As a result the *pm* pointers of the other children will not need to be changed. The time spent by

the *promote* is $O(r)$, and the number of comparisons performed is $O(1)$. After the *promote*, a *heapify* must be called to maintain the heap property for the promoted root.

Fill-gaps. Given a relaxed binomial tree T of rank $r1$ such that the rank of the tree to its right in the queue is $r2$, where $r1 > r2 + 1$, several *split* operations are performed on T. A tree of rank i can be split into two or three trees of rank $i - 1$ by performing one or two *split* operations, respectively. While the ranks of the trees resulting from the split are greater than $r2$, a *split* is repeatedly performed on the right tree among these trees. As a result, there will be at most one tree of rank $r1$ (if there was two before this procedure), one or two trees of each of the ranks $r1 - 1, r1 - 2, \ldots, r2 + 2$, and two or three trees of rank $r2 + 1$. The possibility of having three trees of the same rank violates the rules. If this happens, the leftmost two trees of rank $r2 + 1$ are merged to form a tree of rank $r2 + 2$. This violation may propagate while performing such merge operations, until there are no three trees of the same rank; a case that is insured to be fulfilled if the result of the merge is a tree of rank $r1$. As a final result of this procedure, there will be at most two trees of rank $r1$, one or two trees of each of the ranks $r1 - 1, r1 - 2, \ldots, r2 + 1$. The time spent by the *fill-gaps* procedure is $O(r1 - r2)$.

Maintain-minimum. After deleting the minimum node we need to keep track of the new minimum. Checking the values of all the roots leads to a $\Theta(\log n)$ cost for the *delete-minimum* operation, where n is the size of the queue. The solution is to reuse the idea of the prefix minimum pointers. A pointer is used with every root, in the *root-list*, that points to the node with the smallest value among the roots to its left, including itself. We call these pointers the suffix minimum (*sm*) pointers of the roots. The *sm* pointer of the rightmost root points to the root with the minimum value. After deleting the minimum node, maintaining the affected pointers (the pointers to the right of the deleted root) can be done from left to right. If the rank of the deleted root is r, the number of the affected pointers is at most $2(r + 1)$ (there may be two trees of each possible rank value). This process is crucial to achieve the claimed bounds for the heap operations. A more efficient procedure to implement this step would improve the constants of the heap operations, as explained in Section 6.

Delete-minimum. First, the root of the tree T with the minimum value is removed. A *promote* is performed on T, followed by a *heapify*. As a result of the *promote*, the rank of T may decrease by one, and there may be three trees with the same rank. In this case, a *merge* operation is performed on T and the tree to its right, restoring the property of having at most two trees with the same rank. Next, a *fill-gaps* is performed on the tree T. The final step is to perform a *maintain-minimum* to keep track of the new minimum.

Theorem 1. *Starting with an empty distribution-sensitive binomial queue, the amortized cost of the insert operation is* $O(1)$*, and that of the delete-minimum is* $O(\log \overline{K})$*. The worst-case cost of these operations is* $O(\log n)$*.*

Proof. The worst-case cost follows from the way the operations are implemented, the fact that the rank of any tree is $O(\log n)$, and that the number of trees in the heap is $O(\log n)$.

We use a potential function [23] to derive the amortized bounds. For each possible rank value for the roots of the trees in the queue there is either $0, 1,$ *or* 2 trees. After the i^{th} operation, let N_0^i be the number of rank values that is not represented by any trees, N_1^i be the number of rank values that is represented by one tree, and N_2^i be the number of rank values that is represented by two trees. Let Φ^i be the potential function, such that $\Phi^i = c_1 N_0^i + c_2 N_2^i$, where c_1 and c_2 are constants to be determined. The value of Φ^0 is 0.

First, assume that the operation $i+1$ is an *insert* operation that involved t merges. If as a result of this insertion two trees with the same rank are merged, then there should have been two trees with this rank before the insertion and only one remains after the insertion. This implies that $N_2^{i+1} - N_2^i \leq -t+1$ and $N_0^{i+1} - N_0^i \leq 0$. The amortized cost is bounded by $O(t) - c_2 t + c_2$. By selecting c_2 greater than the constant involved in the O() notation in this relation, the amortized cost of the insertion is c_2.

Next, assume that the operation $i+1$ is a *delete-minimum* performed on the root of a tree T of rank $r1$. The actual cost is $O(r1)$. Let $r2$ be the rank of the tree to the right of T before the operation is performed. The number of nodes of this tree is upper bounded by D_m, where m is the number of the current *delete-minimum* operation (D_m is the number of elements preceding this deleted element in $Pre(Q)$ at this moment). As a result of the *fill-gaps* procedure: $N_0^{i+1} - N_0^i \leq -(r1 - r2 - 2)$ and $N_2^{i+1} - N_2^i \leq r1 - r2 - 1$. Hence, the amortized cost is bounded by $O(r1) - (c_1 - c_2)(r1 - r2 - 1) + c_1$. By selecting c_1, such that $c_1 - c_2$ is greater than the constant in the O() notation in this relation, the amortized cost of the *delete-minimum* is $O(r2)$ which is $O(\log D_m)$. It follows that the cost of these m *delete-minimum* operations is $O(\sum_{i=1}^{m} \log D_i)$. Jensen's inequality implies $\sum_{i=1}^{m} \log D_i \leq m \log(\frac{1}{m} \sum_{i=1}^{m} D_i)$. Since all our procedures have the *inversion-preserving* property, then $\frac{1}{m} \sum_{i=1}^{m} D_i \leq \overline{K}$. It follows that the amortized cost of the *delete-minimum* operation is $O(\log \overline{K})$. □

5 Applications

We expect our data structure to be useful for several applications, from which we mention some examples:

Adaptive sorting. Given a sequence of n elements, a distribution-sensitive binomial queue is built in $O(n)$ by repeatedly inserting these elements. By repeatedly deleting the minimum node from the queue we get a sorted sequence of the input. The time spent to sort such a sequence is $O(n \log \overline{K})$. If the elements

are inserted in reverse order, $\overline{K}$ will be the average number of inversions in the input sequence, and our algorithm is optimal [13,17,18]. Our heap structure is more flexible since it allows interleaved insertions and minimum-deletions. Hence, it can be used in on-line adaptive sorting and order statistics.

Geometric applications. There are several geometric applications that require the usage of a heap structure. For example, in the *sweep-line* paradigm [5] the usage of a priority queue is essential. Our heap may be used if the events to be handled follow some specific distributions; a case where deleting the minimum of an n-node heap may require $o(\log n)$. The existence of some problems, where the geometric nature implies that the expected time that the inserted events spend in the heap before being deleted is small, needs to be investigated.

Discrete event simulation. e.g. future event set algorithms. In such applications a list of future events is to be maintained, and at every step the next occurring event in the list is processed inserting some new events. These events may follow some probability distribution, and hence their processing may be faster using our structure. For a survey on discrete event simulation, see [6].

6 Improving the Constants

The constant factor of the number of comparisons of the *heapify* in the $O(\log \overline{K})$ is 2, and that of the *maintain-minimum* is 2, for a total of at most $4 \log_2 \overline{K} + O(1)$ comparisons per *delete-minimum*. Next, we sketch the way to implement *maintain-minimum* in $O(\log \log \overline{K})$, achieving an overall bound of $2 \log_2 \overline{K} + O(\log \log \overline{K})$ for the number of comparisons. The roots of the trees are kept in a set of heaps, such that all the nodes whose ranks are in the range from 2^i to $2^{i+1} - 1$, for possible integers i, are kept in the same heap. These heaps are arranged in an increasing order of their sizes, maintaining *sm* pointers from right to left (The constant in the smaller terms may even be improved by having a hierarchy of levels of heaps instead of using the *sm* pointers at this level.). Deleting the minimum among these heaps takes $O(\log r)$ if the rank of the deleted node is r, implying a bound of $O(\log \log \overline{K})$. We need to maintain this set of heaps whenever the roots of the main trees change. This requires inserting and deleting such nodes in and from the heaps whenever necessary. Using our original analysis, it follows that the number of the main operations bounds the number of such heap operations. Our goal is to insert or delete an element in these heaps in $O(1)$. We can use any of the heap implementations that perform insert in $O(1)$ and delete-minimum in $O(\log n)$. We use a method of delayed deletions. Whenever a node needs to be deleted from this second level of heaps it is marked. Before inserting a new node, it is first checked if it already exists as a marked node, and hence unmarking it. Whenever the number of the marked nodes reaches half the total number of nodes in one of these heaps, this heap is rebuilt getting rid of the marked nodes. Achieving an $O(1)$ is possible for the deletion because of the

nature of the application, which insures that a marked node will never become the minimum of a heap before being reinserted.

References

1. M. Brown. *Implementation and analysis of binomial queue algorithms.* SIAM J. Comput. 7 (1978), 298–319.
2. M. Brown and R. Tarjan. *Design and analysis of data structures for representing sorted lists.* SIAM J. Comput. 9 (1980), 594–614.
3. S. Carlsson and J. I. Munro. *An implicit binomial queue with constant insertion time.* 1st SWAT. In LNCS 318 (1988), 1–13.
4. R. Cole. *On the dynamic finger conjecture for splay trees. Part II: The proof.* SIAM J. Comput. 30 (2000), 44–85.
5. M. De Berg, M. Kreveld, M. Overmars and O. Shwarzkopf. *Computational geometry-algorithms and applications.* Springer-Verlag, (1997)
6. L. Devroye. *Nonuniform random variate generation.* Springer-Verlag, (1986).
7. E. Doberkat. *Deleting the root of a heap.* Acta Informatica, 17 (1982), 245–265.
8. R. Dutton. *Weak-Heapsort.* BIT, 33 (1993), 372–381.
9. S. Edelkamp and I. Wegener. *On the performance of weak-heapsort.* STACS. In LNCS 1770 (2000), 254–260.
10. A. Elmasry. *Priority queues, pairing and adaptive sorting.* 29th ICALP. In LNCS 2380 (2002), 183–194.
11. A. Elmasry. *A new proof for the sequential access theorem for splay trees.* WSES, ADISC. In Theoretical and Applied Mathematics, (2001) 132–136.
12. M. Fredman, R. Sedgewick, D. Sleator, and R. Tarjan. *The pairing heap: a new form of self_adjusting heap.* Algorithmica 1,1 (1986), 111–129.
13. L. Guibas, E. McCreight, M. Plass and J. Roberts. *A new representation of linear lists.* ACM STOC 9 (1977), 49–60.
14. J. Iacono. *Improved upper bounds for pairing heaps.* 7th SWAT. In LNCS (2000), 32–45.
15. J. Iacono. *Distribution sensitive data structures.* Ph.D. thesis, Rutgers University. (2001).
16. J. Iacono and S. Langerman. *Queaps.* International Symposium on Algorithms and Computation. In LNCS 2518 (2002) 211–218.
17. D. Knuth. *The Art of Computer Programming. Vol III: Sorting and Searching.* Addison-wesley, second edition (1998).
18. H. Mannila. *Measures of presortedness and optimal sorting algorithms.* IEEE Trans. Comput. C-34 (1985), 318–325.
19. Th. Porter and I. Simon. *Random insertion into a priority queue structure.* IEEE Trans. Software Engineering, 1 SE (1975), 292–298.
20. D. Sleator and R. Tarjan. *Self-adjusting binary search trees.* J. ACM 32(3) (1985), 652–686.
21. R. Sundar. *On the deque conjecture for the splay algorithm.* Combinatorica 12 (1992), 95–124.
22. R. Tarjan, *Sequential access in splay trees takes linear time.* Combinatorica 5 (1985), 367–378.
23. R. Tarjan. *Amortized computational complexity.* SIAM J. Alg. Disc. Meth. 6 (1985), 306–318.
24. J. Vuillemin. *A data structure for manipulating priority queues.* Comm. ACM 21(4) (1978), 309–314.

Optimal Worst-Case Operations for Implicit Cache-Oblivious Search Trees

Gianni Franceschini and Roberto Grossi

Dipartimento di Informatica, Università di Pisa, via Buonarroti 2, 56127 Pisa, Italy
Fax +39-050-2212726, {francesc,grossi}@di.unipi.it

Abstract. We close an open issue on dictionaries dating back to the sixthies. An array of n keys can be sorted so that searching takes $O(\log n)$ time. Alternatively, it can be organized as a heap so that inserting and deleting keys take $O(\log n)$ time. We show that these bounds can be simultaneously achieved in the worst case for searching *and* updating by suitably maintaining a permutation of the n keys in the array. The resulting data structure is called implicit as it uses just $O(1)$ extra memory cells beside the n cells for the array. The data structure is also cache-oblivious, attaining $O(\log_B n)$ block transfers in the worst case for any (unknown) value of the block size B, without wasting any single cell of memory at any level of the memory hierarchy.

1 Introduction

In this paper we consider the classical dictionary problem in which a set of n distinct keys $a_1, a_2, \ldots, a_n$ is maintained over a total order, where the only operations allowed on the keys are reads/writes and comparisons using the standard RAM model of computation [1]. The dictionary supports the operations of searching, inserting and deleting an arbitrary key x.

Implicit dictionaries solve the problem by maintaining a plain permutation of $a_1, a_2, \ldots, a_n$ to encode the data structures [17]. When employed in this context, heaps [19] have the drawback of requiring $O(n)$ time for searching, while inserting or deleting a key in the middle part of sorted arrays may take $O(n)$ time [15]. A longstanding question is whether there exists an organization of the keys in an array of n cells combining the best qualities of sorted arrays and heaps, so that each operation requires $O(\log n)$ time. Previous work since the sixties did not achieve polylog time in both searching and updating. We refer the reader to [10] for a history of the problem.

The first milestone in this direction is the implicit AVL tree in the eighties, showing for the first time that polylog time is possible, namely $O(\log^2 n)$, by encoding bits in chunks of $O(\log n)$ permuted keys [16]. It was conjectured a bound of $\Theta(\log^2 n)$ because $\Theta(\log n)$ pointers of $\Theta(\log n)$ bits are decoded/encoded in the worst case to execute an operation in the implicit AVL tree.

The second milestone is the implicit B-tree, attaining $O(\log^2 n/\log n \log n)$ time [11]. Notwithstanding the small improvement in main memory, this recent

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 114–126, 2003.
© Springer-Verlag Berlin Heidelberg 2003

result disproved the conjecture of the eighties, making viable the possibility of getting a bound of $O(\log n)$. The implicit B-tree uses nodes of relatively large fan-out that are augmented with a permuted directory to support fast searching inside each node. For a *known* block size $B = \Omega(\log n)$, it supports the operations in $O(\log_B n)$ block transfers like regular B-trees, while scanning r contiguous elements requires $O(\log_B n + r/B)$ block transfers.

The subsequent results leading to the flat implicit tree [9] probably represents the third milestone. It is the first implicit data structure with optimal $O(\log n)$ time for searching and $O(\log n)$ amortized time for updating. Specifically, the result of $O(\log n \log\log n)$ in [7] uses exponential trees of height $O(\log\log n)$, exploiting in-place algorithms to amortize the bounds and introducing different kinds of chunks of $O(\log n)$ contiguous keys to delay the expensive reorganizations of the updates. The result in [10] obtains $O(\log n)$ amortized time with a two-layer tree of constant height (except very few cases), adapting the redistribution technique of [3,14] to the implicit model. Its cache-oblivious evolution in [8] attains the amortized bounds of $O(\log_B n)$, where the cache-obliviousness of the model lies in the fact that the block transfer size B is unknown to the algorithms operating in the model [13]. The top layer uses a van Emde Boas permutation [13] of the keys as a directory, and the bottom layer introduces compactor zones to attain cache-obliviousness. Compared to implicit B-trees, the update bounds are amortized and scanning is not optimal. On the other hand, achieving an optimal scanning is still an open problem in explicit cache-oblivious dictionaries even with amortized update bounds of $O(\log_B n)$. The implicit B-tree attains this goal with worst-case bounds as it is aware of the block size B.

In this paper we focus on the worst-case complexity of implicit dictionaries. The best bound is that of $O(\log^2 n/\log\log n)$ with the implicit B-trees. For explicit cache-oblivious data structures, the best space occupancy in [5] is $(1+\epsilon)n$ cells for any $\epsilon > 0$ with an $O(1 + r/B)$ scanning cost for r keys, but the update bounds are amortized, whereas the worst-case result in [4] uses more space. Here, we propose a new scheme for implicit data structures that takes $O(\log n)$ time and $O(\log_B n)$ block transfers in the worst case for any unknown B, as in the cache-oblivious model. The optimality of our data structure is at any level of the memory hierarchy as it uses just $n + O(1)$ cells. This closes the problem of determining a permutation of the keys in an array, so that both searching and updating are logarithmic in the worst case as explicit dictionaries.

We introduce new techniques to design our data structures. First, we use some spare keys and some chunks, called filling chunks, to allocate nodes of the tree in an implicit way. When we actually need a chunk, we replace the filling chunk with the routing chunk, and relocate the filling chunk. We also design a bottom layer that can be updated very quickly. We reuse techniques from previous work, but we apply them in a novel way since we have to perform the memory management of the keys in the array. Consequently, our algorithms are slightly more involved than algorithms for explicit data structures, as the latter assume to have a powerful memory manager performing the "dirty" work for them in a transparent way. Instead, we have to carefully orchestrate data

movement as we cannot leave empty slots in any part of the array. In the full paper, we show how to extend to our data structure to a multiset, namely, containing some repeated keys.

The paper is organized as follows. In Section 2, we review some basic techniques that we apply to implicit data structures. We then describe our main data structure in two parts, in Section 3–4, putting them together in Section 5 for the sketch of the final analysis of the supported operations.

2 Preliminary Algorithmic Tools

We encode data by a pairwise (odd-even) permutation of keys [16]. To encode a pointer or an integer of b bits by using $2b$ distinct keys $x_1, y_1, x_2, y_2, \ldots, x_b, y_b$, we permute them in pairs x_i, y_i with the rule: if the ith bit is 0, then $\min\{x_i, y_i\}$ precedes $\max\{x_i, y_i\}$; else, the bit is 1 and $\max\{x_i, y_i\}$ precedes $\min\{x_i, y_i\}$.

Adjacent keys in the array are grouped together into *chunks*, where each chunk contains $O(k)$ (pairwise permuted) keys encoding a constant number of integers and pointers, each of $b = O(\log n)$ bits. The keys in any chunk belong to a certain interval of values, and the chunks are pairwise disjoint when considered as intervals, thus yielding a total order on any set of the chunks. We introduce some terminology on the chunks to clarify their different use. We have *routing* chunks that help us in routing the search of individual keys, and *filling* chunks that provide a certain flexibility in filling the entries of the array in that we can keep them in no particular order. Access to the latter is via the routing chunks. The number of keys in a chunk is fixed to be either k or $k - \alpha$ for a certain constant $\alpha > 1$, which is clear from the context. We also use a set of *spare* keys that can be individually relocated and referenced for a finer level of flexibility in filling the array, associating $O(1)$ spare keys to some chunks. When considered as intervals, the chunks include the spare keys although the latter physically reside elsewhere in the array.

Our algorithms employ some powerful tools to achieve their worst-case and cache-oblivious bounds. One tool is Willard's algorithm [18] and its use in Dietz-Sleator lists [6]. Suppose we have an array Z of N slots (for a fixed N) storing a *dynamic* set S of $n \leq N$ objects, drawn from a totally ordered universe. At any time, for every pair of object $s_1, s_2 \in S$, if $s_1 < s_2$ then the slot storing s_1 precedes that storing s_2. The data structure proposed by Willard in [18] achieves this goal using a number of $O(\log^2 N)$ arithmetical operations, comparisons and moves, in the worst case, for the insertion or the deletion of an individual object in Z. In our use of Willard's scheme, the routing chunks play the role of the full slots while the filling chunks that of the empty slots. It is possible to insert a new routing chunk (thus replacing a filling chunk that goes elsewhere) and delete a routing chunk (putting in its place a filling chunk taken somewhere). These operations have to maintain the invariant of Willard's scheme according to the total order of the routing chunks stored in the slots. Since the slots are of size $O(k)$ in our case, the bounds of Willard's scheme have to multiplied by a factor of $O(k)$ time or $O(k/B)$ block transfers to insert or delete a routing chunk.

Another useful tool is the van Emde Boas (VEB) layout of Prokop [13]. Given a complete binary search tree T with $n = 2^i - 1$ nodes, the VEB layout of T allows for searching with $O(\log_B n)$ block transfers in a cache-oblivious fashion. Brodal et al. [5] describe how to avoid pointers in the VEB layout, still using extra memory. Franceschini and Grossi [9] show how to make the VEB layout implicit in the form of a VEB permutation of the n keys.

The last tool is for memory management of nodes of variable size with compactor lists [11] and compactor zones [9]. Nodes in the design of implicit data structures are sets of permuted keys that should be maintained as contiguous as possible. For this, nodes of the *same* size are kept together in a segment of contiguous cells (the compactor zone) or in a linked list of fixed size allocation units (the compactor list). Their use make possible to avoid to create empty cells during the operations since the nodes of the same size are collected together. However, when a node changes size, we have to relocate the node from one compactor zone (or list) to another. Since we want to achieve worst-case bounds, we use compactor lists for nodes of size $\Theta(\sqrt{\log n})$ since they are efficient with small size nodes, and compactor zones for nodes of size $\Theta(\log n)$ since they can be incrementally maintained still permitting searching. For larger nodes, we use a different approach described in Section 4.

3 Districts of Chunks

The array of n keys is partitioned into $O(\log \log n)$ portions as in Frederickson [12], where the pth portion stores 2^{2^p} keys, except the last portion, which can store less keys than expected. Inserting or deleting a key in any portion can be reduced to performing the operation (possibly with a different key) in the last portion, while searching is applied to each portion. Achieving a logarithmic cost in each portion sums up to $O(\log_B n)$ block transfers, which is the final cost of the supported operations.

In the rest of the paper we focus on the last portion A of the array, assuming without loss of generality that A is an array of n keys, where $N = 2^{2^p}$ is the maximum size of A for some given integer $p > 0$ and $n \leq N$ is sufficiently large to let us fix $k = \Theta(\log N) = \Theta(\log n)$. (The implicit model assumes that A occupies just $n + O(1)$ cells and that it can be extended to the right one cell at a time up to $n = N$ cells.) This condition is guaranteed using Frederickson's partitioning.

The first $O(\log N)$ keys of A form a preamble encoding some bookkeeping information for A. We partition the rest of A into two parts, the layer $\mathscr{D}$ of the *districts* and the layer $\mathscr{B}$ of the *buckets*. We defer the discussion of layer $\mathscr{B}$ to Section 4. Here, we focus on the districts in layer $\mathscr{D}$ in which we use chunks of size $k - \alpha$ for a certain constant $\alpha > 1$. We partition the initial part of layer $\mathscr{D}$ into a number (at most logarithmic) of consecutive districts $D_0, D_1, \ldots$, so that each D_i contains 2^i chunks and $\Theta(2^i)$ spare keys according to the invariants we give next. Here, we denote the zone of $\mathscr{D}$ to the right of the districts by F (see Figure 1).

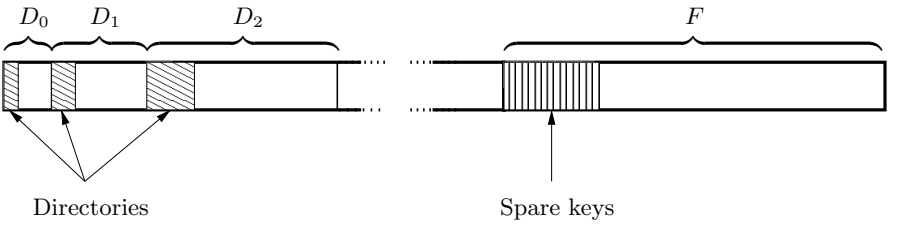


Fig. 1. The districts in layer $\mathcal{D}$.

1. The chunks in layer $\mathcal{D}$ are routing chunks and filling chunks, each with $\alpha = \Theta(1)$ spare keys associated. The routing chunks occur only in the districts $D_0, D_1, \ldots$, while the filling chunks can occur anywhere in $\mathcal{D}$ (i.e., both in $D_0, D_1, \ldots$ and in F).
2. The total left-to-right sequence of routing chunks among all districts in $\mathcal{D}$ is in order, while the filling chunks are not in any particular order. Given any two routing chunks (as closest as possible), the sequence of filling chunks can be arbitrarily long.
3. With each routing chunk c, there are $\Theta(1)$ filling chunks associated. Their number can range between two suitable constants, so that the overall number of filling chunks in F is at least 2^{i+1}. The filling chunks associated with c are the nearest to c in the total order of the chunks, and the pointers to them are encoded in c.
4. The first $\Theta(2^i)$ positions of each district D_i are initially occupied by some spare keys associated with the filling chunks in $\mathcal{D}$. We require that, at any time, the number of these positions is a multiple of the chunk size. The keys in these positions form a *directory* for quickly searching the routing chunks in D_i.
5. The routing chunks in D_i are to the right of their directory, and the first chunk c immediately after the directory is always routing. We maintain the smallest key of c as a spare key that is stored in the preamble of A. In this way, we can discover in which district to search by first reading $O(\log n)$ adjacent spare keys in that preamble.
6. The rest of the spare keys are in F, at the beginning (a multiple of the chunk size) and at the end (any number of them). We incrementally move the spare keys from the end of F to the beginning of F (or vice versa), when adding (or removing) routing chunks in D_i, the rightmost district in $\mathcal{D}$. When the number of routing chunks in $\mathcal{D}$ is sufficiently large, the keys at the beginning of F are already organized to create D_{i+1}, thus shortening F and preparing for D_{i+2} (if any). An analogous situation occurs when D_i has no more routing chunks, and D_{i-1} becomes the rightmost district.

How to search a key in $\mathcal{D}$. The organization mentioned in points 1–6 is not yet suitable for searching. As mentioned in point 5, we can identify efficiently in which district, say D_i, we must go on searching. Once we identify the correct

routing chunk in D_i, it is a simple task to examine its associated filling chunks. Consequently, it suffices to show how to search a key in D_i, so that a routing chunk can be identified with $O(\log_B n)$ block transfers.

With this goal in mind, we set up the directory of D_i following the VEB permutation mentioned in Section 2. Since this scenario is well exploited and studied in previous work [9], we do not describe the one-to-one mapping between the VEB permutation in the directory and the $2^i - 1$ nodes of a complete binary tree. In turn, the nodes of the tree are in one-to-one correspondence with the $2^i - 1$ chunks in D_i. Although the tree is not yet a search tree, we can activate the search path in it for each routing chunk in D_i. At the beginning, the directory is made up of spare keys from filling chunks and no search path is active.

Given a routing chunk c, let u be the corresponding node in the complete binary tree encoded by the VEB permutation. Since c contains $\Theta(\log n)$ keys and the chunks are disjoint as intervals, we can exchange the smallest keys in c with the spare keys found in the upward path from u. The general property we maintain is that the exchanged keys of c must guide the search towards u when searching keys that fall inside c as an interval. In other words, the paths activated for all the routing chunks form a search tree. The nodes along these paths contain keys taken from the routing chunks, while the rest of the keys in the directory are spare keys from the filling chunks. The routing chunks host temporarily the spare keys that they exchanged in the directory. As a result, the spare keys hosted inside the routing chunk c can be retrieved from the pointers encoded in their filling chunks. Vice versa, the keys in c that are currently in the directory stay along some of the nodes in the upward path from u, and they can be retrieved with a cost of $O(\log_B n)$ block transfers.

With this organization of the directory, searching is a standard task with the VEB permutation as each node have now a routing key when needed. What can be observed is that we actually exchange keys in pairs to encode a flag bit indicating whether u has associated spare keys or keys from a routing chunk. The rest of the searching in the VEB permutation is unchanged.

Lemma 1. *Any key x can be searched in $\mathcal{D}$ with $O(k/B+\log_B n)$ block transfers, identifying the (routing or filling) chunk that contains x.*

How to update $\mathcal{D}$. Within a district D_i, we focus on how to maintain its organization of the keys when the routing chunks are added or removed. The first routing chunk in D_{i+1} is to the immediate right of the directory, in which case we exchange $O(\log n)$ keys with the directory. For the following routing chunks, we apply Willard's algorithm to D_i (without its directory) as described in Section 2:

- The number of routing chunks in each district D_i is dictated by Willard's algorithm. In particular, if D_i is the last district, each of $D_0, D_1, \ldots, D_{i-1}$ has the maximum number of routing chunks according to Willard's algorithm, and the rest are filling chunks. Instead, D_i is not necessarily maximal.
- The structural information needed by Willard's algorithm can be entirely encoded and maintained in layer $\mathcal{D}$.

Willard's scheme preserves the distribution of routing chunks among the filling chunks in $O(\log^2 n)$ steps. In each step, it relocates a routing chunk c from one

position to another in D_i by exchanging it with a filling chunk c'. This step requires exchanging the keys of the two chunks incrementally, then performing a search to locate and re-encode the incoming pointer to c'.

However this alone does not guarantee searchability as we need to update the VEB permutation. We therefore divide the step in further $O(\log n)$ substeps that essentially remove c and its search path in the directory and re-insert it into another position, along with its new search path. Specifically, in each substep we retrieve one of the $O(\log n)$ keys of c that are in the directory and put it back in c by exchanging it with the corresponding spare key temporarily hosted in c (note that each spare key requires a search). Next, we exchange c with c', and propagate the same exchange in the VEB permutation of the directory. We then run further $O(\log n)$ substeps to trace the path for the new position of c and exchange its keys so that it is now searchable in the directory. During the substeps, c is the only chunk not searchable in D_i. But we can encode a pointer to it in the preamble, so that searching treats c as a special case. When the task for c is completed, Willard's scheme takes another routing chunk, which becomes the new special case. In summary, each of the $O(\log^2 n)$ steps in Willard's scheme can be divided into further $O(\log n)$ substeps, each costing $O(k + \log n) = O(\log n)$ time and $O(k/B + \log_B n) = O(\log_B n)$ block transfers. It is crucial noting that after each substep, we can run the search as stated in Lemma 1 plus the special case for the current c.

When inserting and deleting routing chunks in a district D_j, for $j < i$, we perform the same steps as in D_i. However we must preserve the property that the number of routing chunks is maximal. This means inserting/deleting a routing chunk also in each of $D_{j+1}, \ldots, D_i$. Since there are $O(\log n)$ districts, we have an extra logarithmic factor in the number of substeps for the districts in the entire layer $\mathcal{D}$.

Theorem 1. *Layer $\mathcal{D}$ can be maintained under insertion and deletion of single routing chunks and filling chunks by performing no more than $O(\text{polylog}(n))$ incremental substeps, each requiring $O(\log n)$ time and $O(\log_B n)$ block transfers. After executing each single substep, searching a key for identifying its chunk takes $O(\log n)$ time and $O(\log_B n)$ block transfers for any (unknown) value of B.*

4 Indirection with Dynamic Buckets

The layer $\mathcal{B}$ of the array A introduced in Section 3 is populated with buckets containing from $\Omega(k^{d-1})$ to $O(k^d)$ keys, for a constant $d \geq 5$. Each bucket is a balanced tree of constant height. A tree is maintained balanced by split and merge operations applied to the nodes. Unlike regular B-trees, the condition that causes a rebalancing for a node is defined with a parameter that depends on the whole size of the subtree rooted in the node (e.g., see the weight-balanced B-trees [2]). We now give a high level description of the buckets assuming that the size of each chunk is k and that we can rely on a suitable memory layout of the nodes. We postpone the discussion of the layout to Section 4.2, which is crucial for both implicitness and cache-obliviousness.

4.1 The Structure of the Buckets

A bucket has a constant number d of levels. Each bucket is associated with either a routing chunk or a filling chunk of layer $\mathcal{D}$, and all the keys in the bucket are greater than those in that chunk.

Leaves. A leaf of a bucket contains from k to $16k$ keys. Moreover, a leaf has associated a maniple that contains from $\sqrt{k}$ to $5\sqrt{k}$ keys and a number of filling chunks that ranges from r to $4r$ for a suitable constant r. The exact value of r concerns the memorization of the internal nodes of the buckets, as clarified in Section 4.2. The filling chunks of a leaf l are maintained in increasing sorted order in a linked list, say $f_1, \ldots, f_s$. Letting m be the maniple associated with l, we have that (1) f_j is the predecessor of f_{j+1} for $1 \leq j < s$, and (2) for each choice of keys $x \in f_s$, $x' \in l$ and $x'' \in m$, we have $x < x' < x''$.

As we shall see, each leaf l, its maniple m and its filling chunks are maintained in a constant number of zone of contiguous memory. Hence, searching in these objects requires a total of $O(k+\log n)$ time and $O(k/B+\log_B n)$ block transfers.

Internal nodes. An internal node contains routing chunks and filling chunks, and the pointer to the jth child is encoded by $O(\log n)$ keys in the jth chunk, which must be routing. Following an approach similar to that in [2], we define the *weight* $w(v)$ of an internal node v at level i (here, the leaves are at level 1) as the number of keys in the leaves descending from v. We maintain the weight ranging from $4^i k^i$ to $4^{i+1}k^i$. For this reason the number of chunks of an internal node can range from k to $16k$. For the root of a bucket, we only require the upper bound on its weight, since the bucket size can be $\Omega(k^{d-1})$ and the number of chunks in the root can be $O(1)$.

In order to pay $O(\log_B n)$ block transfers when searching and updating an internal node v, we maintain a directory of $\Theta(k)$ keys in v, analogously to what done in [11]. Thus the chunks of v are not maintained in sorted order, but their order can be retrieved by scanning the directory in v. In this way, any operation on v involves only $O(1)$ chunks and portions of $\Theta(k)$ contiguous keys each.

Handling insertions and deletions. If we ignore the memory management, the insertion or the deletion of a key in a bucket is a relatively standard task. If x is the key to insert into chunk c, the root of a bucket, we place x in its position inside c, shifting at most k keys to extract the maximum key in that chunk. We obtain the new key x to insert into the node whose pointer is encoded in c. In general, inserting x into a chunk of an internal node u goes along the same lines.

When we reach a leaf l, we perform a constant number of shifts and extractions of the maximum key in its filling chunks $f_1, \ldots, f_s$ and in l itself. We end up inserting a key into the maniple m of l. If the size of m exceeds the maximum allowed, we extract the $\sqrt{k}$ smallest keys from m and insert them into l. If the size of l is less than $16k$, we are done. On the contrary, if also l exceeds the maximum allowed but the number of its filling chunks is still less than $4r$, we extract the smallest chunk of l and create a new filling chunk f_{s+1}.

Instead, if the number of filling chunks is the maximum allowed, $4r$, we "split" the whole group made up of the leaf l, its maniple z and its filling chunks. That is

to say, we partition all the keys so that we have two new groups of the same kind, each group member satisfying all the invariants with their values half on the way between the maximum and the minimum allowed. We also generate a median (routing) chunk that have to be inserted in the parent of l, encoding a pointer in that chunk to the new leaf. We then examine all the ancestors of l, except the root, splitting every ancestor that exceeds its maximum allowed weight, obtaining two nodes of roughly the same weight. Deleting a key is analogous, except that we merge two internal nodes, although we may split once after a merge when the resulting node is too big. For the leaves we need merging and borrowing with an individual key. Merging and splitting the root of a bucket fall inside the control of a mechanism for the synchronization between layer $\mathscr{D}$ and layer $\mathscr{B}$, described in Section 5.

4.2 Memory Layout

We now discuss how to store the buckets in a contiguous portion of memory, which is divided into three areas.

- The *filling area* stores all filling chunks of layer $\mathscr{B}$ and the routing chunks of the internal nodes of the buckets.
- The *leaf area* stores all the leaves of the buckets using a new variation of the technique of *compactor zones* [9] that is suitable for de-amortization.
- The *maniple area* stores all the maniples using a set of *compactor lists*.

Filling area. We use the filling chunks to allocate the internal nodes. We need here to make some clear remarks on what we mean by "allocate." Suppose we want to allocate an empty node v with $16k$ chunks. We take a segment of $16k$ filling chunks that are contiguous and devote them to v. Since each filling chunk can be placed everywhere in the memory, when we need to insert a routing chunk c into v, we can replace the leftmost available filling chunk in v with c, moving that filling chunk elsewhere at the cost of searching one of its keys and of re-encoding the pointer to it, with $O(\log n)$ time and $O(k/B)$ block transfers.

Keeping the above remark in mind, we logically divide the filling zone into segments of $16k$ filling chunks each, since we can have a maximum of $16k$ routing chunks for an internal node. A segment is considered "free memory" if it contains only filling chunks. An internal node v with t routing chunks is stored in a segment with the first t routing chunks permuted and the remaining $16k - t$ filling chunks. When a routing chunk needs to be inserted into an internal node v whose weight is not maximal, we put the chunk in place of a filling chunk in the segment assigned to v. The replaced filling chunk will find a new place in

- either the segment of the child u of v, if u is an internal node that splits,
- or between the filling area and the leaf area, if u is a leaf that splits (the filling area increases by one chunk).

The deletion of a routing chunk in v is analogous. We replace the chunk with a filling chunk that arises either from the two merged children of v, if these children are internal nodes, or from the last position of the filling area, if these

children are leaves (and the filling area decreases by one chunk). Thus, using the directory for the routing as described above, we are able to insert or delete a chunk in an internal node in $O(\log n)$ time and $O(k/B)$ block transfers.

When we have to split an internal node v in two nodes v', v'', we allocate a new segment a for v'' while re-using the segment of v for v', and exchange incrementally the routing chunks in the segment of v' with filling chunks of a, the segment for v''. We exchange a constant number of chunks at each step, and these $s = O(k)$ steps are spread through the subsequent s operations operating through v. Note that, during this transition, v is considered not split but only partitioned in two segments instead of one. The execution of a merge is analogous. The invariants defined on the buckets guarantee that we can terminate an incremental transfer before that a further split or merge occurs.

The management of segments is through a simple linked list of free segments. The constant r that bounds the minimum number of filling chunks associated with a leaf can be easily chosen so that we can guarantee that there exists a sufficient number of filling chunks in layer $\mathcal{B}$ for all internal nodes.

Leaf area. The size of the leaves ranges from k to $16k$ keys, and vary by $\sqrt{k}$ keys at a time. Using the technique of the compactor zones, we maintain $15\sqrt{k}+1$ zones of contiguous memory, one for each possible size. Each zone is indexed by the size of the leaves it contains. The zones are in order by this index, so that zone s precedes zone $s+\sqrt{k}$, for each $s = k, k+\sqrt{k}, k+2\sqrt{k}, \ldots, 16k-\sqrt{k}$. When we have to add $\sqrt{k}$ keys to a leaf l of size s, we would like to extract l out of all compactor zones, moving l near to the $\sqrt{k}$ keys to be added by rotating each traversed zone by s keys. As a result, all the leaves are in a contiguous portion of memory except for a single leaf that can be "broken" in two pieces because of the rotation. This scheme is simple and powerful but too costly. We achieve our worst-case bounds with a two-step modification of this scheme. The first step exploits the fact that, for each leaf l,

1. $\Omega(\sqrt{k})$ update operations occur in its maniple between two consecutive variations of $\sqrt{k}$ in the size of l;
2. $\Omega(k)$ update operations occur in its maniple between two consecutive variations of k in the size of l (due to the creation/destruction of a filling chunk);
3. $\Omega(k)$ update operations occur in its filling chunks and its maniple between two consecutive splits or merges of l.

Consequently, we have a sufficient number of operations to perform incrementally the updates involving a leaf l. The basic idea is to execute a constant number of rotations from zone to zone in a single operation.

The second step introduces two *commuting sub-zones* between any two compactor zones. These two sub-zones work like the compactor zones but contain blocks of keys in transit between zones (see Figure 2). For any pair of sub-zones, the first sub-zone contains the blocks of $k+\sqrt{k}$ keys that have to be inserted in or deleted from a leaf. The second sub-zone contains

- chunks that have to be inserted or deleted in a leaf;
- all the chunks of the leaves to be split or merged.

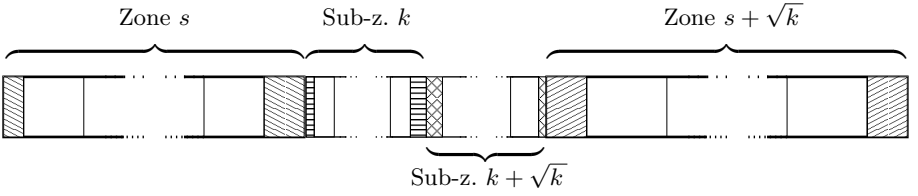


Fig. 2. Compactor zones and sub-zones with broken items highlighted.

For example, when a leaf reaches its maximum number of keys, it is transformed into a linked list of $O(1)$ chunks going to the second sub-zone near zone $16k$. At this point, we incrementally move these chunks until we reach the sub-zone near zone $8k$; we split the list into two parts and put them as two new leaves of size $8k$. Note that the leaf is still searchable while traversing the zones.

Maniple area. The maniple area is handled with compactor lists [11]. However, we use allocation units of size $\sqrt{k}$, and so the structural information for them must be encoded in the leaves associated with the maniples. Each time we need a structural information (e.g., next allocation unit in a list), we perform a search to locate the corresponding leaf. There are $O(\sqrt{k})$ heads of size at most $\sqrt{k}$, so the whole head area occupies $O(k)$ positions and can be scanned each time.

Theorem 2. *Searching, inserting and deleting a key in a bucket of layer $\mathcal{B}$ takes $O(\log n)$ time and $O(\log_B n)$ block transfers for any (unknown) value of B.*

5 Synchronization between Layer $\mathcal{D}$ and Layer $\mathcal{B}$

We combine the two layers described in Sections 3–4 by using a simple variation of the Dietz-Sleator list [6]. Every other $\Omega(\text{polylog}(n))$ operations in layer $\mathcal{B}$, we eventually split the largest bucket and we merge the smallest bucket. This causes the insertion and the deletion of a routing chunk in layer $\mathcal{D}$. By setting up the suitable multiplicative constants, we provide a time slot that is sufficient to complete the algorithms operating in layer $\mathcal{D}$ by Theorem 1.

Theorem 3. *An array of n keys can be maintained under insertions and deletions in $O(\log n)$ worst-case time per operation using just $O(1)$ RAM registers, so that searching a key takes $O(\log n)$ time. The only operations performed on the keys are comparisons and moves. They require $O(\log_B n)$ block transfers in the worst case for the cache-oblivious model, where the block size B is unknown to the operations.*

References

1. Alfred V. Aho, John E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, 1974.

2. L. Arge and J. S. Vitter. Optimal dynamic interval management in external memory. In IEEE, editor, *37th Annual Symposium on Foundations of Computer Science: October 14–16, 1996, Burlington, Vermont*, pages 560–569, USA, 1996. IEEE Computer Society Press.
3. M. A. Bender, E. D. Demaine, and M. Farach-Colton. Cache-oblivious B-trees. In IEEE, editor, *41st Annual Symposium on Foundations of Computer Science: proceedings: 12–14 November, 2000, Redondo Beach, California*, pages 399–409, USA, 2000. IEEE Computer Society Press.
4. Michael A. Bender, Richard Cole, and Rajeev Raman. Exponential structures for efficient cache-oblivious algorithms. *International Colloquium on Automata, Languages and Programming, LNCS*, 2380:195–206, 2002.
5. Gerth Stølting Brodal, Rolf Fagerberg, and Riko Jacob. Cache-oblivious search trees via trees of small height. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 39–48, 2002.
6. P. Dietz and D. Sleator. Two algorithms for maintaining order in a list. In Alfred Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 365–372, 1987. ACM Press.
7. Gianni Franceschini and Roberto Grossi. Implicit dictionaries supporting searches and amortized updates in $O(\log n \log\log n)$. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*, pages 670–678. SIAM, 2003.
8. Gianni Franceschini and Roberto Grossi. Optimal cache-oblivious implicit dictionaries. *International Colloquium on Automata, Languages and Programming, LNCS*, 2003.
9. Gianni Franceschini and Roberto Grossi. Optimal implicit and cache-oblivious dictionaries over unbounded universes. Full version, 2003.
10. Gianni Franceschini and Roberto Grossi. Optimal space-time dictionaries over an unbounded universe with flat implicit trees. Technical report TR-03-03, January 30, 2003.
11. Gianni Franceschini, Roberto Grossi, J. Ian Munro, and Linda Pagli. Implicit B-trees: New results for the dictionary problem. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002.
12. Greg N. Frederickson. Implicit data structures for the dictionary problem. *Journal of the ACM*, 30(1):80–94, 1983.
13. M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In IEEE, editor, *40th Annual Symposium on Foundations of Computer Science: October 17–19, 1999, New York City, New York,*, pages 285–297, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1999. IEEE Computer Society Press.
14. Alon Itai, Alan G. Konheim, and Michael Rodeh. A sparse table implementation of priority queues. In Shimon Even and Oded Kariv, editors, *International Colloquium on Automata, Languages and Programming*, volume 115 of *Lecture Notes in Computer Science*, pages 417–431, 1981.
15. D. E. Knuth. *The Art of Computer Programming III: Sorting and Searching.* Addison–Wesley, Reading, Massachusetts, 1973.
16. J. Ian Munro. An implicit data structure supporting insertion, deletion, and search in $O(\log^2 n)$ time. *Journal of Computer and System Sciences*, 33(1):66–74, 1986.
17. J. Ian Munro and Hendra Suwanda. Implicit data structures for fast search and update. *Journal of Computer and System Sciences*, 21(2):236–250, 1980.

18. Dan E. Willard. A density control algorithm for doing insertions and deletions in a sequentially ordered file in good worst-case time. *Information and Computation*, 97(2):150–204, April 1992.
19. J. W. J. Williams. Algorithm 232: Heapsort. *Communications of the ACM*, 7:347–348, 1964.
20. Andrew C. Yao. Should tables be sorted? *J. Assoc. Comput. Mach.*, 31:245–281, 1984.

Extremal Configurations and Levels in Pseudoline Arrangements*

Micha Sharir[1,2] and Shakhar Smorodinsky[1]

[1] School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel
{michas,smoro}@post.tau.ac.il,
[2] Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA;

Abstract. This paper studies a variety of problems involving certain types of extreme configurations in arrangements of (x-monotone) pseudo-lines. For example, we obtain a very simple proof of the bound $O(nk^{1/3})$ on the maximum complexity of the k-th level in an arrangement of n pseudo-lines, which becomes even simpler in the case of lines. We thus simplify considerably the previous proof by Tamaki and Tokuyama (and also simplify Dey's proof for lines).

We also consider *diamonds* and *anti-diamonds* in (simple) pseudo-line arrangements, where a diamond (resp., an anti-diamond) is a pair u, v of vertices, so that u lies in the double wedge of v and vice versa (resp., neither u nor v lies in the other double wedge). We show that the maximum size of a diamond-free set of vertices in an arrangement of n pseudo-lines is $3n - 6$, by showing that the induced graph (where each vertex of the arrangement is regarded as an edge connecting the two incident curves) is planar, simplifying considerably a previous proof of the same fact by Tamaki and Tokuyama. Similarly, we show that the maximum size of an anti-diamond-free set of vertices in an arrangement of n pseudo-lines is $2n - 2$. We also obtain several additional results, which are listed in the introduction.

In some of our results, we use a recent duality transform between points and pseudo-lines due to Agarwal and Sharir, which extends an earlier transform by Goodman (that applied only in the projective plane). We show that this transform maps a set of vertices in a pseudo-line arrangement to a topological graph whose edges are drawn as x-monotone arcs that connect pairs of the dual points, and form a set of extendible pseudo-segments (they are pieces of curves that form a pseudo-line arrangement in the dual plane). This allows us (a) to 'import' known results on this kind of topological graphs to the context of pseudo-lines; (b) to extend techniques that have been originally applied only for geometric graphs (whose edges are drawn as straight segments), thereby obtaining new results for pseudo-line arrangements, or for the above-class of x-monotone

* Work on this paper has been supported by a grant from the Israel Science Fund (for a Center of Excellence in Geometric Computing). Work by Micha Sharir has also been supported by NSF Grants CCR-97-32101 and CCR-00-98246, by a grant from the U.S.-Israeli Binational Science Foundation, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 127–139, 2003.
© Springer-Verlag Berlin Heidelberg 2003

topological graphs; and (c) to derive new techniques, facilitated by the passage to the dual setting, that apply in the more general pseudo-line context, and extend and simplify considerably the earlier proofs. This paper contains examples of all three kinds of results.

1 Introduction

Let Γ be a collection of n *pseudolines* in the plane, which we define to be graphs of continuous totally-defined functions, each pair of which intersect in exactly one point, and the curves cross each other at that point. In what follows we assume general position of the pseudolines, meaning that no three pseudolines pass through a common point, and that the x-coordinates of any two intersection points of the pseudolines are distinct. Let E be a subset of the vertices of the arrangement $A(\Gamma)$. E induces a graph $G = (\Gamma, E)$ on Γ (in what follows, we refer to such a graph as a *pseudoline graph*). For each pair (γ, γ') of distinct pseudolines in Γ, we denote by $W(\gamma, \gamma')$ the *double wedge* formed between γ and γ', that is, the (open) region consisting of all points that lie above one of these pseudolines and below the other. We also denote by $W^c(\gamma, \gamma')$ the complementary (open) double wedge, consisting of all points that lie either above both curves or below both curves.

Definition 1. *We say that two edges (γ, γ') and (δ, δ') of G form a* diamond *if the point $\gamma \cap \gamma'$ is contained in the double wedge $W(\delta, \delta')$, and the point $\delta \cap \delta'$ is contained in the double wedge $W(\gamma, \gamma')$.*

Definition 2. *We say that two edges (γ, γ') and (δ, δ') of G form an* anti-diamond *if the point $\gamma \cap \gamma'$ is not contained in the double wedge $W(\delta, \delta')$, and the point $\delta \cap \delta'$ is not contained in the double wedge $W(\gamma, \gamma')$; that is, $\gamma \cap \gamma'$ lies in $W^c(\delta, \delta')$ and $\delta \cap \delta'$ lies in $W^c(\gamma, \gamma')$.*

Definition 3. *(a) A collection S of x-monotone bounded Jordan arcs is called a collection of* pseudosegments *if each pair of arcs of S intersect in at most one point, where they cross each other.*
(b) S is called a collection of extendible pseudosegments *if there exists a set Γ of pseudolines, with $|\Gamma| = |S|$, such that each $s \in S$ is contained in a unique pseudoline of Γ.*

See [8] for more details concerning extendible pseudosegments.

Definition 4. *(a) A* drawing *of a graph $G = (\Gamma, E)$ in the plane is a mapping that maps each vertex $v \in \Gamma$ to a point in the plane, and each edge $e = uv$ of E to a Jordan arc connecting the images of u and v, such that no three arcs are concurrent at their relative interiors, and the relative interior of no arc is incident to a vertex.*
(b) If the images of the edges of E form a family of extendible pseudo-segments then we refer to the drawing of G as an (x-monotone) generalized geometric graph.

(The term *geometric graphs* is usually reserved to drawings of graphs where the edges are drawn as straight segments.)

In this paper we prove the following results.

Duality between pseudoline graphs and generalized geometric graphs. The first main result of this paper establishes an equivalence between pseudoline graphs and generalized geometric graphs.

We first derive the following weaker result, which has an easy and self-contained proof.

Theorem 1. *Let Γ and G be as above. Then there is a drawing of G in the plane such that two edges e and e' of G form a diamond if and only if their corresponding drawings cross each other an odd number of times.*

After the original preparation of this paper, Agarwal and Sharir [5] established a duality transformation in arrangements of pseudolines, which has several useful properties and other applications. Using their technique, we derive the following stronger result:

Theorem 2. *(a) Let Γ and G be as above. Then there is a drawing of G in the plane, with the edges constituting a family of extendible pseudosegments, such that, for any two edges e, e' of G, e and e' form a diamond if and only if their corresponding drawings cross each other.*
(b) Conversely, for any graph $G = (V, E)$ drawn in the plane with its edges constituting a family of extendible pseudosegments, there exists a family Γ of pseudolines and a 1-1 mapping φ from V onto Γ, so that each edge $uv \in E$ is mapped to the vertex $\varphi(u) \cap \varphi(v)$ of $\mathcal{A}(\Gamma)$, such that two edges in E cross each other if and only if their images are two vertices of $\mathcal{A}(\Gamma)$ that form a diamond.

Applications. As an immediate corollary of Theorem 2 (which can also be derived from Theorem 1, using the fact [25] that any graph drawn in the plane such that every pair of edges on four distinct vertices cross an even number of times, is planar), we obtain

Theorem 3. *Let Γ and G be as above. If G is diamond-free then G is planar and thus $|E| \leq 3n - 6$.*

Theorem 3 has been proven by Tamaki and Tokuyama [23], using a more involved argument. This was the underlying theorem that enabled them to extend Dey's improved bound of $O(n^{4/3})$ on the complexity of a single level in an arrangement of lines [11], to arrangements of pseudolines. Note that the planarity of G is obvious for the case of lines: If we dualize the given lines into points, using the duality $y = ax + b \mapsto (a, b)$ and $(c, d) \mapsto y = -cx + d$, presented in [13], and map each edge (γ, γ') of G to the straight segment connecting the points dual to γ and γ', we obtain a crossing-free drawing of G. Hence, Theorem 3 is a natural (though harder to derive) extension of this property to the case of pseudolines.

We note also that the converse statement of Theorem 3 is trivial: Every planar graph can be realized as a diamond-free pseudoline graph (in fact, in an

arrangement of lines): We draw the graph as a straight-edge graph (which is always possible [14]), and apply the inverse duality to the one just mentioned.

In more generality, we can take any theorem that involves generalized geometric graphs (whose edges are extendible pseudosegments), and that studies the crossing pattern of these edges, and 'transport' it into the domain of pseudoline graphs. As an example of this, we have:

Theorem 4. *Let Γ and G be as above. (i) If G contains no three edges which form pairwise diamonds then G is quasi-planar (in the terminology of [2]; see below), and thus its size is $O(n)$. (ii) If G contains no k edges which form pairwise diamonds (for any fixed $k \geq 4$) then the size of G is $O(n \log n)$ (with the constant of proportionality depending on k).*

In its appropriate reformulation in the context of generalized geometric graphs, Theorem 4(i) corresponds to a result of Agarwal et al. [2] on quasi-planar graphs. A *quasi-planar* (respectively, *k-quasi-planar*) graph is a graph that can be drawn in the plane such that no three (respectively, k) of its edges are pairwise crossing. It was shown in [2] that the size of a quasi-planar graph is $O(n)$. This result was extended by Valtr [26] to the case $k \geq 4$ and our Theorem 4(ii) is a similar interpretation of Valtr's bound in the context of pseudoline graphs. Our reformulations are valid, for both parts of the theorem, since both the results of [2, 27] hold for graphs whose edges are extendible pseudosegments.

Definition 5. *A* thrackle *is a drawing of a graph in the plane so that every pair of edges either have a common endpoint and are otherwise disjoint, or else they intersect in exactly one point where they cross each other.*

The notion of a thrackle is due to Conway, who conjectured that the number of edges in a thrackle is at most the number of vertices. The study of thrackles has drawn much attention. Two recent papers [18] and [7] obtain linear bounds for the size of a general thrackle, but with constants of proportionality that are greater than 1. The conjecture is known to hold for straight-edge thrackles [20], and, in Section 6, we extend the result, and the proof, to the case of graphs whose edges are extendible pseudosegments. That is, we show:

Theorem 5. *Let Γ and G be as above. If every pair of edges connecting four distinct vertices (that is, curves of Γ) in G form a diamond, then the size of G is at most n.*

Our proof extends ideas used by Perles in the proof for the straight edge case.

Pseudoline graphs without anti-diamonds. We now turn to study pseudoline graphs that do not have any anti-diamond. We show:

Theorem 6. *Let Γ and G be as above. If G is anti-diamond-free then $|E| \leq 2n - 2$.*

Theorem 6 is an extension, to the case of pseudolines, of a (dual version of a) theorem of Katchalski and Last [15], refined by Valtr [27], both solving a problem posed by Kupitz. The theorem states that a straight-edge graph on n points in the plane, which does not have any pair of parallel edges, has at most $2n-2$ edges. A pair of segments e, e' is said to be *parallel* if the line containing e does not cross e' and the line containing e' does not cross e. (For straight edges, this is equivalent to the condition that e and e' are in convex position.) The dual version of a pair of parallel edges is a pair of vertices in a line arrangement that form an anti-diamond. Hence, Theorem 6 is indeed an extension of the result of [15,27] to the case of pseudolines. The proof, for the case of straight-edge graphs, has been recently simplified by Valtr [28]. Our proof, obained independently, can be viewed as an extension of this new proof to the case of pseudolines.

Note that Theorem 6 is not directly obtainable from [15,27,28], (a) because Theorem 2 does not cater to anti-diamonds, and (b) because the analysis of [15, 27,28] only applies to straight-edge graphs.

The complexity of the k-level in an arrangement of pseudolines. We provide a simpler proof of the following major result in combinatorial and computational geometry:

Theorem 7. *The maximum complexity of the k-level in an arrangement of n pseudolines is $O(nk^{1/3})$.*

The k-level in the arrangement of a set Γ of n pseudolines is the (closure of) the set of all points that lie on curves of Γ and have exactly k other curves passing below them. This is a central structure in arrangements, with a long and rich history, and with many applications, both in discrete and in computational geometry; See e.g., [19]. In a recent breakthrough, Dey [11] has shown that the complexity (number of vertices) of the k-level in an arrangement of n lines is $O(nk^{1/3})$ (the best known lower bound is only near-linear [24]). This bound was extended to the case of pseudolines by Tamaki and Tokuyama [23], using a very complicated proof. We present a much simpler proof (than both proofs in [11] and [23]) for the general case of pseudolines.

Incidences and many faces in pseudoline arrangements. Finally, as an application of Theorem 3, we provide yet another simple proof of the following well-known result in a much-studied research area:

Theorem 8. *(a) The maximum number of incidences between m distinct points and n distinct pseudolines is $\Theta(m^{2/3}n^{2/3} + m + n)$.*
(b) The maximum number of edges bounding m distinct faces in an arrangement of n pseudolines is $\Theta(m^{2/3}n^{2/3} + n)$.

The proof is in some sense 'dual' to the proofs based on Székely's technique [12, 22].

The proof of Theorem 8(b) can be extended to yield the following result, recently obtained in [3], where it has been proved using the dual approach, based on Székely's technique.

Theorem 9. *The maximum number of edges bounding m distinct faces in an arrangement of n extendible pseudo-segments is $\Theta((m+n)^{2/3}n^{2/3})$.*

2 Drawing Pseudoline Graphs

In this section we prove Theorems 1 and 2. Both proofs use the same drawing rule for realizing pseudoline graphs as geometric graphs. The difference is that the stronger properties of Theorem 2 follow from the more sophisticated machinery of point-pseudoline duality, developed in [5]. On the other hand, the proof of Theorem 1 is simple and self-contained.

Proof of Theorem 1: Let ℓ be a vertical line such that all vertices of the arrangement $A(\Gamma)$ lie to the right of ℓ. Enumerate the pseudolines of Γ as $\gamma_1, \ldots, \gamma_n$, ordered in increasing y-coordinates of the intersection points $p_i = \ell \cap \gamma_i$. We construct a drawing of G in the plane, using the set $P = \{p_1, \ldots, p_n\}$ as the set of vertices.

For each edge $(\gamma_i, \gamma_j) \in E$, we connect the points p_i and p_j by a y-monotone curve $e_{i,j}$ according to the following rules. Assume, without loss of generality, that $i > j$. If $i = j + 1$ (so that p_i and p_j are consecutive intersection points along ℓ) then $e_{i,j}$ is just the straight segment p_ip_j (contained in ℓ). Otherwise, $e_{i.j}$ is drawn very close to ℓ, and generally proceeds upwards (from p_j to p_i) parallel to ℓ either slightly to its left or slightly to its right. In the vicinity of an intermediate point p_k, the edge either continues parallel to ℓ, or converges to p_k (if $k = i$), or switches to the other side of ℓ, crossing it before p_k. The decision on which side of p_k the edge should pass is made according to the following

> **Drawing rule:** If the pseudoline γ_k passes above the apex of $W(\gamma_i, \gamma_j)$ then $e_{i,j}$ passes to the left of p_k, otherwise $e_{i,j}$ passes to the right of p_k.

This drawing rule is a variant of a rule recently proposed in [4] for drawing, and proving the planarity, of another kind of graphs related to arrangements of pseudocircles or pseudo-parabolas. Note that this rule does not uniquely define the drawing.

We need the following technical lemma:

Lemma 1. *Let $x_1 < x_2 < x_3 < x_4$ be four real numbers. (i) Let $e_{1,4}$ and $e_{2,3}$ be two x-monotone Jordan arcs with endpoints at $(x_1, 0), (x_4, 0)$ and $(x_2, 0), (x_3, 0)$, respectively, so that $e_{1,4}$ does not pass through $(x_2, 0)$ or through $(x_3, 0)$. Then $e_{1,4}$ and $e_{2,3}$ cross an odd number of times if and only if $e_{1,4}$ passes around the points $(x_2, 0)$ and $(x_3, 0)$ on different sides.*

(ii) Let $e_{1,3}$ and $e_{2,4}$ be two x-monotone Jordan arcs with endpoints at $(x_1, 0), (x_3, 0)$ and $(x_2, 0), (x_4, 0)$, respectively, so that $e_{1,3}$ does not pass through $(x_2, 0)$ and $e_{2,4}$ does not pass through $(x_3, 0)$. Then $e_{1,3}$ and $e_{2,4}$ cross an odd number of times if and only if $e_{1,3}$ passes below $(x_2, 0)$ and $e_{2,4}$ passes below $(x_3, 0)$, or $e_{1,3}$ passes above $(x_2, 0)$ and $e_{2,4}$ passes above $(x_3, 0)$.

Proof. In case (i), let f_1 and f_2 be the two real (partially defined) continuous functions whose graphs are $e_{1,4}$ and $e_{2,3}$, respectively. Similarly, for case (ii), let f_1 and f_2 be the functions whose graphs are $e_{1,3}$ and $e_{2,4}$, respectively.

Consider the function $g = f_1 - f_2$ over the interval $[x_2, x_3]$. By the intermediate-value theorem, $g(x_2)$ and $g(x_3)$ have different signs if and only if g vanishes an odd number of times over this interval. This completes the proof of the Lemma.

Let $e_1 = e_{x,y}$, $e_2 = e_{z,w}$ be the drawings of two distinct edges in G that do not share a vertex. We consider two possible cases:

Case (i): The intervals p_xp_y and p_zp_w (on the line ℓ) are nested. That is, their endpoints are ordered, say, as p_z, p_x, p_y, p_w in y-increasing order along the line ℓ. By Lemma 1, e_1 and e_2 cross an odd number of times if and only if e_2 passes around the points p_x and p_y on different sides. On the other hand, it is easily checked that the drawing rule implies that e_1 and e_2 form a diamond in G if and only if e_2 passes around the points p_x and p_y on different sides. Hence, in this case we have that e_1 and e_2 form a diamond if and only if they cross an odd number of times.

Case (ii): The intervals p_xp_y and p_zp_w 'interleave', so that the y-order of the endpoints of e_1 and e_2 is, say, p_x, p_z, p_y, p_w, or a symmetrically similar order. By Lemma 1, e_1 and e_2 cross an odd number of times if and only if e_1 passes around the point p_z on the same side that e_2 passes around the point p_y. On the other hand, the drawing rule for e_1 and e_2 easily implies that e_1 and e_2 form a diamond if and only if e_1 passes around the point p_z on the same side that e_2 passes around the point p_y.

It is also easily checked that, in the case where the intervals p_xp_y and p_zp_w are disjoint, the edges e_1 and e_2 do not form a diamond, nor can their drawings intersect each other. This completes the proof of the theorem. □

Proof of Theorem 2: The drawing rule used in the proof of Theorem 1 is in fact a special case of the duality transform between points and (x-monotone) pseudolines, as obtained recently by Agarwal and Sharir [5]. Specifically, we apply this result to Γ and to the set G of the given vertices of $\mathcal{A}(\Gamma)$. The duality of [5] maps the points of G to a set G^* of x-monotone pseudolines, and maps the pseudolines of Γ to a set Γ^* of points, so that a point $v \in G$ lies on (resp., above, below) a curve $\gamma \in \Gamma$ if and only if the dual pseudoline v^* passes through (resp., above, below) the dual point γ^*. Finally, in the transformation of [5], the points of Γ^* are arranged along the x-axis in the same order as that of the intercepts of these curves with the vertical line ℓ defined above.

We apply this transformation to Γ and G. In addition, for each vertex $v \in G$, incident to two pseudolines $\gamma_1, \gamma_2 \in \Gamma$, we trim the dual pseudoline v^* to its portion between the points γ_1^*, γ_2^*.

This yields a plane drawing of the graph G, whose edges form a collection of extendible pseudo-segments. The drawing has the following main property:

Lemma 2. *Let $v = \gamma_1 \cap \gamma_2$ and $w = \gamma_3 \cap \gamma_4$ be two vertices in G, defined by four distinct curves. Then v and w form a diamond if and only if the corresponding edges of the drawing cross each other.*

Proof. The proof is an easy consequence of the proof of Theorem 1 given above. In fact, it suffices to show that the duality transformation of [5] obeys the drawing rule used in the above proof, with an appropriate rotation of the plane by 90 degrees. So let $\gamma_i, \gamma_j, \gamma_k \in \Gamma$ such that γ_k passes above (resp., below) $\gamma_i \cap \gamma_j$, and such that γ_k meets the vertical line ℓ at a point between $\gamma_i \cap \ell$ and $\gamma_j \cap \ell$. Our drawing rule then requires that the edge $p_i p_j$ pass to the left (resp., to the right) of p_k. On the other hand, the duality transform, preserving the above/below relationship, makes the edge $\gamma_i^* \gamma_j^*$ pass below (resp., above) γ_k^*. Hence the two rules coincide, after an appropriate rotation of the plane, and the lemma is now an easy consequence of the preceding analysis.

Lemma 2 thus implies Theorem 2(a). To prove the converse part (b), let $G = (V, E)$ be a graph drawn in the plane so that its edges form a collection of extendible pseudo-segments, and let Λ denote the family of pseudolines containing the edges of E. Apply the point-pseudoline duality transform of [5] to V and Λ. We obtain a family V^* of pseudolines and a set Λ^* of points, so that the incidence and the above/below relations between V and Λ are both preserved. It is now routine to verify, as in the case of point-line duality, that two edges $u_1 v_1$ and $u_2 v_2$ of E cross each other if and only if the corresponding vertices $u_1^* \cap v_1^*$, $u_2^* \cap v_2^*$ of $\mathcal{A}(V^*)$ form a diamond. This completes the proof of Theorem 2. □

The immediate implications of these results, namely Theorems 3 and 4, follow as well, as discussed in the introduction.

3 The Complexity of a k-Level in Pseudoline Arrangements

In this section we provide a simpler proof of the well-known $O(nk^{1/3})$ upper bound on the complexity of the k-level in an arrangement of n pseudolines (see [11,23]).

Let Γ be the given collection of n pseudolines, and let E be the set of vertices of the k-level, where k is a fixed constant ($0 \leq k \leq n-2$). Theorem 2 and a standard probabilistic argument allow us to derive the following extension of the Crossing Lemma of [6,16]; We omit the proof here.

Lemma 3 (Extended Crossing Lemma). *Let $G(\Gamma, E)$ be a pseudoline graph on n pseudolines, with $|E| \geq 4n$. Then G has $\Omega(|E|^3/n^2)$ diamonds.*

Remark: In the restricted case where Γ is a collection of real lines, Lemma 3 is a dual version of the Crossing Lemma of [6,16].

Dey [11] has shown that the number of diamonds in G is at most the total number of vertices in the arrangement $A(\Gamma)$ that lie at level less than k. It is well known (see e.g. [10]) that the overall complexity of the first k levels in an arrangement of n lines or pseudolines is $O(nk)$. Hence, this fact, combined with the lower bound discussed above, yield the $O(nk^{1/3})$ upper bound on the complexity of the k-level. We provide here an alternative simpler proof that the

number of diamonds in G is at most $k(n-k-1)$, without using the bound on the complexity of the first k levels.

We use the fact that the vertices of the k-level can be grouped into vertex sets of k 'concave' vertex-disjoint chains $c_1, \ldots, c_k$. Each such chain c_i is an x-monotone (connected) path that is contained in the union of the pseudolines of Γ, such that all the vertices of c_i are at level k. Informally, as we traverse c_i from left to right, whenever we reach a vertex of $A(\Gamma)$, we can either continue along the pseudoline we are currently on, or make a right (i.e., downward) turn onto the other pseudoline, but we cannot make a left (upward) turn; in case the pseudolines are real lines, c_i is indeed a concave polygonal chain. The simple construction of these chains is described in [1]: Each chain starts on each of the k lowest pseudolines at $x = -\infty$, and makes (right) turns *only* at vertices of the k-level. In a symmetric manner we can group the vertices of the k-level into $n-k-1$ 'convex' vertex-disjoint chains, by starting the chains along the $n-k-1$ highest pseudolines at $x = -\infty$, and by making *left* turns only at vertices of the k-level. Let (p, p') be a diamond, where p and p' are vertices at level k. Assume without loss of generality that p lies to the left of p'. Let c_1 be the unique *concave* chain that contains p and let c_2 be the unique *convex* chain that contains p'. For a given vertex v in $A(\Gamma)$, let $W_r(v)$ (resp. $W_l(v)$) denote the (interior of the) right (resp. left) wedge of the double-wedge formed by the two pseudolines defining v. Consider the right wedges of vertices of c_1. It is easy to see (from the 'concavity' of c_1) that those wedges are pairwise disjoint (see also [1]). A similar argument holds for the left wedges of the vertices of c_2. Since $p' \in W_r(p)$ and $p \in W_l(p')$, it follows that c_2 does not meet the lower edge of $W_r(p)$, but meets the upper edge of this wedge. This can happen for at most one vertex of c_1, because of the disjointness of the right wedges of its vertices. Hence p is uniquely determined from the pair (c_1, c_2), and, symmetrically this also holds for p'.

Thus the number of diamonds in the k-level is at most the number of pairs (c_1, c_2) of a concave chain and a convex chain; that is, at most $k(n-k-1)$. This completes the proof of Theorem 7.

4 Yet Another Proof for Incidences and Many Faces in Pseudoline Arrangements

In this section we provide yet another proof of the well-known (worst-case tight) bounds given in Theorem 8.

We will prove only part (b) of the theorem; part (a) can then be obtained by a simple and known reduction (see, e.g., [9]); alternatively, it can be obtained by a straightforward modification of the proof of (b), given below.

Let Γ be the given collection of n pseudolines, and let $f_1, \ldots, f_m$ be the m given faces of the arrangement $\mathcal{A}(\Gamma)$. Let E denote the set of all vertices of these faces, excluding the leftmost and rightmost vertex, if any, of each face. Since every bounded face has at least one vertex that is not leftmost or rightmost, and since the number of unbounded faces is $O(n)$, it follows that the quantity

that we wish to bound is $O(|E| + n)$. By Lemma 3, if $|E| \geq 4n$ then the graph $G(\Gamma, E)$ has $\Omega(|E|^3/n^2)$ diamonds. Let (p, p') be a diamond, where p is a vertex of some face f and p' is a vertex of another face f'. (It is easily verified that if p and p' bound the same face then they cannot form a diamond.) Then, using the Levy Enlargement Lemma [17], there exists a curve γ_0 that passes through p and p', such that $\Gamma \cup \{\gamma_0\}$ is still a family of pseudolines. In this case γ_0 must be contained in the two double wedges of p and p', and thus it avoids the interiors of f and of f'; that is, γ_0 is a 'common tangent' of f and f'. As in the case of lines, it is easy to show that a pair of faces can have at most four common tangents of this kind. Hence, the number of diamonds in G cannot exceed $2m^2$. Putting everything together, we obtain $|E| = O(m^{2/3}n^{2/3} + n)$. □

5 Graphs in Pseudoline Arrangements without Anti-diamonds

So far, the paper has dealt exclusively with the existence or nonexistence of diamonds in graphs in pseudoline arrangements. We now turn to graphs in pseudoline arrangements that do not contain any *anti-diamond*. Recall that the notion of an anti-diamond is an extension, to the case of pseudolines, of (the dual version of) a pair of edges in (straight-edge) geometric graphs that are in convex position (so-called 'parallel' edges). Using Theorem 2 (and the analysis in its proof), one obtains a transformation that maps an anti-diamond-free pseudoline graph (Γ, G) to a generalized geometric graph, whose edges form a collection of extendible pseudo-segments, with the property that, for any pair e, e' of its edges, defined by four distinct vertices, either the pseudoline containing e crosses e' or the pseudoline containing e' crosses e.

We present a much shorter and simpler proof of Theorem 6 than those of [15, 27], that applies directly in the original pseudoline arrangement, and is similar in spirit to the recent simplified proof of Valtr [28] for the case of straight-edge geometric graphs.

Proof of Theorem 6: We construct two sequences A and B whose elements belong to Γ, as follows. We sort the intersection points of the pseudolines of Γ that correspond to the edges of G in increasing x-order, and denote the sorted sequence by $P = \langle p_1, \ldots, p_m \rangle$. For each element p_i of P, let γ_i and γ_i' be the two pseudolines forming (meeting at) p_i, so that γ_i lies below γ_i' to the left of p_i (and lies above γ_i' to the right). Then the i-th element of A is γ_i and the i-th element of B is γ_i'.

Lemma 4. *The concatenated cyclic sequence $C = A \| B$ does not contain a subcycle of alternating symbols of the form $a \cdots b \cdots a \cdots b$, for $a \neq b$.*

Proof. Omitted.

A *run* in C is a maximal contiguous subsequence of identically labeled elements. If we replace each run by a single element, the resulting sequence C^* is

a Davenport-Schinzel cycle of order 2 on n symbols, as follows from Lemma 4. Hence, the length of C^* is at most $2n-2$ [21].

Note that it is impossible to have an index $1 \leq i \leq |G|$ such that the i-th element of A is equal to the $(i+1)(mod|G|)$-st element of A and the i-th element of B is equal to the $(i+1)(mod|G|)$-st element of B. Indeed, if these elements are a and b, respectively, then we obtain two vertices of $\mathcal{A}(\Gamma)$ (the one encoded by the i-th elements of A and B and the one encoded by the $(i+1)$-st elements) that are incident to both a and b, which is impossible. In other words, for each $i = 1, \ldots, |G|$, a new run must begin either after the i-th element of A or after the i-th element of B (or after both). This imply that the length of C^* is greater or equal to $|G|$. Hence we have: $|G| \leq |C^*| \leq 2n-2$. This completes the proof of Theorem 6. □

6 Pseudolines and Thrackles

Let G be a thrackle with n vertices, whose edges are extendible pseudo-segments. We transform G, using the pseudoline duality, to an intersection graph in an arrangement of a set Γ of n pseudolines. The edge set of G is mapped to a subset E of vertices of $\mathcal{A}(\Gamma)$, with the property that every pair of vertices of E, not sharing a common pseudoline, form a diamond.

Theorem 10. $|E| \leq n$.

Proof: The proof is an extension, to the case of pseudoline graphs (or, rather, generalized geometric graphs drawn with extendible pseudo-segments), of the beautiful and simple proof of Perles, as reviewed, e.g., in [20].

Fix a pseudoline $\gamma \in \Gamma$ and consider the vertices in $E \cap \gamma$. We say that $v \in E \cap \gamma$ is a *right-turn* (resp., *left-turn*) vertex with respect to γ if, to the left of v, γ lies above (resp., below) the other pseudoline incident to v.

If γ contains three vertices $v_1, v_2, v_3 \in E$, appearing in this left-to-right order along γ, such that v_1 and v_3 are right-turn vertices and v_2 is a left-turn vertex, then all vertices of E must lie on γ, because the intersection of the three (open) double wedges of v_1, v_2, v_3 is empty, as is easily checked. In this case $|E| \leq n-1$ and the theorem follows. A similar argument holds when v_1 and v_3 are left-turn and v_2 is a right-turn vertex.

Hence we may assume that, for each $\gamma \in \Gamma$, the left-turn vertices of $E \cap \gamma$ are separated from the right-turn vertices of $E \cap \gamma$ along γ.

For each $\gamma \in \Gamma$, we delete one vertex of $E \cap \gamma$, as follows. If $E \cap \gamma$ consists only of left-turn vertices, or only of right-turn vertices, we delete the rightmost vertex of $E \cap \gamma$. Otherwise, these two groups of vertices are separated along γ, and we delete the rightmost vertex of the left group.

We claim that after all these deletions, E is empty. To see this, suppose to the contrary that there remains a vertex $v \in E$, incident to two pseudolines $\gamma_1, \gamma_2 \in \Gamma$, such that γ_1 lies below γ_2 to the left of v. Clearly, v is a left-turn vertex with respect to γ_1, and a right-turn vertex with respect to γ_2.

The deletion rule implies that, initially, $E \cap \gamma_1$ contained either a left-turn vertex v_1^- that lies to the left of v, or a right-turn vertex v_1^+ that lies to the right of v. Similarly, $E \cap \gamma_2$ contained either a right-turn vertex v_2^- that lies to the left of v, or a left-turn vertex v_2^+ that lies to the right of v. It is now easy to check that, in each of the four possible cases, the respective pair of vertices, (v_1^-, v_2^-), (v_1^+, v_2^-), (v_1^-, v_2^+), or (v_1^+, v_2^+), do not form a diamond, a contradiction that shows that, after the deletions, E is empty. Since we delete at most one vertex from each pseudoline, it follows that $|E| \leq n$. □

Acknowledgments. The authors would like to thank Pankaj Agarwal, Boris Aronov, János Pach, and Pavel Valtr for helpful discussions concerning the problems studied in this paper.

References

1. P.K. Agarwal, B. Aronov, T. M. Chan and M. Sharir On levels in arrangements of lines, segments, planes, and triangles *Discrete Comput. Geom.*, 19 (1998), 315–331.
2. P.K. Agarwal, B. Aronov, J, Pach, R. Pollack and M. Sharir, Quasi-planar graphs have a linear number of edges, *Combinatorica* 17 (1997), 1–9.
3. P.K. Agarwal, B. Aronov and M. Sharir, On the complexity of many faces in arrangements of pseudo-segments and of circles, *Discrete Comput. Geom.*, The Goodman-Pollack festschrift, to appear.
4. P.K. Agarwal, E. Nevo, J. Pach, R. Pinchasi, M. Sharir and S. Smorodinsky, Lenses in arrangements of pseudodisks and their applications, *J. ACM*, to appear.
5. P.K. Agarwal and M. Sharir, Pseudoline arrangements: Duality. algorithms and applications, *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms* (2002), 781–790.
6. M. Ajtai, V. Chvátal, M. Newborn and E. Szemerédi, Crossing-free subgraphs, *Ann. Discrete Math* 12 (1982), 9–12.
7. G. Cairns and Y. Nikolayevsky, Bounds for generalized thrackles, *Discrete Comput. Geom.*, 23 (2000), 191–206.
8. T.M. Chan, On levels in arrangements of curves, *Proc. 41st IEEE Symp. Found. Comput. Sci.* (2000), 219–227.
9. K. Clarkson, H. Edelsbrunner, L. Guibas, M. Sharir and E. Welzl, Combinatorial complexity bounds for arrangements of curves and spheres, *Discrete Comput. Geom.* 5 (1990), 99–160.
10. K. Clarkson and P. W. Shor Applications of random sampling in computational geometry, II *Discrete Comput. Geom.* 4 (1989), 387–421.
11. T. K. Dey, Improved bounds on planar k-sets and related problems, *Discrete Comput. Geom.* 19 (1998), 373–382.
12. T. Dey and J. Pach, Extremal problems for geometric hypergraphs, *Discrete Comput. Geom.* 19 (1998), 473–484.
13. H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-verlag, Heidelberg, 1987.
14. I. Fáry, On straight-line representation of planar graphs, *Acta Sciientiarum Mathematicarum (Szeged)* 11 (1948), 229–233.
15. M. Katchalski and H. Last, On geometric graphs with no two edges in convex position, *Discrete Comput. Geom.* 19 (1998), 399–404.

16. F. T. Leighton, *Complexity Issues in VLSI*, MIT Press, Cambridge, MA, 1983.
17. F. Levi, Die Teilung der projektiven Ebene durch Gerade oder Pseudogerade, *Ber. Math-Phys. Kl. Sächs. Akad. Wiss.* 78 (1926), 256–267.
18. L. Lovász, J, Pach and M. Szegedy, On Conway's thrackle conjecture, *Discrete Comput. Geom.* 18 (1997), 369–376.
19. J. S. B. Mitchell and J. O'Rourke. Computational geometry column 42. *Internat. J. Comput. Geom. Appl.* (2001). Also in SIGACT News 32(3):63-72 (2001), Issue 120. See also: http://www.cs.smith.edu/ orourke/TOPP/
20. J. Pach, Geometric graph theory, in *Surveys in Combinatorics* (J.D. Lamb and D.A. Preece, eds.), London Mathematical Society Lecture Note Series 267, Cambridge University Press, 1999, 167–200.
21. M. Sharir and P.K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, New York, 1995.
22. L. Székely, Crossing numbers and hard Erdős problems in discrete geometry, *Combinatorics, Probability and Computing* 6 (1997), 353–358.
23. H. Tamaki and T. Tokuyama, A characterization of planar graphs by pseudo-line arrangements, *Proc. 8th Annu. Internat. Sympos. Algorithms Comput.* (ISAAC '97), Springer-Verlag Lecture Notes Comput. Sci., Vol. 1350, 1997, 133–142.
24. G. Tóth Point sets with many k-sets *Discrete Comput. Geom.* 26 (2001), 187–194.
25. W. T. Tutte, Toward a theory of crossing numbers, *J. Combinat. Theory* 8 (1970), 45–53.
26. P. Valtr, Graph drawings with no k pairwise crossing edges, *Lecture Notes Comput. Sci. Springer-Verlag* 1353 (1997), 205–218.
27. P. Valtr, On geometric graphs with no k pairwise parallel edges, *Discrete Comput. Geom.* 19 (1998), 461–469.
28. P. Valtr, Generalizations of Davenport-Schinzel sequences, in *Contemporary Trends in Discrete Mathematics* (J. Nešetřil, J. Kratochvil, F.S. Roberts, R.L. Graham, Eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 49 (1999), 349–389.

Fast Relative Approximation of Potential Fields

Martin Ziegler*

University of Paderborn, 33095 GERMANY; ziegler@upb.de

Abstract. Multi-evaluation of the Coulomb potential induced by N particles is a central part of N-body simulations. In 3D, known subquadratic time algorithms return approximations up to given *absolute* precision. By combining data structures from Computational Geometry with fast polynomial arithmetic, the present work obtains approximations of prescribable *relative* error $\varepsilon > 0$ in time $\mathcal{O}(\frac{1}{\varepsilon} N \cdot \operatorname{polylog} N)$.

1 Introduction

From the very beginning a major application of computers consisted in the simulation of physical objects. Nowadays for instance so-called N*-Body Simulations* have become quite standard a tool ranging from very small particles (Molecular Dynamics) to entire galaxies (Astrophysics). Among the different kinds of attracting/repelling forces governing the motion of such point-like objects, Coulomb's (equivalently: Newton Gravitation) is both most important and most challenging: because of its slow spacial decay ('*long-range interaction*'), a fixed object experiences influences from almost any other one in the system. Naively, this leads to quadratic cost $\mathcal{O}(N^2)$ for simulating its evolution over (physical) time step by step $t \mapsto t+\tau \mapsto t+2\tau \mapsto \ldots \mapsto t+T$.

Formally, let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^3$ denote the particles' positions in physical space and $c_1, \ldots, c_N \in \mathbb{R}$ their respective charges — in case of gravitational: their masses. The respective potential and force acted by particle $\#k$ upon particle $\#\ell$ is then given (up to constants) by

$$\varphi_{k\ell} \quad = \quad c_k \cdot \frac{1}{\|\boldsymbol{x}_k - \boldsymbol{x}_\ell\|_2} \qquad \text{and} \qquad \boldsymbol{f}_{k\ell} \quad = \quad c_k \cdot \frac{\boldsymbol{x}_k - \boldsymbol{x}_\ell}{\|\boldsymbol{x}_k - \boldsymbol{x}_\ell\|_2^3} \tag{1}$$

where $\|\boldsymbol{x}\|_2 = \sqrt{\sum_i x_i^2}$ denotes Euclid's norm. Consequently, the *total* potential or force experienced by particle ℓ,

$$\boldsymbol{\Phi}_\ell \quad = \quad \sum_{k \neq \ell} \varphi_{k\ell} \qquad \text{or} \qquad \boldsymbol{F}_\ell \quad = \quad \sum_{k \neq \ell} f_{k\ell} \ , \tag{2}$$

has to be computed for each $\ell = 1, \ldots, N$ repeatedly and thus better fast. A straightforward way according to (1,2) proceeds by evaluating N sums, each ranging over $N-1$ terms: $\mathcal{O}(N^2)$. Even when exploiting symmetry to save a factor of 2, this asymptotic

* Supported by PaSCo, DFG Graduate College no.693

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 140–149, 2003.
© Springer-Verlag Berlin Heidelberg 2003

severely limits scientists' desire to simulate $N \gtrsim 10^5$ particles over large scales of physical time $T \gtrsim 10^6 \tau$.

In the 2D case, a major breakthrough was achieved by GERASOULIS [6] who devised an algorithm with quasilinear cost $\mathcal{O}(N \cdot \log^2 N)$. His approach is based on fast arithmetic for complex polynomials, identifying $\mathbb{R}^2$ with $\mathbb{C}$.

In the practically more important 3D case, state-of-the-art implementations use Tree Codes and Multipole Expansions as invented by BARNES & HUT [1], taken care of worst-case distributions [4], and further improved by GREENGARD & ROKHLIN [5]. In this framework PAN & REIF & TATE [7] designed an algorithm using $\mathcal{O}(N \cdot \log N)$ many (cheap integer) operations and only $\mathcal{O}(p^2 \cdot N)$ floating point instructions to approximate the potential. They call $p = \log(C/\varepsilon)$ the "*accuracy*" of the output, where ε denotes the error bound to be achieved and $C = \sum |c_i|$ the total charge. Let us point out that C/ε, although being relative w.r.t. the total charge, does *not* necessarily describe the output precision in the relative sense; in fact, p specifies the number of terms considered in the multipole expansion for approximating the true value of the potential up to *absolute* error ε. Particularly in spacial areas of low field $\Phi_\ell \ll C$, this notion of approximation can turn out as unsatisfactory.

The present work complements [7] by approximating all Φ_ℓ up to arbitrary but fixed *relative* error $\varepsilon > 0$ within quasilinear time $\mathcal{O}(N \cdot \operatorname{polylog} N)$. A first step, Sect.2 recalls that the Euclidean norm in $\mathbb{R}^d$ permits approximation up to relative error $\varepsilon > 0$ by a certain other norm whose unit ball is a simplicial polyhedron having at most $f \leq \mathcal{O}(1/\varepsilon^{(d-1)/2})$ facets. Section 4's central Theorem 4 states that, when replacing in (1) the Euclid norm by such one, all Φ_ℓ together can be obtained exactly using only $\mathcal{O}(fN \cdot \log^{d+2} N)$ operations. Throwing things together this yields in 3D our main

Result 1 *Given $c_1, \ldots, c_N > 0$ and $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^3$, one can approximate $\Phi_1, \Phi_2, \ldots, \Phi_N$ according to (2) — i.e., the value of the gravitational/Coulomb potential induced by masses/charges of respective strengths c_k at positions $\boldsymbol{x}_k$ — each up to relative error $\varepsilon > 0$, using $\mathcal{O}(\frac{1}{\varepsilon} N \cdot \log^5 N)$ operations.*

Technically speaking, we combine the Range Tree data structure from Computational Geometry [2] with fast polynomial arithmetic. Both ingredients and the way to exploit them for the above problem are discussed in Section 5 and 6, respectively.

2 Approximating the Euclidean Norm

As is well-known [9], the unit ball $B = \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x}\| \leq 1\}$ of some norm $\|\cdot\|$ in $\mathbb{R}^d$ is a closed, centrally symmetric, bounded, convex set with 0 in its interior. Conversely, any such set B gives rise to a norm

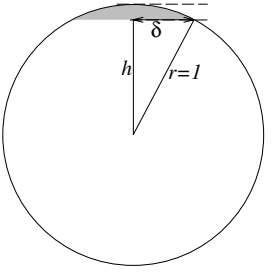
$$\mathbb{R}^d \quad \ni \quad \boldsymbol{x} \quad \mapsto \quad \inf\{\lambda > 0 : \boldsymbol{x}/\lambda \in B\}$$

having B as unit ball. We shall approximate the Euclidean norm up to relative error $\varepsilon > 0$ by replacing its smooth unit ball B with a simplicial[1] polyhedron P having 'few'

[1] each $(d-1)$-dimensional face (=*facet*) is a simplex

facets such that $(1-\varepsilon)B \subseteq P \subseteq B$. $P \cap (-P) \subseteq B$ is then a centrally symmetric, closed, bounded, and convex body containing $(1-\varepsilon)B$; it thus induces the desired norm.

Consider on the Euclidean unit ball B a spherical cap of small radius $\delta > 0$ as indicated to the right. Elementary geometry yields that the distance to the origin of any point on B's surface, after cutting off such a cap, is decreased from $r = 1$ to no less than $h = 1 - \mathcal{O}(\delta^2)$; cf. the figure below. Now recall [8] that the surface of the d-dimensional Euclidean ball can be covered by $\mathcal{O}(1/\delta)^{d-1}$ spherical caps of radius $\delta > 0$. In fact to prove this claim, ROGERS constructs a *triangulation* of B's surface of this size. We choose $\delta := \sqrt{\varepsilon}$ and take that triangulation (rather than his caps) to obtain P as above.



$h = \sqrt{1-\delta^2} \approx 1 - \delta^2/2$
for $\delta \ll 1$

EPCOT ILLUSTRATES
ROGERS' CONSTRUCTION IN 3D

3 Dealing with the 1-Norm

In Sect.6 we shall prove the central

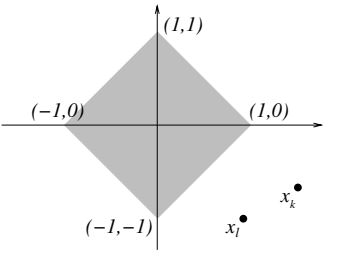
Theorem 2. *Let $\psi_1, \ldots, \psi_N$ denote rational functions in $z \in \mathbb{C}$, given by the coefficients of their respective nominator and denominator polynomials, all having degree at most $\Delta \in \mathbb{N}$. Upon input of these $2\Delta N$ coefficients, of $z_1, \ldots, z_M \in \mathbb{C}$, and of $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N \in \mathbb{R}^d$ and $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M \in \mathbb{R}^d$, it is possible to jointly compute the M sums*

$$\sum_{\substack{k=1 \\ \boldsymbol{a}_k \prec \boldsymbol{b}_\ell}}^{N} \psi_k(z_\ell), \quad \ell = 1, \ldots, M \qquad \textit{in time} \quad \mathcal{O}\big((M + \Delta N) \cdot \log^d(N) \cdot \log^2(\Delta N)\big)$$

where $\boldsymbol{a} \prec \boldsymbol{b} \;:\Leftrightarrow\; \forall i = 1...d : \; a_i \prec_i b_i$ *and* $\prec_i \in \{\leq, <, >, \geq\}$ *is arbitrary but fixed.*

Expressions of the form $\sum_{k:\boldsymbol{a}_k \prec \boldsymbol{b}_\ell}$ over a semi-group for given $\boldsymbol{a}$'s and one $\boldsymbol{b}_\ell$ are known in Computational Geometry as Orthogonal Range Queries. However in our case, *several* such queries are to be answered for $\ell = 1, \ldots, M$; furthermore, one has to account for the preprocessing and the more complex semi-group operations involving rational functions.

To further exemplify Theorem 2, let us apply it to the computation of all $\Phi_1, \ldots, \Phi_N$ in (2) for the case of $\|\boldsymbol{x}\|_1 = \sum |x_i|$. Observe that in $\mathbb{R}^d$ this 1-norm is a simplicial polyhedral norm with 2^d facets. Moreover, restricted to some *hyper-quadrant* $\prod_{i=1}^d [0, \pm\infty)$, $\|\cdot\|_1$ is a *linear* map. In particular



$$\frac{c_k}{\|\boldsymbol{x}_k - \boldsymbol{x}_\ell\|_1} \quad = \quad \frac{c_k}{\sum_{i=1}^d x_{ki} \; - \; z}\Bigg|_{z:=\sum_{i=1}^d x_{\ell i}} \quad =: \quad \psi_k(z)$$

provided $\boldsymbol{x}_k \geq \boldsymbol{x}_\ell$ holds componentwise. Also notice that ψ_k is a rational function in $z \in \mathbb{R}$ of degree $\Delta = 1$. By letting $\boldsymbol{a}_k := \boldsymbol{x}_k, \boldsymbol{b}_\ell := \boldsymbol{x}_\ell, M := N$, one can thus compute the N sums $\sum_{k:\boldsymbol{x}_k \prec \boldsymbol{x}_\ell} \psi_{k,\ell}$ according to Theorem 2 within total time $\mathcal{O}(N \cdot \log^{d+2} N)$.

In fact by partitioning $\mathbb{R}^d \setminus \{0\}$ into disjoint (half-open/-closed) hyper-quadrants, it is possible to decompose each Φ_ℓ into 2^d many sub-sums, each one calculateable within the above time bound. In 2D for instance,

$$\Phi_\ell \quad = \quad \Phi_\ell^{(\geq,>)} \; + \; \Phi_\ell^{(<,\geq)} \; + \; \Phi_\ell^{(\leq,<)} \; + \; \Phi_\ell^{(>,\leq)}$$

where for example

$$\Phi_\ell^{(<,\geq)} \quad = \quad \sum_{\substack{k: x_{k1} < x_{\ell 1} \\ \wedge x_{k2} \geq x_{\ell 2}}} c_k \cdot \overbrace{1/\|\boldsymbol{x}_k - \boldsymbol{x}_\ell\|_1}^{=1/(x_{k2}-x_{k1}+z), \quad z:=x_{\ell 1}-x_{\ell 2}}$$

is again of the form covered by Theorem 2. Generalizing to higher dimensions in a straight-forward way, we thus have the following

Corollary 3. *A 'modified' Coulomb/gravitational potential of N particles, namely w.r.t. the 1-norm $\|\cdot\|_1$ on $\mathbb{R}^d$ instead of the Euclidean one, can be evaluated at all particles' positions exactly within only $\mathcal{O}(N \cdot \log^{d+2} N)$ operations.*

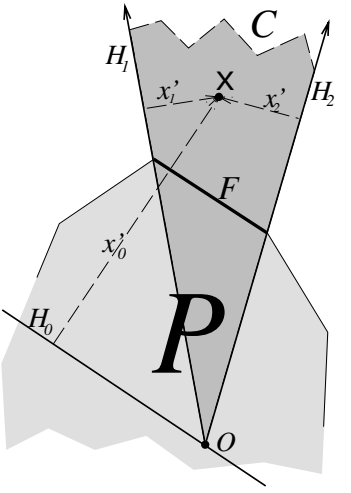
4 Dealing with Simplicial Polyhedral Norms

After this example, we now consider an arbitrary simplicial polyhedral norm $\|\cdot\|$ and its respective unit ball P. Again deal separately with each cone $C = \bigcup_{\lambda>0} \lambda F$ spanned by the origin 0 and one of P's facets F; then calculate the partial sums $\Phi_\ell^C = \sum_{k:\boldsymbol{x}_k - \boldsymbol{x}_\ell \in C} \varphi_{k,\ell}$.

In case of the 1-norm, each such C was some hyper-quadrant; fortunately the general C can be reduced to such hyper-quadrants by a simple linear transformation. To this end consider the d walls of C (simplex!) and their respective supporting hyperplanes $H_1, \ldots, H_d$, oriented such that C lies on their positive sides; let x'_i denote the signed distance of $\boldsymbol{x} \in \mathbb{R}^d$ to H_i. Linearity of $' : \mathbb{R}^d \to \mathbb{R}^d$, $\boldsymbol{x} \mapsto \boldsymbol{x}'$ yields:

$$\boldsymbol{x}_k - \boldsymbol{x}_\ell \in C \quad \Leftrightarrow \quad \boldsymbol{x}'_k - \boldsymbol{x}'_\ell \in [0,\infty)^d \ .$$

Finally consider H_0, the hyperplane supporting F translated to pass through the origin and oriented such that C lies on its positive side; let x'_0 denote the signed distance of $\boldsymbol{x}$ to H_0. Then for $\boldsymbol{x}_k - \boldsymbol{x}_\ell \in C$, $\|\boldsymbol{x}_k - \boldsymbol{x}_\ell\| = x'_{k0} - x'_{\ell 0}$; hence



UNIT BALL P OF A 2D POLYHEDRAL NORM

$$\sum_{k:\boldsymbol{x}_k - \boldsymbol{x}_\ell \in C} \varphi_{k,\ell} \quad = \quad \sum_{\boldsymbol{x}'_k \geq x'_\ell} \psi'_k(x'_{\ell 0}), \qquad \psi'_k(z) := c_k/(x'_{k0} - z)$$

is of the form covered by Theorem 2 and therefore can be calculated within $\mathcal{O}(N \cdot \log^{d+2} N)$ steps; indeed, each of the N linear transformations $\mathbb{R}^d \ni \boldsymbol{x}_k \mapsto (x'_{k0}, \boldsymbol{x}'_k) \in \mathbb{R}^{1+d}$ is computable with constant cost (d fixed). And that completes the proof to

Theorem 4. *Based on Theorem 2, one can evaluate a 'modified' Coulomb/ gravitational potential of N particles, namely w.r.t. a simplicial polyhedral norm $\|\cdot\|$ on $\mathbb{R}^d$ instead of the Euclidean one, exactly within $\mathcal{O}(fN \cdot \log^{d+2} N)$ operations where f denotes the number of facets of $P := \{x \in \mathbb{R}^d : \|x\| \leq 1\}$.*

It thus remains to prove Theorem 2. To this end, we shall construct and exploit a Range Tree data structure as is known from Computational Geometry. However for our purpose, nodes correspond to (ranges of) rational functions rather than to (ranges of) points; similarly, a query does not report points but performs multi-evaluations. The costs inferred by the latter and some other operations on rational functions are recalled in the next section before proceeding in Sect.6 to the actual proof of Theorem 2.

5 Fast Arithmetic for Rational Functions

An important ingredient to our algorithm is fast arithmetic for polynomials in one complex variable z, based on Fast Fourier Transformation (FFT). For later reference, we briefly collect the running times for two basic operations.

Lemma 5. *a) The product of two polynomials of degree at most n, both input and output represented by respective lists of coefficients, can be computed in time $\mathcal{O}(n \cdot \log n)$.* [3, THEOREM 2.8]

b) A polynomial of degree at most n can be multi-evaluated at m given points simultaneously within total time $\mathcal{O}\big((m+n)\cdot\log^2 n\big)$. [3, COROLLARY 3.20]

It is worth while emphasizing that the above upper complexity bounds, although quoted from a theory book, are well-known in practice and correspond to highly efficient and numerically stable algorithms.

Lemma 5 obviously holds as well for rational functions in one complex variable z represented in the form *nominator/denominator*, both being polynomials of maximal degree n and given by their respective coefficient lists. To be precise when talking about arithmetic on rational functions, let us agree the value of $\psi = \alpha/\beta$ with $\alpha, \beta \in \mathbb{C}[Z]$ at some zero z of β to be undefined $\psi(z) := \infty$ even in case of a *removable* singularity, i.e., even when $\alpha(z) = 0$ is a zero of same or higher order. Furthermore $z \pm \infty := \infty$, $z \cdot \infty := \infty$, and $z/\infty := \infty$ for all $z \in \mathbb{C} \cup \{\infty\}$.

Corollary 6. *a) The product of two rational functions of degree at most n can be computed in time* $\mathcal{O}(n \cdot \log n)$.

b) A rational function of degree at most n can be evaluated at m given points simultaneously within total time $\mathcal{O}\big((m+n)\cdot\log^2 n\big)$.

c) The sum of two rational functions of degree at most n can be computed in time $\mathcal{O}(n \cdot \log n)$.

The last claim holds by calculating the sum of two rational functions according to

$$\psi_1 \;+\; \psi_2 \quad = \quad \frac{\alpha_1(z)}{\beta_1(z)} \;+\; \frac{\alpha_2(z)}{\beta_2(z)} \quad = \quad \frac{\alpha_1(z)\cdot\beta_2(z) + \alpha_2(z)\cdot\beta_1(z)}{\beta_1(z)\cdot\beta_2(z)}$$

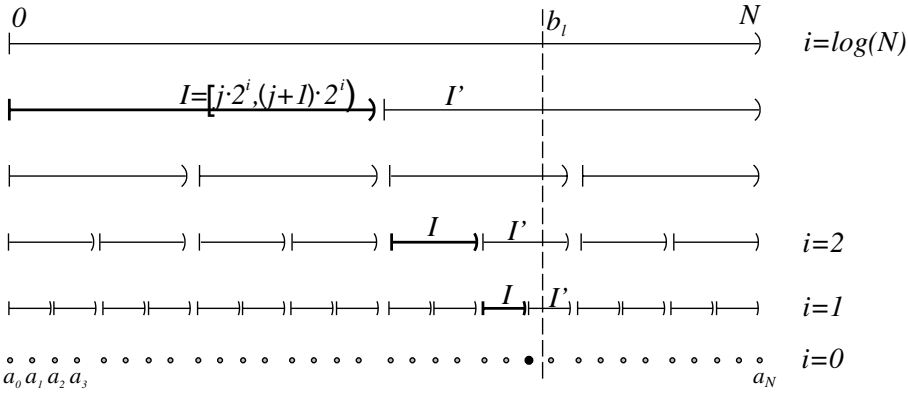
using three polynomial multiplications and two additions. The result is then of degree no more than $\deg(\psi_1) + \deg(\psi_2)$.

6 Interval Tree of Rational Functions

We finally come to prove the announced

Theorem 2. *Let $\Psi = \{\psi_1, \ldots, \psi_N\}$ denote a family of rational functions in $z \in \mathbb{C}$ of degree at most $\Delta \in \mathbb{N}$, given by their respective coefficients. Upon further input of a family $Z = \{z_0, \ldots, z_{M-1}\} \subset \mathbb{C}$, of $A = \{\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{N-1}\} \subset \mathbb{R}^d$, and of $B = \{\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{M-1}\} \subset \mathbb{R}^d$, it is possible to jointly compute the M sums* $\sum_{k:\boldsymbol{a}_k \prec \boldsymbol{b}_\ell} \psi_k(z_\ell), \qquad \ell = 0, ..., M-1$
in time $\mathcal{O}\big((M+\Delta N)\cdot\log^d(N)\cdot\log^2(\Delta N)\big)$.

The proof proceeds by induction on d, starting with dimension 1. W.l.o.g. let N a power of 2, $a_0 < a_1 < \ldots < a_{N-1} < a_N := \infty$, $\quad' \prec' = '<'$.



EXAMPLE OF AN Interval Tree, FOR SIMPLICITY ON DATA $a_k \equiv k$.

The Interval Tree is a complete binary tree with N leafs; each level $i = 0, 1, \ldots, \log(N)$ corresponds to a partition of the real interval $[a_0, \infty) \subseteq \mathbb{R}$ into $N/2^i$ sub-intervals $I = \big[a_{j\cdot 2^i}, a_{(j+1)\cdot 2^i}\big)$, $j = 0, \ldots, N/2^i - 1$. For each such $I = I_{i,j}$, $j < N/2^i - 1$, let $I' := I_{i,j+1}$ denote its right neighbor.

The algorithm performs the following steps:

A) Construct the above tree.
B) For each $i = 0, \ldots, \log(N)$
C) For each $\ell = 0, \ldots, M-1$
determine the unique I_{ij} containing b_ℓ, $j = 0, \ldots, N/2^i - 1$;
endfor ℓ; this gives rise to a partition $Z_{ij} := \{z_\ell : b_\ell \in I_{ij}\}$ of Z.
D) For each $j = 0, \ldots, N/2^i - 1$
i) compute (the coefficients of) $\psi_I := \sum_{k:a_k \in I} \psi_k, \quad I = I_{ij}$;
ii) multi-evaluate ψ_I on those z_ℓ with $b_\ell \in I'$ as obtained in C)
endfor j
endfor i.
E) For each $\ell = 0, \ldots, M-1$
compose the desired sum $\sum_{k:a_k < b_\ell} \psi_k(z_\ell)$ by adding up at most $\log(N)$ many of the precomputed $\psi_I(z_\ell)$.

Looking at the above sketch of an Interval Tree, one easily confirms that the latter composition is always feasible. Concerning the analysis, Step E) obviously contributes $\mathcal{O}(N \cdot \log N)$ to the total running time. Step A) costs $\mathcal{O}(N \cdot \log N)$. Within the j-loop, Step C) can be performed in time $\mathcal{O}\big(M \cdot \log(N/2^i)\big)$ by means of binary search. Step i) of the i-loop costs $\mathcal{O}\big(\Delta 2^i \cdot \log(\Delta N)\big)$ according to Corollary 6d); indeed, ψ_I is a rational function of degree at most $\Delta 2^i$ and furthermore the sum of two others, say ψ_J and $\psi_{J'}$, the already have been computed during pass $i-1$. Step ii) finally takes $\mathcal{O}\big((m_{ij} + \Delta 2^i) \cdot \log^2(\Delta N)\big)$ according to Corollary 6b), $m_{ij} := |Z_{ij}|$.

Since the Z_{ij} partition Z, $\sum_j m_{ij} \le M$; one single pass of the i-loop costs

$$\mathcal{O}\big(M \cdot \log(N/2^i)\big) \quad + \quad \mathcal{O}\big(\Delta N \cdot \log(\Delta N)\big) \quad + \quad \mathcal{O}\big((M + \Delta N) \cdot \log^2(\Delta N)\big)$$

which is $\leq \mathcal{O}\big((M+\Delta N)\cdot\log^2(\Delta N)\big)$. Counter i running from 0 to $\log(N)$ infers another factor of $\log(N)$, and that completes the induction start. □

Observe that already the just proved case $d=1$ allows to reproduce GERASOULIS' result [6] on generalized Hilbert Matrices (although with one additional log-factor):

Corollary 7. *Given $c_1,\ldots,c_N \in \mathbb{C}$ and pairwise distinct $z_1,\ldots,z_N \in \mathbb{C}$, one can compute the N sums* $\displaystyle\sum_{k\neq\ell}\frac{c_k}{z_k-z_\ell},\quad \ell=1,\ldots,N$ *within time* $\mathcal{O}(N\cdot\log^3 N)$.

Indeed, $\psi_k(z) := \frac{c_k}{z_k-z}$ is a rational function of degree $\Delta=1$; hence according to Theorem 2, both parts of $\sum_{k\neq\ell}\psi_k(z_\ell) = \sum_{k<\ell}\psi_k(z_\ell) + \sum_{k>\ell}\psi_k(z_\ell)$ can be obtained for $\ell=1,\ldots,N$ within the claimed running time by choosing $a_k := k \in \mathbb{R}^{\mathbf{1}}$ and $b_\ell := \ell \in \mathbb{R}^{\mathbf{1}}$.

7 Range Tree of Rational Functions

The induction step $d \mapsto d+1$ in our proof for Theorem 2 borrows Range Trees from Computational Geometry [2]. For data $(\alpha_k, \boldsymbol{a}_k) \in \mathbb{R}^{1+d}$, this is defined recursively to be a (1D) Interval Tree on data $\alpha_k \in \mathbb{R}$ with nodes consisting of d-dimensional Range Trees on (appropriate ranges of) data $\boldsymbol{a}_k \in \mathbb{R}^d$.

In our case, we use it to store, process, and multi-evaluate sums of rational functions. For $(\beta_\ell, \boldsymbol{b}_\ell) \in \mathbb{R}^{1+d}$, $\ell=0,\ldots,M-1$, the algorithm proceeds as follows:

A) Construct an Interval Tree on data α_k, $k=0,\ldots,N-1$.
B) For each $i=0,\ldots,\log(N)$
C) For each $j=0,\ldots,N/2^i-1$
i) determine $A_{ij} := \{\boldsymbol{a}_k : \alpha_k \in I_{ij}\}$ and $\Psi_{ij} := \{\psi_k : \alpha_k \in I_{ij}\}$;
ii) determine $B_{ij} := \{\boldsymbol{b}_\ell : \beta_\ell \in I_{ij}\}$ and $Z_{ij} := \{z_\ell : \beta_\ell \in I_{ij}\}$;
iii) apply Theorem 2 to these *sub*collections $\Psi_{i,j} \subseteq \Psi$, $Z_{i,j+1} \subseteq Z$, $A_{i,j} \subseteq A$, and $B_{i,j+1} \subseteq B$.
This will yield all sums $\displaystyle\sum_{\substack{k:\alpha_k\in I\\ \boldsymbol{a}_k<\boldsymbol{b}_\ell}}\psi_k(z_\ell)$ for $\beta_\ell \in I'$.
endfor j
endfor i.
D) For each $\ell=0,\ldots,M-1$
compose the desired sum $\displaystyle\sum_{\substack{k:\alpha_k<\beta_\ell\\ \boldsymbol{a}_k<\boldsymbol{b}_\ell}}\psi_k(z_\ell)$ by adding up at most $\log(N)$ many of the precomputed $\displaystyle\sum_{\substack{k:\alpha_k\in I\\ \boldsymbol{a}_k<\boldsymbol{b}_\ell}}\psi_k(z_\ell)$ with $\beta_\ell \in I'$.

Again, the latter composition is always feasible due to the Interval Tree's properties. Like in Section 6, the running times are dominated by Step iii) of the innermost loop. Let

$m_{ij} = |Z_{ij}| = |B_{ij}|$; observe $n := |\Psi_{ij}| = |A_{ij}| = 2^i \leq N$ and again $\sum_j m_{ij} \leq M$. By induction hypothesis, a whole run of the j-loop thus infers cost

$$\mathcal{O}\Big(\sum_{j=0}^{N/2^i-1} \big((m_{ij} + \Delta n) \cdot \log^d(n) \cdot \log^2(\Delta n)\big) \Big) \leq \mathcal{O}\big((M + \Delta N) \cdot \log^d(N) \cdot \log^2(\Delta N)\big)$$

and the i-loop gives another factor $\log(N)$. □

8 Conclusion

We presented a quasilinear time algorithm for a central problem of N-body simulations: determining for each particle the potential it experiences by the other particles. Other than previous approaches, our approximation achieves guaranteed *relative* errors. Whereas [7] used a synthesis of Algebraic and Numerical Computing, we combine Algebraic Computing with Computational Geometry. This technique yields (Theorem 4) that for a simplicial polyhedral norm with 'few' facets rather than the Euclidean one, the potential fields can be evaluated *exactly* in quasilinear time. Since the Euclid norm permits approximation by simplicial polyhedral norms with 'few' facets (Sect.2), the claim follows.

In fact, Theorem 4 immediately generalizes to higher integral (positive or negative) *powers* of simplicial polyhedral norms:

Theorem 8. *Fix $d \in \mathbb{N}$, $q \in \mathbb{Z}$, and let $\|\cdot\|$ denote a simplicial polyhedral norm in $\mathbb{R}^d$ with f facets. Then one can compute, upon input of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^d$ and $c_1, \ldots, c_N \in \mathbb{R}$, all*

$$\Phi_\ell \quad = \quad \sum_{k \neq \ell} c_k \cdot \|\boldsymbol{x}_k - \boldsymbol{x}_\ell\|^q, \qquad \ell = 1, \ldots, N$$

exactly using $\mathcal{O}\big(fqN \cdot \log^d(N) \cdot \log^2(qN)\big)$ operations.

To be fair, Result 1 requires *positive* strengths $c_k > 0$. In case of gravitation that condition is naturally met by all masses; whereas Coulomb charges (ions) do occur with different signs. In practice one would simply treat the positive and negative ones separately and then subtract their respective contributions. However in strict terms of worst-case analysis, doing so violates the relative error bounds: the difference of two approximations may lead to infinite relative deviations.

Another issue, the high powers of $\log(N)$ in our results might already in 3D not be as negligible as the common notion "*quasilinear time*" suggests. Of course, $\mathcal{O}(N \cdot \mathrm{polylog}\, N)$ does pay off eventually against the naive $\mathcal{O}(N^2)$ approach as $N \to \infty$; *when* break-even occurs is about to be examined in practical implementations. Theoretically, Fractional Cascading [2] might help removing at least one factor $\log(N)$. Also it is quite conceivable that multi-evaluation (Lemma 5b) has only complexity $\mathcal{O}\big((m+n) \cdot \log^{\mathbf{1}} n\big)$, thus saving another $\log(N)$.

However the most central open question is of course this: *Can 3D Coulomb fields be evaluated exactly in subquadratic time?*

References

1. Barnes, J., and P. Hut: "A hierarchical $\mathcal{O}(N \cdot \log N)$ force-calculation algorithm", pp. 446–449 in *Nature* Vol.**324** (1986).
2. de Berg, M., and M. van Kreveld, and M. Overmars, and O. Cheong: "Computational Geometry", Springer (2000).
3. Bürgisser, P., and M. Clausen, and M.A. Shokrollahi: "Algebraic Complexity Theory", Springer (1997).
4. Callahan, P.B., and S.R. Kosaraju: "A Decomposition of Multidimensional Point Sets with Application to k-Nearest-Neighbors and n-Body Potential Fields", pp. 67–90 in *J. ACM* vol.**42** (1995).
5. Cheng, H., and L. Greengard, and V. Rokhlin: "A Fast Adaptive Multipole Algorithm in Three Dimensions", pp. 468–498 in *Journal of Computational Physics* Vol.**155** (1999).
6. Gerasoulis, A.: "A Fast Algorithm for the Multiplication of Generalized Hilbert Matrices with Vectors", pp. 179–188 in *Mathematics of Computation* Vol.**50** No.**181** (1988).
7. Pan, V., and J.H. Reif, and S.R. Tate: "The Power of Combining the Techniques of Algebraic and Numerical Computing: Improved Approximate Multipoint Polynomial Evaluation and Improved Multipole Algorithms", pp. 703–713 in *Proc. 32th Annual IEEE Symposium on Foundations of Computer Science* (FOCS'1992); see also
   Reif, J.H., and S.R. Tate: "N-body Simulation I: Fast Algorithms for Potential Field Evaluation and Trummer's Problem", *Tech. Report* No.**N-96-002**, Univ. of North Texas Dept. of Computer Science (1996).
8. Rogers, C.A.: "Covering a Sphere with Spheres", pp. 157–164 in *Mathematika* Vol.**10** (1963).
9. Rudin, W.: "Functional Analysis", McGraw-Hill (1991).

The One-Round Voronoi Game Replayed

Sándor P. Fekete[1] and Henk Meijer[2]

[1] Department of Mathematical Optimization, TU Braunschweig, Pockelsstr. 14, D-38106 Braunschweig, Germany, s.fekete@tu-bs.de.
[2] Department of Computing and Information Science, Queen's University, Kingston, Ontario K7L 3N6, Canada, henk@cs.queensu.ca

Abstract. We consider the one-round Voronoi game, where player one ("White", called "Wilma") places a set of n points in a rectangular area of aspect ratio $\rho \leq 1$, followed by the second player ("Black", called "Barney"), who places the same number of points. Each player wins the fraction of the board closest to one of his points, and the goal is to win more than half of the total area. This problem has been studied by Cheong et al. who showed that for large enough n and $\rho = 1$, Barney has a strategy that guarantees a fraction of $1/2 + \alpha$, for some small fixed α.

We resolve a number of open problems raised by that paper. In particular, we give a precise characterization of the outcome of the game for optimal play: We show that Barney has a winning strategy for $n \geq 3$ and $\rho > \sqrt{2}/n$, and for $n = 2$ and $\rho > \sqrt{3}/2$. Wilma wins in all remaining cases, i.e., for $n \geq 3$ and $\rho \leq \sqrt{2}/n$, for $n = 2$ and $\rho \leq \sqrt{3}/2$, and for $n = 1$. We also discuss complexity aspects of the game on more general boards, by proving that for a polygon with holes, it is NP-hard to maximize the area Barney can win against a given set of points by Wilma.

Keywords: Computational geometry, Voronoi diagram, Voronoi game, Competitive facility location, NP-hardness.

1 Introduction

When determining success or failure of an enterprise, *location* is one of the most important issues. Probably the most natural way to determine the value of a possible position for a facility is the distance to potential customer sites. Various geometric scenarios have been considered; see the extensive list of references in the paper by Fekete, Mitchell, and Weinbrecht [6] for an overview.

One particularly important issue in location theory is the study of strategies for competing players. See the surveys by Tobin, Friesz, and Miller [7], by Eiselt and Laporte [4], and by Eiselt, Laporte, and Thisse [5].

A simple geometric model for the value of a position is used in the *Voronoi game*, which was proposed by Ahn et al. [1] for the one-dimensional scenario and extended by Cheong et al. [2] to the two- and higher-dimensional case. In this game, a site s "owns" the part of the playing arena that is closer to s than to any other site. Both considered a two-player version with a finite arena Q. The players, White ("Wilma") and Black ("Barney"), place points in Q; Wilma plays first. No point that has been occupied can be

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 150–161, 2003.
© Springer-Verlag Berlin Heidelberg 2003

changed or reused by either player. Let W be the set of points that were played by the end of the game by Wilma, while B is the set of points played by Barney. At the end of the game, a Voronoi diagram of $W \cup B$ is constructed; each player wins the total area of all cells belonging to points in his or her set. The player with the larger total area wins.

Ahn et al. [1] showed that for a one-dimensional arena, i.e., a line segment $[0, 2n]$, Barney can win the *n-round game*, in which each player places a single point in each turn; however, Wilma can keep Barney's winning margin arbitrarily small. This differs from the *one-round game*, in which both players get a single turn with n points each: Here, Wilma can force a win by playing the odd integer points $\{1, 3, \ldots, 2n-1\}$; again, the losing player can make the margin as small as he wishes. The used strategy focuses on "key points". The question raised in the end of that paper is whether a similar notion can be extended to the two-dimensional scenario. We will see in Section 3 that in a certain sense, this is indeed the case.

Cheong et al. [2] showed that the two- or higher-dimensional scenario differs significantly: For sufficiently large $n \geq n_0$ and a square playing surface Q, the second player has a winning strategy that guarantees at least a fixed fraction of $1/2 + \alpha$ of the total area. Their proof uses a clever combination of probabilistic arguments to show that Barney will do well by playing a random point. The paper gives rise to some interesting open questions:

- How large does n_0 have to be to guarantee a winning strategy for Barney? Wilma wins for $n = 1$, but it is not clear whether there is a single n_0 for which the game changes from Wilma to Barney, or whether there are multiple changing points.
- For sufficiently "fat" arenas, Barney wins, while Wilma wins for the degenerate case of a line. How exactly does the outcome of the game depend on the aspect ratio of the playing board?
- What happens if the number of points played by Wilma and Barney are not identical?
- What configurations of white points limit the possible gain of black points? As candidates, square or hexagonal grids were named.
- What happens for the multiple-round version of the game?
- What happens for asymmetric playing boards?

For rectangular boards and arbitrary values of n, we will give a precise characterization of when Barney can win the game. If the board Q has aspect ratio ρ with $\rho \leq 1$, we prove the following:

- Barney has a winning strategy for $n \geq 3$ and $\rho > \sqrt{2}/n$, and for $n = 2$ and $\rho > \sqrt{3}/2$. Wilma wins in all remaining cases, i.e., for $n \geq 3$ and $\rho \leq \sqrt{2}/n$, for $n = 2$ and $\rho \leq \sqrt{3}/2$, and for $n = 1$.
- If Wilma does not play her points on an orthogonal grid, then Barney wins the game.

In addition, we hint at the difficulties of more complex playing boards by showing the following:

- If Q is a polygon with holes, and Wilma has made her move, it is NP-hard to find a position of black points that maximizes the area that Barney wins.

This result is also related to recent work by Dehne, Klein, and Seidel [3] of a different type: They studied the problem of placing a single black point within the convex hull of a set of white points, such that the resulting black Voronoi cell in the unbounded Euclidean plane is maximized. They showed that there is a unique local maximum.

The rest of this paper is organized as follows. After some technical preliminaries in Section 2, Section 3 shows that Barney always wins if Wilma does not place her points on a regular orthogonal grid. This is used in Section 4 to establish our results on the critical aspect ratios. Section 5 presents some results on the computational complexity of playing optimally in a more complex board. Some concluding thoughts are presented in Section 6.

2 Preliminaries

In the following, Q is the playing board. Q is a rectangle of *aspect ratio* ρ, which is the ratio of the length of the smaller side divided by the length of the longer side. Unless noted otherwise (in some parts of Section 5), both players play n points; W denotes the n points played by Wilma, while B is the set of n points played by Barney. All distances are measured according to the Euclidean norm. For a set of points P, we denote by $V(P)$ the (Euclidean) Voronoi diagram of P. We call a Voronoi diagram $V(P)$ a regular grid if

- all Voronoi cells are rectangular, congruent and have the same orientation;
- each point $p \in P$ lies in the center of its Voronoi cell.

If e is a Voronoi edge, $C(e)$ denotes a Voronoi cell adjacent to e. If $p \in P$, then $C(p)$ denotes the Voronoi cell of p in $V(P)$. $\partial C(p)$ is the boundary of $C(p)$ and $|C(p)|$ denotes the area of $C(p)$. $|e|$ denotes the length of an edge e. Let x_p and y_p denote the x- and y-coordinates of a point p.

3 A Reduction to Grids

As a first important step, we reduce the possible configurations that Wilma may play without losing the game. The following holds for boards of any shape:

Lemma 1. *If $V(W)$ contains a cell that is not point symmetric, then Barney wins.*

Proof. Suppose there is a point $w \in W$ and a line through w that does not bisect the area of $C(w)$; say, we have $|C(w)|/2+2\alpha$ on one side of the line. This means that by placing a point close to w, Barney can claim at least $|C(w)|/2 + 2\alpha - \alpha/n$ of the cell $C(w)$. In each other cell $C(w)$ of $V(W)$ Barney can place a point close enough to w to claim an area of at least $|C(w)|/2 - \alpha/n$. Therefore Barney has gained at least $|Q|/2 + \alpha$.

This means that for each $w \in W$, any line $l(\varphi)$ that encloses an angle φ with the positive x-axis and passes through w bisects $C(w)$. Let $r(\varphi)$ be the distance of w to the point on the boundary of $C(w)$ that is stabbed by a ray emanating from w at angle φ. As $\partial C(w)$ is a convex curve, $r(\varphi)$ is a continuous function. Furthermore, we see that $|C(p)| = \frac{1}{2}\int_0^{2\pi} r^2(\varphi)d\varphi$, and the portion of $C(p)$ enclosed between angles φ_1 and φ_2

is $\frac{1}{2}\int_{\varphi_1}^{\varphi_2} r^2(\varphi)d\varphi$. As an infinitesimal rotation of $l(\varphi)$ about w does not change the area on either side, it follows that $r(\varphi) = r(-\varphi)$ for any φ, meaning that $C(w)$ is point symmetric. □

The following theorem is based on this observation and will be used as a key tool for simplifying our discussion in Section 4.

Theorem 1. *If the board is a rectangle and if $V(W)$ is not a regular grid, then Barney wins.*

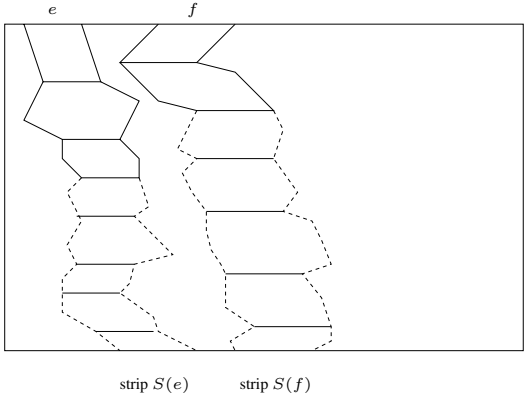


Fig. 1. Playing board with two strips.

Proof. We assume that Barney cannot win, and will show that this implies that $V(W)$ is a regular grid. By Lemma 1, we may assume that all cells of $V(W)$ are point symmetric. Let e_0 be a Voronoi edge of $V(W)$ on the top side of the board. Consider the Voronoi cell C_0 adjacent to e_0. Because C_0 is point symmetric, it contains an edge e_1 that is parallel to e_0 with $|e_0| = |e_1|$. Let C_1 be the cell adjacent to and below e_1. It contains an edge e_2 with $|e_2| = |e_1|$. Similarly define the cells $C_2, C_3, \ldots$. So cell C_i lies below C_{i-1}. Therefore there is a cell C_{k-1} such that e_k lies on the bottom edge of the board. We call $S(e_0) = \{C_0, C_1, C_2, \ldots, C_{k-1}\}$ the strip of e_0. Because C_i is convex, any horizontal line that intersects the board has an intersection with $S(e_0)$ of length $\geq |e_0|$. Consider two different Voronoi edges, e and f, on the top side of the board, with their respective strips $S(e)$ and $S(f)$. Because Voronoi cells are convex and do not have corners with angles of size π, these strips cannot intersect, i.e. do not have a cell in common. For an illustration, see Figure 1. Let S be the collection of strips of e for all Voronoi edges e of $V(W)$ on the top side of the board. The intersection of a line with S has a length which is at least as large as the sum of the lengths of the edges along the top side of the board. Because strips do not intersect, this intersection is exactly as long as the top side of the board. This implies that S covers the whole board and that any horizontal line that intersects the board has an intersection with $S(e_0)$ of length exactly equal to $|e_0|$.

Let e be the left most edge on the top side of the board. The left hand side of $S(e)$ is the left hand side of the board. This implies that each cell in $S(e)$ is a rectangle. By

the same argument, each cell in $V(W)$ is a rectangle. Let v and w be two points in W such that $C(v)$ and $C(w)$ in $V(W)$ have an edge e in common. The distance between v and e is the same as the distance between w and e. Because both $C(v)$ and $C(w)$ are point symmetric and rectangular, it follows that they have the same size. So $V(W)$ is a regular grid. □

4 Critical Aspect Ratios

In this section we prove the main result of this paper: if $n \geq 3$ and $\rho > \sqrt{2}/n$, or $n = 2$ and $\rho > \sqrt{3}/2$, then Barney wins. In all other cases, Wilma wins. The proof proceeds by a series of lemmas. We start by noting the following easy observation.

Lemma 2. *Barney wins, if and only if he can place a point p that steals an area strictly larger than $|Q|/2n$ from W.*

Proof. Necessity is obvious. To see sufficiency, note that Wilma is forced to play her points in a regular grid. Barney places his first point p such that it gains an area of more than $|Q|/2n$. Let w be a point in W. If Barney places a point on the line through w and p, sufficiently close to w but on the opposite side of p, he can claim almost half of the Voronoi cell of w. By placing his remaining $n - 1$ points in this fashion, he can claim a area larger than $|Q|/2n$. □

Next we take care of the case $n = 2$; this lemma will also be useful for larger n, as it allows further reduction of the possible arrangements Wilma can choose without losing.

Lemma 3. *If $n = 2$ and $\rho > \sqrt{3}/2$, then Barney wins. If the aspect ratio is smaller, Barney loses.*

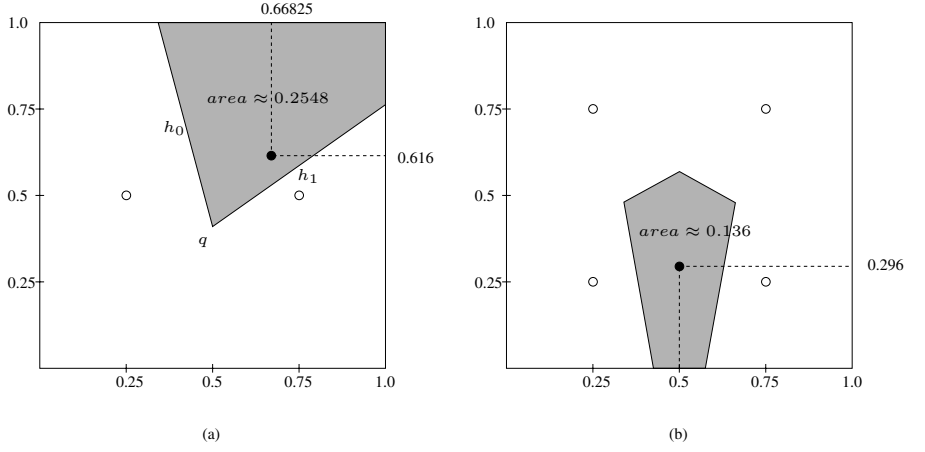


Fig. 2. Barney has gained more than a quarter (a) more than an eighth (b) of the playing surface.

Proof. Assume without loss of generality that the board has size ρ by 1. Suppose that the left bottom corner of Q lies on the origin. By Theorem 1 we know that Wilma has to place her points at $(0.5, \rho/4)$ and $(0.5, 3\rho/4)$ or at $(0.25, \rho/2)$ and $(0.75, \rho/2)$. If Wilma places her points at $(0.5, \rho/4)$ and $(0.5, 3\rho/4)$, then it is not hard to show that she will lose. So assume that Wilma places her points at $(0.25, \rho/2)$ and $(0.75, \rho/2)$. For Barney to win, he will have to gain more than $\rho/4$ with his first point. Suppose Barney places his point at location p. Without loss of generality, assume that $x_p \geq 0.5$ and $y_p \geq \rho/2$. If $y_p = \rho/2$ then Barney gains at most $\rho/4$, so we may assume that $y_p > \rho/2$. Placing a point p with $x_p > 0.75$ is not optimal for Barney: moving p in the direction of $(0.5, \rho/2)$ will increase the area gained. It is not hard to show that for $x_p = 0.75$, Barney cannot gain an area of size $\rho/4$. So we may assume that $0.5 \leq x_p < 0.75$. Let b_0 be the bisector of p and $(0.25, \rho/2)$. Let b_1 be the bisector of p and $(0.75, \rho/2)$. Let q be the intersection of b_0 and b_1. The point q lies on the vertical line through $x = 0.5$. If q lies outside the board Q, then $|C(p)| < \rho/4$, so assume that q lies in Q. Let h_0 be the length of the line segment on b_0, between q and the top or left side of the board. Let h_1 be the length of the line segment on b_1, between q and the top or right side of the board. Consider the circle C center at q which passes through p, $(0.25, \rho/2)$ and $(0.75, \rho/2)$.

If b_0 does not intersect the top of the board then neither does b_1. In this case we can increase $|C(p)|$ by moving p to the left on C and we can use this to show that $|C(p)| < \rho/4$. If both b_0 and b_1 intersect the top of the board we have $h_0 \leq h_1$. We can increase h_1 and decrease h_0 by moving P to the right on C. So $|C(p)|$ can be increased until b_1 intersects the top right corner of the board. If b_0 intersect the top of the board and b_1 intersect the right top corner we have $h_0 \leq h_1$. If we move p to the right on C, both h_0 and h_1 will decrease. The area $|C(p)|$ will increase as long as $h_0 < h_1$ and reaches its maximum value when $h_0 = h_1$. Therefore the maximum exists when at the moment that p approaches $(0.75, \rho/2)$, we have $h_0 > h_1$. When $p = (0.75, \rho/2)$, we have $h_0 = \rho - y_q$ and $h_1 = \sqrt{(1/4 + (\rho - 2y_q)^2)}$. From $h_0 > h_1$ we can derive that $\rho > \sqrt{3}/2$. With his second point Barney can gain an area of size $0.25 - \varepsilon$ for an arbitrary small positive value of ε by placing the point close to $(0.25, \rho/2)$. So Barney can gain more than half the board.

If the aspect ratio is $\leq \sqrt{3}/2$, Barney can gain at most $\rho/4$ with his first move by placing his point at $(x, \rho/2)$ with $0.25 < x < 0.75$. It can be shown that with his second point he can gain almost, but not exactly a quarter. □

The gain for Barney is small if ρ is close to $\sqrt{3}/2$. We have performed computer experiments to compute the gain for Barney for values of $\rho > \sqrt{3}/2$. Not surprisingly, the largest gain was for $\rho = 1$. If the board has size 1×1, Barney can gain an area of approximately 0.2548 with his first point, by placing it at (0.66825,0.616) as illustrated in Figure 2(a).

Lemma 4. *Suppose that the board is rectangular and that $n = 4$. If Wilma places her point on a regular 2×2 grid, Barney can gain 50.78% of the board.*

Proof. Assume that the board has size $\rho \times 1$. By Lemma 1 we know that Wilma has to place her points on the horizontal line at height $\rho/2$, on the vertical line at $x = 0.5$ or at the points $(0.25, \rho/4)$, $(0.25, 3\rho/4)$, $(0.75, \rho/4)$ and, $(0.75, 3\rho/4)$. If Wilma does not

place her points on a line, it can be computed that Barney wins at least $\rho(1/8 + 1/128)$ by placing a point at $(0.5, \rho/4)$. In addition Barney can gain a little more than $3\rho/8 - \varepsilon$ by placing his remaining three points at $(0.25 - 4\varepsilon/3, \rho/4)$, $(0.25 - 4\varepsilon/3, 3\rho/4)$, and $(0.75 + 4\varepsilon/3, 3\rho/4)$. So Barney will gain a total area of size $\rho(1/2 + 1/128) - \varepsilon$. As $1/2 + 1/128 = 50.78125$, the result follows. □

The value in the above lemma is not tight. For example, if Wilma places her point in a 2×2 grid on a square board, we can compute the area that Barney can gain with his first point. If Barney places it at (0.5,0.296), he gains approximately 0.136. For an illustration, see Figure 2(b). By placing his remaining three points at $(0.25-4\varepsilon/3, 0.25)$, $(0.25-4\varepsilon/3, 0.75)$, and $(0.75+4\varepsilon/3, 0.75)$ Barney can gain a total area of size of around $0.511 - \varepsilon$ for arbitrary small positive ε. For non-square boards, we have found larger wins for Black. This suggests that Barney can always gain more than 51% of the board if Wilma places her four points in a 2×2 grid.

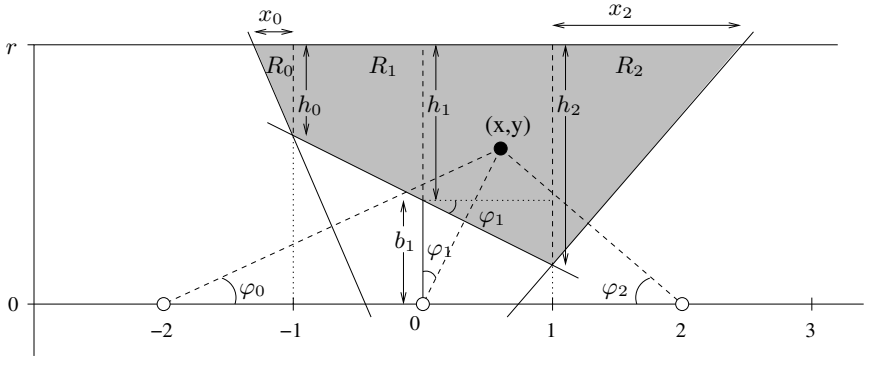


Fig. 3. Wilma has placed at least three stones on a line.

The above discussion has an important implication:

Corollary 1. *If $n \geq 3$, then Wilma can only win by placing her points in a $1 \times n$ grid.*

This sets the stage for the final lemma:

Lemma 5. *Let $n \geq 3$. Barney can win if $\rho > \sqrt{2}/n$; otherwise, he loses.*

Proof. It follows from Corollary 1 that Wilma should place her points in a $1 \times n$ grid. Assume that Q has size $2r \times 2n$ and that the left bottom point of Q lies at $(-3, -r)$ and the top right point at $(2n - 3, r)$. Wilma must place her points at $(-2, 0), (0, 0), (2, 0)$, $\ldots$, $(2n - 4, 0)$. From Lemma 2 we know that in order to win, Barney has to find a location $p = (x, y)$ with $|V(p)| > 2r$.

If $r > \sqrt{3}$, we know from Lemma 3 that Barney can take more than a quarter from two neighboring cells of Wilma, i.e. Barney takes more than $8r/4 = 2r$ with his first point. Therefore assume that $r \leq \sqrt{3}$. We start by describing the size and area of a

potential Voronoi cell for Barney's first point. Without loss of generality, we assume that $p = (x, y)$ with $y, x \geq 0$ is placed in the cell of Wilma's point $(0, 0)$, so $x \leq 1$, $y \leq r$.

If $y > 0$ and if Barney gains parts of three cells of $V(W)$ with his first point, we have a situation as shown in Figure 3. It is not hard to see that he can steal from at most three cells: p has distance more than 2 from all cells not neighboring on Wilma's cells $V(-2, 0)$ and $V(2, 0)$, which is more than the radius of $\sqrt{r^2 + 1} \leq 2$ of those cells with respect to their center points. We see that

$$b_1 = \frac{y}{2} + \frac{x^2}{2y}, \qquad \tan\varphi_1 = \frac{x}{y}, \qquad \tan\varphi_2 = \frac{y}{2 - x}.$$

As shown in Figure 3, the Voronoi cell of p consists of three pieces: the quadrangle R_1 (stolen from $V(0, 0)$), the triangle R_0 (stolen from $V(-2, 0)$), and the triangle R_2 (stolen from $V(2, 0)$). Furthermore,

$$|R_1| = 2h_1 = 2(r - b_1) = 2r - y - \frac{x^2}{y} \text{ and } |R_2| = \frac{x_2 h_2}{2},$$

with $h_2 = r - b_1 + \tan\varphi_1 = r - \frac{y}{2} - \frac{x^2}{2y} + \frac{x}{y}$, and $x_2 = h_2 \tan\varphi_2 = \frac{\left(r - \frac{y}{2} - \frac{x^2}{2y} + \frac{x}{y}\right)y}{2-x}$, so

$$|R_2| = \frac{\left(ry - \frac{y^2}{2} - \frac{x^2}{2} + x\right)^2}{2y(2 - x)}, \text{ and analogously } |R_0| = \frac{\left(ry - \frac{y^2}{2} - \frac{x^2}{2} - x\right)^2}{2y(2 + x)}.$$

We first consider $r \leq \sqrt{2}$ and assume that Barney can win, i.e., he can gain an area larger than $2r$ with his first point. If $y = 0$, then $|V(p)| = 2r$, so we may assume that $y > 0$. From Lemma 3, we know that Barney will not win if he only steals from two of Wilma's cells, so we may assume that Barney steals from three cells. Therefore we can use results from previous equations; from $|R_0| + |R_1| + |R_2| > 2r$ some simplification ultimately yields

$$y^3\left(\frac{y}{2} - 2\sqrt{2}\right) > x^2\left(2 - \frac{x^2}{2} - y^2\right).$$

As the left hand side is negative for $0 < y \leq \sqrt{2}$, we conclude that the right hand side must also be negative; clearly, it is minimized for $x = 1$, so we get

$$y^3\left(\frac{y}{2} - 2\sqrt{2}\right) > \left(2 - \frac{1}{2} - y^2\right),$$

and conclude that $\sqrt{2} \geq y \geq \sqrt{3/2}$, yielding the contradiction

$$4 \;\geq\; \frac{y^4}{2} + y^2 > \frac{3}{2} + 2\sqrt{2}y^3 \;>\; 4.$$

So the best Barney can do is gain an area of size $2r$ with all his points and tie the game. However, notice that the contradiction also holds if $|R_0| + |R_1| + |R_2| = 2r$. So Barney cannot gain an area of size $2r$ if he places his point at (x, y) with $y > 0$ and

steals from three cells of $V(W)$. In Lemma 3 it was shown that Barney will gain less than $2r$ if he places his point at (x, y) with $y > 0$ and steals from two cells of $V(W)$. Therefore Barney must place his points at (x, y) with $y = 0$. This reduces the problem to a one-dimensional one, and we know from [1] that in that case Barney will lose.

Secondly we consider $\sqrt{2} < r \leq \sqrt{3}$. Suppose Barney places his first point at $(0, y)$ with $y > 0$. Clearly he will steal from three cells of $V(W)$. From previous equations we derive that

$$|R_0| + |R_1| + |R_2| = \frac{r^2 y}{2} - \frac{r y^2}{2} + \frac{y^3}{8} + 2r - y,$$

so because of $y > 0$ we have

$$|R_0| + |R_1| + |R_2| \ > \ 2r \Leftrightarrow y^2 - 4ry + 4r^2 - 8 \ > \ 0 \Leftrightarrow 0 < y < 2(r - \sqrt{2}).$$

So Barney wins if he places a point at $(0, y)$ with $0 < y < 2(r - \sqrt{2})$. □

The total area $|R_0| + |R_1| + |R_2|$ is maximal for $y^* = (4r - 2\sqrt{r^2 + 6})/3$. Experiments have confirmed that Barney maximizes the area for his first point at $(0, y^*)$.

Summarizing, we get:

Theorem 2. *If $n \geq 3$ and $\rho > \sqrt{2}/n$, or $n = 2$ and $\rho > \sqrt{3}/2$, then Barney wins. In all other cases, Wilma wins.*

5 A Complexity Result

The previous section resolves most of the questions for the one-round Voronoi game on a rectangular board. Clearly, there are various other questions related to more complex boards; this is one of the questions raised in [2]. Lemma 1 still applies if Wilma's concern is only to avoid a loss. Moreover, it is easily seen that all of Wilma's Voronoi cells must have the same area, as Barney can steal almost all the area of the largest cell by placing two points in it, and no point in the smallest cell. For many boards, both of these conditions may be impossible to fulfill. It is therefore natural to modify the game by shifting the critical margin that decides a win or a loss. We show in the following that it is NP-hard to decide whether Barney can beat a given margin for a polygon with holes, and all of Wilma's stones have already been placed. (In a non-convex polygon, possibly with holes, we measure distances according to the geodesic Euclidean metric, i.e., along a shortest path within the polygon.)

Theorem 3. *For a polygon with holes, it is NP-hard to maximize the area Barney can claim, even if all of Wilma's points have been placed.*

Proof. We give an outline of the proof, based on a reduction from Planar 3SAT, which is known to be NP-complete [8]. For clearer description, we sketch the proof for the case where Barney has fewer points to play; in the end, we hint at what can be done to make both point sets the same size. First, the planar graph corresponding to an instance I of Planar 3Sat is represented geometrically as a *planar rectilinear layout*, with each vertex corresponding to a horizontal line segment, and each edge corresponding to a

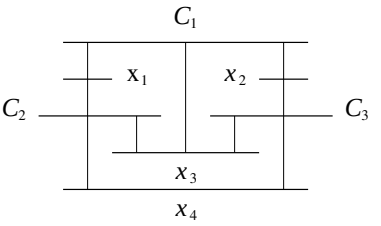


Fig. 4. A geometric representation of the graph G_I for the Planar 3SAT instance $I = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee x_4)$.

vertical line segment that intersects precisely the line segments corresponding to the two incident vertices. There are well-known algorithms (e.g., [11]) that can achieve such a layout in linear time and linear space. See Figure 4.

Next, the layout is modified such that the line segments corresponding to a vertex and all edges incident to it are replaced by a loop – see Figure 5. At each vertex corresponding to a clause, three of these loops (corresponding to the respective literals) meet. Each loops gets represented by a very narrow corridor.

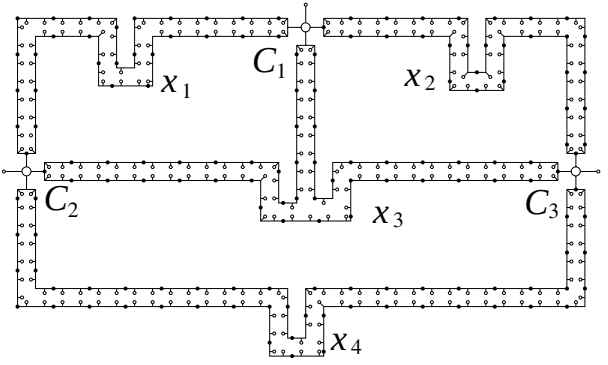


Fig. 5. A symbolic picture of the overall representation: The location of white points is indicated by white dots (with area elements on variable loops not drawn for the sake of clarity). The location of black points (indicated by black dots) corresponds to the truth assignment $x_1 = 0$, $x_2 = 1$, $x_3 = 0$, $x_4 = 1$, which satisfies I. See Figure 6 for a closeup of the gadgets.

Now we place a sequence of extra *area gadgets* at equal distances $2d_1$ along the variable loop. Let n_i be the number of elements along the loop for variable x_i, and let $N = \sum_{i=1}^{n} n_i$, and $\varepsilon = 1/N^3$. (By construction, N is polynomially bounded.) The two basic ingredients of each such gadget is a white point "guarding" an area element of size $A = 1/N$, i.e., being at distance $d_1 + \varepsilon$ from it. Finally, for each clause, we place an extra gadget as shown in Figure 6. Similar to the area gadgets along the variable loops, it consists of a white point guarding an area element of size $A = 1/N$ at distance $d_2 + \varepsilon$. Thus, the overall number of white points is $|W| = N + m$. By making the corridors

sufficiently narrow (say, $1/N^3$ wide), the overall area for the corridors is small (e.g., $O(1/N^2)$.) The total area of the resulting polygon is $1 + m/N + O(1/N^2)$.

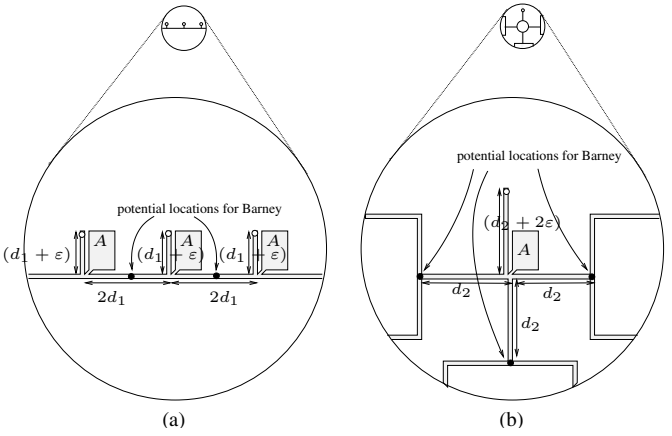


Fig. 6. Area gadget (left) and clause gadgets (right)

Now it is easy to see that for any satisfying truth assignment for I, there is a position of $N/2$ black points that steals all the area elements, i.e., claims an area of $1 + m/N$.

To see the converse, assume Barney can claim an area of at least $q + m/N$, i.e., he can steal all area elements. Note that no position of a black point can steal more than two area elements on a variable; stealing two requires placing it at less than distance $d_1 + \varepsilon$ from both of them. As the $N/2$ black points must form a perfect matching of the N area elements, we conclude that there are only two basic ways to cover all area elements of a variable x_i by not more than $n_i/2$ black points, where each location may be subject to variations of size $O(\varepsilon)$ One of these perfect matchings corresponds to setting x_i to *true*, the other to *false*. If this assignment can be done in a way that also steals all area elements of clause gadgets, we must have a satisfying truth assignment. □

By adding some extra area elements (say, of size $3A$) right next to $N/2 + m$ of the white points along variable gadgets, and increasing $|B|$ to $N + m$, we can modify the proof to apply to the case in which $|W| = |B|$. Similarly, it is straighforward to shift the critical threshold such that Wilma is guaranteed a constant fraction of the board.

6 Conclusion

We have resolved a number of open problems dealing with the one-round Voronoi game. There are still several issues that remain open. What can be said about achieving a fixed margin of win in all of the cases where Barney can win? We believe that our above techniques can be used to resolve this issue. As we can already quantify this margin if Wilma plays a grid, what is still needed is a refined version of Lemma 1 and Theorem 1 that guarantees a fixed margin as a function of the amount that Wilma deviates from a grid. Eventually, the guaranteed margin should be a function of the aspect ratio. Along

similar lines, we believe that it is possible to resolve the question stated by [2] on the scenario where the number of points played is not equal.

There are some real-life situations where explicit zoning laws enforce a minimum distance between stones; obviously, our results still apply for the limiting case. It seems clear that Barney will be at a serious disadvantage when this lower bound is raised, but we leave it to future research to have a close look at these types of questions.

The most tantalizing problems deal with the multiple-round game. Given that finding an optimal set of points for a single player is NP-hard, it is natural to conjecture that the two-player, multiple round game is PSPACE-hard. Clearly, there is some similarity to the game of Go on an $n \times n$ board, which is known to be PSPACE-hard [9] and even EXPTIME-complete [10] for certain rules.

However, some of this difficulty results from the possibility of capturing stones. It is conceivable that at least for relative simple (i.e., rectangular) boards, there are less involved winning strategies. Our results from Section 4 show that for the cases where Wilma has a winning strategy, Barney cannot prevent this by any probabilistic or greedy approach: Unless he blocks one of Wilma's key points by placing a stone there himself (which has probability zero for random strategies, and will not happen for simple greedy strategies), she can simply play those points like in the one-round game and claim a win. Thus, analyzing these key points may indeed be the key to understanding the game.

References

1. H.-K. Ahn, S.-W. Cheng, O. Cheong, M. Golin, and R. van Oostrum. Competitive facility location along a highway. In *Proceedings of the Ninth Annual International Computing and Combinatorics Conference*, volume 2108, pages 237–246, 2001.
2. O. Cheong, S. Har-Peled, N. Linial, and J. Matousek. The one-round voronoi game. In *Proceedings of the Eighteenth Annual ACM Symposium on Computational Geometry*, pages 97–101, 2002.
3. F. Dehne, R. Klein, and R. Seidel. Maximizing a Voronoi region: The convex case. In *Proceedings of the Thirteenth Annual International Symposium on Algorithms and Computation*, volume 2518, pages 624–634, 2001.
4. H. Eiselt and G. Laporte. Competitive spatial models. *European Journal of Operational Research*, 39:231–242, 1989.
5. H. Eiselt, G. Laporte, and J.-F. Thisse. Competitive location models: A framework and bibliography. *Transportation Science*, 27:44–54, 1993.
6. S. P. Fekete, J. S. B.Mitchell, and K. Weinbrecht. On the continuous Weber and k-median problems. In *Proceedings of the Sixteenth Annual ACM Symposium on Computational Geometry*, pages 70–79, 2000.
7. T. Friesz, R. Tobin, and T. Miller. Existence theory for spatially competitive network facility location models. *Annals of Operations Research*, 18:267–276, 1989.
8. D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11, 2:329–343, 1982.
9. D. Lichtenstein and M. Sipser. Go is polynomial-space hard. *Journal of the ACM*, 27:393–401, 1980.
10. J. Robson. The complexity of go. In *Information Processing: Proceedings of IFIP Congerss*, pages 413–4417, 1983.
11. P. Rosenstiehl and R. E. Tarjan. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete and Computational Geometry*, 1:343–353, 1986.

Integrated Prefetching and Caching with Read and Write Requests

Susanne Albers[1] and Markus Büttner[1]

Institute of Computer Science, Freiburg University,
Georges-Köhler-Allee 79, 79110 Freiburg, Germany.
{salbers,buettner}@informatik.uni-freiburg.de

Abstract. All previous work on integrated prefetching/caching assumes that memory reference strings consist of read requests only. In this paper we present the first study of integrated prefetching/caching with both read and write requests. For single disk systems we analyze popular algorithms such as *Conservative* and *Aggressive* and give tight bounds on their approximation ratios. We also develop a new algorithm that performs better than *Conservative* and *Aggressive*. For parallel disk systems we present a general technique to construct feasible schedules. The technique achieves a load balancing among the disks. Finally we show that it is NP-complete to decide if an input can be served with f fetch and w write operations, even in the single disk setting.

1 Introduction

Prefetching and caching are powerful and extensively studied techniques to improve the performance of storage hierarchies. In prefetching missing memory blocks are loaded from slow memory, e.g. a disk, into cache before their actual reference. Caching strategies try to keep actively referenced blocks in cache. Both techniques aim at reducing processor stall times that occur when requested data is not available in cache. Most of the previous work investigated prefetching and caching in isolation although they are strongly related: When prefetching a block, one has to evict a block from cache in order to make room for the incoming block. Prefetch operations initiated too early can harm the cache configuration. Prefetch operations started too late diminish the effect of prefetching. Therefore, there has recently been considerable research interest in integrated prefetching and caching [1,2,4,5,6,7,8,9]. The goal is to develop strategies that coordinate prefetching and caching decisions.

All the previous work on integrated prefetching/caching assumes that memory reference strings consist of read requests only, i.e. we only wish to read data blocks. In other words, memory blocks are read-only and do not have to be written back to disk when they are evicted from cache. However, in practice reference strings consist of both read and write requests. In a write request we wish to modify and update a given data block. Of course, modified blocks must be written to disk when they are evicted from cache.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 162–173, 2003.
© Springer-Verlag Berlin Heidelberg 2003

In this paper we present the first study of integrated prefetching/caching with read and write requests. It turns out that integrated prefetching/caching is considerably more complicated in the presence of write requests. The problem is that prefetch and write operations compete with each other and it is not clear when to schedule which disk operation. Moreover, compared to the read-only case, it is not true anymore that in a prefetch operation we always evict the block from cache whose next request is furthest in the future. To save a write-back operation it might be better to evict an unmodified block, even if it is requested again soon. Finally, even if it were known when to initiate write operations, there is no simple rule that determines which blocks to write to disk.

Cao et al. [4] introduced a formal model for integrated prefetching/caching. We also use this model but generalize it to take into account read and write requests. We are given a request sequence $\sigma = r_1, \ldots, r_n$ consisting of n requests. Each request specifies the block to be accessed and the reference type. If $r_i = b_i$, then r_i is a read request to block b_i. If $r_i = b_i^*$, then the reference is a write request where we want to modify b_i. We first assume that all blocks reside on a single disk. To serve a request, the requested block must be in cache. The cache can simultaneously hold k blocks. Serving a request to a block in cache takes 1 time unit. If a requested block is not in cache, then it must be fetched from disk, which takes F time units. A fetch operation may overlap with the service of requests to blocks already in cache. If a fetch, i.e. a prefetch, of a block is initiated at least F requests before the reference to the block, then the block is in cache at the time of the request and no processor stall time is incurred. If the fetch is started only i, $i < F$, requests before the reference, then the processor has to stall for $F - i$ time units until the fetch is finished. When a fetch operation is initiated, a block must be evicted from cache to make room for the incoming block. A block that was modified since the last time it was brought into cache can only be evicted if it has been written back to disk after its last write request. Such a write-back operation takes W time units and can be scheduled any time before the eviction. If the operation overlaps with the service of $i \leq W$ requests, then $W - i$ units of processor stall time are incurred to complete the write operation. In this submission, unless otherwise stated, we assume for simplicity that $W = F$. The goal is to minimize the total processor stall time incurred on the entire request sequence. This is equivalent to minimizing the elapsed time, which is the sum of the processor stall time and the length of the request sequence. We emphasize here that the input σ is completely known in advance.

To illustrate the problem, consider a small example. Let $\sigma = b_1, b_2^*, b_2, b_3, b_4^*, b_3, b_4, b_3, b_5, b_1, b_4, b_2$. Assume that we have a cache of size $k = 4$ and that initially blocks b_1, b_2, b_3 and b_4 reside in cache. Let $F = W = 3$. The first missing block is b_5. We could initiate the fetch for b_5 when starting the service of the request b_2^*. The fetch would be executed while serving requests b_2^*, b_2 and b_3. When starting this fetch, we can only evict b_1, which is requested again after b_5. We could initiate the fetch for b_1 when serving request b_5 and evict b_3. Two units of stall time would be incurred before request b_1, so that the total elapsed time is equal to 14 time units. A better option is

to write b_2 back to disk after request b_2^* and then to initiate a fetch for b_5 by evicting b_2. Both disk operations finish in time before request b_5 because the write operation may overlap with the service of the read request to b_2. When serving request b_5 we could start fetching b_2 by evicting b_3. Again this operation would be finished in time so that the elapsed time of this schedule is equal to 12 time units.

Integrated prefetching and caching is also interesting in parallel disk systems. Suppose that we have D disks and that each memory block always resides on exactly one of the disks. Fetch and write operations on different disks may be executed in parallel. Of course we can take advantage of the parallelism given by a multiple disk system. If the processor incurs stall time to wait for the completion of a fetch or write operation, then fetch and write operations executed in parallel on other disks also make progress towards completion during that time. Again we wish to minimize the total elapsed time.

Previous work: As mentioned before, all previous work on integrated prefetching/caching [1,2,4,5,6,7,8,9] assumes that request sequences consist of read request only. Cao et al. [4], who initiated the research on integrated prefetching/caching, introduced two popular algorithms called *Conservative* and *Aggressive* for the single disk problem. *Conservative* performs exactly the same cache replacements as the optimum offline paging algorithm [3] but starts each fetch at the earliest possible point in time. Cao et al. showed that *Conservative* achieves an approximation ratio of 2, i.e. for any request sequence the elapsed time of *Conservative*'s schedule is at most twice the elapsed time of an optimal schedule. This bound is tight. The *Aggressive* algorithm starts prefetch operations at the earliest reasonable point in time. Cao et al. proved that *Aggressive* has an approximation ratio of at most $\min\{1 + F/k, 2\}$ and showed that this bound is tight for $F = k$. In practical applications, F/k is typically 0.02. Kimbrel and Karlin [7] analyzed *Conservative* and *Aggressive* in parallel disk systems and showed that the approximation guarantees are essentially equal to D. They also presented an algorithm called *Reverse Aggressive* and proved an approximation guarantee of $1 + DF/k$.

In [1] it was shown, that an optimal prefetching/caching schedule for a single disk can be computed in polynominal time based on a linear programming approach. The approach was extended to parallel disk systems and gave a D-approximation algorithm for the problem of minimizing the stall time of a schedule. The algorithm uses $D - 1$ extra memory locations in cache.

Our contribution: This paper is an in-depth study of integrated prefetching/caching with read an write requests. We first address the single disk problem. In Section 2 we investigate implementations of *Conservative* and *Aggressive* and prove that *Conservative* has an approximation ratio of 3. We show that this bound is tight. We also show that *Aggressive* achieves an approximation guarantee of $\min\{2 + 2F/k, 4\}$ and that this bound is tight for $F = k$. Hence, surprisingly, for large ratios of F/k *Conservative* performs better than *Aggressive*. This is in contrast to the algorithms' relative performance in the read-only case.

In Section 3 we develop a new prefetching/caching algorithm that has an approximation ratio of 2 and hence performs better than *Conservative* and *Aggressive* for all F and k. The basic idea of the new strategy is to delay cache replacements for a few time units.

The complexity of integrated prefetching/caching in the presence of write requests is unknown. However, Section 4 indicates that the problem is probably NP-hard. More precisely we prove that it is NP-complete to decide if a given request sequence can be served with at most f fetch and w write operations.

In Section 5 we study systems with D parallel disks. To speed up write operations, many parallel disk systems have the option of writing memory blocks back to an arbitrary disk and not necessarily to the disk where the block was stored previously. Of course, old copies of a block become invalid. Hence the disk where a given block resides may change over time. We present a general technique for constructing feasible prefetching/caching schedules in two steps. In the first step an algorithm determines fetch and write operations without considering on which disks the involved blocks reside. The second step assigns disks to all the fetch and write operations so that a load balancing is achieved for all the disks. Using a parallel, synchronized implementation of the *Conservative* algorithm in step 1 we obtain schedules whose elapsed time is at most 5 times the elapsed time of an optimal schedule plus an additive term that depends on the initial disk configuration. Replacing *Conservative* by *Aggressive* and investing $\lceil D/2 \rceil$ additional memory locations in cache the ratio of 5 drops to 4.

2 Analysis of *Conservative* and *Aggressive*

In this section we study the single disk setting. We extend the algorithms *Conservative* and *Aggressive* to request sequences consisting of both read and write requests and analyze their performance. *Conservative* executes exactly the same cache replacements as the optimum offline paging algorithm MIN [3] while initiating a fetch at the earliest reasonable point in time, i.e. the block to be evicted should not be requested before the block to be fetched. Modified blocks to be evicted may be written back to disk anytime before their eviction.

Theorem 1. *For any request sequence σ, the elapsed time of Conservative's schedule is at most 3 times the elapsed time of an optimal schedule. This bound is nearly tight, i.e. there are request sequences for which the ratio of Conservative's elapsed time to OPT's elapsed time is at least* $(3F + 2)/(F + 2)$.

Proof. The upper bound of 3 is easy to see. Consider an arbitrary request sequence σ and suppose that *Conservative* performs m cache replacements. In the worst case each replacements takes $2F$ time units: The algorithm may need $W = F$ time units to write the block to be evicted to disk; F time units are incurred to fetch the new block. Let $Cons(\sigma)$ be the total elapsed time of *Conservative*'s schedule. Then $Cons(\sigma) \leq |\sigma| + 2Fm$. *Conservative*'s cache replacements are determined by the MIN algorithm, which incurs the minimum number of

cache replacements for any request sequence. Thus the optimum algorithm performs at least m cache replacements on σ, each of which takes at least F time units. We have $OPT(\sigma) \geq \max\{|\sigma|, Fm\}$ and hence $Cons(\sigma) \leq 3 \cdot OPT(\sigma)$.

For the construction of the lower bound we assume $k \geq 3$ and use $k-2$ blocks $A_1, \ldots, A_{k-2}$ as well as $k-2$ block $B_1, \ldots, B_{k-2}$ and three auxiliary blocks X, Y and Z. The requests to blocks $A_1, \ldots, A_{k-2}$, X, Y and Z will always be read requests whereas the requests to $B_1, \ldots, B_{k-2}$ will always be write requests. We use the asterisk to denote write requests, i.e. B_i^* is a write request modifying block B_i, $1 \leq i \leq k-2$. The request sequence is composed of subsequences σ_A and σ_B, where $\sigma_A = Z^F, A_1, Z^F, A_2, \ldots, Z^F, A_{k-2}$ and $\sigma_B = B_1^*, \ldots, B_{k-2}^*$. Let $\sigma' = \sigma_A, \sigma_B, Z, X, \sigma_A, \sigma_B, Z, Y$. The request sequence σ is an arbitrary number of repetitions of σ', i.e. $\sigma = (\sigma')^i$, for some positive integer i. To establish the lower bound we compare *Conservative*'s elapsed time on σ' to OPT's elapsed time on σ'. In the analysis the two algorithms start with different cache configurations but at the end of σ' the algorithms are again in their initial configuration.

We assume that initially *Conservative* has blocks $A_1, \ldots, A_{k-2}$, Y and Z in cache. During the service of the first σ_A in σ' *Conservative* first evicts Y to load B_1. This fetch overlaps with the service of requests. While serving the first σ_B, *Conservative* evicts B_i to load B_{i+1}, for $i = 1, \ldots, k-3$. Each operation generates $2F$ units of stall time because the evicted block has to be written to disk and the fetch cannot overlap with the service of requests. Then *Conservative* evicts B_{k-2} to fetch X. Again the operation takes $2F$ time units but can overlap with the service of the request to Z. The algorithm now has $A_1, \ldots, A_{k-2}$, X and Z in cache. It serves the second part of σ' in the same way as the first part except that in the beginning X is evicted to load B_1 and in the end B_{k-2} is evicted to load Y so that the final cache configuration is again $A_1, \ldots, A_{k-2}$, Y and Z. To serve σ', *Conservative* needs $Cons(\sigma') = 2((k-2)(F+1)+1+(k-2)(2F+1)) = 2((k-2)(3F+2)+1)$ time units.

For the analysis of OPT on σ' we assume that OPT has initially $B_1, \ldots, B_{k-2}$, Y and Z in cache. Blocks $B_1, \ldots, B_{k-2}$ and Z are never evicted. In the first part of σ' OPT evicts Y to load A_1 and then evicts A_i to load A_{i+1}, for $i = 1, \ldots, k-3$. These fetches are executed during the sevice of the requests to Z. While serving σ_B OPT evicts A_{k-2} to load X and the cache then contains $B_1, \ldots, B_{k-2}$, X and Z. In the second part of σ' the operations are the same except the roles of X and Y interchange. OPT's cache configuration at the end of σ' is again $B_1, \ldots, B_{k-2}$, Y and Z. The elapsed time is $OPT(\sigma') = 2((k-2)(F+1) + \max\{F, k-1\} + 1)$.

Hence, for $F < k$, the ratio of *Conservative*'s elapsed time to OPT's elapsed time on σ' is

$$\frac{Cons(\sigma')}{OPT(\sigma')} = \frac{(k-2)(3F+2)+1}{(k-2)(F+1)+k} \geq \frac{3F+2}{F+2}$$

and the desired bound follows by repeating σ' often enough. □

The *Aggressive* algorithm proposed by Cao et al. [4] works as follows. Whenever the algorithm is not in the middle of a fetch, it determines the next block b in the request sequence missing in cache as well as the block b' in cache whose next request is furthest in the future. If the next request to b is before the next

request to b', then *Aggressive* initiates a fetch for b evicting b' from cache. We consider two extension of this algorithm to request sequences with read and write requests. If b' has to be written back to disk, then *Aggressive1* executes the write operation immediately before initiating the fetch for b and incurs F units of stall time before that fetch operation. *Aggressive2* on the the other hand overlaps the write-back operation as much as possible with the service of past and future requests at the expense of delaying the fetch for b. More formally, assume that *Aggressive2* finished the last fetch operation immediately before reqeust r_i and that r_j, $j \geq i$ is the first request such that the next request to b is before the next request to b'. If b' has to be written back to disk, start the wirte operation at the earliest $r_{i'}$, $i' \geq i$, such that b' is not requested between $r_{i'}$ and r_j. Overlap the operation as much as possible with the service of request.

While *Aggressive1* is very easy to analyze, *Aggressive2* is a more intuitive implementation of an aggressive strategy. We show that the approximation ratios of *Aggressive1* and *Aggressive2* increase by a factor of 2 relative to the approximation ratio of the standard *Aggressive* strategy. For *Aggressive1* this is easy to see. The algorithm performs exactly the same fetches and evictions as the *Aggressive* algorithm if all references were read requests. In the worst case each cache replacement takes $2F$ instead of F time units as the evicted block has to be written to disk. For *Aggressive2* the bound is not obvious. The problem is that *Aggressive2* finishes fetch operations on read/write request sequences later than *Aggressive* if all requests were read references. This affects the blocks to be evicted in future fetches and hence the cache replacements are different. The proof of the following theorem is omitted due to space limitations.

Theorem 2. *For any request sequence σ, the elapsed time of Aggressive1 and Aggressive2 on σ is at most $2\min\{1 + F/k, 2\}$ times the elapsed time of OPT on σ.*

Cao et al. [4] showed that for $F = k - 2$, the approximation ratio of *Aggressive* on request sequences consisting of read requests is not smaller than 2. We prove a corresponding bound for *Aggressive1* and *Aggressive2*.

Theorem 3. *For $F = k$, the approximation ratios of Aggressive1 and* Aggressive2 *are not smaller than 4.*

Proof. Let $k \geq 4$. For the construction of the lower bound we use $k - 3$ blocks $A_1, \ldots, A_{k-3}$, two blocks B_1 and B_2 as well as two blocks C_1 and C_2. Hence we work with a universe of size $k + 1$ so that there is always one block missing in cache. The reference to $A_1, \ldots, A_{k-3}, C_1$ and C_2 will always be write requests. The references to B_1 and B_2 will always be read requests.

Let $\sigma' = \sigma_1, \sigma_2$, where $\sigma_1 = A_1^*, B_1, A_2^*, \ldots, A_{k-3}^*, C_1^*, B_2, C_2^*$ and $\sigma_2 = A_1^*, B_2, A_2^*, \ldots, A_{k-3}^*, C_2^*, B_1, C_1^*$. The sequence σ_1 and σ_2 are identical except that the positions of B_1 and B_2 as well as C_1 and C_2 are interchanged. Let $\sigma = (\sigma')^i$, for some $i \geq 1$, i.e. σ' is repeated an arbitrary number of times. We compare the elapsed time of *Aggressive1* and *Aggressive2* on σ' to the elapsed time of OPT on σ' and assume that our approximation algorithms initially have

$A_1, \ldots, A_{k-3}, B_1, B_2$ and C_1 in cache. We first consider *Aggressive1*. At the beginning of σ_1 all blocks in cache are requested before the missing block C_2. Hence *Aggressive1* can start the fetch for C_2 only after the service of the request to A_1 in σ_1. It incurs F units of stall time before the request to B_1 in order to write A_1 to disk and then evicts A_1 to load C_2. The fetch is completed immediately before the request to C_2, where 1 unit of stall time must be incurred. To load the missing block A_1, which is first requested in σ_2, *Aggressive1* writes C_1 to disk immediately before the request to C_2, generating F additional units of stall time before that request. Then C_1 is evicted to load A_1 and $F-1$ units of stall time must be incurred before the request to A_1. At that point *Aggressive1* has blocks $A_1, \ldots, A_{k-3}, B_1, B_2$ and C_2 in cache. The cache replacements in σ_2 are the as as in σ_1, except that the roles of C_1 and C_2 change. At the end of σ' *Aggressive1* has again blocks $A_1, \ldots, A_{k-3}, B_1, B_2$ and C_1 in cache, which is identical to the initial configuration.

Aggressive2's schedule on σ' is the same except that (a) $F+1$ units of stall time are incurred before the last request in σ_1 and σ_2 and (b) $2F-1$ units of stall time are generated before the first requests in σ_1 and σ_2. Hence both algorithms need $2(4F+1)$ time units to serve a subsequence σ'. The optimal algorithm always keeps $A_1, \ldots, A_{k-3}, C_1$ and C_2 in cache and only swaps B_1 and B_2. It needs $2(F+4)$ time units to serve σ'. Since $F = k$, we obtain a performance ratio of $(4k+1)/(k+4)$, which can be arbitrarily close to 4. □

3 New Algorithms

We present an algorithm that achieves an approximation ratio of 2 and hence performs better than *Conservative* and *Aggressive*. Intuitively, the following strategy delays the next fetch operation for F time units and then determines the best block to be evicted.

Algorithm Wait: Whenever the algorithm is not in the middle of a fetch or write operation, it works as follows. Let r_i be the next request to be served and r_j, $j \geq i$, be the next request where the referenced block is not in cache at the moment. If all the k blocks currently in cache are requested before r_j, then the algorithm serves r_i without initiating a write or fetch operation. Otherwise let $d = \min\{F, j-i\}$ and let S be the set of blocks referenced by write requests in $r_i, \ldots, r_{i+d-1}$. Immediately before serving r_{i+d} the algorithm initiates a fetch for the block requested by r_j. It evicts the block b whose next request is furthest in the future among blocks in cache that are not contained in S. If b has been modified since the last time it was brought into cache, the algorithm writes b to disk while serving $r_i, \ldots, r_{i+d-1}$, incurring $F-d$ units of stall time. Otherwise $r_i, \ldots, r_{i+d-1}$ are served without executing a write or fetch operation.

Theorem 4. *The Wait algorithm achieves an approximation ratio of 2.*

For the analysis of *Wait* (and *Aggressive2*) we need a dominance concept introduced by Cao et al. [4]. Given a request sequence σ, let $c_A(t)$ be the index of the next request at time t when A processes σ. Suppose that $c_A(t) = i$. For any

j with $1 \leq j \leq n-k$, let $h_A(t,j)$ be the smallest index such that the subsequence $\sigma(i), \ldots, \sigma(h_A(t,j))$ contains j distinct block not in cache at time t. We also refer to $h_A(t,j)$ as *A's jth hole*. Given two prefetching/caching algorithms A and B, *A's cursor at time t dominates B's cursor at time t'* if $c_a(t) \geq c_B(t')$. Moreover, *A's holes at time t dominate B's holes at time t'* if $h_A(t,j) \geq h_B(t',j)$, for all $1 \leq j \leq n-k$. Finally *A's state at time t dominates B's state at time t'* if A's cursor at time t dominates B's cursor at time t' and A's holes at time t dominate B's holes at time t'. Cao et al. proved the following lemma.

Lemma 1. **[4]** *Suppose that A (resp. B) initiates a fetch at time t (resp. t') and that both algorithms fetch the next missing block. Suppose that A replaces the block whose next request is furthest in the future. If A's state at time t dominates B's state at time t', then A's state at time $t+F$ dominates B's state at time $t'+F$.*

Proof (of Theorem 4). We construct time sequences $t_0, t_1, t_2, \ldots$ and $t'_0, t'_1, t'_2, \ldots$ such that (a) *Wait*'s state at time t_l dominates OPT's state at time t'_l, (b) *Wait* is not in the middle of a fetch or write operation at time t_l and (c) $t_{l+1} - t_l \leq 2(t'_{l+1} - t'_l)$, for all $l \geq 0$. Condition (c) then implies the theorem.

Setting $t_0 = t'_0 = 0$, conditions (a–c) hold initially. Suppose that they hold at times t_l and t'_l and let r_i the next request to be served by *Wait*. If at time t_l all blocks in *Wait*'s cache are requested before the next missing block, then *Wait* serves r_i without initiating a write or fetch operation. We set $t_{l+1} = t_l + 1$ and $t'_{l+1} = t_{l+1} + 1$. Conditions (b) and (c) hold. Since at time t_{l+1} *Wait*'s holes occur at the latest possible positions, *Wait*'s state at time t_{l+1} dominates OPT's state at time t'_{l+1}. In the remainder of this proof we assume that at time t_l there is a block in *Wait*'s cache whose next request is after r_j, where r_j is the reference of the next missing block.

Let t_{l+1} be the time when *Wait* completes the next fetch and let $t'_{l+1} = t'_l + F$. We have $t_{l+1} - t_l \leq 2F$ and hence condition (c) holds. Also, *Wait* is not in the middle of a fetch or write operation at time t_{l+1}. We have to argue that *Wait*'s state at time t_{l+1} dominates OPT's state at time t'_{l+1}. First, *Wait*'s cursor at time t_{l+1} dominates OPT's cursor at time t'_{l+1}. This is obvious if *Wait* does not incur stall time to complete the fetch. If *Wait* does incur stall time, then OPT's cursor cannot pass *Wait*'s cursor because the index of *Wait*'s next hole at time t_l is at least as large as the index of OPT's next hole at time t'_l and OPT needs at least F time units to complete the next fetch.

If OPT does not initiate a fetch before t'_{l+1}, we are easily done. The indices of *Wait*'s $n-k$ holes increase when moving from t_l to t_{l+1} while OPT's holes do not change between t'_l and t'_{l+1}. Hence *Wait*'s holes at time t_{l+1} dominate OPT's holes at time t'_{l+1} and we have the desired domination for the states. If OPT does initiate a fetch before t'_{l+1}, then the analysis is more involved. Let a be the block evicted by OPT during the fetch and let b be the block evicted by *Wait* during the first fetch after t_l. If the next request to b is not earlier than the next request to a, then *Wait*'s holes at time t_{l+1} dominate OPT's holes at time t'_{l+1} and we have again domination for the states. Otherwise, let $d = \min\{F, j - i\}$. *Wait* initiates the next fetch after t_l immediately before serving r_{i+d}. OPT cannot

initiate the first fetch after t'_l after r_{i+d}. If $d = F$, this follows from the fact that *Wait*'s cursor at time t_l dominates OPT's cursor at time t'_l and OPT initiates the fetch before $t'_l + F$. If $d < F$, then the statement holds because the index of *Wait*'s next hole at time t_l is at least as large as the index of OPT's next hole at time t'_l and r_{i+d} is the next missing block for *Wait*.

Recall that we study the case that the next request to block b is before the next request to a. Block a is not in the set S of blocks referenced by write requests in $r_i, \ldots, r_{i+d-1}$ because a would have to be written back to disk after its last write reference in $r_i, \ldots, i_{i+d-1}$. This write opertion would take F time units after t_l and could not be completed before t_{l+1}. As argued at the end of the last paragraph, *Wait*'s cursor at the time when *Wait* initiates the fetch dominates OPT's cursor when OPT initiates the fetch. By the definition of the algorithm, *Wait* evicts the block whose next request is furthest in the future among blocks not in S. We have $a \notin S$. Since *Wait* does not evict block a but the next request to a is after the next request to b it must be the case that a is not in *Wait*'s cache at the time when the algorithm initiated the first fetch after t_l. Hence a is not in *Wait*'s cache at time t_l and corresponds to one of *Wait*'s holes at time t_l.

Consider OPT's holes at time t'_l that are after *Wait*'s first hole $h_W(t_l, 1)$ at time t_l. If these holes are a subset of *Wait*'s holes at time t_l, then OPT's holes at time t'_{l+1} with index larger than $h_W(t_l, 1)$ are a subset of *Wait*'s holes at time t_{l+1}. The reason is that, as argued above, *Wait* also has a hole at the next request to a, the block evicted by OPT during the fetch. Note that all of *Wait*'s holes at time t'_l have index larger than $h_W(t_l, 1)$. Hence *Wait*'s holes at time t_{l+1} dominate OPT's holes at time t'_{l+1}.

If OPT's holes at time t'_l with index larger than $h_W(t_l, 1)$ are not a subset of *Wait*'s holes at time t_l, then let $h_{OPT}(t'_l, s')$ be the largest index such that $h_{OPT}(t'_l, s') > h_W(t_l, 1)$ and *Wait* does not have a hole at the request indexed $h_{OPT}(t'_l, s')$. The block referenced by that request cannot be in S because OPT would not be able to write the block back to disk before $t_l + F$. Hence the next request to the block b evicted by *Wait* cannot be before $h_{OPT}(t'_l, s')$. At time t_l let s be the number of *Wait*'s holes with index smaller than $h_{OPT}(t'_l, s')$. At time t_{l+1}, the first hole is filled. Hence *Wait*'s first $s - 1$ holes at time t_{l+1} dominate OPT's first holes at time t'_{l+1}. *Wait*'s remaining holes at time t'_{l+1} have an index of at least $h_{OPT}(t'_l, s')$ and OPT's holes at time t'_{l+1} with an index larger than $h_{OPT}(t'_l, s')$ are a subset of *Wait*'s holes because, as mentioned before, the next request to block a evicted by OPT is a hole for *Wait*. Hence *Wait*'s last $n-k-(s-1)$ holes at time t_{l+1} dominate OPT's last $n-k-(s-1)$ holes at time t'_{l+1}. Thus *Wait*'s state at time t_{l+1} dominates OPT's state at time t'_{l+1}. □

4 Complexity

Theorem 5. *Given a request sequence σ, it is NP-complete to decide if there exists a prefetching/caching schedule for σ that initiates at most f fetch and at most w write operations.*

The proof is omitted due to space limitations.

5 Algorithms for Parallel Disk Systems

In this we study integrated prefetching and caching in systems with D parallel disks. To speed up write operations, many parallel disk systems have the option of writing a memory block to an arbitrary location in the disk systems and not necessarily to the location where the block was stored previously. In particular, blocks may be written to arbitrary disks. As an example, suppose that block b has to be written to disk and that only disk d is idle at the moment. Now disk d can simply write b to the available location closest to the current head position. Of course, if a block is written to a location different from the one where the block was stored previously, the old copy of the block becomes invalid and cannot be used in future fetch operations. We assume that at any time, for any block there exists exactly one valid copy in the parallel disk system.

Given the ability to write blocks to arbitrary disks, we are able to design prefetching/caching algorithms that achieve a constant performance ratio independent of D. In particular we are able to construct efficient prefetching/caching schedules in two steps. Given a request sequence σ, we first build up a schedule S without considering from which disks blocks have to be fetched and to which disks they have to be written back. The algorithm *Loadbalance* described below then assigns fetch and write operations to the different disks. The algorithm works as long as S is synchronized and executes at most $\lceil D/2 \rceil$ parallel disk operations at any time. Moreover blocks evicted from cache must be written back to disk every time, even if they have not been modified since the last time they were brought into cache.

A schedule is *synchronized* if any two disk operations either are executed in exactly the same time interval or do not overlap at all. Formally, for any two disk operations executed from time t_1 to t_1' and from time t_2 to t_2', with $t_1 \leq t_2$ we require (1) $t_1 = t_2$ and $t_1' = t_2'$ or (2) $t_1' < t_2$.

Algorithm Loadbalance: The algorithm takes as input a synchronized prefetching/caching schedule S in which at most $\lceil D/2 \rceil$ disk operations are performed at any time. Blocks are written back to disk each time they are evicted from cache. The schedule is feasible except that disk operations have not yet been assigned to disks. The assignment is now done as follows. The initial disk configuration specifies from which disk to load a block when it is fetched for the first time in S. As for the other assignments, the algorithm considers the write operations in S in order of increasing time when they are initiated; ties are broken arbitrarily. Let w be the write operation just considered and b be the block written back. Let f be the operation in S that fetches b back the next time. Assign w and f to a disk that is not yet used by operations executed in parallel with w and f. Such a disk must exist because a total of $2(\lceil D/2 \rceil - 1)$ disk operations are performed in parallel with w and f.

We next present algorithms for computing schedules S that have the properties required by *Loadbalance*. We first develop a parallel implementation of the *Conservative* algorithm.

Algorithm Conservative: Consider the requests in the given sequence σ one by one. Let r_i be the next request for which the referenced block is not in cache.

The algorithm schedules up to $\lceil D/2 \rceil$ cache replacements immediately before r_i as follows. In each step let a be the next block missing in cache and b be the block in cache whose next request is further in the future. If the next request to a is before the next request is to b, then evict b in order to load a. Suppose that $d \leq \lceil D/2 \rceil$ cache replacements are determined in this way. Let $a_1, \ldots, a_n$ be the blocks loaded and $b_1, \ldots, b_n$ be the blocks evicted. Schedule a set of d synchronized write operations in which $b_1, \ldots, b_d$ are written, followed by a set of d synchronized fetch operations in which $a_1, \ldots, a_n$ are loaded immediately before r_i. These disk operations do not overlap with the service of requests. In the following we refer to such a combination of write and fetch operations as an access interval.

Applying *Loadbalance* to a schedule constructed by *Conservative*, we obtain a feasible prefetching/caching schedule for a given σ, provided that we modify the schedule as follows. If an access interval fetches two blocks that are loaded for the first time in the schedule and reside on the same disk in the initial disk configuration, then schedule an additional fetch operation before the given request r_i.

Theorem 6. *For any σ, the elapsed time of the schedule constructed by Conservative and Loadbalance is at most 5 times the elapsed time of an optimal schedule plus FB. Here B is the number of distinct blocks requested in σ.*

Proof. Given an arbitrary request sequence σ, let I be the number of access intervals generated by *Conservative*. The total elapsed time of the schedule constructed by *Conservative* and *Loadbalance* is bounded by $|\sigma| + (W + F)I + FB$. The additive FB is necessary to bound the fetch time for blocks loaded for the first time in the schedule. Because of initial disk configuration, it might not be possible to execute these fetch operations in parallel with other fetches. We will show that the elapsed time of an optimal schedule is at least $\max\{|\sigma|, F\lceil I/2 \rceil\}$. Since $W \leq F$, the theorem then follows.

It suffices to show that $F\lceil I/2 \rceil$ is a lower bound on the elapsed time of an optimal schedule because the lower bound of $|\sigma|$ is obvious. Let S be an optimal schedule for $|\sigma|$. We partition the fetch operations in σ into *sets of fetches*. For this purpose we sort the fetch operations in S by increasing starting times; ties are broken arbitrarily. The first set of fetches contains the first fetch operation f and all the fetches that are initiated before f is finished. In general, suppose that $i - 1$ sets of fetches have been constructed so far. The ith set of fetches contains fetch operations that are not yet contained in the $i - 1$ first sets. It contains the first such fetch f as well as all fetch operations that are initiated before f terminates. Let J be the number of sets thus created. The first fetches in these J sets are non-overlapping and hence the optimum algorithm spends at least FJ time units fetching blocks.

Lemma 2. *It is possible to modify the schedule S such that it is identical to* Conservative*'s schedule and the total fetch time is at most $2FJ$.*

The proof is omitted. Since the total fetch time of *Conservative*'s schedule is IF, the desired bound then follows. □

We next give an implementation of the *Aggressive* algorithm. It uses $\lceil D/2 \rceil$ extra memory locations in cache.

Algorithm Aggressive+: Let r_i be the next request to be served and r_j be the next request where the referenced block is not in cache. Let $d = \min\{j - i, F\}$. Determine the largest number d, $d \leq \lceil D/2 \rceil$, such that there exist d blocks in cache whose next requests after r_{i+d-1} are later than the first references of the next d blocks missing in cache. If $d = 0$, then serve r_i without initiating a fetch. Otherwise, when serving r_i, initiate d synchronized fetch operations in which the next d missing blocks are loaded into $\lceil D/2 \rceil$ extra cache locations. When these fetches are complete, evict the d blocks from cache whose next requests are furthest in the future and write them back to disk in a synchronized write operation. The $\lceil D/2 \rceil$ extra cache locations are available again. Note that the write operations start with the service of r_{i+d}.

Again we apply *Loadbalance* to a schedule constructed by *Aggressive+*. The proof of the next theorem is omitted.

Theorem 7. *Given a request sequence σ, the elapsed time of the schedule constructed by Aggressive+ and Loadbalance is at most 4 time the elapsed time of an optimal schedule plus FB, where B is the number of distinct blocks requested in σ.*

References

1. S. Albers, N. Garg and S. Leonardi. Minimizing stall time in single and parallel disk systems. *Journal of the ACM*, 47:969–986, 2000. Preliminary version STOC98.
2. S. Albers and C. Witt. Minimizing stall time in single and parallel disk systems using multicommodity network flows. *Proc. 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, Springer LNCS 2129, 12–23, 2001.
3. L.A. Belady. A study of replacement algorithms for virtual storage computers. *IBM Systems Journal*, 5:78–101, 1966.
4. P. Cao, E.W. Felten, A.R. Karlin and K. Li. A study of integrated prefetching and caching strategies. *Proc. ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 188–196, 1995.
5. P. Cao, E.W. Felten, A.R. Karlin and K. Li. Implementation and performance of integrated application-controlled caching, prefetching and disk scheduling. *ACM Transaction on Computer Systems (TOCS)*, 14:311–343, 1996.
6. A. Gaysinsky, A. Itai, and H. Shachnai. Strongly competitive algorithms for caching with pipelined prefetching. *Proc. of the 9th Annual European Symposium on Algorithms (ESA01)*, Springer LNCS 2161, 49–61, 2001.
7. T. Kimbrel and A.R. Karlin. Near-optimal parallel prefetching and caching. *SIAM Journal on Computing*, 29:1051 – 1082, 2000. Preliminary version in FOCS96.
8. T. Kimbrel, P. Cao, E.W. Felten, A.R. Karlin and K. Li. Integrated parallel prefetching and caching. *Proc. ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 1996.
9. T. Kimbrel, A. Tomkins, R.H. Patterson, B. Bershad, P. Cao, E.W. Felten, G.A. Gibson, A.R. Karlin and K. Li. A trace-driven comparison of algorithms for parallel prefetching and caching. *Proc. of the ACM SIGOPS/USENIX Association Symposium on Operating System Design and Implementation*, 1996.

Online Seat Reservations via Offline Seating Arrangements*

Jens S. Frederiksen and Kim S. Larsen

Department of Mathematics and Computer Science
University of Southern Denmark, Odense, Denmark
{svalle,kslarsen}@imada.sdu.dk

Abstract. When reservations are made to for instance a train, it is an on-line problem to accept or reject, i.e., decide if a person can be fitted in given all earlier reservations. However, determining a seating arrangement, implying that it is safe to accept, is an off-line problem with the earlier reservations and the current one as input. We develop optimal algorithms to handle problems of this nature.

1 Introduction

In Danish as well as other European long-distance train systems, it is very common to make reservations. Near weekends and holidays, almost all tickets are reserved in advance. In the current system, customers specify their starting and ending stations, and if there is a seat available for the entire distance between the two stations, a reservation is granted, and the customer is given a car and seat number which uniquely specifies one seat in the train set. The problem of giving these seat numbers on-line has been studied extensively [7,8,6,4,3], and the conclusion is that no matter which algorithm is used, the result can be quite far from optimal. How far depends on the pricing policy. For unit price tickets, a factor of about two is lost, depending on more specific assumptions. If the price depends linearly on the distance, measured in number of stations, then the result can be much worse.

We give a very simple example of how mistakes are possible in this scenario. Much more advanced examples can be found in the literature cited above. In the example, we assume the stations are named A, B, C, and D, and we assume that the train has only two seats, seat 1 and seat 2. The first reservation is (A, B), and without loss of generality, we give the seat number 1. The next reservation is (C, D). If we give seat 2 to this reservation, then the next reservation will be (A, D), which we must reject, even though it could have been accommodated had we given seat 1 the second time as well. If, on the other hand, we give seat 1 to the reservation (C, D), then we might get first (A, C), which we can give seat 2, and then (B, D), which we must reject. Thus, no matter which decision

* Supported in part by the Danish Natural Science Research Council (SNF) and in part by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 174–185, 2003.
© Springer-Verlag Berlin Heidelberg 2003

we make on the second reservation, we may accommodate fewer than possible, if we knew the entire future.

Because of these results, it is tempting to switch to a different system, where seat numbers are not given in response to a reservation, but instead announced later. Many people expect that soon we will almost all be equipped with PDAs (personal digital assistants) or just cell phones, so it will be practically feasible to send the seat number to the customer shortly before the train may be boarded. An electronic bulletin board inside the train could inform the remaining customers of their seat number. Notice that in both of the example scenarios above, it would be possible to seat all customers, if seat numbers are not determined until after all reservations are made.

Computing a seating arrangement off-line is a well-known problem. Partly because the problem is equivalent to the channel-assignment problem [12] and partly because the underlying abstract problem is coloring of interval graphs [13]. In [12], it is shown that the problem can be solved in the optimal time $\theta(n \log n)$ in the decision tree model, where the optimality follows by a reduction from the element uniqueness problem [10].

The problem we consider is in some sense in between the on-line and off-line problems described above, since we wish to compute the optimal off-line solution, but we must decide for each reservation whether or not the inclusion of this current reservation into the collection of already accepted reservations will still allow for a solution, given the number of seats available. Thus, we want a data structure with an operation *insert*, which inserts a reservation into the data structure if the resulting collection allows for a solution using at most N seats, where N is a fixed constant. If not, then the reservation should be rejected. We also want an operation *output*, which from the data structure extracts a seating arrangement. We assume that each reservation is accompanied by some form of identifier (reservation number, cell phone number, or similar) such that each customer can be notified regarding his or her allocated seat. The output must be sorted by increasing starting station. Finally, we want an operation *delete* such that customers may cancel their reservation.

We show that in the pointer machine model [14], we can provide a data structure with the optimal complexities of $O(\log p)$ for *insert* and $O(n)$ for *output*, where n is the current number of reservations, and p is the current number of different stations, which could be a lot smaller than n and also smaller than the number of possible stations. The complexity of *delete* will also be $O(\log p)$. Furthermore, the updating operations make only $O(1)$ structural changes if a red-black tree is used as the underlying search tree, and the space requirements are $O(n)$.

In fact, our data structure will allow us to perform more insertions of reservations during the process, provided that the outputting process has not yet gotten to the starting station of the reservation to be inserted. Similarly, deletions of reservations can be carried out when the starting station of the reservation has not yet been reached. The total time spent on outputting will still be $O(n)$, where n is the total number of intervals, which have been inserted and not deleted. The

fact that this gradual outputting can be done efficiently may be even more interesting in non-train scenarios, if our algorithm is used to allow computers to reserve some resources for particular time intervals, e.g., in a variant of the channel-assignment problem.

Our structure is similar to segment trees (in [9], this data structure is reported to have been described first in [5]) and dynamic segment trees [15]. However, segment trees have a fixed number of leaves, whereas we want to add new endpoints dynamically as they are required. This can be handled by dynamic segment trees, but these are fairly complicated (which is not surprising because they solve a more involved problem). For the dynamic segments trees of [15], *insert* is $O(\log n)$ and *delete* is $O(a(i,n)\log n)$, where a is related to the inverse Ackermann function [1] and i is a constant. This function grows extremely slowly and can for all practical purposes be considered a constant. The time complexity is only amortized because the structure must be rebuild occasionally. The space requirements are $O(n \log n)$.

It may be possible to adjust dynamic segment trees to solve our problem. However, the problem scenarios are as a starting point not comparable since dynamic segment trees must be able to answer stabbing queries, whereas we must be able to provide an efficient *output* operation and also efficiently disallow *insert* operations if and only if some stabbing query after the insertion would yield a set with a cardinality greater than N. In the main part of the paper, for simplicity, we refer to and compare with the better known segment trees.

2 The Algorithms

In this section, we follow the graph tradition and talk about intervals, endpoints, and colors instead of reservations, stations, and seat numbers, respectively.

We first discuss which attributes we expect different units of data to be equipped with in our algorithms.

Intervals have left and right endpoints, which we refer to as *begin* and *end*. The intervals are closed to the left and open to the right. Intervals may also have a *color*. If necessary, we assume that intervals are also equipped with a unique identifier such that otherwise identical intervals can be distinguished.

The data structure we propose is a binary tree where the leaves represent the set of all the different endpoints which have been used. They appear in the leaves in sorted order from left to right. The tree is build from nodes which contain the following information: Since the tree is binary, there is a *left* and *right* reference. The attribute *cover* stores the interval covered by a node. For a leaf node, this is the interval from its endpoint to the endpoint of the next leaf, and for an internal node, this is the union of all the intervals of the leaves in its subtree. At any leaf node, the intervals which begin or end at the endpoint of the leaf are stored in the attributes *BeginList* and *EndList*, respectively.

To help us decide how many colors are necessary to color the intervals, we use two extra variables in each node, k and Δk. For any path from a node in the tree to a leaf in its subtree, we define its Δ-length as the sum of all the Δk

values of the nodes of the path. By updating the Δk and k values appropriately, we first of all make sure that the Δ-length of a path from the root to any leaf is exactly the density of the cover interval of the leaf, i.e., the number of intervals added to the structure which overlap the cover interval. Furthermore, we ensure that the k value of any node is the maximal Δ-length from this node to any leaf in its subtree. For the root, this is the maximal density of the tree.

As a basis for our data structure, we use a worst-case logarithmically balanced search tree such as a red-black tree [11] or an AVL-tree [2], for instance. This means that in addition to the attributes for tree nodes described above, attributes appropriate for rebalancing should also be present, but since the exact choice of tree is irrelevant, we just assume that the necessary attributes are present. Our use of it is similar to segment trees. However, segment trees have a fixed number of leaves, whereas we want to add new endpoints dynamically as they are required.

A segment tree is designed for storing intervals and the leaves represent all possible endpoints in sorted order from left to right. The tree is used for searching for intervals which contain a certain point. Each interval (an identifier or a reference to it) is stored in at least one, but maybe in several nodes of the tree. This can be in internal nodes as well as leaves. An interval is stored in a given node u if and only if all the possible endpoints in the leaves of the subtree rooted by u are contained in the interval and no node between u and the root of the entire tree has that property. The effect obtained by this is that on a search for a point in the data structure, each interval containing that point will be found exactly once on the search path.

Our approach is similar in the way that we initially update at exactly the same locations. However, most places we only increase a counter. The actual interval is only stored at the leaves corresponding to its endpoints. Another difference is that the counter values cannot necessarily remain in the same location throughout the computations (as intervals would in a segment tree) because the tree structure is altered dynamically.

For clarity, we assume that the starting point is a leaf node covering the interval $-\infty$ to ∞ with $k = \Delta k = 0$ and empty *BeginList* and *EndList*.

To ensure that the two demands regarding k and Δk are met, we initialize the Δk values to zero. When inserting a new interval into the structure, we increment the Δk value of exactly one node on any path from the root node to a leaf, the cover interval of which intersects the new interval. All other nodes maintain their Δk values. Subsequently, we update the k values bottom-up. The algorithms for insertion can be seen in Fig. 1.

With slightly more complicated code, it is possible to combine searches down the tree. However, this will only improve the complexity by a constant factor. For readability, we have divided it up, so that we first check whether insertion is at all possible, then we insert the endpoints (if they are not already present) and update the corresponding *BeginList* and *EndList*, and as the last step we update the counters.

```
proc insert(tree: Node, x: Interval)
    if okToInsert(tree, x, N) then
        insertEndpoint(tree, x.begin, true, x)
        insertEndpoint(tree, x.end, false, x)
        insertInterval(tree, x)

func okToInsert(n: Node, x: Interval, c: Integer): Boolean
    if n.cover ∩ x = ∅ then
        return True
    else if n is a leaf or n.cover ⊆ x then
        return c ≥ n.k + 1
    else
        c′ ← c − n.Δk # Calculate the number of colors left
        return okToInsert(n.left, x, c′) and okToInsert(n.right, x, c′)

proc insertEndpoint(tree: Node, b: Real, d: Boolean, x: Interval)
    n ← findLeaf(tree, b) # Finds maximal a such that a ≤ b
    if n.cover.begin ≠ b then
        # Split n as described
        n ← n.right
        # Rebalance tree bottom-up if necessary
    if d then n.BeginList.append(x) else n.EndList.append(x)

proc insertInterval(n: Node, x: Interval)
    if n.cover ⊆ x then
        n.Δk ← n.Δk + 1
        n.k ← n.k + 1
    else
        if n.left.cover ∩ x ≠ ∅ then
            insertInterval(n.left, x)
        if n.right.cover ∩ x ≠ ∅ then
            insertInterval(n.right, x)
        n.k ← max(n.left.k, n.right.k) + Δk
```

Fig. 1. The insert operation.

It is *insertEndpoint* which uses the tree dynamically. If an endpoint is not present, it is inserted by performing a local operation at the leaf where the search ends. The setting of the attributes in the new node is shown in Fig. 2, where it is demonstrated how one leaf is replaced by one internal node and two leaves.

After this change, the tree may need rebalancing. This is done differently for different balanced tree schemes. However, we only assume that it is done bottom-up by at most a logarithmic number of local constant-sized transformation on the search path. Such transformations on a search tree can always be expressed

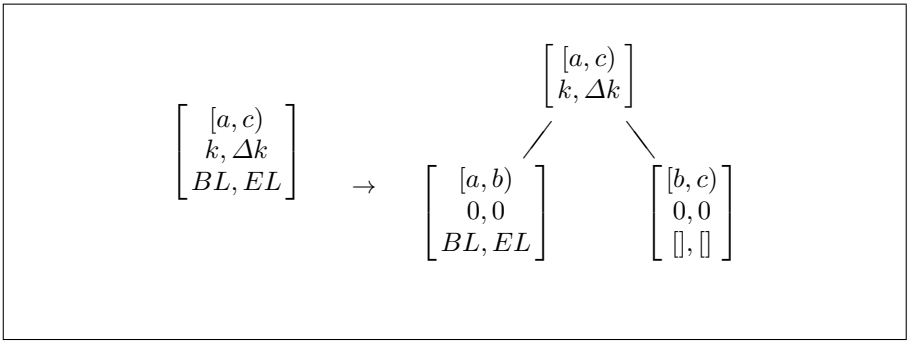


Fig. 2. A split operation performed on a leaf initially containing the interval $[a, c)$. In the nodes, the first line shows the *cover interval* and the second line shows the k and Δk value of the node. The third line shows the *BeginList* and *EndList* of leaf nodes. The new endpoint b is inserted.

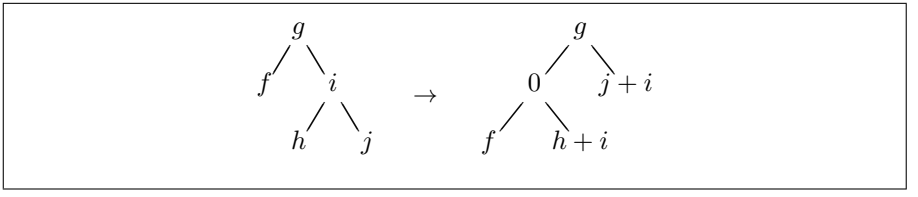


Fig. 3. A left rotation with old and new Δk values shown.

as a constant number of rotations. In Fig. 3, we show how attributes should be set in connection with a left rotation. A right rotation is similar.

Note that the new k values can be calculated using the Δk values, and the new *cover* values for the two internal nodes of the operation can be recomputed using their children.

The considerations for *delete* are similar. We must update the density information by deleting the interval, we must remove the actual reservation from a leaf, and we must delete the endpoints if no other intervals share them. The actions reverse actions taken during an *insert*. The *delete* operation is shown in Fig. 4. In Fig. 5, we show how a node is removed from the tree in the case where no other intervals share the endpoint. Notice how the updates to the Δk-values preserve the invariants. For the first case, where the node to be deleted is a left child of its parent, b must be changed to a c on the path from the point of deletion up towards the root, until the procedure reaches the root or a node which has the deleted node in its right subtree. From that node, the b's must also be changes to c's on the path down to the predecessor of the deleted node (the node containing $[a, b)$ before the update).

As for insertion, rebalancing is a matter of carrying out a number of rotations, so the details given for insertions cover this case as well.

Finally, the *output* operation is shown in Fig. 6.

```
proc delete(tree: Node, x: Interval)
    deleteInterval(tree, x)
    deleteEndpoint(tree, x.begin, true, x)
    deleteEndpoint(tree, x.end, false, x)

proc deleteEndpoint(tree: Node, b: Real, d: Boolean, x: Interval)
    n ← findLeaf(tree, b) # Finds leaf containing the endpoint b
    if d then n.BeginList.remove(x) else n.EndList.remove(x)
    if n.BeginList.isEmpty() and n.EndList.isEmpty() then
        # Delete n as described
        # Rebalance tree bottom-up if necessary

proc deleteInterval(n: Node, x: Interval)
    if n.cover ⊆ x then
        n.Δk ← n.Δk − 1
        n.k ← n.k − 1
    else
        if n.left.cover ∩ x ≠ ∅ then
            delete(n.left, x)
        if n.right.cover ∩ x ≠ ∅ then
            delete(n.right, x)
        n.k ← max(n.left.k, n.right.k) + Δk
```

Fig. 4. The delete operation.

Finally, we note that in an actual implementation, some of the values we use can be computed rather than stored.

First, it is only necessary to store the k values in the nodes, since the Δk value for any node n can be calculated as $n.\Delta k = n.k - \max(n.left.k, n.right.k)$.

Second, it is sufficient to store the starting of the cover intervals in the nodes. The other endpoint can be computed as we traverse the path. This would also eliminate the need for the traversal down towards the predecessor of a deleted node to change b's to c's.

3 Correctness and Complexity

We argue that the algorithms presented are correct, and discuss their complexity.

3.1 Correctness

Regarding correctness, there are two essential properties our structure should have. First, it should allow an insertion if and only if the resulting graph can be colored using at most N colors. Second, a legal coloring using at most N

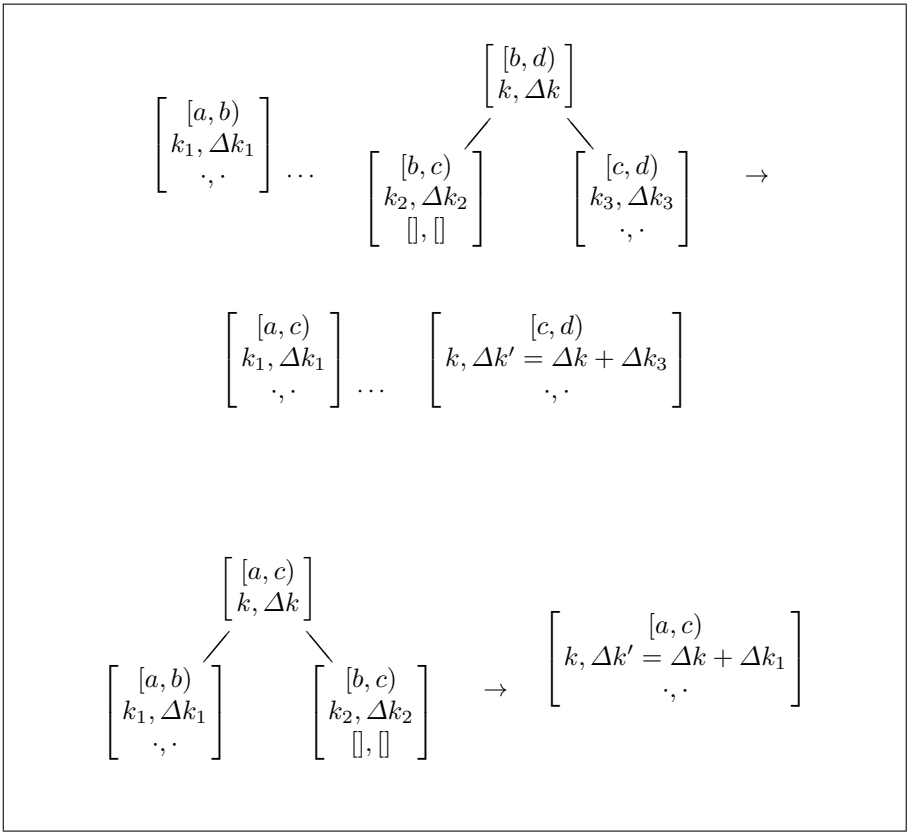


Fig. 5. A delete operation performed on a node containing the interval $[b, c)$. There are two cases depending on whether the node to be deleted is the left or right child of its parent.

colors should be printed by the outputting procedure. Third, a deletion should correctly undo an insertion.

Regarding the first point, we claim that for any path from the root node to a leaf node, its Δ-length is exactly the same as the number of intervals inserted into the tree which intersect the cover interval of the leaf node, i.e., the density of the cover interval of the leaf. Furthermore, we claim that for any node, its k value is the maximum Δ-length of a path to a leaf in its subtree. This is true because the insertion and the deletion of an interval ensures it and rotations preserve it.

An insertion of an interval ensures it by increasing Δk in nodes such that their *cover* intervals are disjoint while together covering the inserted interval exactly and furthermore updating the k values bottom up. Similarly for deletions. Rotations preserve it by ensuring that Δk values remain associated with the correct intervals and recomputing the k values based on the Δk values.

```
proc output(tree: Node)
   s ← new Stack of N Colors
   # Optional wait until first station is reached
   for each Leaf v in tree using in-order do
      for each Interval x in v.EndList do
         s.push(x.color)
      for each Interval x in v.BeginList do
         x.color ← s.pop()
         print x
      # Optional wait until next station is reached
```

Fig. 6. The output operation.

Now, if k intervals overlap at a given point, this defines a clique of size k in the corresponding interval graph. Interval graphs are *perfect* [13] which means that the size of the largest clique equals the minimum number of colors required to color the graph.

When deciding whether or not the insertion of an interval is possible, *okToInsert* is used. By using the Δk values, this function keeps track of how many colors are left in the recursion on the way to the bottom of the tree. An insertion is only accepted if it will not increase the maximum Δ-length from the root of the tree to more than the allowed number of colors.

Regarding the second point, we must argue that we output a legal coloring which means that we use at most N colors and no two overlapping intervals receive the same. The fact that no two overlapping intervals receive the same color is ensured by the stacking mechanism where the color is simply removed from the stack of available colors when it is used for an interval and it is not pushed onto the stack again until that interval has ended. The fact that we use at most N colors follows from the fact that the number of colors in use (the ones which are not on the stack) is exactly the density at the given point.

3.2 Complexity

If the underlying search tree guarantees $O(\log p)$ searches and rebalancing, where p is the number of leaves (which is the same as the number of different endpoints), then *insertEndpoint* is also clearly $O(\log p)$.

Regarding *insertInterval*, the argument for its complexity is similar to the corresponding argument for segment trees. At a first glance, it seems that the searching down the tree could split into many different paths. However, we argue that this is not the case.

In general, the search may stop (the first if-part) or continue (the else-part) either to the left or to the right, or possibly in both directions. For a number (possibly zero) of steps, we may from each node just continue down one of the

two paths. Then at some node u, we may have to continue down both of them. We argue that there are no further real splits off the two search paths from that point on.

Let us consider the search down the left-most path. At the left child of u, we know (since there was also a search splitting off to the right) that the interval to be inserted covers the right-most point in our subtree. This is the essential property (we refer to it as the right-cover property), and it will be maintained on the rest of the search down the path.

At any node on this path, starting with the left child of u, if we continue down to our left child, then the recursive call to the right child will fall into the if-case and therefore terminate immediately because of the right-cover property. At the same time, the right-cover property will hold for the search to the left. If there is no search to the left, but only to the right, the right-cover property also clearly holds in that case.

The analysis for *okToInsert* is similar to *insertInterval*, except that instead of checking directly before calling, we use an additional recursive call when deciding whether the cover interval of a node intersects the interval to be inserted.

For deletion, the argument is similar. However, we assume that the user reservation encodes a pointer to the reservation. The reservations stored in the BeginLists and EndLists are kept in a doubly-linked list such that they can be removed in constant time.

The work of *output* consists of a linear time traversal of the nodes of the tree which is $O(p) \subseteq O(n)$, where p is the number of different endpoints used in the intervals, plus some constant work per interval which is then also $O(n)$.

Finally, the space requirements are $\theta(n)$: the procedure *insertEndpoint* uses constant extra space per interval, and the procedure *insertInterval* only increments integers already present in the structure.

3.3 Optimality

Regarding optimality, clearly $\Omega(n)$ is required to output the result. If, as we do, *output* is provided in $O(n)$, *insert* must be $\Omega(\log n)$, in the cases where $p \in \Theta(n)$. Otherwise, we can solve the off-line problem in $o(n \log n)$, and this has been proven impossible in the decision tree model in [12] by a simple reduction from the more well-known element uniqueness problem [10], which is known to be $\theta(n \log n)$.

However, this only settles optimality for $p \in \Theta(n)$. We now assume that $p \in o(n)$ and argue that also in this case is the result optimal.

Let us first consider the following sorting problem: we are given a sequence of n distinct objects $x_1, x_2, \ldots, x_n$, equipped with keys of which $p \in o(n)$ are distinct. We argue that in the decision tree model, the time to sort such sequences is $\Omega(n \log p)$. By sorting, we here mean outputting the objects in an order such that the keys of the objects are nondecreasing.

First, we obtain a lower bound on the number of possible outputs. We can think of the number of different ways we can place the x_i's in p distinct boxes under the restriction that none of them may be empty. We first remove p objects

with distinct keys from the sequence, placing them in each their box, thereby removing the restriction. The remaining $n-p$ objects can be placed in the p different boxes in p^{n-p} different ways. The number of binary comparisons we would have to use in the worst-case to choose correctly between p^{n-p} different possible outputs is $\log(p^{n-p})$, assuming that we can balance our decision tree perfectly; otherwise it only gets worse. Now, $\log(p^{n-p}) = (n-p)\log p \in \Omega(n \log p)$, since $p \in o(n)$.

As a simple corollary, n intervals with at most p different endpoints cannot in general be sorted on starting point faster than $\Omega(n \log p)$.

However, this sorting problem can be solved using the data type discussed in this paper. Let $N = n$ so that all intervals will fit, use *insert* to insert each interval one at a time, and *output* to obtain the result. Thus, if the problem in this paper is not in $\Omega(n \log p)$, the sorting problem above would not be either, and that would be a contradiction.

4 Concluding Remarks

Without making the data structure more complicated, it is possible to make some minor extensions.

As presented here, we use a constant number N as the number of seats available. It would not be a problem to make this value dynamic, as long as it is never changed to a value smaller than the k value of the root of the tree.

Furthermore, the intervals we consider are all closed to the left and open to the right. This can easily be extended to the general case as in [15], where either side may be open or closed, by using alternately open and closed intervals in the leaves of the structure: $(-\infty, a_1)$, $[a_1, a_1]$, (a_1, a_2), $[a_2, a_2]$, ...

In some special cases, it is also straight-forward to implement *split* and *join* operations on the tree. If we for *split* require that no intervals in the tree contain the splitting point inside the interval, and for *join* require that the intervals in the two trees do not intersect each other, then both operations can be implemented in $O(\log p)$ time.

As a more general remark, it is important to notice that we do not assume that the stations which are used are numbered from 1 through p. In fact, we do not even assume that they are integers. One can think of the stations as floating point numbers. One could consider a less dynamic version of the problem and assume that stations are numbered from 1 through p, treating p as a constant. This would make it possible to obtain different theoretical results and better results in practice, in the cases where p really is small. However, the results would be less general and therefore not necessarily as easily applicable to other problems, such as the channel-assignment problem. The theoretical treatment would also be entirely different, since if elements are known to be from a small interval of integers, many problems become computationally much easier.

References

1. Wilhelm Ackermann. Zum Hilbertschen Aufbau der reellen Zahlen. *Mathematische Annalen*, 99:118–133, 1928.
2. G. M. Adel'son-Vel'skiĭ and E. M. Landis. An Algorithm for the Organisation of Information. *Doklady Akadamii Nauk SSSR*, 146:263–266, 1962. In Russian. English translation in *Soviet Math. Doklady*, 3:1259–1263, 1962.
3. Eric Bach, Joan Boyar, Leah Epstein, Lene M. Favrholdt, Tao Jiang, Kim S. Larsen, Guo-Hui Lin, and Rob van Stee. Tight Bounds on the Competitive Ratio on Accommodating Sequences for the Seat Reservation Problem. *Journal of Scheduling*, 6(2):131–147, 2003.
4. Eric Bach, Joan Boyar, Tao Jiang, Kim S. Larsen, and Guo-Hui Lin. Better Bounds on the Accommodating Ratio for the Seat Reservation Problem. In *Sixth Annual International Computing and Combinatorics Conference*, volume 1858 of *Lecture Notes in Computer Science*, pages 221–231. Springer-Verlag, 2000.
5. J. L. Bentley. Solutions to Klee's Rectangle Problems. Tech. report, Carnegie-Mellon University, 1977.
6. Joan Boyar, Lene M. Favrholdt, Kim S. Larsen, and Morten N. Nielsen. Extending the Accommodating Function. In *Eighth Annual International Computing and Combinatorics Conference*, volume 2387 of *Lecture Notes in Computer Science*, pages 87–96. Springer-Verlag, 2002.
7. Joan Boyar and Kim S. Larsen. The Seat Reservation Problem. *Algorithmica*, 25(4):403–417, 1999.
8. Joan Boyar, Kim S. Larsen, and Morten N. Nielsen. The Accommodating Function: a generalization of the competitive ratio. *SIAM Journal on Computing*, 31(1):233–258, 2001.
9. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd edition, 2000.
10. David P. Dobkin and Richard J. Lipton. On the Complexity of Computations under Varying Sets of Primitives. *Journal of Computer and System Sciences*, 18(1):86–91, 1979.
11. Leo J. Guibas and Robert Sedgewick. A Dichromatic Framework for Balanced Trees. In *19th Annual IEEE Symposium on the Foundations of Computer Science*, pages 8–21, 1978.
12. U. I. Gupta, D. T. Lee, and Joseph Y.-T. Leung. An Optimal Solution for the Channel-Assignment Problem. *IEEE Transactions on Computers*, 28(11):807–810, 1979.
13. Tommy R. Jensen and Bjarne Toft. *Graph Coloring Problems*. John Wiley & Sons, 1995.
14. Peter van Emde Boas. Machine Models and Simulations. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 1, pages 1–66. Elsevier Science Publishers, 1990.
15. Marc J. van Kreveld and Mark H. Overmars. Union-Copy Structures and Dynamic Segment Trees. *Journal of the Association for Computing Machinery*, 40(3):635–652, 1993.

Routing and Call Control Algorithms for Ring Networks

R. Sai Anand * and Thomas Erlebach

Computer Engineering and Networks Lab
Eidgenössische Technische Hochschule Zürich
{anand|erlebach}@tik.ee.ethz.ch

Abstract. A vast majority of communications in a network occurs between pairs of nodes, each such interaction is termed a *call*. The job of a call control algorithm is to decide which of a set of calls to accept in the network so as to maximize the number of accepted calls or the profit associated with the accepted calls. When a call is accepted it uses up some network resources, like bandwidth, along the path through which it is routed. The call control algorithm needs to make intelligent trade-offs between resource constraints and profits. We investigate two variants of call control problems on ring networks; in the first, the algorithm is allowed to determine the route connecting the end nodes of a call, while in the second, the route is specified as part of the input. For the first variant, we show an efficient algorithm that achieves the objective of routing and maximizing the *number* of accepted calls within an additive constant of at most 3 to an optimal algorithm. For the fixed path variant, we derive a 2-approximation for maximizing the *profits* (which could be arbitrary) of accepted calls. For several important special cases we show polynomial time optimal algorithms.

1 Introduction

Motivation. Optical fiber based networks are increasingly replacing the traditional copper cable based ones in modern day telecommunication. They provide substantial advantages in terms of high bandwidth and capability to carry multiple types of traffic. This is in tandem with the emergence of high bandwidth applications like video-conferencing, multimedia, video on demand etc. SONET is a dominant technology standard for optical networks today. The building block of these networks, called a SONET ring, is one in which network nodes are connected together in a ring with optical fiber cables. It is therefore interesting and important to study communication problems, such as call control, that arise in ring networks.

Call admission control is a basic problem in communication networks. Within the bandwidth limitations on network links that carry data, the call control problem is to optimize the profits accrued on traffic that can be carried across the network. More concretely, the situation is the following. Network elements, like edge routers, receive a sequence of requests from nodes to establish connections with other nodes. Each such connection takes up some bandwidth along the path through which it is routed. The call control algorithm at the router needs to make a decision as to which among these

* Supported by the joint Berlin/Zurich graduate program Combinatorics, Geometry and Computation (CGC), financed by ETH Zürich and the German Science Foundation (DFG).

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 186–197, 2003.
© Springer-Verlag Berlin Heidelberg 2003

requests it can accept on the network at any given time. For every request that is accepted and routed a profit is made. In a typical setting, requests for connections arrive one after the other and the algorithm needs to make an on-line decision. That is, a decision to accept or reject a request cannot depend on the future but may be based on decisions made in the past. In this paper, however, we shall concentrate on the off-line problem where all requests are known beforehand. There are two reasons for this. Firstly, an off-line algorithm serves as a basis for evaluating on-line ones. Secondly, there are real life scenarios like advance reservations, where the traffic requests are indeed known in advance.

A Graph Model. The call control problem can be modeled as a graph problem. The communication network is represented as a capacitated graph $G = (V, E, w)$. The nodes and links connecting nodes are identified with the vertices and edges of the graph respectively. The bandwidth on links corresponds to capacities on edges. A connection between two nodes is a path in the graph. Thus, the bandwidth constraints on the links would mean that the number of paths through an edge $e \in E$ is at most its capacity $w(e)$. If the objective is to maximize the number of connections to be accepted, in the graph we need to maximize the total number of paths. If the objective is to maximize the profit, in the graph model the sum of the profits of accepted paths should be maximized. There are two versions of call control depending on whether the call control algorithm can decide on the path it will assign to a connection or the route is pre-specified. We shall formalize these notions in the following sections.

Previous Work. Call control problems have been studied for several topologies in both the on-line and off-line settings. For chain networks, the off-line version with equal bandwidths on edges, is polynomially solvable as the problem can be modeled as a maximum k-colorable induced subgraph of interval graphs. A clever linear time implementation of this approach is presented in [6]. For the ring topology, a polynomial time optimal algorithm was given in [1], to maximize the number of accepted connections when the path for a connection is pre-specified. This result also solves the off-line problem in chains when the edge capacities are arbitrary.

When all edges have bandwidth of unity, the off-line call control problem is the well known maximum edge disjoint paths problem (MEDP). MEDP is polynomially solvable for chains [6] and undirected trees [10]. In undirected and bidirected rings, MEDP can be solved optimally as well [13,11]. However, for bidirected trees of arbitrary degree and trees of rings, MEDP has been proved to be APX-complete [8,7].

On-line versions of call control have also been investigated when preemption is allowed. Here, a connection once established can be removed in favour of another that is requested later on. Garay et al., in [9], study the on-line preemptive version on chains with unit edge capacities to obtain an $O(\log n)$-competitive algorithm, where n is the number of vertices in the chain. For chain networks, a randomized $O(1)$-competitive algorithm is given in [2] when all edge capacities are equal. Instead of maximizing the number of accepted connections, the objective of minimizing the number of rejected connections was considered in [5]. They showed 2-competitive preemptive algorithms for chains with arbitrary capacities and for arbitrary graphs with unit edge capacities. For the off-line version, they give an $O(\log m)$-approximation algorithm for arbitrary graphs with arbitrary capacities, where m is the number of edges in the graph.

Our Results. We study the call control problem in rings, where all connections demand unit bandwidth on the links, in two variants. In the first variant, the algorithm is allowed to determine the route of connections by itself. For this problem, we give an efficient algorithm that accepts and routes at most 3 fewer connections compared to an optimal algorithm. In the second variant, the routes for connections are predetermined and connections have arbitrary profits associated with them. We give an approximation algorithm for this case that achieves at least half the profit as compared to an optimal algorithm. Moreover, for various special cases, we provide optimal polynomial algorithms or PTASes. One of the special cases subsumes the problem considered in [1]. The PTAS is obtained when the profits are proportional to the length of the routes. For space reasons, in this extended abstract we sometimes skip whole or part of proofs and completely omit the details of the PTAS. The interested reader is invited to look at the technical report [4] for the details.

Section 2 details the first variant which is called the Routing and Call Admission Control problem (RCAC). Section 3 presents results on the second variant, Pre-routed Call Admission Control problem (PCAC). It should be remarked that the computational complexities of both these problems are as yet unresolved and are interesting open problems.

2 RCAC on Rings

Terminology: A *call* in a communication network is a pair of distinct nodes between which a connection needs to be established. It is usually specified as an (unordered) pair of the nodes, also called end points of the call. A *route* for a call is one of several possible paths connecting its end points. A *call set* is a set of all calls in the network which are presented to the call control algorithm for a decision on acceptance and routing. With this terminology in place, we are now ready to define the Routing and Call Admission Control (RCAC) problem on rings.

Input and Objective of RCAC: The input instance to RCAC is a ring (or cycle) $R = (V, E)$ on n vertices, a call set S of m calls and a capacity function $w : E \to \mathbb{Z}^+$. A route for a call on the ring is one of the two possible paths. A *feasible solution* to the instance is a subset $S' \subseteq S$ such that every call $\{u, v\} \in S'$ is routed and the number of routes that pass through any edge e in E is at most $w(e)$. The objective of RCAC is a feasible solution $OPT \subseteq S$ such that $|OPT|$ is the maximum possible and for every call in OPT, a route is specified.

In the rest of the presentation, we abuse notation and let OPT stand for the optimal feasible set and its cardinality. Our approach to solving the RCAC problem is to formulate it as an integer linear program and to round the optimal fractional solution of the relaxed program to a feasible solution. We shall show that the feasible solution so generated is very close to an optimal solution.

An Integer Linear Program for RCAC: The formulation of the integer linear program (ILP) for the RCAC problem is a natural one. Let the call set be $S = \{\{u_i, v_i\} : i = 1, 2, ..., m\}$. We shall refer to a call by its index i. Further, we consider a fixed embedding of the ring on a plane and assign a clockwise direction. Let the n edges be numbered (and referred to from here on) $1, 2, ..., n$, with edge j incident to edge

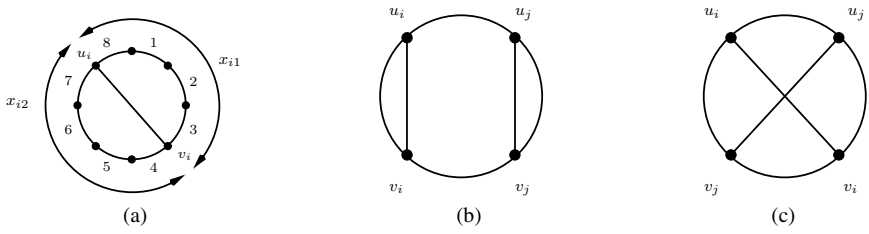


Fig. 1. (a) Call i and its two indicator variables. (b) Parallel calls. (c) Crossing calls.

$(j+1) \bmod n$ at the former's clockwise end vertex (0 is identified with n). For each call i, introduce two indicator variables x_{i1} and x_{i2} corresponding to the two possible routes. The first of them corresponds to the path containing edge 1 and the other to the path that does not. See Figure 1(a) for an illustration. For edge $j = 1, 2, ..., n$, let $S_j = \{x_{ik} :$ route x_{ik} contains edge $j,\ i = 1, 2, ..., m,\ k = 1, 2\}$. Now, the ILP looks as follows:

$\max \sum_{i=1}^{m} (x_{i1} + x_{i2})$
subject to
$\sum_{x_{ik} \in S_j} x_{ik} \leq w(j), j = 1, 2, ..., n$
$x_{i1} + x_{i2} \leq 1,\ i = 1, 2, ..., m$
$x_{ik} \in \{0, 1\},\ i = 1, 2, ..., m,\ k = 1, 2$

Relaxing the above ILP changes the last of the constraints by admitting all fractional values between 0 and 1. The relaxed LP can be solved in time polynomial in n, m and $\log_2 w(.)$. Denote the fractional optimal solution vector as $x^* = (x^*_{i1}, x^*_{i2})_{i=1}^{m}$ and the objective value by OPT^*. It will be helpful to think of the vector x as a function on the set of routes of the calls into the real interval $[0, 1]$. Hence, we shall refer to x as a *route function* and as a $\{0, 1\}$-*route function*, if the components of x are either 0 or 1.

2.1 Rounding Scheme

Before describing the rounding scheme it is useful to distinguish a relation between pairs of calls. Two calls $i = \{u_i, v_i\}$ and $j = \{u_j, v_j\}$ are said to be *parallel* if either their end points appear as u_i, u_j, v_j, v_i while traversing the ring in clockwise fashion or they share a common end point. Observe that since the pair of vertices in a call are unordered the order in which vertices of each call, namely u_i, v_i, themselves appear is immaterial. If two calls are not parallel then they are called *crossing*. Equivalently, a clockwise traversal encounters these end points in the order u_i, u_j, v_i, v_j. A simple observation is that when two calls are parallel one of the routes of the first call is totally contained in a route of the second and vice-versa. Parallel and crossing calls are illustrated in Figures 1(b) and 1(c).

The rounding scheme starts off by doing a preliminary set of transformations on parallel and crossing calls so that the components of the fractional optimal vector x^* are in a particular "canonical" form. It should be remarked that while we change the values

of the components of x^* we do not affect either the feasibility of the resultant vector nor the objective value. We proceed to describe them below.

Transformations on Parallel Calls: Let i and j be two parallel calls with the path $x_{i1}(x_{j2})$ contained in path $x_{j1}(x_{i2})$ and $x^*_{i1}, x^*_{i2}, x^*_{j1}, x^*_{j2} > 0$. The goal behind this transformation is to set at least one of the fractional values $x^*_{i1}, x^*_{i2}, x^*_{j1}, x^*_{j2}$ to zero. Let $y = x^*_{i1} + x^*_{i2} + x^*_{j1} + x^*_{j2}$. We set, $x^*_{i1} \leftarrow \min\{1, x^*_{i1} + x^*_{j1}\}; x^*_{j2} \leftarrow \min\{1, x^*_{i2} + x^*_{j2}\}$. Now, if $x^*_{i1} \geq x^*_{j2}, x^*_{i2} \leftarrow 0; x^*_{j1} \leftarrow y - x^*_{i1} - x^*_{j2}$ else, $x^*_{i2} \leftarrow y - x^*_{i1} - x^*_{j2}; x^*_{j1} \leftarrow 0$.
Transformations on Crossing Calls: Consider two crossing calls i and j with x^*_{i1}, $x^*_{i2}, x^*_{j1}, x^*_{j2} > 0$ and neither of $x^*_{i1} + x^*_{i2}, x^*_{j1} + x^*_{j2}$ are unity. The aim of this transformation is to either set at least one of the variables to zero or make one of the sums $x^*_{i1} + x^*_{i2}, x^*_{j1} + x^*_{j2}$ equal unity. This is achieved in a slightly more involved transformation shown below:
Set $\epsilon_i = 1-(x^*_{i1}+x^*_{i2}); \epsilon_j = 1-(x^*_{j1}+x^*_{j2})$ and $y = \min\{\frac{\epsilon_i}{2}, \frac{\epsilon_j}{2}, x^*_{i1}, x^*_{i2}, x^*_{j1}, x^*_{j2}\}$. If $y = \frac{\epsilon_i}{2}(\frac{\epsilon_j}{2}), x^*_{ik} \leftarrow x^*_{ik} \pm y; x^*_{jk} \leftarrow x^*_{jk} \mp y; k \in \{1,2\}$. If $y = x^*_{i1}$ or x^*_{i2} (x^*_{j1}or x^*_{j2}), $x^*_{ik} \leftarrow x^*_{ik} \mp y; x^*_{jk} \leftarrow x^*_{jk} \pm y; k \in \{1,2\}$.
(The (top) bottom signs in $\pm$ and $\mp$ hold when y takes the values (not) in the brackets.)

These transformations performed on every pair of calls, partitions the call set into four categories according to the values of their corresponding indicator variables in the optimal solution vector x^*:
A) Calls for which both the corresponding indicator variables are set to zero. Let the set be denoted by $S_{(a)}$ and the sum of their x^* values by $x^*(S_{(a)}) = 0$.
B) Calls for which exactly one of the corresponding indicator variables is non-zero. Let the set be denoted by $S_{(b)}$ and the sum of their x^* values by $x^*(S_{(b)})$.
C) Calls which are pairwise crossing but the sum of their (non-zero) indicator variables equals unity. Let the set be denoted by $S_{(c)}$ and the sum of their x^* values by $x^*(S_{(c)})$.
D) At most one call for which the sum of the (non-zero) indicator variables is less than one. Let the call be D and the sum $x^*_{D1} + x^*_{D2} < 1$ with $0.5 > x^*_{D2} \leq x^*_{D1}$, say.

We shall now show rounding schemes for class B and C calls.

Rounding of Class B Calls. Since calls in class B have one of their two indicator variables set to zero, the route function x can be restricted to be defined on the unique route for each call that received a non-zero value. Instead of calls, we need only concentrate on the unique path for each call in class B. Accordingly, we show the rounding on a set of paths.

Lemma 2.1. (Rounding on the line) *Let S be a set of paths on a line $L = (V, E)$ with capacity function $w : E \rightarrow \mathbb{Z}^+_0$ on the edges. Let $x : S \rightarrow [0,1]$ be a function that assigns fractional values to the set of paths and $x(S) = \sum_{s \in S} x(s)$. Further, let $x(.)$ be such that the sum of x values of paths through an edge e on the line is at most $w(e)$. Then $\exists$ a function $x' : S \rightarrow \{0,1\}$ such that $\forall e, \sum_{s \in S: s \text{ contains } e} x'(s) \leq w(e)$ and $x'(S) = \sum_{s \in S} x'(s) \geq \lceil x(S) \rceil$.*
Proof. (sketch) Order the paths according to increasing order of their right end points. Round up the x value of the first path. Then round down the x values of paths intersecting the first path so as to satisfy edge capacity constraints. Now, an induction type argument proves the lemma. □

This rounding lemma for the line serves as a starting step to round the values for paths on the ring. The next lemma captures this.

Lemma 2.2. (Rounding on the ring) *Let S be a set of paths on a ring $R = (V, E)$ with capacity function $w : E \to \mathbb{Z}_0^+$ on the edges. Let $x : S \to [0, 1]$ be a function that assigns fractional values to the set of paths and $x(S) = \sum_{s \in S} x(s)$. Further, let $x(.)$ be such that the sum of x values of paths through an edge e on the ring is at most $w(e)$ and for some edge e_{sat} the sum is exactly $w(e_{sat})$. Then $\exists$ a function $x' : S \to \{0, 1\}$ such that $\forall e, \sum_{s \in S : s \text{ contains } e} x'(s) \leq w(e)$ and $x'(S) = \sum_{s \in S} x'(s) \geq \lceil x(S) \rceil - 1$.*

Proof. Consider the edge e_{sat} in the ring and the set of paths $S_{e_{sat}} \subseteq S$ that pass through it. If there were two paths s_{e_1}, s_{e_2} through e_{sat} such that the former is contained in the latter then consider the following reassignment of their x values; $x(s_{e_1}) \leftarrow \min\{1, x(s_{e_1}) + x(s_{e_2})\}, x(s_{e_2}) \leftarrow x(s_{e_2}) + x_{old}(s_{e_1}) - x(s_{e_1})$, where $x_{old}(s_{e_1})$ is the value of $x(s_{e_1})$ before the reassignment. With this reassignment it is easy to see that all paths through e_{sat} which have x values in $(0, 1)$ are not strictly contained in each other. Call these paths $S^{(1)}_{e_{sat}} = \{s_1, s_2, ..., s_k\}$ where the order in which they appear is according to the increasing order of their clockwise end points. Let e_j be the smallest index such that $\sum_{i=1}^{e_j} x(s_i) \geq j, j = 1, 2, ..., x(S^{(1)}_{e_{sat}}) - 1$. Define $x'(s_{e_j}) = 1, j = 1, 2, ..., x(S^{(1)}_{e_{sat}}) - 1$ and $x'(s_i) = 0$, for $s_i \in S^{(1)}_{e_{sat}} \setminus \{s_{e_1}, s_{e_2}, ..., s_{e_{x(S^{(1)}_{e_{sat}})-1}}\}$. Also, set $x'(s) = x(s)$, for $s \in S_{e_{sat}} \setminus S^{(1)}_{e_{sat}}$. Recall that for the paths in $S_{e_{sat}} \setminus S^{(1)}_{e_{sat}}$ the x values are either 0 or 1.

Argument: For any edge e' in the ring, the sum of the x' values of paths in $S^{(1)}_{e_{sat}}$ that pass through it is at most the sum of their x values rounded down.

Proof. See [4]. □

Now, consider all paths that do not pass through e_{sat}, they lie on the line obtained by removing the edge e_{sat} from the ring. Therefore, we can invoke Lemma 2.1, to obtain a x' function on them which satisfies the condition that the sum of x' values passing through any edge is at most the rounded up value of the sum of their x values. This, combined with the statement of the above argument implies that the x' values of paths in S that pass through any edge e of the ring sum up to at most the capacity of that edge, $w(e)$. Further, we have, $x'(S) = x'(S_{e_{sat}}) + x'(S \setminus S_{e_{sat}}) \geq \lceil x(S) \rceil - 1$. □

Lemma 2.2 immediately suggests a rounding scheme for class B calls such that rounded values at any edge sum up to at most the rounded up value of the sum of their x^* values and at the same time we lose at most one from their cumulative sum. We note that if none of the x^* values at an edge sum exactly to the rounded up value at an edge then we can increase at least one of the x^* values to satisfy the condition or make all x^* equal 1. This is summarized in Corollary 2.3.

Corollary 2.3. (Rounding class B calls) *Given a set of class B calls $S_{(b)}$ with a corresponding route function x^*. There exists a $\{0, 1\}$-route function x' such that (i) at every edge the sum of the x' values through it is at most the rounded up value of the sum of the x^* values and (ii) $x'(S_{(b)}) = \sum_{i \in S_{(b)}} (x'_{i1} + x'_{i2}) \geq \lceil x^*(S_{(b)}) \rceil - 1$.*

Figure 2 shows an example of the rounding of class B calls.

Rounding of Class C Calls. Our next step is to describe a rounding for the class C calls. The general idea behind the rounding is that we can reassign the x^* values corresponding to a call to be either $0, 0.5$ or 1 without losing on their contribution to the objective value or feasibility. These x^* values can then be rounded to 0 or 1. However, to maintain feasibility

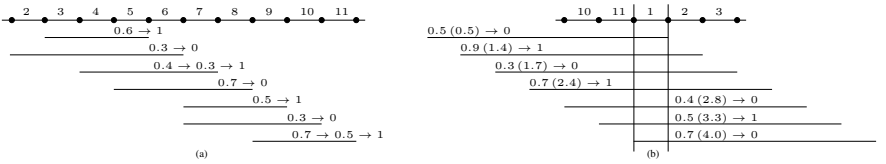


Fig. 2. Rounding of Class B calls. (a) shows rounding of routes not through edge 1. (b) shows rounding of routes through edge 1. Figure does not show routes which received x^* values 0 or 1.

we will need to throw away a constant number of calls, bounded by 2 from above. We start with a lemma that does the rounding when the two variables corresponding to a call are exactly 0.5 each. A definition is in order before we state the lemma: Two edges in a ring are said to be *diametrically opposite* if they connect the end points of two crossing calls. Without loss in generality, we assume that every vertex of the ring is an end point of a call. For presentation's sake, we introduce a fictitious call with index 0, which refers to none of the calls in the input instance.

Lemma 2.4. (Rounding pairwise crossing calls) *Given a set of m mutually crossing calls in a ring with $2m$ vertices and a route function x such that $x_{i1} = x_{i2} = 0.5, i = 1, 2, ..., m$. There exists a $\{0, 1\}$-route function x' and a call $j, 0 \leq j \leq m$, such that (i) $x'_{j1} = x'_{j2} = 0$, (ii) $x'_{i1} + x'_{i2} = 1,\ i \neq j$, and (iii) the sum of the x' values at any edge is at most the sum of the x values rounded up.*

Proof. See [4]. □

Recall that class C calls had their corresponding x^* values summing to exactly one. We have just shown that if these x^* values are 0.5 each then there exists a rounding that loses at most 1 call compared to an optimal solution. The next step is to show how to achieve half-integral values from arbitrary ones. First, we will discard one of the calls from the set of crossing calls. Next, for the remaining calls we appeal to the powerful Okamura-Seymour theorem in [12] to get half-integer values.

Theorem 2.5. (Okamura-Seymour Theorem) *If $G = (V, E)$ is a planar graph with edge capacities $w(e), e \in E$, and can be drawn such that vertices $s_i, t_i, i = 1, ..., k$ are all on the boundary of the infinite region, then the following are equivalent:*
(i) For $1 \leq i \leq k$ there is a flow F_i from s_i to t_i of value q_i such that $\forall e \in E$,

$$\sum_{i=1}^{k} |F_i(e)| \leq w(e)$$

(ii) For each $X \subseteq V$,

$$\sum_{e \in \partial(X)} w(e) \geq \sum_{i \in D(X)} q_i$$

($\partial(X) \subseteq E$ is the set of edges with one end in X and the other in $V \setminus X$. $D(X) \subseteq \{1, 2, ..., k\}$ is $\{i : 1 \leq i \leq k, \{s_i, t_i\} \cap X \neq \emptyset \neq \{s_i, t_i\} \cap (V \setminus X)\}$.) Furthermore, if q and w are integer valued, then the flows F_i may be chosen half-integer valued.

The relation between the theorem and our problem is readily apparent. The ring is a planar graph and all the vertices in it indeed lie on the outer infinite face. The flows correspond to the paths connecting the end vertices. Thus, if we are able to show that the mutually crossing C calls satisfy condition (ii) of the theorem then we can obtain

half-integer valued flows (or equivalently, half-integer values for the routes of a call). Lemma 2.7 addresses this. But, first we need to identify one call among the class C calls which will be discarded for the above theorem to be applied.

We start with some more terminology. Given a ring on $2m$ vertices, two edges are *almost diametrically opposite* if they have $m-2$ edges between them. Note that between any pair of diametrically opposite edges there are exactly $m-1$ edges. For every edge there is exactly one diametrically opposite edge and there are two almost diametrically opposite edges. For a set of m mutually crossing calls with a route function x with $x_{i1}+x_{i2}=1, \forall i$, the total of rounded down sums of x values at diametrically opposite edges is at least $m-1$ and for almost diametrically opposite edges is at least $m-2$.

Lemma 2.6. *Given a set of m mutually crossing calls with a route function x such that $x_{i1}+x_{i2}=1, \forall i$, and $x_{i1}, x_{i2} \notin \{0,1\}$ and an edge e_0 such that the total of the rounded down sum of x values through it and the rounded down sum of x values through its almost diametrically opposite edge is $m-2$. There exist two consecutive edges in the ring such that the rounded down sums of the x values through them are equal.*

Proof. Assume to the contrary that for every pair of consecutive edges the rounded down values are unequal. Let the sum of the x values at an edge e be $x(e)$. For two consecutive edges e, e' it is true that $|x(e)-x(e')|<1$. Therefore, $||\lfloor x(e)\rfloor - \lfloor x(e')\rfloor| = 1$. Consider the edge e_0 and one of its almost diametrically opposite edges e_{m-1} in the ring. They have $m-2$ edges between them (traversing the ring in one of the two possible ways). Denote the edge that has exactly $k-1$ edges between it and e_0 in this above traversal by $e_k, k = \{1,2,...,m-1\}$. It can be proved that $\lfloor x(e_0)\rfloor - \lfloor x(e_k)\rfloor \in \{\pm k, \pm(k-2), \pm(k-4), ..., \pm(k-2\lfloor \frac{k}{2} \rfloor)\}$. Indeed, for $k=1$, it is trivially true. For $k \geq 2$, $\lfloor x(e_0)\rfloor - \lfloor x(e_k)\rfloor \in \{\lfloor x(e_0)\rfloor - \lfloor x(e_{k-1})\rfloor \pm 1\}$, since $||\lfloor x(e_{k-1})\rfloor - \lfloor x(e_k)\rfloor| = 1$. From here, the above statement follows. We now have $\lfloor x(e_0)\rfloor - \lfloor x(e_{m-1})\rfloor \in \{\pm(m-1), \pm(m-3), ..., \pm(m-(2\lceil \frac{m}{2} \rceil - 1))\}$. But, $\lfloor x(e_0)\rfloor + \lfloor x(e_{m-1})\rfloor = m-2$, implying $\lfloor x(e_0)\rfloor - \lfloor x(e_{m-1})\rfloor = m-2-2\lfloor x(e_{m-1})\rfloor$. Or $\lfloor x(e_0)\rfloor - \lfloor x(e_{m-1})\rfloor \in \{\pm(m-2), \pm(m-4), ..., \pm(m-2\lfloor \frac{m}{2} \rfloor)\}$. A contradiction. Thus our hypothesis that no two successive edges have equal rounded down sum of x values is impossible, proving the claim. □

A consequence of Lemma 2.6 is that we can identify one call among the class C calls (essentially, the call, one of whose end points is incident on the consecutive edges identified by Lemma 2.6) such that its removal will make the remaining calls satisfy the Okamura Seymour condition. More accurately, the total of the rounded down sum of x values at any two edges is at least the number of calls (other than the removed call) which cross these two edges (A call is said to *cross* a set of edges in the ring if its end vertices lie in two different components of the graph obtained after removal of the set of edges from the ring). Thus, with edge capacities equal to the rounded down x sums, condition (ii) of Okamura Seymour is satisfied (for a rigorous proof, see [4]). We get,

Lemma 2.7. (Half-integer rounding of crossing calls) *Given a set of m mutually crossing calls on a ring and a route function x such that $x_{i1}+x_{i2}=1, \forall$ calls i. There is a half-integer route function x' and a call j such that (i) $x'_{j1}=x'_{j2}=0$, (ii) $x'_{i1}+x'_{i2}=1$, $\forall$ calls $i \neq j$, and (iii) the sum of the x' values at an edge e is at most the sum of the x values rounded down.*

Lemma 2.7 in conjunction with Lemma 2.4 yields an integer rounding that is close to the fractional optimum by an additive constant of at most 2. We can now state the performance guarantee of the rounding scheme of crossing calls in the corollary below:

Corollary 2.8. (Rounding class C calls) *Given a set of class C calls $S_{(c)}$ on a ring with a corresponding route function x^*. There exists a $\{0,1\}$-route function x' such that (i) for every edge e the sum of x' values of routes through it is at most the rounded down value of the sum of the x^* values and (ii) $x'(S_{(c)}) \geq x^*(S_{(c)}) - 2$.*

Proof. Lemma 2.7 shows a rounding of x^* to half-integer x values losing one on the sum of the x^* values. Applying Lemma 2.4 on those calls that got x values 0.5 for both their variables we get a $\{0,1\}$ route function x'. The sum of the x' values is at most one less than the sum of the x values. Thus, in total we lose at most 2 from the sum of x^* values. Condition (i) follows from Lemma 2.7. □

Assembling the Pieces. Finally, we shall piece together the different parts for solving the RCAC problem. Starting from the optimal fractional solution x^* to the relaxed LP, we adjust the values such that x^* is in the canonical form with respect to parallel and crossing calls, as set forth in the beginning of Section 2.1. If there is a class D call then make it a class B call by setting the lower of the two indicator variables to zero. Next, perform the rounding on class B and class C calls as described in the Corollaries 2.3 and 2.8. For class B calls the sum of the rounded values at any edge is at most the rounded up value of the original x^* values and for class C calls it is at most the rounded down value of the original x^* values. Thus combining the two sums at an edge will satisfy its capacity constraint. In other words, the rounded solution is a feasible one. As regards the objective value, $OPT^* = x^*(S_{(a)}) + x^*(S_{(b)}) + x^*(S_{(c)}) + x^*_{D1} + x^*_{D2} \leq x'(S) + 3.5$. But, $\lfloor OPT^* \rfloor$ is an upper bound on the objective value of the integer linear program. Therefore, the rounded solution is at most 3 away from an integer optimal solution to the ILP. Yielding,

Theorem 2.9. ("Almost" optimal RCAC solution) *Given an instance of RCAC: a set S of m calls on a ring $R = (V, E)$ with integer edge capacities $w(e)$, $e \in E$. There is a polynomial time algorithm that produces a feasible solution routing at most 3 fewer calls compared to an optimal solution.*

3 PCAC on Rings

We turn to the pre-routed variant of the call control problem in rings, namely PCAC. This problem, for example, applies to unidirectional rings where each call is routed by the clockwise path from the sender to the receiver. In addition to a fixed route for each call which is specified in the input, every call has a non-negative profit associated with it. Formally, the PCAC problem is the following:

Input and Objective of PCAC: The input to PCAC consists of a ring $R = (V, E)$ on n vertices, a call set S of m calls together with, for each call, one of the two paths as its pre-specified route, a profit function $p : S \to \mathbb{Z}_0^+$ and a capacity function $w : E \to \mathbb{Z}^+$. Here, a *feasible solution* is a subset $S' \subseteq S$ such that the number of routes of calls in S' through an edge $e \in E$ is at most the capacity $w(e)$ of the edge. The profit of the feasible solution S', denoted $p(S')$, is the sum of the profits of the calls in S'. The objective is a feasible solution $OPT \subseteq S$ with maximum possible profit.

As in the approach to solving RCAC, we shall formulate this problem as an ILP and then show a rounding mechanism. However, unlike for the RCAC variant we obtain only a 2-approximation for the general problem with arbitrary profits.

An ILP for PCAC: Let the call set be $S = \{\{u_i, v_i\} : i = 1, 2, ..., m\}$. Since the routes for each call are specified with the call set we have exactly one indicator variable x_i for call i corresponding to whether the call is accepted and routed along this path. Let $S_e = \{i : \text{ call } i \text{ is routed through edge } e\}$, for edge $e \in E$. The ILP can now be stated as:

$\max \sum_{i=1}^{m} p(i) \cdot x_i$
subject to
$\sum_{i \in S_e} x_i \leq w(e),\ e \in E$
$x_i \in \{0, 1\}$

As in RCAC, we shall call the vector x a route function. A route function is called *feasible* if it satisfies the capacity constraints on the edges.

A 2-approximation: An easy corollary to Lemma 2.1 is that there is a feasible $\{0, 1\}$-route function to a set of paths on the line that routes at least as many calls as any arbitrary route function. When the set of paths on the line have profits p associated with them a similar statement is true with respect to the sum of the profits. However, this does not follow from the rounding scheme described in Lemma 2.1 but from the theory of totally unimodular matrices and network flows. A matrix A is said to be totally unimodular if the determinant of every square sub-matrix of A (including A, if A is a square matrix) is $0, 1,$ or -1. A consequence of a matrix A being totally unimodular is that if it appears as the constraint matrix of a linear program $\max\{c^T x : Ax \leq b, 0 \leq x \leq 1\}$, the LP has an integer optimal solution whenever b is integral. It is a well known fact that a $(0, 1)$-matrix in which the ones appear consecutively in every column (or row) is totally unimodular. From these observations, we have the following lemma.

Lemma 3.1. *Given a set of paths S with profits $p : S \to \mathbb{Z}_0^+$ on a line $L = (V, E)$ and a capacity function $w : E \to \mathbb{Z}^+$. There exists a $\{0, 1\}$-route function x' such that (i) x' is feasible, (ii) $\sum_{i \in S} p(i) \cdot x'(i) \geq \sum_{i \in S} p(i) \cdot x(i)$, for every feasible route function x, and (iii) x' can be computed efficiently.*

The 2-approximation for the PCAC problem is rather simple. Identify an edge e on the ring which has the least capacity $w(e) = w_{min}$. Lemma 3.1 asserts that an optimal feasible set among all calls not routed through e can be found in polynomial time. Next, from the set of calls through e pick the $w(e)$ calls with highest profits to get a second feasible set. Choose the set which has maximum profit between the two. This set will have at least half the profit of an optimal solution to the PCAC problem. This algorithm and its analysis can be modified to yield an approximation ratio of at most $n/(n - L)$, where L is the maximum length of route of any call. This ratio is better than 2 for $L < n/2$.

3.1 Optimal Algorithms for Special Cases of PCAC

In this subsection, we consider three special cases of PCAC on rings and show optimal algorithms that run in time polynomial in the size of the input. We consider: (a) calls have routes of equal length and their profits are arbitrary, (b) calls have "proper" routes

(defined later) and profits are arbitrary, and (c) calls have arbitrary routes but the profit of a call whose route is contained in that of another is at least as great as the profit of the latter. Our algorithms for all of these special cases of PCAC are based on network flow techniques and we shall use the following theorem from [3, p. 315] to derive the results.

Theorem 3.2. *Any linear program that contains (a) at most one* $+1$ *and at most one* -1 *in each column or (b) at most one* $+1$ *and at most one* -1 *in each row, can be transformed into a minimum cost flow problem.*

Theorem 3.2 implies that such a linear program can be solved in strongly polynomial time but also the optimal solution vector to the linear program is integral if the right hand side is an integral vector. Since the edge capacities are integral in our instances, we shall see that we obtain integral optimal solutions for them.

Calls with Paths of Equal Length. For convenience, let us assume that no two calls which have the same end points have been assigned the same route. We shall drop this condition later on. Assume that all routes of calls have equal length of L. Let the vertices of the ring be numbered $0, 1, ..., n-1$ in a clockwise fashion and edge i be incident on vertices i and $i+1 \bmod n, i = 0, 1, ..., n-1$. Let the call set be rearranged such that call i is routed by a path containing vertices i through $(i+L) \bmod n, i = 0, 1, ..., n-1$. If no such call appears in the original call set then introduce such a call with profit 0. This does not alter the original instance. With this rearrangement of indices, for any edge j, precisely the following calls pass through it: namely, calls with indices $(j-L+1) \bmod n$ through j. If $j \geq L-1$ this implies all calls with indices $j-L+1$ through j. If $j < L-1$, all calls with indices 0 through j and those with indices $(j-L+1) \bmod n$ through $n-1$. Thus, we can rewrite the relaxation of the ILP stated at the beginning of Section 3 as:

$$\max \sum_{i=0}^{n-1} p(i) \cdot x_i$$
subject to
$$\sum_{i=j-L+1}^{j} x_i \leq w(j),\ n-1 \geq j \geq L-1$$
$$\sum_{i=0}^{j} x_i + \sum_{i=(j-L+1) \bmod n}^{n-1} x_i \leq w(j),\ 0 \leq j < L-1$$
$$0 \leq x_i \leq 1,\ i = 0, 1, ..., n-1$$

Now, define $X(-1) = 0, X(k) = \sum_{i=0}^{k} x_i,\ k = 0, 1, ..., n-1$. Substituting these new variables in the above LP we obtain (unless we use $\bmod n$, -1 is NOT $n-1$):

$$\max \sum_{i=0}^{n-1} p(i) \cdot (X(i) - X(i-1))$$
subject to
$$X(j) - X(j-L) \leq w(j),\ n-1 \geq j \geq L-1$$
$$X(j) + X(n-1) - X((j-L) \bmod n) \leq w(j),\ 0 \leq j < L-1$$
$$0 \leq X(i) - X(i-1) \leq 1, i = 0, 1, ..., n-1$$
$$X(-1) = 0$$

Naturally, for integer solutions, $X(n-1)$ is an integer between 0 and n. Thus, we can set $X(n-1) = t$, for some integer $t, 0 \leq t \leq n$. This reduces the constraint matrix to one where each row has at most one $+1$ and one -1 by taking $X(n-1)$ to the right hand side. That the above LP has an integer optimal solution, obtained using network flow techniques, can be deduced from Theorem 3.2 (see comments appearing immediately after the theorem). Integer solution for the modified LP implies integer solutions for the original LP as $x_i = X(i) - X(i-1), i = 0, 1, ..., n-1$. Note also that if X_t^* denotes a

feasible vector for the above LP with $X(n-1) = t$ then $\lambda X^*_{t_1} + (1-\lambda)X^*_{t_2}$ is a feasible solution to the LP when $X(n-1) = \lambda t_1 + (1-\lambda)t_2$. Thus, the approach to solve the original problem is a modified binary search for values of $X(n-1)$ between 0 and n.

In the foregoing argument we had assumed that no two calls had the same route if they shared the same end points. This can be easily patched. First, order the distinct routes for calls as before. Next, among calls having the same routes order arbitrarily. For this order of calls the above arguments go through.

Calls with "Proper" Routes & Calls with Restricted Profits. When the input to PCAC is such that no route of a call is strictly contained in that of another, the set of routes is said to be proper. The input is said to have restricted profits, if for any pair of parallel calls, the profit of the call whose route is completely contained in that of another is at least as great as the profit of the latter. For both these cases, we can transform the LP into the form required by Theorem 3.2. We omit the details. Note that equal profits for all calls, studied in [1], is a special case of restricted profits.

References

1. U. Adamy, C. Ambuehl, R.S. Anand, and T. Erlebach. Call control in rings. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming ICALP 2002*, LNCS 2380, pages 788–799, 2002.
2. R. Adler and Y. Azar. Beating the logarithmic lower bound: randomized preemptive disjoint paths and call control algorithms. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms SODA 1999*, pages 1–10, 1999.
3. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: Theory, algorithms and application*. Prentice-Hall, New York, USA, 1993.
4. R.S. Anand and T. Erlebach. Routing and call control algorithms for ring networks. Technical Report TIK-Report 171, ETH Zürich, May 2003. Available electronically at `ftp://ftp.tik.ee.ethz.ch/pub/publications/TIK-Report171.pdf`.
5. A. Blum, A. Kalai, and J. Kleinberg. Admission control to minimize rejections. In *Proceedings of the 7th Workshop on Algorithms and Data Structures WADS 2001*, LNCS 2125, pages 155–164, 2001.
6. M.C. Carlisle and E.L. Lloyd. On the k-coloring of intervals. *Discrete Applied Mathematics*, 59:225–235, 1995.
7. T. Erlebach. Approximation algorithms and complexity results for path problems in trees of rings. In *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science MFCS 2001*, LNCS 2136, pages 351–362, 2001.
8. T. Erlebach and K. Jansen. The maximum edge-disjoint paths problem in bidirected trees. *SIAM Journal on Discrete Mathematics*, 14(3):326–355, 2001.
9. J.A. Garay, I.S. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithm. *Journal of Algorithms*, 23:180–194, 1997.
10. N. Garg, V.V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.
11. C. Nomikos, A. Pagourtzis, and S. Zachos. Minimizing request blocking in all-optical rings. In *IEEE INFOCOM*, 2003.
12. H. Okamura and P. Seymour. Multicommodity flows in planar graphs. *Journal of Combinatorial Theory, Series B*, 31:75–81, 1981.
13. P.J. Wan and L.Liu. Maximal throughput in wavelength-routed optical networks. In *Multichannel Optical Networks: Theory and Practice*, volume 46 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 15–26. AMS, 1998.

Algorithms and Models for Railway Optimization*

Dorothea Wagner

University of Karlsruhe, Department of Computer Science, Germany

Abstract. Mobility is increasing in a way that calls for systematic traffic planning in a broad context. In Europe the railways are requested to play a central role in this development. Future developments and improvements of European railways will have an impact on people's lives and therefore on society in general. The problems arising in this context are large and highly complex. Here are many interesting and challenging algorithmic problems waiting to be studied. Research topics include the network design, line planning, time table generation, crew scheduling, rolling stock rostering, shunting, time table information and delay management.

In this talk we present models and algorithmic methods for several of these problems. We will discuss the interplay between algorithmic aspects and practical issues like availability and quality of data. The focus will be on two topics from network design and time table information respectively where we have ongoing cooperation with railway companies. As an example from network design, we will consider a scenario where the effects of introducing new train stops in the existing railway network is studied. For time table information whose algorithmic core problem is the computation of shortest paths we discuss new algorithmic issues arising from the huge size of the underlying data.

1 Introduction

Railway systems, as all transport systems, can be modeled in a uniform way as network systems. Planning and optimization in this context are typical examples of structured, combinatorial problems, such as scheduling, network flows, shortest paths and routing problems. However, the conditions and circumstances are induced by real-world demands. Therefore, a first step consists in transforming such complex practical problems into a simplified model still describing its most important characteristics.

Many traffic optimization problems are $\mathcal{NP}$-hard. Discrete optimization techniques have been successfully applied in the past [4]. However, because of the increasing size of today's railway systems the applicability of these methods is limited. On the other hand experiences with traffic optimization problems have

* The author gratefully acknowledges financial support from the Human Potential Programme of the European Union under contract no. HPRN-CT-1999-00104 (AMORE) and the Deutsche Forschungsgemeinschaft under grant WA 654/12-1.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 198–206, 2003.
© Springer-Verlag Berlin Heidelberg 2003

shown that a careful analysis of the real-world input data can lead to a tremendous data reduction and make even very large instances tractable for exact methods.

In this talk we present models and algorithmic methods for several of these problems. We will discuss the interplay between algorithmic aspects and practical issues like availability and quality of data. First, we will give a short overview of the railway optimization process and the algorithmic problems occurring in this context. The focus will then be on two topics where properties of the input data, respectively the problem size have major influence on the model of choice and the applicability and efficiency of algorithmic methods.

2 The Railway Optimization Process

Starting with a given transport demand, the goal is to design an optimal complete railway system satisfying this demand. Of course, this is a highly complex optimization problem. It is therefore decomposed into sub problems that are solved separately. However, these cannot be considered independently. The main steps respectively sub problems consist of

- network design,
- line planning,
- time tabling,
- traffic planning,
- rolling stock rostering,
- crew scheduling,
- shunting, maintenance,
- operation control,
- time table information,
- ticket pricing.

In order to know the transport demand, reliable information on the number of passengers and amount of cargo transport requirements are needed. The basis for the railway optimization process is then the so-called origin-destination matrix (short OD-matrix). As it is often the case in real-world scenarios, the estimation of the OD-matrix is already a challenging issue. Actually, the way a mathematical model for any of the sub problems is chosen depends very much on the informations available about the transport demand and other relevant parameters. Examples on this will be given in the next two sections. See also [4] and [31].

3 Network Design

Planning and evaluation of measures for building up and extending railway systems require algorithmic techniques to handle the distribution of traffic load on a network according to its capacity. Such a problem is already quite complex for one national railway system. Today, railway systems are not build from scratch but there is already infrastructure. Accordingly, the issue in network design is to increase the attractiveness of train travel in an existing railway network.

3.1 Station Location

Given a railway network together with information on the population and their use of the railway infrastructure, we are considering the effects of introducing new train stops in the existing railway network. One effect may e.g. concern the accessibility of the railway infrastructure to the population, measured in how far people live from their nearest train stop. Other effects are the change in travel time for the railway customers that is induced by new train stops and the increased cost for the railway company to build and operate new stops. As part of a project with the largest German rail company (DB), we studied different objective functions taking both mentioned effects into account [13], [32].

A first goal might be to establish as few stops as possible in such a way, that (nearly) all customers are covered, and to simplify the regions where people live by considering only points in the plane instead. This leads to the following problem, which is already $\mathcal{NP}$-hard in general.

Definition 1 (Covering Along Lines). *Given a set $\mathcal{P}$ of integer-coordinate points in the plane, a connected set $\mathcal{L}$ which is given as the union of a finite number of line segments in the plane, and positive integers d and $K < |\mathcal{P}|$, can the points of $\mathcal{P}$ be covered by at most K discs of diameter d, all with center points in $\mathcal{L}$?*

However, in the special case of only one line segment, *Covering Along Lines* is a set covering problem whose constraint has the *consecutive ones property* (e.g., [11]). This result can be extended to sets of line segments, if no settlement can be covered by two stops from different line segments. It is known that the problem is polynomially solvable by linear programming if the consecutive ones property holds. More efficient procedures for solving set covering problems, where the covering matrix has the consecutive ones property are based on the theory of network matrices, or use a transformation to a shortest path problem, see [30].

Analysis of the real data of DB show that for large parts of the network only one line segment needs to be considered for most settlements. Accordingly, a strategy can be developed where the network is decomposed into sub problems satisfying the consecutive ones property. As a sub case, the station location on two intersecting lines occurs. This case is studied in [19] and a polynomial approach for solving the problem for sufficiently large angle formed by the two lines is developed. In a preliminary study [20], the decomposition strategy is combined with data reduction techniques and leads to promising results for the real data of DB.

In [13], the minimization of the overall traveling time over all customers, which is given by the access time of the customers to their (Euclidean) closest station, their travel time within the vehicle and their time to go from the final station to their destination, was considered. It turned out that due to the savings in the access times, it is possible to decrease the overall travel times by establishing (many) new stations. This result was obtained by a genetic algorithm, using the real data of DB.

Similar scenarios are also considered in [17]. In [24], [25] and [26] the stop location problem for the public transportation network in Brisbane, Australia is studied, where either only the actual stops are considered, or it is assumed that a finite candidate set of new stops is given. This leads to an unweighted set covering problem (like the one tackled in [34]). In the context of stop location this problem has been solved by [25] using the Lagrangian-based set covering heuristic of [5]. Very recently, another discrete stop location model has been developed in [18]. They investigate which candidate stops along one given line in Sevilla should be opened, taking into account constraints on the interstation space. Finally, the more realistic case where the settlements are modeled by polygons is considered in [33].

4 Time Table Information

European railway timetable information today consists of much more than determining a fastest railway connection between two stations. First, the underlying data set has enormously increased within the last decade. For example, the timetable information system Hafas [12], which is provided by DB, and which is used not only by Germany, but also by Austria, Denmark, Switzerland, and many more European countries, contains the national railway connections data from nearly all European railway companies. Furthermore, more and more data from local transport systems, including even city bus connections, are integrated. The growing size of data underlying such timetable information systems calls for sophisticated speed-up techniques. That is, although the algorithmic question to be answered here is "easy" (polynomially solvable) from a theoretical point of view, running time in practice is a real issue. Of course, the problem becomes even harder, when, in addition to the time needed for a connection, its price, number of interchange stations, type of trains etc. are requested.

In order to satisfy long distance queries efficiently, a condensation of large parts of the underlying graph can be advantageous. On the other hand, algorithms for solving shortest paths problems can be improved by using the geography underlying railway systems.

4.1 Models

The first step in time table information is to model the problem in a way that subsequent queries asking for optimal itineraries can be efficiently answered. The main target that underlies the modeling is to process a vast number of on-line queries as fast as possible. In railway systems, we are concerned with a specific, query-intensive scenario, where a central server is directly accessible to any customer either through terminals in train stations or through a web interface, and has to answer a potentially infinite number of queries. The main goal in such an application is to reduce the *average response time* for a query.

Two main approaches have been proposed for modeling timetable information: the *time-expanded* approach [23,29,36,37], and the *time-dependent* approach [3,27,28]. The common characteristic of both approaches is that a query

is answered by applying some shortest path algorithm to a suitably constructed digraph; see also [22] for a discussion of graph models for time-table information. Techniques for solving general pareto-optimal problems have been presented in [23]. In [21] modeling complex real-world aspects with focus on space consumption is considered.

The time-expanded approach [36] constructs the time-expanded digraph in which every node corresponds to a specific time event (departure or arrival) at a station and edges between nodes represent either elementary connections between the two events (i.e., served by a train that does not stop in-between), or waiting within a station. Depending on the problem that we want to solve, the construction assigns specific fixed weights to the edges. This naturally results in the construction of a very large (but usually sparse) graph. The time-dependent approach [3] constructs the time-dependent digraph in which every node represents a station and two nodes are connected by an edge if the corresponding stations are connected by an elementary connection. The weights on the edges are assigned "on-the-fly", i.e., the weight of an edge depends on the time in which the particular edge will be used by the shortest path algorithm to answer the query.

The two most frequently encountered timetable problems are the earliest arrival and the minimum number of changes problems. In the *earliest arrival* problem, a query consists of a departure and an arrival station, and a departure time (including the departure day). Connections are valid if they depart at least at the given departure time, and the goal is to find the valid connection that minimizes the difference between the arrival time and the given departure time. There are two variants of the problem depending on whether train changes within a station are assumed to take negligible time (simplified version) or not. In the *minimum number of changes* problem, a query consists only of a departure station A and an arrival station B. Trains are assumed to operate daily (and there is no restriction on the number of days a timetable is valid). All connections from A to B are valid, and the goal is to find the valid connection that minimizes the number of train changes when considering an itinerary from A to B. Then combinations of the above problems can be seen as bicriteria or pareto-optimal problems.

For the time-expanded model, the simplified version of the earliest arrival problem has been extensively studied [36,37]. In [3] it is argued (theoretically) that the time-dependent approach is better than the time-expanded one when the simplified version of the earliest arrival problem is considered. We will report on a recent paper [38] that compares the time-expanded and the time-dependent approaches with respect to modeling aspects and performance.

4.2 Geometric Speed-up Techniques

One of the features of travel planning in general and time table information especially, is the fact that the network does not change for a certain period of time while there are many queries for shortest paths. This justifies a heavy preprocessing of the network to speed up the queries. Although pre-computing

and storing the shortest paths for all pairs of nodes would give us "constant-time" shortest-path queries, the quadratic space requirement for traffic networks with 10^5 and more nodes prevents us from doing so. In [36], we explored the possibility to reduce the search space of Dijkstra's algorithm in time table information by using precomputed information that can be stored in $O(n + m)$ space. One key idea is the use of angular sectors to reduce the search space for the online shortest-path computations.

In this talk we will report on more general results from a recent study [39]. The following very fundamental observation on shortest paths is used. That is, in general, an edge that is not the first edge on a shortest path to the target can be ignored safely in any shortest path computation to this target. More precisely, we apply the following concept:

- In the preprocessing, for each edge e, the set of nodes $S(e)$ is stored that can be reached by a shortest path starting with e.
- While running Dijkstra's algorithm, edges e for which the target is not in $S(e)$ are ignored.

As storing all sets $S(e)$ would need $O(mn)$ space, we relax the condition by storing a geometric object for each edge that contains *at least* $S(e)$. Remark that this does in fact still lead to a correct result, but may increase the number of visited nodes to more than the strict minimum (i.e. the number of nodes in the shortest path). In order to generate the geometric objects, an embedding of the graph is used. For the application of travel information systems, such a layout is for example given by the geographic locations of the nodes. It is however not required that the edge lengths are derived from the layout. In fact, for some of our experimental data this is even not the case. Actually, results from [2] show that such an embedding can be even computed "artificially" from the travel time informations contained in the time table data using graph drawing methods.

In an experimental study [39] we examined the impact of various different *geometric objects* and consider Dijkstra for general embedded graphs. It turns out that the number of nodes visited by Dijkstra's algorithm can be reduced to 10%.

4.3 Multi-level Graphs

Several of the approaches used so far in traffic engineering introduce speed-up techniques based on *hierarchical decomposition*. For example, in [1,6,14,15] graph models are defined to abstract and store road maps for various routing planners for private transport. Similarly, in [35] a space reduction method for shortest paths in a transportation network is introduced. The idea behind such techniques is to reduce the size of the graph in which shortest path queries are processed by replacing precomputed shortest paths by edges. The techniques are hierarchical in the sense that the decomposition may be repeated recursively. Several theoretical results on shortest paths are based on the same intuition regarding planar graphs [9,10,16] and graphs of small treewidth [7,8].

In [36], a first attempt is made to introduce and evaluate a speed-up technique based on hierarchical decomposition, called *selection of stations*. Based on a small set of selected vertices an auxiliary graph is constructed, where edges between selected vertices correspond to shortest paths in the original graph. Consequently, shortest path queries can be processed by performing parts of the shortest path computation in the much smaller and sparser auxiliary graph. In [36], this approach is extensively studied for one single choice of selected vertices, and the results are quite promising.

In this talk, we will report on a subsequent detailed and systematic experimental study of such a space reduction approach given in [37]. We introduce the *multi-level graph model* that generalizes the approach of [36]. A multi-level graph $\mathcal{M}$ of a given weighted digraph $G = (V, E)$ is a digraph which is determined by a sequence of subsets of V and which extends E by adding multiple levels of edges. This allows to efficiently construct a subgraph of $\mathcal{M}$ which is substantially smaller than G and in which the shortest path distance between any of its vertices is equal to the shortest path distance between the same vertices in G. Under the new framework, the auxiliary graph used in [36] – based on the selection of stations – can be viewed as adding just one level of edges to the original graph. A distance-preserving speed-up technique based on a hierarchical decomposition using the multi-level graph model was implemented and evaluated on train data of the German railways. The processed queries are a snapshot of the central Hafas server in which all queries of customers of all ticket offices in Germany were recorded over several hours. From the time-table information, the so-called *time-expanded train graph* is generated in a preprocessing step. Based on that graph, for various numbers l of levels and sequences of subsets of vertices the corresponding multi-level graphs evaluated.

The study concentrates in measuring the improvement in the performance of Dijkstra's algorithm when it is applied to a subgraph of $\mathcal{M}$ instead of being applied to the original train graph. The experiments demonstrate a clear speed-up of the hierarchical decomposition approach based on multi-level graphs. Given the complexity of the recursive construction of the multi-level graph (or of similar models proposed in the literature), this concept might appear to be more of theoretical interest than of practical use. To our surprise, our experimental study with multi-level graphs for this specific scenario exhibited a considerable improvement in performance regarding the efficient computation of on-line shortest path queries. For the best choice of all parameters considered we obtained a speed-up of about 11 for CPU time and of about 17 for the number of edges hit by Dijkstra's algorithm.

References

1. R. Agrawal and H. Jagadish. Algorithms for Searching Massive Graphs. *IEEE Transact. Knowledge and Data Eng.*, Vol. 6, 225–238, 1994.
2. U. Brandes, F. Schulz, D. Wagner, and T. Willhalm. Travel Planning with Self-Made Maps. *Proceedings of 3rd Workshop Algorithm Engineering and Experiments (ALENEX '01)*, volume 2153 of *Springer LNCS*, 132–144, 2001.

3. G. S. Brodal and R. Jacob. Time-dependent networks as models to achieve fast exact time-table queries. Technical Report ALCOMFT-TR-01-176, ALCOM-FT, September 2001.
4. M. R. Bussiek, T. Winter and U. T. Zimmermann. Discrete optimization in public rail transport. *Mathematical Programming* 79(3), pp.415–444, 1997.
5. A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. *Operations Research*, 47(5):730–743, 1999.
6. A. Car and A. Frank. Modelling a Hierarchy of Space Applied to Large Road Networks. *Proc. Int. Worksh. Adv. Research Geogr. Inform. Syst.* (IGIS '94), 15–24, 1994.
7. S. Chaudhuri and C. Zaroliagis. Shortest Paths in Digraphs of Small Treewidth. Part I: Sequential Algorithms. *Algorithmica*, Vol. 27, No. 3, 212–226, Special Issue on Treewidth, 2000.
8. S. Chaudhuri and C. Zaroliagis. Shortest Paths in Digraphs of Small Treewidth. Part II: Optimal Parallel Algorithms. *Theoretical Computer Science*, Vol. 203, No. 2, 205–223, 1998.
9. G. Frederickson. Planar graph decomposition and all pairs shortest paths. *Journal of the ACM*, Vol. 38, Issue 1, 162–204, 1991.
10. G. Frederickson. Using Cellular Graph: Embeddings in Solving All Pairs Shortest Path Problems. *Journal of Algorithms*, Vol. 19, 45–85, 1995.
11. M.R. Garey and D.S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
12. Hafas is a trademark of Hacon Ingenieurgesellschaft mbH, Hannover, Germany. See `http://bahn.hafas.de`.
13. H.W. Hamacher, A. Liebers, A. Schöbel, D. Wagner, and F. Wagner. Locating new stops in a railway network. *Electronic Notes in Theoretical Computer Science*, 50(1), 2001.
14. K. Ishikawa, M. Ogawa, S. Azume, and T. Ito. Map Navigation Software of the Electro Multivision of the '91 Toyota Soarer. *IEEE Int. Conf. Vehicle Navig. Inform. Syst.*, 463–473, 1991.
15. S. Jung and S. Pramanik. HiTi Graph Model of Topographical Road Maps in Navigation Systems. *Proc. 12th IEEE Int. Conf. Data Eng.*, 76–84, 1996.
16. D. Kavvadias, G. Pantziou, P. Spirakis, and C. Zaroliagis. Hammock-on-Ears Decomposition: A Technique for the Efficient Parallel Solution of Shortest Paths and Other Problems. *Theoretical Computer Science*, Vol. 168, No. 1, 121–154, 1996.
17. E. Kranakis, P. Penna, K. Schlude, D.S. Taylor, and P. Widmayer. Improving customer proximity to railway stations. Technical report, ETH Zürich, 2002. To appear in *Proceedings 5th Conference on Algorithms and Complexity, (CIAC'03)*, 2003.
18. G. Laporte, J.A. Mesa, and F.A. Ortega. Locating stations on rapid transit lines. *Computers and Operations Research*, 29:741–759, 2002.
19. F. M. Mammana, S. Mecke and D. Wagner. The station location problem on two intersecting lines. Submitted.
20. S. Mecke and D. Wagner. In preparation.
21. M. Schnee, M. Müller-Hannemann and K. Weihe. Getting train timetables into the main storage. *Electronic Notes in Theoretical Computer Science*, 66, 2002.
22. R. Möhring. *Angewandte Mathematik – insbesondere Informatik*, pages 192–220. Vieweg, 1999.

23. M. Müller-Hannemann and K. Weihe. Pareto shortest paths is often feasible in practice. In *Proceedings 5th Workshop on Algorithm Engineering*, volume 2141 of *Springer LNCS*, pages 185–198, 2001.
24. A. Murray, R. Davis, R.J. Stimson, and L. Ferreira. Public transportation access. *Transportation Research D*, 3(5):319–328, 1998.
25. A. Murray. Coverage models for improving public transit system accessibility and expanding access. Technical report, Department of Geography, Ohio State University, 2001.
26. A. Murray. Strategic analysis of public transport coverage. *Socio-Economic Planning Sciences*, 35:175–188, 2001.
27. A. Orda and R. Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM*, 37(3), 1990.
28. A. Orda and R. Rom. Minimum weight paths in time-dependent networks. *Networks*, 21, 1991.
29. S. Pallottino and M. G. Scutellà. *Equilibrium and Advanced Transportation Modelling*, chapter 11. Kluwer Academic Publishers, 1998.
30. A. Schöbel. Set covering problems with consecutive ones property. Technical report, Universität Kaiserslautern, 2001.
31. A. Schöbel. *Customer-oriented optimization in public transportation.* Habilitation Thesis, 2002.
32. A. Schöbel, H.W. Hamacher, A. Liebers, and D. Wagner. The continuous stop location problem in public transportation. Technical report, University of Kaiserslautern, Wirtschaftsmathematik, 2002. Report in Wirtschaftsmathematik Nr. 81/2001. Submitted.
33. A. Schöbel and M. Schröder. Covering population areas by railway stops. *Proceedings of OR 2002, Klagenfurt*, 2002.
34. C. Toregas, R. Swain, C. ReVelle, and L. Bergman. The location of emergency facilities. *Operations Research*, 19:1363–1373, 1971.
35. L. Siklóssy and E. Tulp. The Space Reduction Method: A method to reduce the size of search spaces. *Information Processing Letters*, 38(4), 187–192, 1991.
36. F. Schulz, D. Wagner, and K. Weihe. Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *Journal of Experimental Algorithmics*, volume 5, article 12, 2000.
37. F. Schulz, D. Wagner, and C. Zaroliagis. Using multi-level graphs for timetable information. *Proceedings 4th Workshop on Algorithm Engineering and Experiments (ALENEX 2002)*, volume 2409 of *Springer LNCS*, 43–59, 2002.
38. F. Schulz, D. Wagner, and C. Zaroliagis. Two approaches for time-table information: A comparison of models and performance. Submitted.
39. D. Wagner and T. Willhalm. Geometric speed-up techniques for finding shortest paths in large sparse graphs. Technical Report 183, Preprints in Mathematics and Computer Science at University of Konstanz, 2003. Submitted.

Approximation of Rectilinear Steiner Trees with Length Restrictions on Obstacles

Matthias Müller-Hannemann and Sven Peyer

Research Institute for Discrete Mathematics
Rheinische Friedrich-Wilhelms-Universität Bonn
Lennéstr. 2, 53113 Bonn, Germany
{muellerh,peyer}@or.uni-bonn.de
http://www.or.uni-bonn.de/~muellerh/
http://www.or.uni-bonn.de/~peyer/

Abstract. We consider the problem of finding a shortest rectilinear Steiner tree for a given set of points in the plane in the presence of rectilinear obstacles. The Steiner tree is allowed to run over obstacles; however, if we intersect the Steiner tree with some obstacle, then no connected component of the induced subtree must be longer than a given fixed length L. This kind of length restriction is motivated by its application in VLSI design where a large Steiner tree requires the insertion of buffers (or inverters) which must not be placed on top of obstacles.
We show that the length-restricted Steiner tree problem can be approximated with a performance guarantee of 2 in $O(n \log n)$ time, where n denotes the size of the associated Hanan grid. Optimal length-restricted Steiner trees can be characterized to have a special structure. In particular, we prove that a certain graph, which is a variant of the Hanan grid, always contains an optimal solution. Based on this structural result, we can improve the performance guarantee of approximation algorithms for the special case that all obstacles are of rectangular shape or of constant complexity, i.e. they are represented by at most a constant number of edges. For such a scenario, we give a $\frac{5}{4}\alpha$-approximation and a $\frac{2k}{2k-1}\alpha$-approximation for any integral $k \geq 4$, where α denotes the performance guarantee for the ordinary Steiner tree problem in graphs.

Keywords: Rectilinear Steiner trees, obstacles, VLSI design, approximation algorithms

1 Introduction and Overview

Problem definition. The rectilinear Steiner tree problem is a key problem in VLSI layout. In this paper we study the rectilinear Steiner tree problem in the presence of rectilinear obstacles. To define the problem, an *edge* is a horizontal or vertical line connecting two points in the plane. A *rectilinear tree* is a connected acyclic collection of edges which intersect only at their endpoints. A *rectilinear Steiner tree* for a given set of terminals is a rectilinear tree such that each terminal is an endpoint of some edge in the tree. In this paper, distances are

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 207–218, 2003.
© Springer-Verlag Berlin Heidelberg 2003

always measured based on the L_1 metric if not otherwise stated. The length of a tree is the sum of the lengths of all its edges. A shortest rectilinear Steiner tree is called *Steiner minimum tree (SMT)*.

Throughout this paper, an *obstacle* is a connected region in the plane bounded by one or more simple rectilinear polygons such that no two polygon edges have an inner point in common (i.e. an obstacle may contain holes). For a given set of obstacles $\mathcal{O}$ we require that the obstacles be disjoint, except for possibly a finite number of common points. By ∂O we denote the boundary of an obstacle O. Every obstacle O is weighted with a factor $w_O \geq 1$ (regions not occupied by an obstacle and boundaries of obstacles all have unit weight). These weights are used to compute a weighted tree length which we want to minimize.

Moreover, we introduce length restrictions for those portions of a tree T which run over obstacles. Namely, for a given parameter $L \in \mathbb{R}_0^+$ we require the following for each obstacle $O \in \mathcal{O}$ and for each strictly interior connected component T_O of $(T \cap O) \setminus \partial O$: the (weighted) length $\ell(T_O)$ of such a component must not be longer than the given length restriction L. Note that the intersection of a Steiner minimum tree with an obstacle may consist of more than one connected component and that our length restriction applies individually for each connected component.

Problem 1 (Length-restricted Steiner tree problem (LRSTP)).
Instance: *A set of terminal points S in the plane, a set of obstacles $\mathcal{O}$ such that no terminal point lies in the interior of some obstacle, and a length restriction $L \in \mathbb{R}_0^+$.*
Task: *Find a rectilinear Steiner tree T of minimum (weighted) length such that for all obstacles $O \in \mathcal{O}$, all connected components T_O of $(T \cap O) \setminus \partial O$ satisfy $\ell(T_O) \leq L$.*

An optimal solution of an instance of the length-restricted Steiner tree problem (LRSTP) is called a *length-restricted Steiner minimum tree (LRSMT)*. Obviously, LRSTP is an NP-hard problem as it contains the rectilinear Steiner minimum tree problem as a special case, which is well-known to be NP-hard [10].

Background and application. The motivation to study the length-restricted Steiner tree problem stems from its application in the construction of buffered routing trees in VLSI design [1], [2], [7].Consider a situation where we want to connect a signal *net*, specified by a source r and a set of sinks S. This gives us an instance of the rectilinear Steiner tree problem with the terminal set $\{r\} \cup S$. A *routing tree* is a tree rooted at the source such that each sink is a leaf. A *buffered routing tree T* is a routing tree with buffers[1] located on its edges. The *subtree driven by a buffer b (or the source)* is the maximal subtree of T which is rooted at b and has no internal buffers. The *capacitive load* of a subtree driven by b is the sum of the wire capacitance of the subtree and the input capacitances of its leaves. The source, as well as each type of buffer, can only drive a certain

[1] A buffer (also called repeater) is a circuit which logically realizes the identity function $id : \{0,1\} \mapsto \{0,1\}, id(x) = x$.

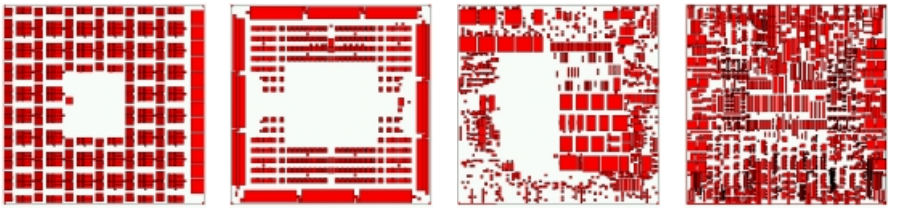


Fig. 1. Typical shape and distribution of obstacles (macros and other circuits) on current chip designs by IBM.

respective maximum load. Hence, the insertion of buffers in a routing tree may be necessary.

Preplaced macros or other circuits play the role of obstacles. Due to the availability of several routing layers, obstacles usually do not block wires, but it is impossible to place a buffer (or inverter) on top of an obstacle. For simplicity, we use the same length restriction for all obstacles in our formulation. However, all our results carry over to the case that each obstacle O has an individual length restriction L_O. In particular, by setting $L_O = 0$ for an obstacle, we can model the case that the interior of O must be completely avoided.

In real world applications, most obstacles are rectangles or of very low complexity. Figure 1 gives an impression of the shape, size and distribution of obstacles on typical chip designs.

Electrical correctness and minimization of power consumption for non-critical nets with respect to timing motivates the minimum buffered routing problem, which we shall define now. The *cost* of a buffered routing tree may, for example, be its total capacitance (wire capacitance plus input capacitance of buffers) as a measure for power consumption, or merely just the number of inserted buffers.

Problem 2 (Minimum Buffered Routing Problem (MBRP)).
Instance: *A source s and sinks $t_1, \ldots, t_k$ with coordinates on a chip image, input capacitances of the sinks, and a library of available buffer types with input capacitances and upper load constraints.*
Task: *Find a minimum cost buffered routing tree connecting the source to all sinks such that the capacitive load of the source and all inserted buffers is within the given load constraints.*

Alpert et al. [2] gave approximation algorithms for MBRP in a scenario without obstacles for a single buffer type. Their algorithms use approximations of the rectilinear Steiner minimum tree as a subroutine because such trees yield a lower bound on the necessary wiring capacitance. However, in the presence of large obstacles no feasible buffering of a given tree might be possible any more.

We introduce length restrictions on obstacles to overcome this problem as they limit the wire capacitance of a connected tree component which runs over some blocked area. Of course, the length restriction parameter has to be chosen carefully with respect to the available buffer library and technology parameters like unit wire capacitance. This is still a simplified model because the load of a

subtree also crucially depends on the input capacitances of its leaves. One way to get rid of this complication would be to require that each internal connected component running over an obstacle has not only a length restriction but also a given upper bound on the number of its leaves (a *fanout restriction*). A second possibility is to introduce a family of length restriction parameters $L_1 \geq L_2 \geq \cdots \geq L_i \geq \ldots$ with the interpretation that for a component T_O with i leaves the length constraint $\ell(T_O) \leq L_i$ applies. In both models it is then always possible to insert additional buffers into a tree such that no load violations occur. As a first step to extending the approximation results for MBRP to the case with obstacles, we look for good approximations of the LRSTP with one of these additional types of restrictions. Just for simplicity of presentation in this paper we consider only the version of LRSTP as defined in Problem 1. However, fanout restrictions as well as fanout dependent length restrictions are easily incorporated into our algorithmic approach and change none of our results with respect to approximation guarantees and asymptotic running times.

Previous work. The literature on the Steiner tree problem is very comprehensive. For an introduction see, for example, the monographs by Hwang, Richards, and Winter [13] and Prömel and Steger [19]. Given a set of terminals in the plane without obstacles, the shortest rectilinear Steiner tree can be approximated in polynomial time to within any desired accuracy using Arora's or Mitchell's celebrated approximation schemes [3], [16].

An obstacle which has to be avoided completely will be referred to as *hard obstacle*. Most previous work dealing with obstacles considered hard obstacles. Given a finite point set S in the plane and a set of obstacles $\mathcal{O}$, the *Hanan grid* [11] is obtained by constructing a vertical and a horizontal line through each point of S and a line through each edge used in the description of the obstacles. The importance of the Hanan grid lies in the fact that it contains a rectilinear Steiner minimum tree. Ganley and Cohoon [9] observed that the rectilinear Steiner tree problem with hard obstacles can be solved on a slightly reduced Hanan grid. Several more variants and generalizations of the Steiner tree problem are solvable on the Hanan grid; for a survey see Zachariasen's catalog [21]. As a consequence, all these variants can be solved as instances of the Steiner tree problem in graphs. (Given a connected graph $G = (V, E)$, a length function ℓ, and a set of terminals $S \subseteq V$, a *Steiner tree* is a tree of G containing all vertices of S. A Steiner tree T is a *Steiner minimum tree* of G if the length of T is minimum among all Steiner trees.) The best available approximation guarantee for the Steiner problem in general graphs is $\alpha = 1 + \frac{\ln 3}{2} \approx 1.55$, obtained by Robins and Zelikovsky [20].

Miriyala, Hashmi and Sherwani [15] solved the case of a single rectangular hard obstacle to optimality and approximated the Steiner tree for a set of rectangular hard obstacles provided that all terminals lie on the boundary of an enclosing rectangle (switchbox instance). Slightly more general, a switchbox instance with a constant number of rectangular hard obstacles can be solved exactly in linear time as was shown by Chiang, Sarrafzadeh and Wong [8].

Rectilinear shortest path problems with hard obstacles and weighted versions have achieved a lot of attention. The strongest result for this kind of problems has been given by Chen, Klenk, and Tu [6] who provide a data structure to answer two-point shortest rectilinear path queries for arbitrary weighted, rectilinear obstacles. Such a data structure can be constructed in $O(n^2 \log^2 n)$ time and space and allows to find a shortest path in $O(\log^2 n + k)$ time, where n is the number of obstacle vertices and k denotes the number of edges on the output path.

Rectilinear shortest path problems with length restrictions have first been considered by Müller-Hannemann and Zimmermann [18] who showed that these problems can easily be solved to optimality (see also Section 2). To the best of our knowledge, the Steiner tree problem with length restrictions on obstacles has previously not been considered.

Our contribution. In Section 2, we show that the length-restricted Steiner tree problem can be approximated with a performance guarantee of 2 in $O(n \log n)$ time where n denotes the number of nodes of the corresponding Hanan grid. This result mainly relies on the fact that we can solve the special case of length-restricted shortest path problems to optimality. Based on that we can use the standard minimum spanning tree approach to obtain a 2-approximation. The running time of $O(n \log n)$ is achieved by using Mehlhorn's implementation [14]. We also show that the guarantee of 2 is tight in this approach for LRSTP.

Then, in Section 3, we show that there are optimal length-restricted Steiner trees bearing a special structure. In particular, we prove that a certain graph which we call the *augmented Hanan grid* always contains an optimal solution. Based on this structural result, we can improve the performance guarantee of approximation algorithms for the special case that all obstacles are of rectangular shape or of constant complexity (i.e. each obstacle can be described by at most a constant number of edges). The restriction to the special cases ensures that the augmented Hanan grid has polynomial size. For such a scenario, we introduce another class of auxiliary graphs G_k, parameterized by some integer $k \geq 3$ with $O(n^{k-2})$ nodes and edges, on which we solve a related Steiner tree problem (now n denotes the size of the augmented Hanan grid.). This yields a $\frac{2k}{2k-1}\alpha$-approximation for any $k \geq 4$, where α denotes the performance guarantee for the ordinary Steiner tree problem in graphs. For $k = 3$, we obtain a $\frac{5}{4}\alpha$-approximation.

Due to space restrictions we had to sketch or to omit several proofs. A complete version is available as a technical report [17].

2 A 2-Approximation of Length-Restricted Steiner Trees

2.1 Shortest Length-Restricted Paths

Instances of the length-restricted Steiner tree problem with only two terminals, i.e. *length-restricted shortest path problems (LRSPP)*, are of special interest for several reasons. In contrast to the general length-restricted Steiner

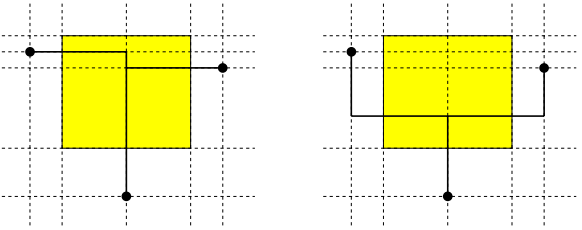


Fig. 2. A small rectilinear Steiner tree instance with three terminals: an optimal Steiner tree without a length restriction lies on the Hanan grid (left), whereas an optimal Steiner tree with such a restriction on the rectangular obstacle does not always lie on the Hanan grid (right).

tree problem, such instances can be solved to optimality in polynomial time. Müller-Hannemann and Zimmermann [18] analyzed the LRSPP and used it as a subroutine for constructing slack-optimized buffer and inverter trees. An efficient solution to the LRSPP is the basis for our 2-approximation of the length-restricted Steiner tree problem. We summarize the most important properties of the LRSPP for later use.

Lemma 1. *[18] Given two terminals s and t, a set of obstacles $\mathcal{O}$ and a length restriction L, there is an optimal length-restricted (s–t)-path using only Hanan grid edges.*

Remark 1. This property does not hold for Steiner trees. A small counter-example with three terminals is shown in Fig. 2.

For a set $\mathcal{O}$ of obstacles described by $n_{\mathcal{O}}$ edges (in total) and a set S of terminals, the size of the associated Hanan grid may have as many as $O((n_{\mathcal{O}} + |S|)^2)$ nodes. For many applications, see again Fig. 1, this is by far too pessimistic. Therefore, in the following we use the actual size of the Hanan grid as a measure of our algorithm's complexity.

Lemma 2. *[18] Given a Hanan grid with n nodes, there is a graph G with $O(n)$ nodes and edges in which all s–t-paths are length-feasible and which contains an optimal length-feasible s–t-path for any pair s, t of terminals. Such a graph can be constructed in $O(n)$ time.*

Lemma 3. *[18] Given a weighted rectilinear subdivision of the plane with an associated Hanan grid of size n where a subset of the regions are obstacles, the weighted shortest path problem with a given length restriction L can be solved by Dijkstra's algorithm in $O(n \log n)$ time.*

2.2 The 2-Approximation

To obtain a 2-approximation for LRSTP, we use well-known 2-approximations for the Steiner tree problem in graphs. Consider an instance $G = (V, E, \ell; S)$

of the Steiner tree problem in graphs, where (V, E) is a connected graph with edge length function ℓ, and S denotes the terminal set. The *distance network* $N_d = (S, E_S, d)$ is a complete graph defined on the set of terminals S: for each pair $s_1, s_2 \in S$ of terminals there is an edge with exactly the length $d(s_1, s_2)$ of a shortest s_1–s_2-path in G. For every vertex $s \in S$ let $N(s)$ be the set of vertices in V that are closer to s (with respect to d) than to any other vertex in S. More precisely, we partition the vertex set V into sets $\{N(s) : s \in S\}$ with $N(s) \cap N(t) = \emptyset$ for $s, t \in S, s \neq t$ with the property

$$v \in N(s) \Rightarrow \ d(v, s) \leq d(v, t) \text{ for all } t \in S,$$

resolving ties arbitrarily. The *modified distance network* $N_d^* = (S, E^*, d^*)$ is a subgraph of N_d defined by

$$E^* := \{(s, t) \,|\, s, t \in S \text{ and there is an edge } (u, v) \in E \text{ with } u \in N(s), v \in N(t)\},$$

and

$$d^*(s, t) := \min\{d(s, u) + \ell(u, v) + d(v, t) \,|\, (u, v) \in E, u \in N(s), v \in N(t)\},$$

for $s, t \in S$. Given an instance $G = (V, E, \ell; S)$ of the Steiner tree problem in graphs with $n = |V|$ and $m = |E|$, Mehlhorn's algorithm [14] computes a Steiner tree with a performance guarantee of 2 in $O(n \log n + m)$ time. Mehlhorn showed that (a) every minimum spanning tree of N_d^* is also a minimum spanning tree of N_d and that (b) N_d^* can be computed in $O(n \log n + m)$ time. The algorithm works as follows:

Algorithm 1. *Mehlhorn's Algorithm [14]*

Input: A Steiner problem instance $G = (V, E, \ell; S)$.
Output: A Steiner tree T for G.

1. Compute the modified distance network N_d^* for $G = (V, E, \ell; S)$.
2. Compute a minimum spanning tree T_d^* in N_d^*.
3. Transform T_d^* into a Steiner tree T for G by replacing every edge of T_d^* by its corresponding shortest path in G.

Theorem 1. *Length-restricted Steiner trees can be approximated with a performance guarantee of 2 in $O(n \log n)$ time.*

Proof. Using the results of the previous section, we can efficiently build up the modified Hanan grid G' from Lemma 2. We apply Mehlhorn's algorithm to G' and obtain a performance guarantee of 2. The claim on the runtime follows immediately as $O(m) = O(n)$. Finally, the obtained tree will be feasible, as no tree in G' violates any length restriction. □

We finish this section by noting that the approximation guarantee for Algorithm 1 is tight. The *Steiner ratio* is the least upper bound on the length of a minimum spanning tree in the distance network divided by the length of a

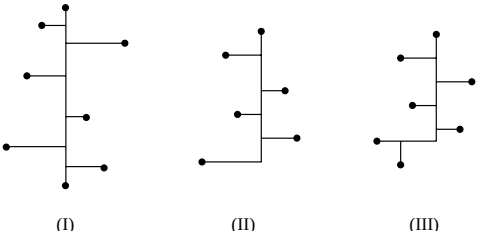


Fig. 3. The three different types of fir trees.

minimum Steiner tree for all instances of the Steiner tree problem. We extend this notion to length restrictions. The *length-restricted Steiner ratio* is the least upper bound on the length of a minimum spanning tree in the distance network containing a length-restricted shortest path between any pair of terminals divided by the length of an LRSMT for all instances of the length-restricted Steiner tree problem. In the case without obstacles the Steiner ratio is $\frac{3}{2}$ as was shown by Hwang [12]. However, in the scenario with obstacles and length restrictions the corresponding Steiner ratio is worse, namely only 2, and therefore not better than for the Steiner tree problem in general graphs.

Lemma 4. *The length-restricted Steiner ratio is 2.*

3 Improved Approximation Ratios

3.1 The Structure of Length-Restricted Steiner Trees

The purpose of this section is to characterize the structure of optimal length-restricted Steiner trees. In particular, we will define a finite graph (a variant of the Hanan grid) which always contains an optimal solution.

First we need some more definitions. For a rectilinear tree, the *degree* of a point is the number of edges it is incident to. All points of degree at least three which are not terminals are called *Steiner points* of the tree. We may assume that a degree two point which is not a terminal is incident to one horizontal and one vertical edge. Such a point is called *corner point*.

Let S be a set of terminals with $|S| \geq 4$ and T be a Steiner minimum tree for S. Then T is called a *fir tree* (see Fig. 3) if and only if every terminal has degree one in T and one of the following two conditions is satisfied (possibly after reflection and/or rotation):

1. All Steiner points lie on a vertical line and every Steiner point is adjacent to exactly one horizontal edge, and these horizontal edges alternatingly extend to the left and to the right. The topmost Steiner point is adjacent to a vertical edge ending in a terminal, the lowest Steiner point is adjacent to a vertical edge either ending in a terminal or at a corner. In the latter case, the horizontal leg extends to the opposite side than the horizontal edge of the lowest Steiner point. (Types (I) and (II) in Fig. 3)

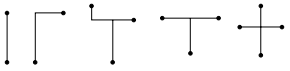
2. All but one Steiner point lie on a vertical line. Every Steiner point but the exceptional one is adjacent to exactly one horizontal edge which alternatingly extend to the left and to the right and ends in a terminal. The exceptional Steiner point is incident to two horizontal edges, one of which ends in a terminal. The other edge is a connection to the lowest Steiner point on the vertical line by a corner from the opposite side than the horizontal edge of the lowest Steiner point. Finally, the topmost and the exceptional Steiner point are both adjacent to a vertical edge that extend upwards and downwards, respectively, and ends in a terminal. (Type (III) in Fig. 3)

The vertical line connecting all or all but one Steiner point is called the *stem* of the fir tree, all horizontal edges will be called *legs*. An edge is called *interior* with respect to some obstacle O if it is contained in O and does not completely lie on the boundary of O.

Lemma 5. *Let S be a terminal set on the boundary of an obstacle O such that in every length-restricted Steiner minimum tree for S*

1. *all terminals are leaves, and*
2. *all tree edges are interior edges with respect to O.*

Then there exists a length-restricted Steiner minimum tree T for S such that it either is a fir tree or has one of the following five shapes (possibly after reflection and/or rotation):



Proof. The proof is a straightforward adaption of almost the same characterization for rectilinear Steiner trees without obstacles; see, for example, the monograph by Prömel and Steger [19], Chapter 10. □

Trees of the fourth and fifth shape will be called *T-shaped* and *cross-shaped*, respectively. The two horizontal edges of a T-shaped tree are its *stem*. Note that the previous lemma asserts that for a set of terminals located on the boundary of an obstacle there is either a LRSMT of the described structure or that the tree decomposes into at least two instances with fewer terminals. Based on these structural insights, we can now define a variant of the Hanan grid, which we call *augmented Hanan grid*.

Definition 1 (augmented Hanan grid). *Given a set S of points in the plane, a set of rectilinear obstacles $\mathcal{O}$ and a length restriction $L \in \mathbb{R}_0^+$, the* augmented Hanan grid *is the graph induced by the following lines:*

1. *for each point $(x, y) \in S$, there is a vertical and a horizontal line going through (x, y),*
2. *each edge of each obstacle is extended to a complete line, and*
3. *for each obstacle $O \in \mathcal{O}$, include a line going through the stem of all those T-shaped trees, and all those fir trees of type (I) or of type (III) which have exactly length L, have only interior edges, and an arbitrary, odd set of points located on the boundary of O as their terminals.*

From its definition it is not clear whether the augmented Hanan grid has polynomial size and can be efficiently constructed. For instances with rectangular obstacles both properties hold: We observe that we need at most four additional lines per obstacle and that we can find all candidate lines easily.

Lemma 6. *If all obstacles in $\mathcal{O}$ are rectangles, then we have to add at most $O(|\mathcal{O}|)$ additional lines to the ordinary Hanan grid to obtain the augmented Hanan grid.*

Similarly, but with more involved counting arguments, one can show that the size of the augmented Hanan grid is still polynomially bounded if each obstacle can be described by at most k edges, where k is some given constant.

Next we note that the augmented Hanan grid has the desired property to contain an optimal solution.

Lemma 7. *The length-restricted Steiner tree problem has an optimal solution which lies completely on the augmented Hanan grid.*

Proof. (Sketch) Choose T among all optimal trees such that (a) T has the structure described in Lemma 5 inside obstacles, and (b) T has the fewest number of (inclusion-maximal) segments $q > 0$ which do not lie on the augmented Hanan grid among all those optimal trees which already fulfill (a). Now one obtains a contradiction by showing how to modify T such that it remains length-minimal and keeps property (a) but contains fewer non-Hanan segments. □

3.2 Improved Approximation for Rectangular Obstacles

In this section, we focus on improved approximation guarantees for instances where all obstacles are rectangles. The basic idea is to construct an instance of the Steiner tree problem in graphs with the property that a Steiner tree in the constructed graph immediately translates back to a feasible length-restricted rectilinear Steiner tree. In addition, the construction is designed to guarantee that the optimal Steiner tree in the graph is not much longer than the optimal LRSMT. This is inspired by approximation algorithms for rectilinear Steiner trees which rely on k-restricted Steiner trees [22], [4]. We say that a Steiner tree is a *k-restricted* Steiner tree if each full component spans at most k terminals. To make this general idea precise, we do the following. Given an instance of the length-restricted Steiner tree problem with rectangular obstacles and an integer $k \geq 2$, we construct the graph G_k in three steps:

1. build up the augmented Hanan grid;
2. delete all nodes and incident edges of the augmented Hanan grid that lie in the strict interior of some obstacle;
3. for each obstacle R, consider each c-element subset of distinct nodes on the boundary of R for $c = 2, \ldots, k$. Compute an optimal (unrestricted) Steiner tree for such a node set. If the length of this tree is less or equal to the given length bound L and if the tree has no edge lying on the boundary of R, then add this tree to the current graph and identify the leave nodes of the tree with the corresponding boundary nodes of R.

The following lemma shows that the construction of G_k can be done efficiently. In particular, in Step 3 we do not have to consider all c-element subsets of nodes on the boundary of a rectangle explicitly. It suffices to enumerate only those subsets of nodes which have optimal Steiner trees according to Lemma 5.

Lemma 8. *If the augmented Hanan grid has n nodes, then (a) G_2 has at most $O(n)$ nodes and edges, and can be constructed in $O(n)$ time, and (b) G_k has at most $O(n^{k-2})$ nodes and edges and can be constructed in $O(n^{k-2})$ time for any $k \geq 3$.*

The following lemma yields the basis for our improved approximation guarantee.

Lemma 9. *Let O be a rectangular obstacle and S a set of terminals on its boundary. Then G_3 contains an optimal Steiner tree that is at most $\frac{5}{4}$ times as long as the optimal length-restricted Steiner tree. For $k \geq 4$, G_k contains an optimal Steiner tree that is at most $\frac{2k}{2k-1}$ times as long as the optimal length-restricted Steiner tree.*

Proof. (Sketch) Let T_{opt} be an LRSMT. We may assume that T_{opt} lies on the augmented Hanan grid (by Lemma 7) and that T_{opt} is a full Steiner tree and all tree edges are interior with respect to O (otherwise one may split T_{opt} into smaller instances and apply the theorem inductively). Zelikovsky [22] and Berman and Ramaiyer [4] defined four 3-restricted Steiner trees that each span the same terminals as T_{opt} with a total length five times $L(T_{opt})$. Thus, the shortest of the four trees has length at most $\frac{5}{4}L(T_{opt})$. For $k \geq 4$, Borchers et al. [5] were able to define a collection of $2k-1$ k-restricted full Steiner trees with total length at most $2k$ times the length of any full tree. □

Combining our previous observations, we obtain the main result of this section.

Theorem 2. *Using an approximation algorithm for the ordinary Steiner tree problem in graphs with an approximation guarantee α, we obtain approximation algorithms for the length-restricted Steiner tree problem subject to rectangular obstacles with performance guarantee $\frac{5}{4}\alpha$ and $\frac{2k}{2k-1}\alpha$ for any $k \geq 4$, respectively.*

Finally, we note again that a similar result holds for a scenario with general obstacles provided each obstacle is bounded by only a constant number of edges.

References

1. C. J. Alpert, G. Gandham, J. Hu, J. L. Neves, S. T. Quay, and S. S. Sapatnekar, *Steiner tree optimization for buffers, blockages and bays*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **20** (2001), 556–562.
2. C. J. Alpert, A. B. Kahng, B. Liu, I. Măndoiu, and A. Zelikovsky, *Minimum-buffered routing of non-critical nets for slew rate and reliability control*, IEEE/ACM International Conference on Computer-Aided Design (ICCAD 2001), 2001, pp. 408–415.

3. S. Arora, *Polynomial time approximation schemes for the Euclidean traveling salesman and other geometric problems*, Journal of the ACM **45** (1998), 753–782.
4. P. Berman and V. Ramaiyer, *Improved approximations for the Steiner tree problem*, Journal of Algorithms **17** (1994), 381–408.
5. A. Borchers, D.-Z. Du, B. Gao, and P. Wan, *The k-Steiner ratio in the rectilinear plane*, Journal of Algorithms **29** (1998), 1–17.
6. D. Z. Chen, K. S. Klenk, and H. T. Tu, *Shortest path queries among weighted obstacles in the rectilinear plane*, SIAM J. on Computing **29** (2000), 1223–1246.
7. W. Chen, M. Pedram, and P. Buch, *Buffered routing tree construction under buffer placement blockages*, Proceedings of 7th ASPDAC and 15th International Conference on VLSI Design, 2002, pp. 381–386.
8. C. Chiang, M. Sarrafzadeh, and C. K. Wong, *An algorithm for exact rectilinear Steiner trees for switchbox with obstacles*, IEEE Transactions on Circuits and Systems — I: Fundamental Theory and Applications **39** (1992), 446–455.
9. J. L. Ganley and J. P. Cohoon, *Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles*, Proceedings of the IEEE International Symposium on Circuits and Systems, 1994, pp. 113–116.
10. M. R. Garey and D. S. Johnson, *The rectilinear Steiner tree problem is NP-complete*, SIAM Journal on Applied Mathematics **32** (1977), 826–834.
11. M. Hanan, *On Steiner's problem with rectilinear distance*, SIAM Journal on Applied Mathematics **14** (1966), 255–265.
12. F. K. Hwang, *On Steiner minimal trees with rectilinear distance*, SIAM Journal on Applied Mathematics **30** (1976), 104–114.
13. F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner tree problem*, Annals of Discrete Mathematics, vol. 53, North-Holland, 1992.
14. K. Mehlhorn, *A faster approximation algorithm for the Steiner problem in graphs*, Information Processing Letters **27** (1988), 125–128.
15. S. Miriyala, J. Hashmi, and N. Sherwani, *Switchbox Steiner tree problem in presence of obstacles*, IEEE/ACM International Conference on Computer-Aided Design (ICCAD 1991), 1991, pp. 536–539.
16. J. S. B. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems*, SIAM Journal on Computing **28** (1999), 1298–1309.
17. M. Müller-Hannemann and S. Peyer, *Approximation of rectilinear Steiner trees with length restrictions on obstacles*, Tech. Report 03924, Research Institute for Discrete Mathematics, Bonn, Germany, 2003.
18. M. Müller-Hannemann and U. Zimmermann, *Slack optimization of timing-critical nets*, Tech. Report 03926, Research Institute for Discrete Mathematics, Bonn, Germany, 2003.
19. H. J. Prömel and A. Steger, *The Steiner tree problem: A tour through graphs, algorithms, and complexity*, Advanced lectures in mathematics, Vieweg, 2002.
20. G. Robins and A. Zelikovsky, *Improved Steiner tree approximation in graphs*, Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, 2000, pp. 770–779.
21. M. Zachariasen, *A catalog of Hanan grid problems*, Networks **38** (2001), 76–83.
22. A. Z. Zelikovsky, *An $\frac{11}{8}$-approximation algorithm for the Steiner problem in networks with rectilinear distance*, Coll. Math. Soc. J. Bolyai **60** (1992), 733–745.

Multi-way Space Partitioning Trees

Christian A. Duncan

Department of Computer Science, University of Miami,
duncan@cs.miami.edu,
http://www.cs.miami.edu/~duncan

Abstract. In this paper, we introduce a new data structure, the multi-way space partitioning (MSP) tree similar in nature to the standard binary space partitioning (BSP) tree. Unlike the super-linear space requirement for BSP trees, we show that for any set of disjoint line segments in the plane there exists a linear-size MSP tree completely partitioning the set. Since our structure is a deviation from the standard BSP tree construction, we also describe an application of our algorithm. We prove that the well-known Painter's algorithm can be adapted quite easily to use our structure to run in $O(n)$ time. More importantly, the constant factor behind our tree size is extremely small, having size less than $4n$.

1 Introduction

Problems in geometry often involve processing sets of objects in the plane or in a higher dimensional space. Generally, these objects are processed by recursively partitioning the space into subspaces. A common approach to partitioning the set involves constructing a binary space partitioning (BSP) tree on the objects. The operation is quite straightforward. We take the initial input and determine in some manner a hyperplane that divides the region. We then partition the space into two subspaces, corresponding to the two half-spaces defined by the hyperplane. The set of objects is also partitioned by the hyperplane, sometimes fragmenting individual objects. The process is then repeated for each subspace and the set of (fragmented) objects until each subspace (cell) contains only one fragment of an object. This requires the assumption that the objects are disjoint; otherwise, we cannot guarantee that every cell subspace contains only one fragment of an object. The final tree represents a decomposition of the space into cells. Each node of the tree stores the hyperplane splitting that subspace and each leaf represents a cell in the decomposition containing at most one fragmented object. For more detailed information see, for example, [9].

In computer graphics, one often wishes to draw multiple objects onto the screen. A common problem with this is ensuring that objects do not obstruct other objects that should appear in front of them. One solves this problem by doing some form of *hidden surface removal*. There are several approaches to solving this problem including the *painter's algorithm* [11]. Like a painter, one attempts to draw objects in a back-to-front order to guarantee that an object is drawn *after* all objects behind it are drawn and thus appears in front of all

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 219–230, 2003.
© Springer-Verlag Berlin Heidelberg 2003

of them. Fuchs et al. [12] popularized the use of BSP trees by applying them to the *painter's algorithm*. Since then BSP trees have been successfully applied to numerous other application areas including shadow generation [4,5], solid modeling [13,15,19], visibility [3,17,18], and ray tracing [14].

The size of the BSP tree, bounded by the number of times each object is partitioned, greatly affects the overall efficiency of these applications. Paterson and Yao [15] showed some of the first efficient bounds on the size of the binary space partition tree. In particular, they showed that a BSP tree of size $O(n \log n)$ can be constructed in the plane and an $O(n^2)$-sized tree can be constructed in $\mathbb{R}^3$, which they prove to be optimal in the worst-case. Recently, Tóth [20] proved that there exist sets of line segments in the plane for which any BSP tree must have at least $\Omega(n \log n / \log \log n)$ size.

By making reasonable and practical assumptions on the object set, improved bounds have been established, see [6,10,16,21]. For example, Paterson and Yao [16] show that a linear-size BSP tree exists when the objects are orthogonal line segments in the plane. Tóth [21] shows a bound of $O(kn)$ when the number of distinct line segment orientations is k. In [6], de Berg et al. show that in the plane a linear size BSP tree exists on sets of fat objects, on sets of line segments where the ratio between the longest and shortest segment is bounded by a constant, and on sets of homothetic objects, that is objects of identical shape but of varying sizes. Our approach is very similar to theirs but with a different aim.

The research in higher-dimensional space is also quite rich but is not the focus of this paper [1,2,7,8,15,16]. We do feel that extending this structure to $\mathbb{R}^3$ is a natural next step for this data structure.

1.1 Our Results

This paper focuses on partitioning a set of n disjoint line segments in the plane. We introduce a new data structure, the multi-way space partitioning (MSP) tree. Unlike standard binary partitioning schemes, MSP trees are produced by partitioning regions into several sub-regions using a spirally shaped cut as described in the next section. We show that for any set of disjoint line segments in the plane there exists a linear-size MSP tree on the set. Unlike previous results on linear-size BSP trees, our segments have no constraints other than being disjoint. More importantly, the constant factors behind our techniques are extremely small. In fact, we show that the constructed tree has size less than $4n$.

Since our structure is a deviation from the standard BSP tree construction, we also describe an application of our algorithm. More specifically, we prove that the painter's algorithm can quite easily be adapted to use our structure to run in $O(n)$ time. We accomplish this by creating a visibility ordering of the cells from a viewpoint v. That is, for any two cells, c_i and c_j, if any line segment from v to a point in c_i intersects c_j then c_j comes before c_i in the ordering. Since many other applications using BSP trees rely on some form of a visibility ordering on the various cell regions, our algorithm should easily adapt to other applications.

2 Multi-way Space Partitioning

For the remainder of this paper, we shall assume we are working exclusively with objects which are disjoint line segments in the plane. Multi-way space partitioning trees store information in a fashion very similar to BSP trees. At each node in the tree, rather than restricting partitioning to a single hyperplane, we also allow a spiral cut to partition the region into multiple disjoint sub-regions. Since every region produced will be convex, when we refer to a region we specifically mean a convex region. As with the BSP tree, every segment that intersects a sub-region is propagated downwards. In some cases, line segments may be split by the (spiral) cut and belong to multiple sub-regions. A leaf in the tree is created when a region contains a single segment. To minimize the size of the MSP tree, we wish to reduce the number of times any segment is divided by cuts. In our construction algorithm, we shall bound the number of times a segment is split to at most three; thus proving a size of less than $4n$. Before we can proceed with the construction and proof of the tree size, we must first define the spiral cut in detail (see Figure 1).

Definition 1. *A* **spiral cut of size** $k \geq 3$ *is a cyclic set of rays* $C = \{c_0, \ldots, c_{k-1}\}$ *such that,*

1. c_i *intersects* c_j *if and only if* $j \equiv i \pm 1 \bmod k$*; only neighboring rays intersect.*
2. c_i *and* c_{i+1} *intersect only at the endpoint of* c_{i+1} *(*mod$k$*, of course).*

Let p_i *be the endpoint of ray* c_i *lying on ray* c_{i-1}*. Let the* **center line segment** l_i *be the segment, lying on* c_i*, formed by the endpoints* p_i *and* p_{i+1}*. Let the* **exterior ray** c'_i *be the ray formed by removing* l_i *from* c_i*. Note that* c'_i *has endpoint* p_{i+1}*.*

Define the **arm region** R_i *to be the V-shaped region lying between the two rays* c_i *and the* c'_{i-1} *defined by* p_i*. Define the* **center region** R_k *to be the convex hull of the set of endpoints* $p_i, i \in \{0, \ldots, k\}$*, which consists of the set of center line segments.*

A point p *lies to the* **right of ray** c_i *if the angle formed from* c_i *to the ray starting at* p_i *passing through* p *is in the range* $(0, \pi)$*. Similarly, a point* p *lies to the* **left of ray** c_i *if the angle is negative. In addition, a ray* c_{i+1} *is to the* **right** *(***left***) of ray* c_i *if any point on* c_{i+1} *is to the right (left) of* c_i*.*

A spiral cut is oriented **clockwise** *(***counterclockwise***) if every consecutive ray is to the right (left) of its previous ray. That is, if* c_{i+1} *is to the right of* c_i *for all* $c_i \in C$*.*

Because the rays are cyclically ordered and only intersect neighboring rays, every *turn* must be in the same direction. Therefore, there are only two types of spiral cuts, clockwise and counterclockwise. As described above, a spiral cut of size k divides the region into $k + 1$ convex sub-regions. There are k sub-regions, $R_0, \ldots, R_{k-1}$, each associated with an arm region of the spiral, and one sub-region R_k in the center of the spiral (see Figure 1).

There are several properties that we can establish that will prove useful in our evaluation of the MSP tree.

Property 1. If the spiral cut, C, is clockwise (counterclockwise), then any point p in the center region R_k lies to the right (left) of every ray $c_i \in C$. For a clockwise spiral cut, let p be any point in an arm region, say R_0. Point p lies to the left of c_0 and the right of c_{k-1}. In addition, there exists a ray c_m such that p lies to the left of all rays c_i for $0 \leq i \leq m$ and to the right of all rays c_i for $m < i \leq k-1$. That is, traversing the cycle from c_0 around to c_{k-1}, divides the cycle into two continuous sequences those with p on the left and those with p on the right. For counterclockwise spiral cuts, the reverse directions apply.

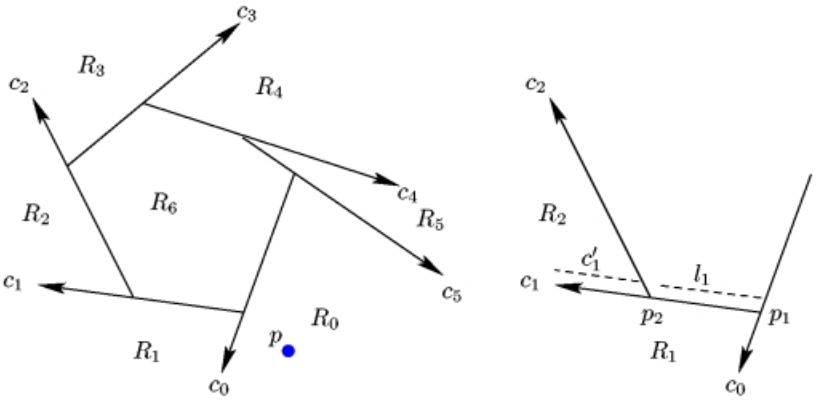


Fig. 1. An example of a clockwise spiral cut $C = \{c_0, c_1, c_2, c_3, c_4, c_5\}$ forming. 6 arm regions and the center region. The point $p \in R_0$ lies to the left of c_0 and c_1 but to the right of all other rays.

2.1 Construction

Assume we are given an initial set of segments S. The general construction algorithm is quite simple, start with an initial bounding region of the segment endpoints. For every region R, if there is only one segment of S in the region, nothing needs to be done. Otherwise, find an appropriate halfplane cut or spiral cut. Then, divide the region into sub-regions $R_0, R_1, \ldots R_k$ which become child regions of R. The line segments associated with the cut are stored in the current node and all remaining line segments in R are then propagated into the appropriate (possibly multiple) sub-regions. Finally, repeat on each of the sub-regions.

What remains to be shown is how to determine an appropriate cut. We do this by classifying our segments into two categories: rooted and unrooted segments (see Figure 2). For any convex region R, a *rooted segment* of R is a segment which intersects both the interior and boundary of R. Similarly, an *unrooted segment* of R is a segment which intersects the interior of R but not its boundary. By this definition unrooted segments of R must lie completely inside the region.

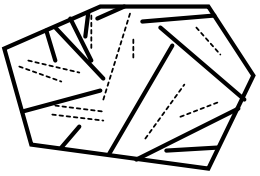


Fig. 2. An example of rooted (solid) and unrooted (dashed) segments in a convex region.

For any region R, let $\mathcal{S}(R)$ represent the set of all segments of S lying (partially) inside R. Let $\mathcal{R}(R) \subseteq \mathcal{S}(R)$ represent the set of rooted segments of S in R and let $\mathcal{U}(R) = \mathcal{S}(R) - \mathcal{R}(R)$ represent the set of unrooted segments of S in R. For any two regions R_1 and R_2 if there exists a segment $s \in S$ such that $s \in \mathcal{U}(R_1)$ and $s \in \mathcal{U}(R_2)$ then $R_1 \subseteq R_2$ or $R_2 \subseteq R_1$. This means that R_1 and R_2 must lie on the same path from the root node to a leaf in the MSP tree. In addition, if $s \in \mathcal{U}(R_1)$ and $s \in \mathcal{R}(R_2)$ then $R_2 \subset R_1$. That is, R_2 must be a descendant of R_1 in the tree.

Let us now see how we can exploit these rooted and unrooted segments. In [6], de Berg et al. show that if a region contains *only* rooted segments then a BSP tree of linear size can be constructed from it. Of course, the challenge is in guaranteeing that this situation occurs. As a result, they first made numerous cuts to partition the initial region R into sub-regions such that every segment was cut at least once but also not too many times. Their result relied on the assumption that the ratio between the longest segment and the shortest segment was some constant value.

We take a somewhat different approach to this problem. We do not mind having unrooted segments in our region and actually ignore them until they are first intersected by a dividing cut, after which they become rooted segments and remain so until they are selected as part of a cut. In our construction, we guarantee that rooted segments are *never* divided by a partitioning cut. That is, only unrooted segments will be cut. This situation can only occur once per segment in S. Let us now see how to find an appropriate partitioning cut.

2.2 Finding a Spiral or Hyperplane Cut

Let us assume we are given some region R. For this subsection, we will completely ignore unrooted segments. Therefore, when we refer to a segment s we always mean a rooted segment $s \in \mathcal{R}(R)$.

Although not necessary, observe that if a rooted segment intersects the boundary of R in two locations then we can choose this segment as a valid partitioning cut. Therefore, for simplicity, we assume that no segment in R intersects the boundary of R more than once.

As in [6], we try to find either a single cut that partitions the region or else a cycle of segments that do. We do this by creating an ordered sequence on the

segments starting with an initial segment $s_0 \in \mathcal{R}(R)$. Let us extend s_0 into R until it either hits the boundary of R or another segment in $\mathcal{R}(R)$. Define this extension to be $\text{ext}(s_0)$. For clarity, note that the extension of s_0 includes s_0 itself. If $\text{ext}(s_0)$ does not intersect any other segment in $\mathcal{R}(R)$, then we take it as a partitioning cut. Otherwise, the extension hits another segment s_1. In this case, we take s_1 to be the next segment in our sequence. The rest of the sequence is completed in almost the same fashion.

Let us assume that the sequence found so far is $\{s_0, s_1, s_2, \ldots, s_i\}$. We then extend s_i until s_i hits either the boundary of R, a previous extension $\text{ext}(s_j)$ for $j < i$, or a new segment s_{i+1}. If it hits the boundary of R, then we can take s_i as a partitioning cut. If it intersects $\text{ext}(s_j)$, then we have completed our cycle, which is defined by the sequence $\mathcal{C}(R) = \{\text{ext}(s_j), \text{ext}(s_{j+1}), \ldots, \text{ext}(s_i)\}$. Otherwise, we repeat with the next segment in our sequence, s_{i+1}.

Since there are a bounded number of segments in $\mathcal{R}(R)$, the sequence must find either a single partition cut s or a cycle $\mathcal{C}$. If it finds a single partition cut s then we can simply divide the region R into two sub-regions by the line formed by s as usual. Otherwise, we use the cycle $\mathcal{C}$ to define a spiral cut.

Let $\text{ext}(s_i)$ and $\text{ext}(s_{i+1})$ be two successive extension segments on the cycle. By the construction of the cycle, $\text{ext}(s_i)$ has an endpoint p_i on $\text{ext}(s_{i+1})$. We, therefore, define the ray for c_i to be the ray starting at p_i and extending outward along $\text{ext}(s_i)$ (see Figure 3). To be consistent with the spiral cut notation, we must reverse the ordering of the cycle. That is, we want p_i to lie on c_{i-1} and *not* c_{i+1}. Also, except possibly for the initial extended segment, every extension $\text{ext}(s_i)$ is a subset of l_i, the center line segments forming the convex center region, R_k.

Since the initial region is convex and by the general construction of the cycle, this new cycle of rays defines a spiral cut. We can now proceed to use this spiral cut to partition our region into multiple regions and then repeat the process until the space is completely decomposed.

2.3 MSP Size

To complete our description of the tree, we only need to analyze its worst-case size. The size of the MSP tree produced by our construction algorithm depends only on the following two conditions:

1. At every stage no rooted segment is partitioned.
2. At every stage no unrooted segment is partitioned more than a constant, c, number of times.

If both of these conditions hold, the size of the tree is at most $(c+1)n$ since an unrooted segment once split is divided into rooted segments only and each ray of the spiral cut corresponds to one rooted segment.

Lemma 1. *Given a convex region R with a set of rooted segments $\mathcal{R}(R)$ and unrooted segments $\mathcal{U}(R)$, a partitioning cut or spiral cut can be found which divides R into sub-regions such that no segment in $\mathcal{R}(R)$ is intersected by the*

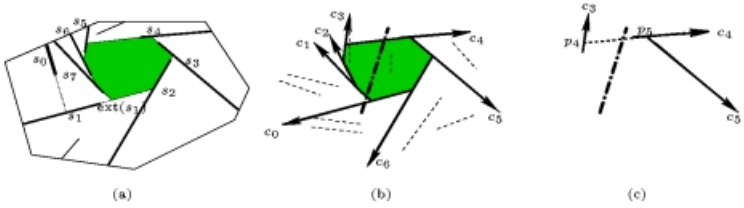


Fig. 3. (a) An example of finding a cycle of cuts. Here, s_0 is the initial cut and the cycle completes when s_7 intersects ext(s_1). Thus, the sequence is $\{s_1, s_2, \ldots, s_7\}$. (b) The resulting spiral cut, $\{c_0, \ldots, c_6\}$. This cycle is formed by the sequence of segments reversed. Observe how the unrooted segments intersect the spiral cut and in particular how the bold dashed unrooted segment is intersected the maximum three times. (c) Here, c_4 is the ray extending from p_4, c'_4 is the ray extending from p_5, l_4 is the segment between p_4 and p_5. The ext(s_4) is the dashed portion starting at p_4 and ending before p_5. Observe how the unrooted segment can intersect only the ext(s_4) if it intersects c_4.

cut except those that are part of the cut and no unrooted segment in $\mathcal{U}(R)$ is intersected by the cut more than three times.

Proof. We construct the sequence $\{s_0, s_1, \ldots s_k\}$ as described in the previous subsection. If we choose a segment s_i, as a partitioning cut, then by our construction it does not intersect any other rooted segment. Also, it can intersect an unrooted segment at most once.

Let us then assume that we have identified a spiral cut $\{c_0, c_1, c_2, \ldots, c_{k-1}\}$. Given the construction of the spiral cut itself, it is clear that no rooted segment that is not part of the cycle is intersected. So, all that is left to prove is that unrooted segments are intersected at most three times.

As described earlier, the rays of the spiral cut can be broken into two pieces, the portion of the ray forming the convex central region R_k and the arm regions R_i for $0 \leq i < k$. In particular, let us look at any three successive rays, say c_0, c_1, and c_2. Recall that p_1 is the endpoint of c_1 and p_2 is the endpoint of c_2. In addition, p_1 lies on c_0 and p_2 lies on c_1. Recall that the *center* line segment l_1 is defined to be the segment from p_1 to p_2 and that the exterior ray c'_1 is the ray extending from p_2 along c_1.

Now, let us look at an unrooted segment $s \in \mathcal{U}(R)$. We first look at the line segments l_i forming R_k. Because the region is convex, s can intersect at most two segments of the convex central region. Let us now look at the exterior ray portions. Recall that each extension, ext(s_i), except for $i = 0$, is a subset of the center line segment l_i. Since the portion of c_i lying inside R is exactly the union of s_i and ext(s_i) and, except for $i = 0$, ext(s_i) is a subset of the center line segment l_i, the portion of c'_i lying inside R is a subset of the segment s_i. Since all segments are disjoint and s is unrooted, s cannot intersect c'_i except for c'_0. As a result, the spiral cut intersects s at most three times (see Figure 3b). □

This lemma along with the construction of the multi-way space partitioning tree leads to the following theorem:

Theorem 1. *Given a set of n disjoint segments $S \subset \mathbb{R}^2$, a multi-way space partitioning tree T can be constructed on S such that $|T| < 4n$ in $O(n^3)$ time.*

Proof. The proof of correctness and size is straightforward from the construction and from Lemma 1. As for the running time, a straightforward analysis of the construction algorithm shows $O(n^2)$ time for finding a single spiral cut and hence the $O(n^3)$ overall time. □

This is most likely not the best one can do for an MSP tree construction. It seems possible to get the time down to near quadratic time. Although it may be difficult to develop an algorithm to compete with the $O(n \log n)$ construction time for a regular BSP tree, we should point out that the BSP tree created is not necessarily optimal and is typically created via a randomized construction.

3 Painter's Algorithm

To illustrate the vitality of the MSP tree, we now show how to apply this structure to the painter's algorithm. In a BSP tree, the traditional approach to solving the painter's algorithm is to traverse the tree in an ordered depth-first traversal. Assume we are given an initial view point, v. At any region R in the tree, we look at the partitioning cut. Ignoring the degenerate case where v lies on the cut itself, v must lie on one side or the other of the cutting line. Let R_1 be the sub-region of R lying on the same side of the line as v and let R_2 be the other sub-region. We then recursively process R_2 first, process the portion of the line segment in the region corresponding to the cutting line, and then process R_1. In this way, we guarantee that at any time a line segment s is drawn it will always be drawn *before* any line segment between s and v.

To see the corresponding approach to traversing the MSP tree, let us generalize the depth-first search. Recall at a region R, we visit all the sub-regions on the opposing side of the cutting line to v and then all sub-regions on the same side as v. Let R_1 be a sub-region of R visited in the search. The ultimate goal is to guarantee that, for any point $p \in R_1$, the line segment $\overline{pv}$ intersects only sub-regions that have not been visited already. Now, let R have multiple sub-regions $R_0, R_1, \ldots, R_k$ rather than just two. We still wish to construct an ordering on the sub-regions such that the following property holds:

- Let p_i be any point in R_i. The line segment $\overline{p_i v}$ does not intersect any region R_j with $j < i$ in our ordering.

Notice if this property holds, then we can traverse each sub-region recursively as before and guarantee that no line segment s is drawn after a line segment appearing between v and s.

3.1 Spiral Ordering

Unfortunately, given a spiral cut, we cannot actually guarantee that such an ordering of the sub-regions always exists from any viewpoint v. However, when

processing a scene one also considers a viewing direction and a viewing plane onto which to project the scene. In this sense, we assume that one has a view line v_p that passes through v and defines a particular viewing half-plane V. Therefore, all line segments **behind** the viewer can in fact be ignored.

Adding a view line does in fact enable us to create an ordering. This particular point will only arise in one specific case. In addition, for applications such as shadow generation requiring full processing of the scene, observe that we may perform the process *twice* using the same view line with opposing normals.

To compute the order using a spiral, it is somewhat easier to describe how to compute the *reverse* ordering. After creating this ordering, we can simply reverse the ordering to get the desired result. Let us define the following ordering on a spiral cut:

Definition 2. *Given a view point v, a viewing half-plane V, and a spiral cut $\{c_0, c_1, \ldots, c_{k-1}\}$. Let $R_0, R_1, \ldots, R_k$ be the sub-regions produced by the cut. A* **visible ordering** *$o(x)$ represents a permutation of the sub-regions such that,*

- *for any point $p_i \in R_i \cap V$, if the line segment $\overline{p_i v}$ intersects a region R_j, then $o(j) \leq o(i)$.*

Moreover, given any ordering, we say a point $p_i \in R_i$ is **visible** *from v if the above condition holds for that point. We also say that v* **sees** *p_i.*

In other words, we visit regions in such a way that v can see every point in a region R_i by passing *only* through previously visited regions. Notice this is the reverse ordering of the painter's algorithm where we want the opposite condition that it only passes through regions that it has not yet visited. A simple flip of the ordering once generated produces the required ordering for the painter's algorithm.

Lemma 2. *Given a view point v, a viewing half-plane V, and a spiral cut $\{c_0, c_1, \ldots, c_{k-1}\}$. Let $R_0, R_1, \ldots, R_k$ be the sub-regions produced by the cut. There exists a visible ordering $o(x)$ on the spiral cut.*

Proof. Let R_i be the region containing the view point v itself. Let $o(i) \to 0$ be the first region in our ordering. Notice that since every region is convex and $v \in R_i$, any point $p \in R_i$ is visible from v.

Without loss of generality, assume that the spiral cut is a clockwise spiral. The argument is symmetrical for counterclockwise spirals. Let us now look at two different subcases. Recall that the spiral cut consists of two parts the center region and the arm regions.

Case 1: Let us first assume that R_i is an arm region. Without loss of generality assume that $R_i = R_0$. We will create our ordering in three stages. In the first stage, we add regions $R_1, R_2, \ldots, R_m$ for some m to be described shortly. We then add the center region R_k and finally we add the remaining regions $R_{k-1}, R_{k-2}, \ldots R_{m+1}$.

Let us begin with the first stage of ordering. Assume that we have partially created the ordering $o(0), o(1), \ldots, o(i)$ and let $R_i = R_{o(i)}$ be the last region

added. Recall that R_i is defined by the rays c_i and c'_{i-1}. Let us now look at the neighboring region R_{i+1} defined by the rays c_{i+1} and $c'_i \subset c_i$.

If v lies to the **left** of c_{i+1}, add R_{i+1} to the ordering. That is, let $o(i+1) \rightarrow i+1$. We claim that all points in R_{i+1} are visible from v. Let p be any point in R_{i+1}. Notice that p also lies to the left of c_{i+1}. Therefore the line segment $\overline{pv}$ cannot intersect ray c_{i+1} and must therefore intersect ray c_i. Let q be the point on this intersection or just slightly passed it. Notice that q lies inside R_i. By induction, q must be visible from v. Therefore, the line segment $\overline{qv}$ intersects only regions with ordering less than or equal to i. In addition, the line segment $\overline{pq}$ intersects only R_{i+1}. Therefore, the line segment $\overline{pv}$ intersects only regions with ordering less than or equal to $i+1$ and p is visible from v.

If v lies to the **right** of c_{i+1}, we are done with the first stage of our ordering, letting $m = i$.[1] We now add the center region R_k into our ordering. That is, let $o(i+1) \rightarrow k$. Again, we claim that all points in R_k are visible from v. Recall from Property 1 that v lies to the right of all rays from c_{m+1} to c_{k-1}, given that v lies in R_0. Let p be any point in R_k. Again, from Property 1 we know that p lies to the right of every ray in the cut. Let R_j be any region intersected by the line segment $\overline{pv}$. If R_j is R_k or R_0 we are done since they are already in the ordering. Otherwise, we know that since R_j is convex, $\overline{pv}$ must intersect the ray c_j. Since p is to the right of c_j as with all rays, this implies that v must lie to the left of c_j. But, that means that c_j cannot be part of c_{m+1} to c_{k-1}. R_j must be one of the regions already visited and so $j \in \{o(0), \ldots, o(m)\}$. Hence, p is visible from v.

We now enter the final stage of our ordering. We shall now add into the ordering the regions from R_{k-1} backwards to R_{m+1}. Let us assume that we have done so up to R_j. We claim that all points in R_j are visible from v. Let p be any point in R_j. Again look at the line segment $\overline{pv}$ and the first (starting from p) intersection point q with another region. This point must certainly lie on one of the two rays c_{j-1} or c_j. Since p is to the right of c_{j-1} (Property 1), if it intersects c_{j-1}, v must lie to the left of c_{j-1}. This means that R_{j-1} is already in the ordering and, as with previous arguments, q is visible from v and hence so is p. If it intersects c_j instead, then q lies either in R_k or R_{j+1}. But again in either case, since we added the center region already and are counting backwards now, both R_k and R_{j+1} are in the ordering. This implies that q is visible from v and so then is p.

Thus, we have constructed a visible ordering of the regions assuming p lies in one of the arm regions. We now need to prove the other case.

Case 2: Let v lie in the center region R_k. In this case, unfortunately, there is no region that is completely visible from v except for the center region. This is where the viewing half-plane V comes into play. Our arguments are almost identical to the above case except we now only look at points in V. For simplicity, let us assume that V is horizontal with an upward pointing normal.

[1] For the sake of simplicity, we are ignoring degenerate cases such as when v lies directly on the line defined by c_{i+1}.

Look at the ray from v going horizontal to the right and let R_i be the first new region hit by this ray. That is, R_i is the region directly to the right of v. Without loss of generality, we can let this region be R_{k-1}. We then add all regions into the ordering starting with the center region and counting backwards from the rightmost region, $R_k, R_{k-1}, R_{k-2}, \ldots, R_m$, where R_m is the last region visible, at least partially intersecting V. We first claim that all point in $R_{k-1} \cap V$ are visible from v. Let p be any point in $R_{k-1} \cap V$. Since p lies to the left of and v lies to the right of c_{k-1}, the line segment $\overline{pv}$ must intersect c_{k-1}. Let q be this intersection point. Since R_{k-1} is the first region to the right of v and p lies *above* the line defined by V, we know that q must actually lie on l_{k-1} or else R_0 would be seen first horizontally by v. This implies that q is seen from v and hence so is p. Let us now assume that we have constructed the ordering up to some region R_i. We claim that all points in $R_i \cap V$ are visible from v. Let p be any point in $R_i \cap V$. Once again from the sidedness of p and v, we know that the line segment $\overline{pv}$ must intersect c_i. Let q be this intersection point. Now, either q lies in $R_k \cap V$ or in $R_{i+1} \cap V$. In either case, both regions have been added to our ordering and so q is visible from v. Therefore, p must also be visible from v. By induction, our ordering is a proper visible ordering and we are done. □

The technique for calculating the ordering is quite straightforward. The algorithm must make one full scan to determine the sub-region containing v. Afterwards, it either marches along one direction, adds in the center region, and marches in the other direction or it adds in the center region first, finds the first region intersected by the viewing half-plane V and marches backwards along the list. In either case, the algorithm can be implemented in at most two scan passes. These observations and the fact that the MSP tree has linear size, leads to the following theorem:

Theorem 2. *Given an MSP tree constructed on a set of n line segments S in $\mathbb{R}^2$, one can perform the painter's algorithm on S in $O(n)$ time.*

4 Conclusion and Open Problems

In this paper, we have described a simple space-partitioning tree that can be constructed in linear size on any set of disjoint line segments in the plane. We hope to improve construction time and reduce the maximum degree for any single node from $O(n)$ to constant degree. More importantly, we would like to focus on a similar technique in $\mathbb{R}^3$ space where BSP trees are known to have very poor sizes. The question arises whether deviating from the standard notion of binary space partitions provides better performance, even in the average case. We feel that answering such a question would demonstrate the greatest promise for this new tree structure. The spiral cut as mentioned for the plane will not immediately translate into higher-dimensions, but we are hopeful that some other deviation from the standard cutting method may produce surprising results.

References

1. P. Agarwal, T. Murali, and J. Vitter. Practical techniques for constructing binary space partitions for orthogonal rectangles. In *Proc. of the 13th Symposium on Computational Geometry*, pages 382–384, New York, June 4–6 1997. ACM Press.
2. P. K. Agarwal, E. F. Grove, T. M. Murali, and J. S. Vitter. Binary space partitions for fat rectangles. *SIAM Journal on Computing*, 29(5):1422–1448, Oct. 2000.
3. J. M. Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations*. PhD thesis, Dept. of CS, U. of North Carolina, July 1990. TR90-027.
4. N. Chin and S. Feiner. Near real-time shadow generation using BSP trees. *Computer Graphics (SIGGRAPH '90 Proceedings)*, 24(4):99–106, Aug. 1990.
5. N. Chin and S. Feiner. Fast object-precision shadow generation for areal light sources using BSP trees. *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, 25(4):21–30, Mar. 1992.
6. de Berg, de Groot, and Overmars. New results on binary space partitions in the plane. *CGTA: Computational Geometry: Theory and Applications*, 8, 1997.
7. M. de Berg. Linear size binary space partitions for fat objects. In *Algorithms—ESA '95, Third Annual European Symposium*, volume 979 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 25–27 Sept. 1995.
8. M. de Berg and M. de Groot. Binary space partitions for sets of cubes. In *Abstracts 10th European Workshop Comput. Geom.*, pages 84–88, 1994.
9. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry Algorithms and Applications*. Springer-Verlag, Berlin Heidelberg, 1997.
10. A. Dumitrescu, J. S. G. Mitchell, and M. Sharir. Binary space partitions for axis-parallel segments, rectangles, and hyperrectangles. In *Proceedings of the 17th annual symposium on Computational geometry*, pages 141–150. ACM Press, 2001.
11. J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, 1990.
12. H. Fuchs, Z. M. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. *Comput. Graph.*, 14(3):124–133, 1980. Proc. SIGGRAPH '80.
13. B. Naylor, J. A. Amanatides, and W. Thibault. Merging BSP trees yields polyhedral set operations. *Comp. Graph (SIGGRAPH '90).*, 24(4):115–124, Aug. 1990.
14. B. Naylor and W. Thibault. Application of BSP trees to ray-tracing and CGS evaluation. Technical Report GIT-ICS 86/03, Georgia Institute of Tech., School of Information and Computer Science, Feb. 1986.
15. M. S. Paterson and F. F. Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete Comput. Geom.*, 5:485–503, 1990.
16. M. S. Paterson and F. F. Yao. Optimal binary space partitions for orthogonal objects. *J. Algorithms*, 13:99–113, 1992.
17. S. J. Teller. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, Dept. of Computer Science, University of California, Berkeley, 1992.
18. S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. *Comput. Graph.*, 25(4):61–69, July 1991. Proc. SIGGRAPH '91.
19. W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. *Comput. Graph.*, 21(4):153–162, 1987. Proc. SIGGRAPH '87.
20. C. D. Tóth. A note on binary plane partitions. In *Proceedings of the seventeenth annual symposium on Computational geometry*, pages 151–156. ACM Press, 2001.
21. C. D. Tóth. Binary space partitions for line segments with a limited number of directions. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 465–471. ACM Press, 2002.

Cropping-Resilient Segmented Multiple Watermarking*
(Extended Abstract)

Keith Frikken and Mikhail Atallah

Purdue University

Abstract. Watermarking is a frequently used tool for digital rights management. An example of this is using watermarks to place ownership information into an object. There are many instances where placing multiple watermarks into the same object is desired. One mechanism that has been proposed for doing this is segmenting the data into a grid and placing watermarks into different regions of the grid. This is particularly suited for images and geographic information systems (GIS) databases as they already consist of a fine granularity grid (of pixels, geographic regions, etc.); a grid cell for watermarking is an aggregation of the original fine granularity cells. An attacker may be interested in only a subset of the watermarked data, and it is crucial that the watermarks survive in the subset selected by the attacker. In the kind of data mentioned above (images, GIS, etc.) such an attack typically consists of cropping, e.g. selecting a geographic region between two latitudes and longitudes (in the GIS case) or a rectangular region of pixels (in an image). The contribution of this paper is a set of schemes and their analysis for multiple watermark placement that maximizes resilience to the above mentioned cropping attack. This involves the definition of various performance metrics and their use in evaluating and comparing various placement schemes.

1 Introduction

Watermarking is a frequently used tool in digital rights management. For example, watermarking can be used for copyright protection [14]; this is done by placing an ownership watermark into the object. Another example is a digital VCR, where watermarks are placed into the object to convey what commands the user is allowed to perform on the object (read only, read and copy, etc.) [14]. Placing multiple watermarks into data has many applications; several examples appear in [13]. One digital rights management application of multiple watermarking is collaborative watermarking. In collaborative watermarking several

* Portions of this work were supported by Grants EIA-9903545 and ISS-0219560 from the National Science Foundation, Contract N00014-02-1-0364 from the Office of Naval Research, by sponsors of the Center for Education and Research in Information Assurance and Security, by Purdue Discovery Park's e-enterprise Center, and by the GAANN fellowship.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 231–242, 2003.
© Springer-Verlag Berlin Heidelberg 2003

organizations may have partial ownership of an object, and each organization wants to place ownership watermarks into the object. A single organization may choose to place multiple watermarks into the same object for various reasons. For example, defense in depth can be achieved by using different watermarking schemes that have different strengths and weaknesses.

Several techniques have been proposed for inserting multiple watermarks into an object including rewatermarking, segmented watermarking, interleaved watermarking, and composite watermarking [18]. Segmented watermarking divides the object into regions and places each watermark into a set of these regions. A scheme for determining regions is given in [4], but in this paper we assume the regions are equal sized rectangles as in [18]. However, we assume that each of these regions contains enough information to hide a single watermark. An attack against the segmented watermarking scheme would be to take a rectangular subset (a cropping) of the data to remove some of the watermarks. A watermark will survive a cropping if that watermark is contained in a region which is fully enclosed within the cropping. The purpose of the work in this paper is to maximize the number of recoverable watermarks for random croppings. For simplicity, we assume that all croppings are equally likely. The rest of this paper does not depend on the exact nature of the object being watermarked (image, GIS, NASA spatial data, etc.), as long as the object can be naturally partitioned into a grid, and is useful if an adversary may find a rectangular subset of the grid of value for stealing.

In the collaborative watermarking application mentioned above, the cropping attack can be carried out by an outsider or by any of the watermarking organizations. We introduce two performance metrics that are important to this application: (i) Maximum Non-Complete Area(MNCA) and (ii) Minimum Non-Full Area(MNFA). The MNCA is the maximum number of tiles that can be in a cropping which does not contain all watermarks; the MNCA provides a bound on the largest area that can be stolen such that one of the watermarks cannot be recovered. Obviously, minimizing the MNCA is a goal for a placement scheme. As a motivation for MNFA, observe that a cropping that is lacking a watermark yet contains more than one copy of another watermark is "bad". Ideally, no such croppings would exist, but short of this it is desirable to maximize the area of such croppings. The MNFA is the minimum number of tiles that can be in a cropping that does not contain all watermarks, but contains at least one duplicate watermark. The motivation for MNFA is that it is the minimum cropping that will allow an attacker to get away with something (i.e. have less watermarks than there are tiles); for any cropping with less tiles than the MNFA the number of watermarks will be the number of tiles, which is the best any placement can do. A placement scheme should attempt to maximize the MNFA. If a single organization uses multiple ownership watermarks then it is possible that only a subset of the watermarks need to be recovered for proof of ownership. If only t watermarks need to be recovered, the placement scheme should minimize the maximum area that does not contain at least t watermarks.

If we treat the watermarks as colors and the data as a grid, watermark placement can be viewed as grid coloring; in this paper we use the term color when discussing placement schemes and we use the terms tile and region interchangeably. This watermark placement problem is similar to a grid coloring problem used for declustering data in a database among multiple disks to parallelize I/O (see Section 2). For simplicity we restrict this work to data tiled along two dimensions. Furthermore, we only consider croppings of the data on tile boundaries, since every cropping contains a subcropping on tile boundaries. We define the area of a cropping to be the number of complete tiles contained in the cropping.

The results of our work include a formal definition of this problem and a formal definition of the above mentioned comparison heuristics (MNCA and MNFA). A scheme is given that colors any grid with M colors so that the MNCA is $O(M)$, and a scheme is given where the MNFA is $\Omega(M)$. Also in the case where only half of the watermarks need to be recovered, we provide a scheme that colors any grid with M colors in such a way that any area containing M tiles contains half of the watermarks when M is a power of 2. Furthermore, a set of experiments were performed to evaluate the performance of several schemes using these two comparison metrics.

The layout of the rest of this paper is as follows. In Section 2, we discuss the distributed database retrieval problem, which is similar to this watermarking placement problem, but has some key differences. In Section 3, we present a formal definition of this problem along with several results about MNCA, MNFA, and other constraints. In Section 4, we briefly discuss the results of our experimental analysis, and we summarize our contributions in Section 5. Due to space limitations, we often give a proof sketch of a claim; the details of these proofs will be given in the full paper.

2 Related Work

A problem that is similar to the watermark placement problem outlined in the previous section is the distributed database declustering problem. Given an n dimensional database divide each dimension uniformly to get tiles. By placing the tiles on different disks the retrieval of records during query processing can be parallelized, which reduces the I/O time to the time that it takes to retrieve the maximum number of tiles stored on the same disk. The problem of placing the records so that the response times for range queries is minimized has been well studied.

Given k disks and m tiles in a range query, an optimal tile placement would require an I/O time of $\lceil \frac{m}{k} \rceil$. It was shown in [1] that this bound is unachievable for all range queries in a grid except in a few limited circumstances. Since there are many cases where no scheme can achieve this optimal bound, several schemes have been developed to achieve performance that is close to optimal. These schemes include Disk Modulo DM [6], CMD [12], Fieldwise eXclusive or [11], and the HCAM approach [7]. These are just a subset of the techniques that have been proposed for declustering.

Suppose we are given k colors. The DM approach [6] assigns tile (x, y) to $(x + y) \bmod k$. The FX approach [11] assigns tile (x, y) to $(x \oplus y) \bmod k$. Cyclic allocation schemes [15] choose a skip value s such that $gcd(k, s) = 1$ and assigns tile (x, y) to $(x + sy) \bmod k$. The choice of the skip value is what defines the scheme. In RPHM (Relatively Prime Half Modulo), the shift value is defined to be the number nearest to $\frac{M}{2}$ that is relatively prime to M. The EXH (Exhaustive) scheme takes all values of s where $gcd(s, M) = 1$ and finds the one that optimizes a certain criterion. Another class of schemes are the permutation schemes [3], in these schemes a permutation ϕ of the numbers in $\{0, ..., k-1\}$ is chosen and then tile (x, y) is assigned color $(x - \phi^{-1}((y) \bmod k))$. Examples of permutation schemes are DM, the cyclic schemes, and GRS. In the GRS scheme [3] the permutation is computed as follows:

1. $\forall i \in \{0, ..., k-1\}$ compute the fractional part of $\frac{2i}{1+\sqrt{5}}$, and call it k_i.
2. Sort the values k_i and use this to define the permutation.

In [2], a coloring scheme was presented that was later found in [16] to be equivalent to $(x \oplus y^R) \bmod k$, where y^R is the $(\lceil \log k \rceil)$-bit reversal of y; in this paper we will call this scheme RFX (Reverse Fieldwise eXclusive-or). Recently, two new directions have been explored: i) the relation between this area and discrepancy theory [5,16], and ii) the use of redundancy [8,17,19], i.e. placing each record on multiple disks.

The database declustering problem appears similar to that of the watermarking representation problem defined in the previous section, but there are key differences:

1. In the database declustering problem the multiplicity of a color is of central importance, whereas in the watermarking placement problem multiplicity of a color in a cropping is irrelevant (as long as it is nonzero).
2. Given a coloring for k colors it is possible to construct a coloring for $k-1$ colors that will have the same MNCA by ignoring the kth color. In the database problem you cannot ignore a color since that tile may need to be retrieved.
3. Given a coloring for k colors it is possible to construct a coloring for $k+1$ colors that will have the same MNFA by ignoring the $(k+1)$st color. In the database problem this is like not using certain disks, which may improve the additive error from an optimal solution, but will not improve overall query performance (there may be a few cases where it does, but these are very limited).

3 Theoretical Results

3.1 Definitions and Basic Properties

Given M watermarks labeled $\{0, ..., M-1\}$ to place into a two dimensional data, which is tiled into a grid with dimension sizes $d_1 \in \aleph$ and $d_2 \in \aleph$, a *coloring* maps a grid location to a watermark and is defined by a function $C : \aleph \times \aleph \rightarrow$

$\{0, ..., M-1\}$. A coloring C is said to be *periodic* with period p if and only if $C(x,y) = C(x+p,y)$ and $C(x,y) = C(x,y+p)$ for all grid locations (x,y). Furthermore, if each watermark is represented every p tiles (in both dimensions) then the coloring is *completely periodic*. More formally, a coloring C is completely periodic with period p if and only if it is periodic with period p and $\forall w \in \{0,1,2,...,M-2,M-1\}, \forall (x,y) \in \aleph \times \aleph, \exists s_x, s_y$ such that $0 \leq s_x < p, 0 \leq s_y < p$ where $C(x+s_x,y) = w$ and $C(x,y+s_y) = w$.

A coloring works for a specific number of watermarks, but a family of colorings can be grouped together to create a *coloring scheme*. A coloring scheme $\{C_M\}_{M=1}^{\infty}$ is a set of colorings indexed by M, where C_M is a coloring for M watermarks. A coloring scheme $\{C_M\}_{M=1}^{\infty}$ is *completely periodic* with period $\{p_M\}_{M=1}^{\infty}$ if and only if the coloring C_M is completely periodic with period p_M for all $M \geq 1$. It is worth noting that the complete period of many coloring schemes is the number of colors itself; these schemes include: DM, the Cyclic schemes, and GRS; this is also true for the FX and RFX schemes when the number of colors is a power of two.

In what follows, whenever we say "rectangular subsection" of a grid, we implicitly include wraparound, e.g. in a 3×5 grid, the region $[2,0] \times [1,3]$ is considered to be rectangular (the reason for allowing wraparound will become apparent after reading Lemma 3–1). Given a coloring C and a rectangular subsection R, define a function W that computes the set of watermarks present in R, note that $W(R,C) = \{C(i,j), \forall (i,j) \in R\}$.

A watermarking entity will have certain desired constraints for a watermark placement scheme. Given an area threshold a and a watermark threshold b then a possible constraint on a scheme is that any cropping containing a or more tiles contains at least b distinct watermarks. More formally, given an area threshold a and a watermark threshold b a *constraint* (a,b) is *satisfied* for a grid G and coloring C if and only if for any rectangular subsection R in G, if $(|R| \geq a) \rightarrow (|W(R,C)| \geq b)$. A constraint (a,b) is said to be *universally satisfiable* if there is a coloring C such that for any grid G, C satisfies (a,b) for G. We consider only constraints (a,b) with $a \geq b$ and $b \leq M$, since it is trivial to prove that other constraints are unsatisfiable. Define a satisfiability function $S(C, M, (d_1, d_2), (a,b))$ that is true if and only if C satisfies the constraint (a,b) in a $d_1 \times d_2$ grid. Define a universally satisfiable function $US(C, M, (a,b))$ which is true if and only if the C universally satisfies constraint (a,b).

Lemma 3–1.: Given M watermarks, a coloring C that has complete period p, and a reasonable constraint (a,b) such that $S(C, M, (p,p), (a,b))$ is true, then $US(C, M, (a,b))$ is also true.

Proof: Suppose we are given an arbitrary grid and a rectangular subsection of that grid, call it R, of size $s_1 \times s_2$, where $s_1 s_2 \geq a$. We must show that $|W(R,C)| \geq b$. If s_1 or s_2 is greater than or equal to p then it is trivial since C has complete period p and thus contains all M watermarks. Assume $s_1 < p$ and $s_2 < p$, thus R fits in a $p \times p$ grid. Now R is a wraparound cropping in some $p \times p$ grid, and since $S(C_M, M, (p,p), (a,b))$ this area contains b watermarks. Therefore, the constraint is satisfied. □

A consequence of this lemma is that for the colorings defined for the database declustering problem, we need only to look at grids the size of the complete period for that coloring to determine if constraints are universally satisfiable. The following lemma shows how constraints that are universally satisfiable imply weaker constraints that are universally satisfiable.

Lemma 3–2.: If $US(C, M, (a, b))$, then: i) $US(C, M+1, (a, b))$, ii) $US(C, M, (a+1, b))$, and iii) $US(C, M, (a, b-1))$

Proof: The first part states that if a constraint can be universally satisfied for M watermarks, then it is universally satisfiable for $M+1$ watermarks. This is obvious since the $(M+1)$st watermark can be ignored, and the same constraint will still be satisfiable. Since any cropping containing $a+1$ tiles must contain a tiles, and likewise any cropping containing b watermarks must contain at least $b-1$ watermarks, the second and third parts are trivially true. □

3.2 Maximum Non-complete Area

Suppose an organization watermarks some data with the tiling method outlined previously; it would be desirable for this organization to know the largest rectangular subsection that does not contain its watermark as a measure of resilience to cropping of the placement scheme. There is such a subsection for every watermark; define the maximum area of all of these subsections as the *Maximum Non-Complete Area(MNCA)*. Formally, the MNCA of a coloring C for M colors is the value k such that $\neg US(C, M, (k, M))$ and $US(C, M, (k+1, M))$. Obviously, it is desirable to minimize the MNCA for a set of watermarks; note that a strictly optimal declustering would have a MNCA of $(M-1)$.

Theorem 3–3. The best achievable MNCA for any coloring of M watermarks, labeled $\{0, \cdots, M-1\}$ is $M-1$ (i.e. optimal) if and only if $M = 1, 2, 3,$ or 5.

Proof Sketch: For M=1, 2, or 3 the DM coloring scheme has optimal MNCA. For $M = 5$ the RPHM coloring has optimal MNCA. To show that the other cases cannot be done optimally, there are two cases to consider, M is even and M is odd.

Case 1: Suppose $M = 2k$ for some k (and ≥ 4), construct a $4 \times M$ grid (4 columns and M rows). BWOC, suppose that this can be colored optimally. The first column must contain all M colors, WLOG color them in sequential order top down as $(0, \cdots, 2k-1)$. Consider $2 \times k$ sections (which must contain all M colors) that have tiles in the first and second columns of the grid.

0	2	0	2
1	3	1	3
2	0	2	0
3	1	3	1

Diagram 1

From these it can be determined that the second column must be colored in the order $(k, \cdots, 2k-1, 0, \cdots, k-1)$. By similar reasoning, the third column must be $(0, \cdots, 2k-1)$ and the fourth column must be $(k, \cdots, 2k-1, 0, \cdots, k-1)$; the above construction is shown in Diagram 1 for a 4×4 grid colored with $M = 4$ colors. But this implies that a $4 \times \lceil \frac{M}{4} \rceil$ cropping only contains $2\lceil \frac{M}{4} \rceil < M$ colors and thus contradicts our assumption that the grid can be colored optimally.

Case 2: We omit this case of $M = 2k+1$ for some k, but it will be contained in the full version of the paper. However, the proof is similar to Case 1, but it is slightly more complicated. □

The previous theorem states that we cannot obtain optimal MNCA for most values of M. In this section we establish an upper bound on the best achievable MNCA of $O(M)$ for M colors. This is done by proving that the MNCA for GRS is $O(M)$ if M is a Fibonacci number, and this is generalized to any number of colors using a smoothing process that is defined after the next theorem.

Theorem 3–4. If a coloring C for M colors has a MNCA of r, then given $k \leq M$ colors it is possible to construct a coloring C' for k colors that has a MNCA no larger than r.

Proof: Suppose coloring C has a MNCA of r for M colors, which implies that $US(C, M, (r+1, M))$. Define a coloring C', where $C'(x, y) = (C(x, y) \bmod k)$. We must show $US(C', k, (r + 1, k))$. Suppose we are given a rectangular subsection R with area at least $r + 1$, and an arbitrary watermark $w \in \{0, 1, 2, ..., k - 1\}$. There must be a tile (x, y) in R, with $C(x, y) = w$ (since $US(C, M, (r + 1, M))$ and $k \leq M$), which implies $C'(x, y) = w$ and thus $US(C', k, (r + 1, k))$. □

The previous theorem implies that the best achievable MNCA for $M - 1$ colors can be no worse than the best achievable MNCA for M colors, or equivalently that the best achievable MNCA for a specific number of colors is a nondecreasing function of M. A coloring scheme that satisfies this property is called *MNCA-smooth*. Many coloring schemes are not MNCA-smooth (EXH, GRS, and FX), but we can modify these schemes so that this property will hold. Define a function *MA* that given a coloring returns the MNCA of the coloring. Given a coloring scheme $\{C_M\}_{M=1}^{\infty}$, define a new coloring scheme $\{D_M\}_{M=1}^{\infty}$ where $D_M = (C_k \bmod M)$ where k is chosen such that $MA(C_k) = \min_{M \leq j \leq MA(C_M)}(MA(C_j))$. This process creates a MNCA-smooth coloring scheme, which has MNCA no larger than $\{C_M\}_{M=1}^{\infty}$ for all values of M.

When the number of watermarks is a Fibonacci number (recall that they satisfy the recurrence $F_1 = 1, F_2 = 1$ and $F_k = F_{k-1} + F_{k-2}$), the GRS coloring scheme has a MNCA no larger than double the number of colors (see Theorem 3–5). Using Theorem 3–4, we can get a general bound of $\frac{10}{3}$ times the number of watermarks for any number of watermarks, see Corollary 3–6. Thus the GRS coloring scheme has a MNCA which is $O(M)$.

Theorem 3–5. The GRS coloring has a MNCA of no more than $2 * F_k$ for $M = F_k$ colors where F_k is the kth Fibonacci number.

Proof Sketch: We need only to consider croppings of an $M \times M$ grid with wraparound since the complete period of GRS is M. Suppose we are given such a cropping. To finish the proof we need the concept of gaps that has been defined for permutation schemes [3]. Given r consecutive rows there will be r instances of any color (one per row); the set of distances between these values (including the wraparound distance) will be the same for any color, and these distances are called the gaps of these rows (See Diagram 2 on the next page for more information on gaps). If an area is non-complete then it must have less columns than the maximum gap. It was shown in [3] and [10] that the maximum gap

for $r(= F_i + s)$ rows where $0 \leq s < F_{i-1}$ is F_{k-i+2}. It can be shown that $(F_i + s)(F_{k-i+2} - 1) < 2F_k$. Thus, given any number of rows the maximum area of a non-complete cropping is less than $2F_k$, hence we have proven that MNCA will be no larger than $2F_k$. □

1	2	3	4	5	0
4	5	0	1	2	3
0	1	2	3	4	5

Diagram 2

Diagram 2 shows a permutation coloring for $(M = 6)$ colors with 3 rows. The gaps between 0's are 2, 3, and 1. Notice that the gaps are the same (not necessarily in the same order) for any color.

Corollary 3–6. For M watermarks the MNCA-smoothed GRS scheme has a MNCA no more than $\frac{10}{3}M$.

Proof: If M is a Fibonacci number, then this bound is clearly true. Suppose M is not a Fibonacci number (note $M \geq 4$) then let F be the next Fibonacci number larger than M, note that $F \leq \frac{5}{3}M$, which is easy to verify with induction. Now we can use GRS for F colors to obtain a coloring for M colors that has a MNCA no larger than $2F$ (by theorem 3-4 and theorem 3-5). So the MNCA will be no larger than $\frac{10}{3}M$. □

3.3 Minimum Non-full Area

Another desirable trait of a watermark placement scheme is for small areas to have unique colors. For a coloring there is a minimum area that does not contain unique colors, call this area the *Minimum Non-Full Area(MNFA)*. Formally, the MNFA of a coloring C for M colors is the value k such that $\neg US(C, M, (k, \min\{M, k\}))$ and $US(C, M, (k-1, \min\{M, k-1\}))$. The MNFA is useful since it is the minimum area for which an attacker can attempt to "get away with something", i.e. a cropping that could contain more watermarks than it actually does. It is desirable to maximize the MNFA of a coloring, and the MNFA for a strictly optimal placement is ∞.

Lemma 3–7. If a coloring has a MNFA that is optimal for M colors, then the coloring will be optimal for MNCA as well.

Proof: Since the MNFA of C is optimal we know that $\forall k, US(C, M, (k, \min\{M, k\}))$, so this must be true for $k = M$, and so $US(C, M, (M, M))$. However, this implies that the MNCA is optimal. □

Theorem 3–8. The MNFA for any coloring of M watermarks is ∞ (i.e. optimal) if and only if $M = 1, 2, 3,$ or 5.

Proof: For M=1, 2, or 3 the DM coloring scheme has optimal MNFA. For $M = 5$ the RPHM coloring has optimal MNFA. If for other values of M there was an optimal coloring for MNFA then this coloring would be optimal for MNCA (by lemma 3-7), but this contradicts theorem 3-4. □

Theorem 3–9. If a coloring C for M colors has a MNFA of r, then given $k \geq M$ colors C has a MNFA $\geq r$ for k colors.

Proof: Since C has a MNFA of r we know that $US(C, M, (r-1, r-1))$, but by applying the first part of lemma 3-2 repeatedly we get $US(C, k, (r-1, r-1))$. □

The previous theorem implies that the best achievable MNFA for $M+1$ colors can be no worse then the best MNFA for M colors, i.e. the best achievable MNFA is a nondecreasing function of M. A coloring scheme that satisfies this property is called *MNFA-smooth*. Many coloring schemes are not MNFA-smooth (EXH, GRS, and FX), but we can modify these schemes so that this property will hold. Like the MNCA, we can define a MNFA-smoothing process. Define a function *MNFA* that given a coloring returns the MNFA of the coloring. Given a coloring scheme $\{C_M\}_{M=1}^{\infty}$, define a new coloring scheme $\{D_M\}_{M=1}^{\infty}$ such that $D_M = C_k$ where k is chosen such that $MNFA(C_k) = \max_{1 \le j \le M}(MNFA(C_j))$. This process creates a MNFA-smooth coloring scheme, which has MNFA no worse than $\{C_M\}_{M=1}^{\infty}$ for all values of M. However, this transformation has a drawback; if this smoothing process is used then some colors will not be used, which means that some watermarks will not be contained in the data. However, this problem can be fixed by treating each color in the smoothed scheme as a group of colors and whenever a tile is assigned to a group it is randomly assigned a watermark from that group. In Theorem 3–10 and Corollary 3–11 we prove a lower bound of $\Omega(M)$ for the best achievable MNFA for any number of colors M. Like the proof for the upper bound on MNCA, we use the GRS coloring scheme to prove this lower bound on MNFA.

Theorem 3–10. The GRS coloring scheme has a MNFA larger than $\frac{3}{7}F_k$ for $M = F_k$ colors where F_k is the kth Fibonacci number.

Proof Sketch: We only need to consider croppings of an $M \times M$ grid with wraparound since complete period of GRS is M. Suppose we are given a such a cropping. we will use the same concept of gaps as in the proof of Theorem 3–5. If an area is non-full then it must have more columns than the minimum gap. It was shown in [3] and [10] that the minimum gap for $r(= F_i + s)$ rows where $0 \le s < F_{i-1}$ is at least F_{k-i}. It can be shown that $(r)(F_{k-i}+1) \ge (F_i)(F_{k-i}+1) > \frac{3}{7}F_k$. Thus given any number of rows there must be at least $\frac{3}{7}F_k$ tiles before there is a duplicate. Hence, the MNFA will be no less than $\frac{3}{7}M$ □

Corollary 3–11. For M watermarks there is a coloring where the MNFA is no less than $\frac{9}{35}M$.

Proof: If M is a Fibonacci number, then this bound is clearly true. Suppose M is not a Fibonacci number (note $M \ge 4$) then let F be the largest Fibonacci number smaller than M, an easy induction shows that $F \ge \frac{3}{5}M$. Now we can use GRS for F colors to obtain a coloring for M colors that has a MNFA no smaller than $\frac{3}{7}F$ (by theorem 3-9 and theorem 3-10). So the MNFA of the MNFA-smoothed scheme will be no smaller than $\frac{9}{35}M$. □

3.4 Other Satisfiability Properties

Suppose that to prove ownership of an item an entity only has to recover about half of its watermarks. The question becomes how much area is needed so that about half of the colors are represented. Theorem 3–12 states that it is possible to color a grid with $M = 2^k$ colors in such a way that any area containing M

tiles has at least $\frac{M}{2} = 2^{k-1}$ distinct colors. Corollary 3–13 generalizes this result for non-powers of two.

Theorem 3–12. Given $M = 2^k$ colors, there is a coloring C such that $US(C, M, (M, \frac{M}{2}))$.

Proof Sketch : Use the RFX coloring scheme for M colors. We only need to consider wraparound croppings in an $M \times M$ grid since the complete period for RFX is M when M is a power of 2. It can be shown that if you partition the columns into 2^s groups each with 2^{k-s} columns (that have a common prefix of size s), then given any column partition and any 2^s consecutive rows (including wraparound), the $2^{k-s} \times 2^s$ cropping defined by the intersection of the column partition and the rows will contain unique colors (and hence all colors). Furthermore, any cropping containing M tiles must have at least $\frac{M}{2}$ tiles in one of these regions, hence there must be at least $\frac{M}{2}$ colors. □.

Corollary 3–13. Given M colors, there is a coloring C such that $US(C, M, (2^{\lfloor \log(M) \rfloor}, 2^{\lfloor \log(M) \rfloor - 1}))$.

Proof: By theorem 3-12, we know that there is a coloring C such that $US(C, 2^{\lfloor \log(M) \rfloor}, (2^{\lfloor \log(M) \rfloor}, 2^{\lfloor \log(M) \rfloor - 1}))$. But since $M \geq 2^{\lfloor \log(M) \rfloor}$, by Lemma 3–2 we can conclude that $US(C, M, (2^{\lfloor \log(M) \rfloor}, 2^{\lfloor \log(M) \rfloor - 1}))$. □

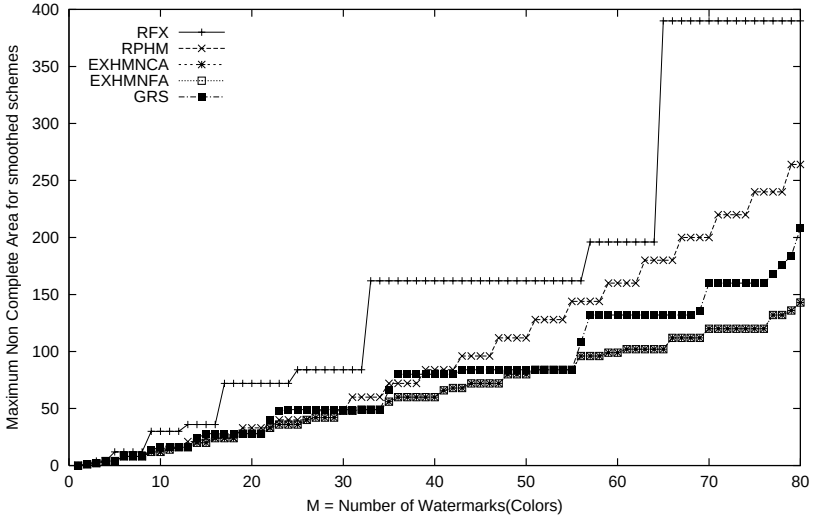


Fig. 1. MNCA of various MNCA-smoothed schemes

4 Experimental Results

To compare colorings we looked at the performance of various schemes with regards to their MNFA and MNCA. The colorings that were examined are: DM,

FX, RFX, RPHM, EXH (optimized for MNCA), EXH (optimized for MNFA), and GRS. Due to page constraints we only include the MNFA of these MNFA-smoothed schemes for up to 80 colors (Figure 1). Note that DM and FX are omitted due to poor performance.

Figure 1 shows that the stronger schemes are EXH and GRS, with EXH slightly outperforming GRS. When smoothing is used the criterion used to optimize EXH appear to have little effect on the performance of the scheme. Similar results occur when the performance criterion is MNFA.

5 Conclusion

Watermarking is a tool for digital rights management, and inserting multiple watermarks into the same data is an important application. A scheme for inserting multiple watermarks into an object consists of tiling the data into uniform rectangles and placing each watermark into a set of tiles; placement of the watermarks in such an environment effects the resilience of the object to croppings. This problem is relates to the distributed database declustering problem, but differs from the latter in significant aspects.

We propose two orthogonal heuristics to compare schemes: MNCA and MNFA. Other than in very limited cases, it is impossible to have optimal performance for either heuristic for every cropping in a grid. Given M colors to place in a grid, the GRS scheme that is smoothed for MNCA has a MNCA of $O(M)$ for any grid, and the GRS scheme that is smoothed for MNFA has a MNFA of $\Omega(M)$. Furthermore, if M is a Fibonacci number then the GRS scheme will achieve both of these bounds; extending both bounds to any number of colors is left for future work. Also, the RFX scheme was proven to have good properties if only half of the watermarks need to be recovered. Furthermore, we performed experiments to evaluate the performance of various schemes with regards to MNCA and MNFA and found that the GRS and EXH schemes have the strongest performance among the colorings schemes that were analyzed.

Acknowledgments. The authors would like to thank Dr. Rei Safavi-Naini for introducing us to this area and Dr. Sunil Prabhakar for his help with the distributed database declustering background.

References

1. K.A.S. Abdel-Ghaffar and A. El Abbadi. Optimal allocation of two-dimension data. In *Int. Conf. on Database Theory*, pages 409–418, Delphi, Greece, Jan. 1997.
2. M. J. Atallah and S. Prabhakar. (Almost) optimal parallel block access for range queries. In *Proc. of the 19th ACM Symposium on Principles of Database Systems (PODS)*, Dallas, Texas, May 2000.
3. R. Bhatia, R. K. Sinha, and C.-M. Chen. Declustering using golden ratio sequences. In *Proc. of Int'l. Conference On Data Engineering (ICDE)*, San Diego, California, March 2000.

4. G. Brisbane, R. Safavi-Naini, and P. Ogunbona. Region-based Watermarking for Images. In *Proceedings of Information Security Workshop (ISW)*, LNCS 1729, 1999, pages 154–166.
5. C. Chen and C. Cheng. From Discrepancy to Declustering: Near-optimal multidimensional declustering strategies for range queries. In *A CM Symposium On Principles of Database Systems (PODS)* 2002 pages 29–38.
6. H. C. Du and J. S. Sobolewski. Disk allocation for cartesian product files on multiple-disk Systems. *ACM Trans of Database Systems*, 7(1):82–101, 1982.
7. C. Faloutsos and P. Bhagwat. Declustering using fractals. In *Proc. of the 2nd Int. Conf. On Parallel und Distributed Information Systems*, pages 18–25, San Diego, CA, Jan 1993.
8. K. Frikken, M. Atallah,S. Prabhakar,R. Safavi-Naini. Optimal Parallel 1/0 for Range Queries through Replication. In *Proc. of DEXA*, LNCS 2453, pages 669–678, 2002.
9. J. Gray, B. Horst, and M. Walker. Parity striping of disc arrays: Low-cost reliable Storage with acceptable throughput. In *Proceedings of the Int. Conf. On Very Large Data Bases*, pages 148-161, Washington DC., August 1990.
10. A. Itai and Z.Rosberg. A golden ratio control policy for a multiple-access channel. In *IEEE Transactions On Automatic Control*, AC-29:712–718, 1984.
11. M. H. Kim and S. Pramanik. Optimal file distribution for partial match retrieval. In *Proc. ACM SIGMOD Int. Conf. On Management of Data*, pages 173–182, Chicago, 1988.
12. J. Li, J. Srivastava, and D. Rotem. CMD: a multidimensional declustering method for parallel database Systems. In *Proceedings of the Int. Conf. On Very Large Data Bases*, pages 3–14, Vancouver, Canada, August 1992.
13. F. Mintzer and G. Braudaway. If one watermark is good, are more better? Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, vol. 4, Phoenix, Arizona, May 1999.
14. F. Mintzer, G. Braudaway, and M. Yeung. Effective and Ineffective Digital Watermarks. In *IEEE ICIP*, volume 111, pages 9–12, Santa-Barbara, Cal, October 1997.
15. S. Prabhakar, K. Abdel-Ghaffar, D. Agrawal, and A. E1 Abbadi. Cyclic allocation of two-dimensional dat a. In *Proc. of the International Conference On Data Engineering (ICDE'98)*, pages 94–101, Orlando, Florida, Feb 1998.
16. Rakesh K. Sinha, Randeep Bhatia, and Chung-Min Chen. Asymptotically optimal declustering schemes for range queries. In *Proc. of 8th International Conference On Database Theory (ICDT)*, pages 144–158, London, UK, January 2001.
17. P. Sanders, S. Egner, and J . Korst. Fast concurrent access to parallel disks. In *11th ACM-SIAM Symposium On Discrete Algorithms*, 2000.
18. N. Sheppard, R. Safavi-Naini, P. Ogunbona. On multiple watermarking. In *ACM Multimedia Conference*, ACM Multimedia 2001, pp. 3–6.
19. A. Tosun and H. Ferhatosmanoglu. Optimal Parallel 1/0 Using Replication. OSU Technical Report OSU-CISRC-ll/Ol-TR26, 2001.

On Simultaneous Planar Graph Embeddings

P. Brass[1], E. Cenek[2], Christian A. Duncan[3], A. Efrat*[4], C. Erten*[4], D. Ismailescu[5], S.G. Kobourov*[4], A. Lubiw[2], and J.S.B. Mitchell**[6]

[1] Dept. of Computer Science, City College of New York, peter@cs.ccny.cuny.edu
[2] Dept. of Computer Science, University of Waterloo, {acenek,alubiw}@uwaterloo.edu
[3] Dept. of Computer Science, Univ. of Miami, duncan@cs.miami.edu
[4] Dept. of Computer Science, Univ. of Arizona, {alon,cesim,kobourov}@cs.arizona.edu
[5] Dept. of Mathematics, Hofstra University, Dan.P.Ismailescu@hofstra.edu
[6] Dept. of Applied Mathematics and Statistics, Stony Brook University, jsbm@ams.sunysb.edu

Abstract. We consider the problem of simultaneous embedding of planar graphs. There are two variants of this problem, one in which the mapping between the vertices of the two graphs is given and another in which the mapping is not given. In particular, given a mapping, we show how to embed two paths on an $n \times n$ grid, and two caterpillar graphs on a $3n \times 3n$ grid. We show that it is not always possible to simultaneously embed three paths. If the mapping is not given, we show that any number of outerplanar graphs can be embedded simultaneously on an $O(n) \times O(n)$ grid, and an outerplanar and general planar graph can be embedded simultaneously on an $O(n^2) \times O(n^2)$ grid.

1 Introduction

The areas of graph drawing and information visualization have seen significant growth in recent years [10,15]. Often the visualization problems involve taking information in the form of graphs and displaying them in a manner that both is aesthetically pleasing and conveys some meaning. The aesthetic criteria alone are the topic of much debate and research, but some generally accepted and tested standards include preferences for straight-line edges or those with only a few bends, a limited number of crossings, good separation of vertices and edges, as well as a small overall area. Some graphs change over the course of time and in such cases it is often important to preserve the "mental map".

Consider a system that visualizes the evolution of software, information can be extracted about the program stored within a CVS version control system [8]. Inheritance graphs, program call-graphs, and control-flow graphs can be visualized as they evolve in time; see Fig. 1. Such tools allow programmers to understand the evolution of a legacy program: Why is the program structured the

* Partially supported by NSF Grant ACR-0222920.
** Partially supported by Metron Aviation, Inc., NASA Ames Research (NAG2-1325), NSF (CCR-0098172), and the U.S.-Israel Binational Science Foundation.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 243–255, 2003.
© Springer-Verlag Berlin Heidelberg 2003

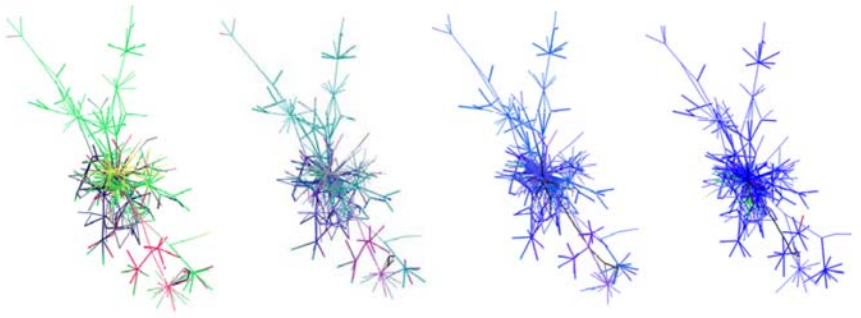


Fig. 1. The inheritance graph of a large Java program as it evolves through time. Different colors indicate different authors. For every time-step that a node does not change, its color fades to blue.

way it is? Which programmers were responsible for which parts of the program during which time periods? Which parts of the program appear unstable over long periods of time and may need to be rewritten? For such a visualization tool, it is essential to preserve the mental map for the graph under scrutiny. That is, slight changes in the graph structure should not yield large changes in the actual drawing of the graph. Vertices should remain roughly near their previous locations and edges should be routed in roughly the same manner as before [10, 15].

While graphs that evolve through time are not necessarily planar, solving the planar case can provide intuition and ideas for the more general case. Thus, the focus of the this paper is on the problem of simultaneous embedding of planar graphs. This problem is related to the thickness of graphs; see [18] for a survey. The thickness of a graph is the minimum number of planar subgraphs into which the edges of the graph can be partitioned. Thickness is an important concept in VLSI design, since a graph of thickness k can be embedded in k *layers*, with any two edges drawn in the same layer intersecting only at a common vertex and vertices placed in the same location in all layers. A related graph property is *geometric thickness*, defined to be the minimum number of layers for which a drawing of G exists having all edges drawn as straight-line segments [11]. Finally, the *book thickness* of a graph G is the minimum number of layers for which a drawing of G exists, in which edges are drawn as straight-line segments and vertices are in convex position [2]. It has been shown that the book thickness of planar graphs is no greater than four [21].

As initiated by Cenek and Lubiw [5], we look at the problem almost in reverse. Assume we are given the layered subgraphs and now wish to simultaneously embed the various layers so that the vertices coincide and no two edges of the same layer cross. Take, for example, two graphs from the 1998 Worldcup; see Fig. 2. One of the graphs is a tree illustrating the games played. The other is a graph showing the major exporters and importers of players on the club level. In displaying the information, one could certainly look at the two graphs separately, but then there would be little correspondence between the two layouts if they

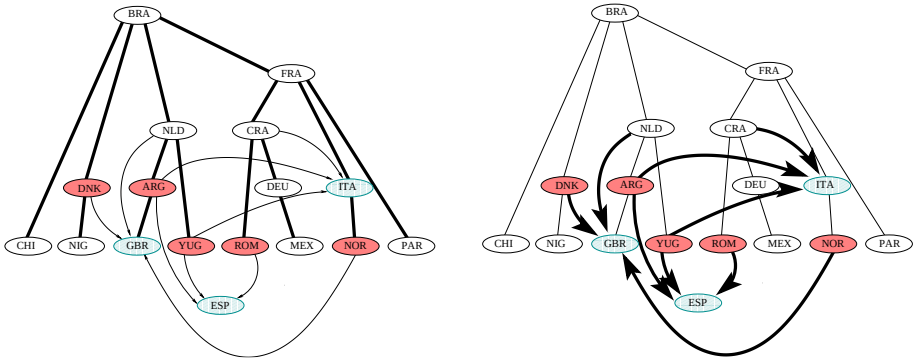


Fig. 2. The vertices of this graph represent the round of 16 teams from Worldcup 1998 (plus Spain). The 8 teams eliminated in the round of 16 are on the bottom; next are the 4 teams eliminated in the quarter-finals, etc.Thick edges in the left drawing indicate matches played. Thick edges in the right drawing indicate export of players on the club level. The light (dark) shaded vertices indicate importers (exporters) of players.

were created independently, since the viewer has no "mental map" between the two graphs. Using a simultaneous embedding, the vertices can be placed in the exact same locations for both graphs, making the relationships more clear. This is different than simply merging the two graphs together and displaying the information as one large graph.

In simultaneous embeddings, we are concerned with crossings but not between edges belonging to different layers (and thus different graphs). Typical graph drawing algorithms lose all information about the separation of the two graphs and so must also avoid such non-essential crossings. Techniques for displaying simultaneous embeddings can be quite varied. One may choose to draw all graphs simultaneously, employing different edge styles, colors, and thickness for each edge set. One may choose a more three-dimensional approach in order to differentiate between layers. One may also choose to show only one graph at a time and allow the users to choose which graph they wish to see by changing the edge set (without moving the vertices). Finally, one may highlight one set of edges over another, giving the effect of "bolding" certain subgraphs, as in Fig. 2.

The subject of simultaneous embeddings has many different variants, several of which we address here. The two main classifications we consider are embeddings with and without predefined vertex mappings.

Definition 1. Given k planar graphs $G_i = (V, E_i)$ for $1 \leq i \leq k$, *simultaneous (geometric) embedding of G_i with mapping* is the problem of finding plane straight-line drawings D_i of G_i such that for every $u \in V$ and any two drawings D_i and D_j, u is mapped to the same point on the plane in all k drawings.

Definition 2. Given k planar graphs $G_i = (V_i, E_i)$ for $1 \leq i \leq k$, *simultaneous (geometric) embedding of G_i without mapping* is the problem of finding plane straight-line drawings D_i of G_i such that given any two drawings D_i and D_j

there exists a bijective mapping $f : V_i \to V_j$. such that $u \in V_i$ and $v \in V_j$ are mapped to the same point in the plane in both drawings.

Note that in the final drawing a crossing between two edges a and b is allowed *only if* there does not exist an edge set E_i such that $a, b \in E_i$.

In both versions of the problem, we are interested in embeddings that map the vertices to a small cardinality set of candidate vertex locations. Throughout this paper, we make the standard assumption that candidate vertex locations are at integer grid points, so our objective is to bound the size of the integer grids required.

The following table summarizes our current results regarding the two versions under various constraints on the type of graphs given; entries in the table indicate the size of the integer grid required.

Graphs	**With Mapping**	**Without Mapping**
G_1: Planar, G_2: Outerplanar	not always possible	$O(n^2) \times O(n^2)$
G_1, G_2: Outerplanar	not always possible	$O(n) \times O(n)$
C_1, C_2: Caterpillar	$3n \times 3n$	$O(n) \times O(n)$ (outerplanar)
C_1: Caterpillar, P_2: Path	$n \times 2n$	$O(n) \times O(n)$ (outerplanar)
P_1, P_2: Path	$n \times n$	$\sqrt{n} \times \sqrt{n}$
C_1, C_2: Cycle	$4n \times 4n$	$\sqrt{n} \times \sqrt{n}$
P_1, P_2, P_3: Path	not always possible	$\sqrt{n} \times \sqrt{n}$

2 Previous Work

Computing straight-line embeddings of planar graphs on the integer grid is a well-studied graph drawing problem. The first solutions to this problem are given by de Fraysseix, Pach and Pollack [9], using a canonical labeling of the vertices in an algorithm that embeds a planar graph on n vertices on the $(2n-4)\times(n-2)$ integer grid and, independently, by Schnyder [19] using the barycentric coordinates method. The algorithm of Chrobak and Kant [7] embeds a 3-connected planar graph on an $(n-2) \times (n-2)$ grid so that each face is convex. Miura, Nakano, and Nishizeki [17] further restrict the graphs under consideration to 4-connected planar graphs with at least four vertices on the outer face and present an algorithm for straight-line embeddings of such graphs on an $(\lceil n/2 \rceil - 1) \times (\lfloor n/2 \rfloor)$ grid.

Another related problem is that of simultaneously embedding more than one planar graph, not necessarily on the same point set. This problem dates back to the circle-packing problem of Koebe [16]. Tutte [20] shows that there exists a simultaneous straight-line representation of a planar graph and its dual in which the only intersections are between corresponding primal-dual edges. Brightwell and Scheinerman [4] show that every 3-connected planar graph and its dual can be embedded simultaneously in the plane with straight-line edges so that the primal edges cross the dual edges at right angles. Erten and Kobourov [13] present an algorithm for simultaneously embedding a 3-connected planar graph and its dual on an $O(n) \times O(n)$ grid.

Bern and Gilbert [1] address a variation of the problem: given a straight-line planar embedding of a planar graph, find suitable locations for dual vertices

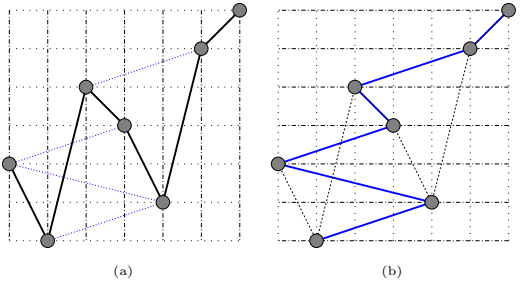


Fig. 3. An example of embedding two paths on an $n \times n$ grid. The two paths are respectively $v_1, v_2, v_3, v_4, v_5, v_6, v_7$ and $v_2, v_5, v_1, v_4, v_3, v_6, v_7$. They are drawn using (a) increasing x-order and (b) increasing y-order.

so that the edges of the dual graph are also straight-line segments and cross only their corresponding primal edges. They present a linear-time algorithm for the problem in the case of convex 4-sided faces and show that the problem is NP-hard for the case of convex 5-sided faces.

3 Simultaneous Embedding with Mapping

We first address the simplest version of the problem: embedding paths.

Theorem 1. *Let P_1 and P_2 be 2 paths on the same vertex set, V, of size n. Then a simultaneous geometric embedding of P_1 and P_2 with mapping can be found in linear time and on an $n \times n$ grid.*

Proof: For each vertex $u \in V$, we embed u at the integer grid point (p_1, p_2), where $p_i \in \{1, 2, \ldots, n\}$ is the vertex's position in the path P_i, $i \in \{1, 2\}$. Then, P_1 is embedded as an x-monotone polygonal chain, and P_2 is embedded as a y-monotone chain; thus, neither path is self-intersecting. See Fig. 3. □

This method can be extended to handle two cycles, but does not extend to more than two paths. We present these results in turn.

Theorem 2. *Let C_1 and C_2 be 2 cycles on the same vertex set of size n, each with the edges oriented clockwise around an interior face. Then a simultaneous geometric embedding (with mapping) of C_1 and C_2 that respects the orientations can be found in linear time on a $4n \times 4n$ grid, unless the two cycles are the same cycle oppositely oriented. In the latter case no such embedding exists.*

Proof: Assume that C_1 and C_2 are not the same cycle oppositely oriented. Then there must exist a vertex v such that the predecessor of v in C_1, say a, is different from the successor of v in C_2, say b. Place v at the point $(0, 0)$, and use the simultaneous path drawing algorithm from Theorem 1 to draw the path in C_1 from v to a as an x-monotone path, and the backward path in C_2 from v back to b as a y-monotone path. Then a will be drawn as the point of maximum x coordinate, and b as the point of maximum y coordinate.

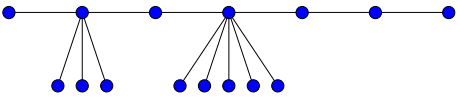


Fig. 4. A caterpillar graph C is drawn with solid edges. The vertices on the top row and the edges between them form the spine. The vertices on the bottom row form the legs of the caterpillar.

Without destroying the simultaneous embedding, we can pull v diagonally to the grid point $(-n, -n)$ and a horizontally out to the right until the line segment av lies completely below the other points. Let c be the predecessor of v in C_2. The line segment cv has slope at least $1/2$. The y-coordinate distance between v and a is at most $2n$. If the x-coordinate distance between v and a is greater than $4n$ then the slope of the segment av becomes less than $1/2$ and and it is below the other points. The same idea applies to b (this time shifting b up vertically) also and we get a grid of total size $4n \times 4n$. □

Theorem 3. *There exist three paths $\mathcal{P} = \bigcup_{1 \le i \le 3} P_i$ on the same vertex set V such that at least one of the layers* must *have a crossing.*

Proof: A path of n vertices is simply an ordered sequence of n numbers. The three paths we consider are: 714269358, 824357169 and 758261439. For example, the sequence 714269358 represents the path $(v_7, v_1, v_4, v_2, v_6, v_9, v_3, v_5, v_8)$. We will write ij for the edge connecting v_i to v_j. There are twelve edges in the union of these paths

$$E = \{14, 16, 17, 24, 26, 28, 34, 35, 39, 57, 58, 69\}.$$

It is easy to see that the graph G consisting of these edges is a subdivision of $K_{3,3}$ and therefore non-planar: collapsing 1 and 7, 2 and 8, 3 and 9 yields the classes {1,2,3} and {4,5,6}.

It follows that there are two nonadjacent edges of G that cross each other. It is easy to check that *every pair of nonadjacent edges* from E appears in *at least one* of the paths given above. Therefore, at least one path will cross itself which completes the proof. □

3.1 Caterpillars

A simple class of graphs similar to paths is the class of caterpillar graphs. Let us first define the specific notion of a caterpillar graph.

Definition 3. *A caterpillar graph $C = (V, E)$ is a tree such that the graph obtained by deleting the leaves, which we call the* **legs** *of C, is a path, which we call the* **spine** *of C; see Fig. 4.*

We describe an algorithm to simultaneously embed two caterpillars on a $3n \times 3n$ grid. As a first step in this direction we argue that a path and a caterpillar can be embedded in a smaller area, as the following theorem shows.

Theorem 4. *Given a path P and a caterpillar graph C, we can simultaneously embed them, with mapping, on an $n \times 2n$ grid.*

Proof: We use much the same method as embedding two paths, with one exception: we allow some vertices to share the same x-coordinate. Let S and L, respectively, denote the spine and the legs of C. For a vertex v let $o_p(v)$ denote v's position in P. If v is in S, then let $o_c(v)$ be its position in S and place v initially at the location $(2o_c(v), o_p(v))$. Otherwise, if $v \in L$, let $o_c(v) = o_c(p(v))$ be its parent's position and initially place v at the location $(2o_c(v) + 1, o_p(v))$.

We now proceed to attach the edges. By preserving the y-ordering of the points, we guarantee that the path has no crossings. In our embedding, we may need to shift, but we shall only perform right shifts. That is, we shall push points to the right of a vertex v by one unit right, in essence inserting one extra grid location when necessary. Note that this step still preserves the y-ordering.

To attach the caterpillar edges, we march along the spine. Let $L(u)$ denote the legs of a vertex u in the spine S. If we do not consider any edges of S then all the legs can be drawn with straight-line edges and no crossings by the initial placement. Now when we attach an edge from u to v on the spine, where $u, v \in S$, it is not planar if and only if there exists $w \in L(u)$ that is collinear with u and v. In this case, we simply shift v and all succeeding points by one unit to the right. We continue the right shift until none of the legs is collinear with u and v. Now, the edge from u to v on the spine is no longer collinear with other vertices. This right shift does not affect the planarity of the legs since the relative x-coordinates of the vertices are still preserved. The number of shifts we made is bounded by $|L(u)|$.

We continue in this manner until we have attached all edges. Let k be the total number of legs of the caterpillar. Then the total number of shifts made is k. Since we initially start with $2 \times (n-k)$ columns in our grid, the total number of columns necessary is $2n - k$. Thus, in the worst case the grid size needed is less than $2n \times n$. □

The algorithm for embedding two caterpillars is also similar but before we can prove our main result for caterpillars, we need an intermediary theorem. In order to embed two caterpillars, we allow shifts in two directions. Let $C_1 = (V, E_1)$ and $C_2 = (V, E_2)$ be two caterpillars. Denote the vertices on the spine of C_1 (C_2) with S_1 (S_2). Let $L_1(u)$ ($L_2(u)$) denote the legs of $u \in S_1$ (S_2). Let $\mathcal{T}_1$ ($\mathcal{T}_2$) be a fixed traversal order of vertices on S_1 (S_2). Let $u(X)$ and $u(Y)$ denote the x-coordinate and y-coordinate of the vertex u, respectively. We will place the vertices such that the following *initial placement invariants* hold:

1. For any $u, v \in V$, $u(X) \neq v(X)$ and $u(Y) \neq v(Y)$.
2. If $u \in S_1$ appears before $v \in S_1$ in $\mathcal{T}_1$ then $u(X) < w(X) < v(X)$ where $w \in L_1(u)$. If $u \in S_2$ appears before $v \in S_2$ in $\mathcal{T}_2$ then $u(Y) < w(Y) < v(Y)$ where $w \in L_2(u)$.

3. The set of vertices belonging to $L_1(u)$ that are above (below) $u \in S_1$ are monotonically increasing in the x-coordinate, and monotonically non-increasing (non-decreasing) in the y-coordinate. Similarly for C_2, the set of vertices belonging to $L_2(u)$ that are to the left (right) of $u \in S_2$ are monotonically increasing in the x-coordinate, and monotonically non-decreasing (non-increasing) in the y-coordinate.

Theorem 5. *The initial placement can be done on an $n \times n$ grid.*

Proof. We start by assigning x-coordinates of the vertices in S_1 by following the order in $\mathcal{T}_1$. The first vertex is assigned 1. We assign $v(X) = u(X) + |L_1(u)| + 1$ where $v \in S_1$ follows $u \in S_1$ in $\mathcal{T}_1$. Similarly we assign y-coordinates of the vertices in S_2, i.e., the first vertex is assigned 1 and $v(Y) = u(Y) + |L_2(u)| + 1$ where $v \in S_2$ follows $u \in S_2$ in $\mathcal{T}_2$.

Next we assign the x-coordinates of the vertices in $L_1(u)$ for each $u \in S_1$. We sort the vertices in $L_1(u)$ based on their y-coordinate distance from u in descending order. For each $w \in L_1(u) \cup \{u\}$, if $w \in S_2$, we use $w(Y)$ for comparison while sorting otherwise $w \in L_2(w')$ for some $w' \in S_2$ and we use $w'(Y) + 1$. Following this sorted order we assign $u(X)+1$, $u(X)+2$, ... to each vertex in $L_1(u)$. While sorting we use the same y-coordinate for two vertices $r, r' \in L_1(u)$ only if $r, r' \in L_2(v)$. In this case their x-coordinates get randomly assigned. However, this is not a problem, since the y-coordinate calculation of the legs in C_2 takes into account the x-coordinates we just calculated, and both the coordinates will then be compatible in terms of the initial placement invariants above. For assigning the y-coordinates of the vertices in $L_2(v)$, first we partition its vertices such that $r, r' \in L_2(v)$ are in the same partition if and only if $r, r' \in L_1(u)$ for some $u \in S_1$. We now calculate the y-coordinates of these partitions in $L_2(v)$ similar to the x-coordinate calculation above (taking the x-coordinate of a random vertex in the partition for comparison in sorting) , but this time considering the exact x-coordinates we just calculated.

After the initial placement we get the arrangement in Fig. 5. It is easy to see that with the initial placement invariants satisfied, for any $u \in S_1$ (S_2), any leg $w \in L_1(u)$ ($L_2(u)$) is visible from u and if we do not consider the edges on the spine, C_1 (C_2) is drawn without crossings.

Theorem 6. *Let C_1 and C_2 be 2 caterpillars on the same vertex set of size n. Then a simultaneous geometric embedding of C_1 and C_2 with mapping can be found on a $3n \times 3n$ grid.*

Proof: In the initial placement, a spine edge between $u, v \in S_1$ is not planar if and only if a vertex $w \in L_1(u)$ is collinear with u and v. We can avoid such collinearities while ensuring that no legs are crossing by shifting some vertices up/right. The idea is to grow a rectangle starting from the bottom-left corner of the grid, and to make sure that parts of C_1 and C_2 that are inside the rectangle are always non-crossing. This is achieved through additional shifting of the vertices up/right.

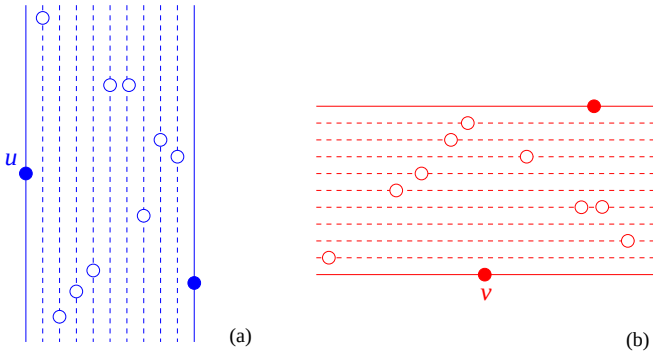


Fig. 5. a) Arrangement of $u \in S_1$ and $L_1(u)$. The legs of u are shown with empty circles. The x-coordinate of each vertex in $L_1(u)$ is determined by its vertical distance from u. b) Arrangement of $v \in S_2$ and $L_2(v)$. The legs of v are shown with empty circles. The y-coordinate of each vertex in $L_2(v)$ is determined by its horizontal distance from v.

First we make the following observation regarding the shifting:

Observation: Given a point set arrangement that satisfies the initial placement invariants, shifting any vertex $u \in V$ and all the vertices that lie above (to the right of) u up (right) by one unit preserves the invariants.

Since shifting a set of points up, starting at a certain y-coordinate, does not change the relative positions of the points, the invariants are still preserved.

We start out with the rectangle $\mathcal{R}_1$ such that the bottom-left corner of $\mathcal{R}_1$ is the bottom-left corner of the grid and the upper-right corner is the location of the closest vertex u, where $u \in S_1$ or $u \in S_2$. Since no other vertices lie in $\mathcal{R}_1$, the parts of C_1, C_2 inside $\mathcal{R}_1$ are non-crossing.

Now assume that after the k^{th} step of the algorithm, the parts of the caterpillars lying inside $\mathcal{R}_k$ are planar. We find the closest vertex v, to $\mathcal{R}_k$, where $v \in S_1$ or $v \in S_2$. There are two cases.

- *Case 1:* v is above $\mathcal{R}_k$, i.e., $x(v)$ is between the x-coordinate of the left edge and right edge of the rectangle. Enlarge $\mathcal{R}_k$ in the *y-direction* so that v lies on the top edge of the rectangle, and call the new rectangle $\mathcal{R}_{k+1}$. Let u (u') be the spine vertex before (after) v in $\mathcal{T}_1$. Let w (w') be the spine vertex before (after) v in $\mathcal{T}_2$. If any one of u, u', w, or w' lies inside $\mathcal{R}_{k+1}$ we check if v is visible from that vertex. If not, we shift v one unit up and enlarge $\mathcal{R}_{k+1}$ accordingly.
- *Case 2:* v is not above $\mathcal{R}_k$. If v is to the right of $\mathcal{R}_k$ we enlarge it in the *x-direction* so that v lies on the right edge of the rectangle, otherwise we enlarge it in both x and y directions so that v lies on the top-right corner. We call the new rectangle $\mathcal{R}_{k+1}$. As in *Case 1*, we check for the visibility of the neighboring vertices along the spines, but in this case we perform a

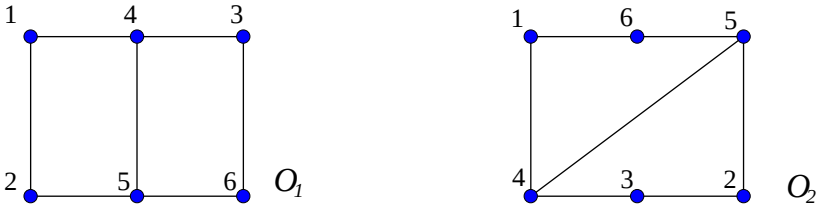


Fig. 6. Given the above mapping between the vertices; the outerplanar graphs O_1 and O_2 can not be embedded simultaneously.

right shift and enlarge $\mathcal{R}_{k+1}$ in the *x-direction* accordingly, if we encounter any collinearities.

When we perform an up/right shift, we do not make any changes inside the rectangle, so the edges drawn inside the rectangle remain non-crossing. Each time we perform a shift we eliminate a collinearity between the newly added vertex v and the vertices lying inside the rectangle. Hence, after a number of shifts all the collinearities involving v and such vertices inside the rectangle will be resolved, and all the edges inside our new rectangle, including the edges involving the new vertex v are non-crossing.

From the above observation shifting the vertices does not violate the initial placement invariants and so the legs of the caterpillars remain non-crossing throughout the algorithm.

Since each leg (in C_1 or C_2) contributes to at most one shifting, the size of the grid required is $(n + k_1) \times (n + k_2)$, where $(k_1 + k_2) < 2n$, thus yielding the desired result. □

3.2 Outerplanar Graphs

Simultaneous embedding of outerplanar graphs is not always possible.

Theorem 7. *There exist two outerplanar graphs which, given a mapping between the vertices of the graphs, cannot be simultaneously embedded.*

Proof: The two outerplanar graphs O_1, O_2 are as shown in Figure 6. The union of O_1, and O_2 contains $K_{3,3}$ as a subgraph, which means that when embedded simultaneously the edges of the two graphs contain at least one intersection. Assume O_1 and O_2 can be simultaneously embedded. Then the crossing in the union of the two graphs must be between an edge of O_1 and an edge of O_2. The edges belonging to O_1 only are 12 and 36. The edges belonging to O_2 only are 23 and 16. However, we can not pick a crossing pair out of these, since each such pairing consists of incident edges which can not cross. Thus there must be another pair either in O_1 or in O_2 which intersects. □

4 Simultaneous Embedding without Mapping

In this section we present methods to embed different classes of planar graphs simultaneously when no mapping between the vertices are provided. For the remainder of this section, when we say simultaneous embeddings we always mean without vertex mappings. This additional freedom to choose the vertex mapping does make a great difference. For example, any number of paths or cycles can be simultaneously embedded. Indeed, in this setting of simultaneous embedding without vertex mappings we do not have any non-embeddability result; it is perhaps the most interesting open question whether any two planar graphs can be simultaneously embedded. We do have a positive answer if all but one of the planar graphs are outerplanar.

Theorem 8. *A planar graph G_1 and any number of outerplanar graphs $G_2, \ldots, G_r$, each with n vertices, can be simultaneously embedded (without mapping) on an $O(n^2) \times O(n^2)$ grid.*

Theorem 9. *Any number of outerplanar graphs can be simultaneously embedded (without mapping) on an $O(n) \times O(n)$ grid.*

Key to the proof of both theorems is the construction of grid subsets in general position, since it is known that any outerplanar graph can be embedded on any point set in general position (no three points collinear):

Theorem 10. *[3,14] Given a set P of n points in the plane, no three of which are collinear, an outerplanar graph H with n vertices can be straight-line embedded on P.*

These embeddings can even be found efficiently. Gritzmann *et al* [14] provide an embedding algorithm for such graphs that runs in $O(n^2)$ time, and Bose [3] further reduces the running time to $O(n \lg^3 n)$.

Theorem 9 then follows from the existence of sets of n points in general position in an $O(n) \times O(n)$ grid. But this is an old result by Erdös [12]: choose the minimum prime number p greater than n (there is a prime between n and $(1+\varepsilon)n$ for $n > n_0(\varepsilon)$), then the points $(t, t^2 \bmod p)$ for $t = 1, \ldots, p$ are a set of $p \geq n$ points in the $p \times p$-grid with no three points collinear. So we can choose the required points in a $(1+\varepsilon)n \times (1+\varepsilon)n$-grid. The smallest grid size in which one can choose n points in general position is known as the 'no-three-in-line'-problem; the only lower bound is $\frac{1}{2}n \times \frac{1}{2}n$, below that there are already three points in the same row or column.

In order to prove Theorem 8, we must embed an arbitrary planar graph, G_1, in addition to the outerplanar graphs; unlike outerplanar graphs, we cannot embed G_1 on any point set in general position. Thus, we begin by embedding G_1 in an $O(n) \times O(n)$ grid using the algorithm of [6]. The algorithm draws any 3-connected planar graph in an $O(n) \times O(n)$ grid under the *edge resolution rule*, and produces a drawing of that graph with the special property that for each vertex and each edge not incident with this vertex, the distance between the vertex

and the edge in the embedding is at least one grid unit. This embedding may still contain many collinear vertices; we resolve this in the next step. We again choose the smallest prime $p \geq n$, and blow up the whole drawing by a factor of $2p$, mapping a previous vertex at (i, j) to the new location $(2pi, 2pj)$. In this blown-up drawing, the distance between a vertex and a non-incident edge is at least $2p$. Now let v_1v_2 be an edge in that drawing, w a vertex not incident to that edge, and let v_1', v_2', w' be arbitrary grid points from the small $p \times p$-grids centered at v_1, v_2, w. Then the distance of v_1', v_2', w' to v_1, v_2, w is at most $\frac{1}{\sqrt{2}}p$, so the distance of w' to the segment $v_1'v_2'$ is at least $(2 - \frac{2}{\sqrt{2}})p > 0$. Thus, any perturbation of the blown-up drawing, in which each vertex v is replaced by some point v' from the $p \times p$-grid centered at v, will still have the same combinatorial structure, and still be a valid plane drawing. We now choose a special such perturbation to obtain a general-position set: If the vertex v_ν was mapped by the algorithm of [6] on the point (i, j), then we map it on the point $(2pi + (\nu \bmod p), 2pj + (\nu^2 \bmod p))$. This new embedding is still a correct embedding for the planar graph, since all vertices have still sufficient distance from all non-incident edges. Further, it is a general-position point set, suitable for the embedding of outerplanar graphs, since by a reduction modulo p the points are mapped on the general-position point set $\{(\nu, \nu^2 \bmod p) : \nu = 1, \ldots, n\}$, and collinearity is a property that is preserved by the mod p-reduction of the coordinates. So we have embedded the planar graph in an $O(n^2) \times O(n^2)$ grid, on a point set in general position, on which now all outerplanar graphs can also be embedded. This completes the proof of Theorem 8.

5 Open Problems

- Can 2 lobster graphs[1] or 2 trees be simultaneously embedded with mapping? We have answered affirmatively for the special case of 2 caterpillars.
- Given a general planar graph G, and a path P with two or more vertices, can we always simultaneously embed with mapping G and P ?
- While, in general, it is not always possible to simultaneously embed (with mapping) two arbitrary planar graphs, can we test in polynomial time whether two particular graphs can be embedded for a given mapping?
- Can any two planar graphs be simultaneously embedded without mapping?

Acknowledgments. We would like to thank Ed Scheinerman for stimulating discussions about different variations of the problem and Esther M. Arkin for her proof of Theorem 3 (independent of our work).

References

1. M. Bern and J. R. Gilbert. Drawing the planar dual. *Information Processing Letters*, 43(1):7–13, Aug. 1992.

[1] A lobster graph is a tree such that the graph obtained by deleting the leaves is a caterpillar.

2. F. Bernhart and P. C. Kainen. The book thickness of a graph. *J. Combin. Theory*, Ser. B 27:320–331, 1979.
3. P. Bose. On embedding an outer-planar graph in a point set. *CGTA: Computational Geometry: Theory and Applications*, 23(3):303–312, 2002.
4. G. R. Brightwell and E. R. Scheinerman. Representations of planar graphs. *SIAM Journal on Discrete Mathematics*, 6(2):214–229, May 1993.
5. E. Cenek. *Layered and Stratified Graphs*. PhD thesis, University of Waterloo, forthcoming.
6. M. Chrobak, M. T. Goodrich, and R. Tamassia. Convex drawings of graphs in two and three dimensions. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 319–328, 1996.
7. M. Chrobak and G. Kant. Convex grid drawings of 3-connected planar graphs. *Intl. Journal of Computational Geometry and Applications*, 7(3):211–223, 1997.
8. C. Collberg, S. G. Kobourov, J. Nagra, J. Pitts, and K. Wampler. A system for graph-based visualization of the evolution of software. In *1st ACM Symposium on Software Visualization*. To appear in 2003.
9. H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.
10. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, Englewood Cliffs, NJ, 1999.
11. M. B. Dillencourt, D. Eppstein, and D. S. Hirschberg. Geometric thickness of complete graphs. *Journal of Graph Algorithms and Applications*, 4(3):5–17, 2000.
12. P. Erdös. Appendix. In K. F. Roth. On a problem of Heilbronn. *J. London Math. Soc.*, 26:198–204, 1951.
13. C. Erten and S. G. Kobourov. Simultaneous embedding of a planar graph and its dual on the grid. In *13th Intl. Symp. on Algorithms and Computation (ISAAC)*, pages 575–587, 2002.
14. P. Gritzmann, B. Mohar, J. Pach, and R. Pollack. Embedding a planar triangulation with vertices at specified points. *American Math. Monthly*, 98:165–166, 1991.
15. M. Kaufmann and D. Wagner. *Drawing graphs: methods and models*, volume 2025 of *Lecture Notes in Computer Science*. Springer-Verlag Inc., New York, NY, USA, 2001.
16. P. Koebe. Kontaktprobleme der konformen Abbildung. *Berichte "uber die Verhandlungen der Sächsischen Akademie der Wissenschaften zu Leipzig. Math.-Phys. Klasse*, 88:141–164, 1936.
17. K. Miura, S.-I. Nakano, and T. Nishizeki. Grid drawings of 4-connected plane graphs. *Discrete and Computational Geometry*, 26(1):73–87, 2001.
18. P. Mutzel, T. Odenthal, and M. Scharbrodt. The thickness of graphs: a survey. *Graphs Combin.*, 14(1):59–73, 1998.
19. W. Schnyder. Planar graphs and poset dimension. *Order*, 5(4):323–343, 1989.
20. W. T. Tutte. How to draw a graph. *Proc. London Math. Society*, 13(52):743–768, 1963.
21. M. Yannakakis. Embedding planar graphs in four pages. *Journal of Computer and System Sciences*, 38(1):36–67, Feb. 1989.

Smoothed Analysis
Motivation and Discrete Models*

Daniel A. Spielman[1] and Shang-Hua Teng[2]

[1] Department of Mathematics, Massachusetts Institute of Technology
[2] Department of Computer Science, Boston University

Abstract. In smoothed analysis, one measures the complexity of algorithms assuming that their inputs are subject to small amounts of random noise. In an earlier work (Spielman and Teng, 2001), we introduced this analysis to explain the good practical behavior of the simplex algorithm. In this paper, we provide further motivation for the smoothed analysis of algorithms, and develop models of noise suitable for analyzing the behavior of discrete algorithms. We then consider the smoothed complexities of testing some simple graph properties in these models.

1 Introduction

We believe that the goals of research in the design and analysis of algorithms must be to develop theories of algorithms that explain how algorithms behave and that enable the construction of better and more useful algorithms. A fundamental step in the development of a theory that meets these goals is to understand why algorithms that work well in practice actually do work well. From a mathematical standpoint, the term "in practice" presents difficulty, as it is rarely well-defined. However, it is a difficulty we must overcome; a successful theory of algorithms must exploit models of the inputs encountered in practice. We propose using smoothed analysis to model a characteristic of inputs common in many problem domains: inputs are formed in processes subject to chance, randomness, and arbitrary decisions. Moreover, we believe that analyses that exploit this characteristic can provide significant insight into the behavior of algorithms. As such analyses will be difficult, and will therefore be instinctively avoided by many researchers, we first argue the necessity of resting analyses on models of inputs to algorithms.

Researchers typically avoid the need to model the inputs to algorithms by performing worst-case analyses. By providing an analysis that does not depend upon the inputs, worst-case analysis provides an incredibly strong guarantee, and it is probably one of the greatest achievement of the theoretical computer science community. However, worst-case analysis provides only one statistic about an algorithm's behavior. In many situations, and especially those in which algorithms are used, it is more important to understand the typical behavior of

* The first author was supported in part by NSF grant CCR-0112487, and the second author was supported in part by NSF grant 99-72532

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 256–270, 2003.
© Springer-Verlag Berlin Heidelberg 2003

an algorithm. Moreover, the typical behavior of an algorithm is often quite different from its worst-case behavior. If the mention of the ill-defined "typical" causes a mathematical mind run to the comfort of the cleanly defined worst-case analysis, it is understandable. It is not even clear that one should try to use mathematics to understand a notion such as "typical behavior", and it is clear that experiments must also play a role. However, the results of experiments are best understood in the context of an abstract theory. Experiments can confirm or contradict a theory; but, mathematically describable theories provide the most desirable encapsulations of knowledge about algorithms. It remains to be seen whether these theories will be mathematically rigorous, reason by analogy with mathematically rigorous statements, or combine theorems with heuristic mathematical arguments as is common in the field of Physical Applied Mathematics.

In smoothed analysis, we exploit the low-order random events influencing the formation of the inputs to algorithms. These influences have many sources including measurement error, constraints imposed by economics or management, and the chain of chance leading to the consideration of any particular situation. Consider, for example, the design of a bridge that may be input to an algorithm. Design constraints are imposed by the surface under the bridge, and the locations of the roadways available to connect the bridge at either edge. A governmental committee will provide a probability distribution over architects, and a given architect will choose different designs in different periods of her career. These designs will then be altered as local politicians push contracts to favored constituents, etc.

By examining different levels of the design process, one can obtain complexity measures varying from average case to worst case. If one just views the entire process as providing one distribution on bridges, then one obtains an average-case complexity measure. If one merely considers the finished bridge, and maximizes over the possible bridges, then one obtains a worst-case complexity measure. By considering the probability distribution after certain choices have been made, and taking a maximum over those choices, one obtains a model between the average-case and worst-case complexity measures.

Of course, we cannot hope to define a mathematical model that precisely captures any of these influences or that captures the levels of refinement of the actual process. But, we can try to define models that capture their spirit and then reason by analogy. Our first attempt [ST01] was to model these influences by subjecting inputs to perturbations. In this model we defined the smoothed complexity of an algorithm to be the maximum over its inputs of its expected running time over random perturbations of those inputs. This running time should be measured in terms of the input length and the magnitude of the perturbations. By varying the magnitudes of the perturbations, we smoothly generate complexity measures between the worst-case and average-case.

However, a model in which inputs are perturbed at random may be unnatural for some problems, and it might be necessary to place some constraints upon the perturbations by insisting that they respect some divisions of the input space. For example, it might be necessary that the bridge be able to support a 20-ton

truck (or SUV), and we should not allow perturbations of the bridge that violate this constraint to enter our probability space. In general, perturbations should probably be restricted to preserve the most significant aspects of an input for a given situation. For example, a natural perturbation of a graph is obtained by adding edges between unconnected vertices and removing edges with some probability. However, a graph subject to such perturbations is highly unlikely to have a large clique, and so it may be meaningless to measure the performance of algorithms for clique under this model. We propose to avoid this problem by studying *property-preserving perturbations*, which we define by restricting a natural perturbation model to preserve certain properties of the input. For example, one could imagine perturbing a graph subject to preserving the size of its largest clique.

We remark that a notion such as property-preserving perturbations is necessary even in average-case analysis. For example, if one desires an average-case analysis of algorithms for max-clique, one should state the running times of the algorithms as functions of the size of the max-clique. Otherwise, the probability mass is concentrated on the graphs without large cliques, and for which the problem is much less interesting. One should not be distracted by the fact that it may be computationally difficult to sample from the resulting conditional distributions under which we must measure the complexity of our algorithms.

Of course, one should not just preserve only the property being calculated by the algorithm: it is natural to require that the perturbations preserve all the most relevant properties of the input. For example, when studying algorithms for minimum bisection, one might consider genus- and bisection-size-preserving graph perturbations. We note that the complexity measure of an algorithm under perturbations that preserve more properties properties is strictly closer to worst-case complexity that a measure under perturbations that preserve a subset of the properties.

1.1 A Mathematical Introduction

In our analysis of the simplex method [ST01], we exploited the most natural model of perturbation for real-number inputs—that of Gaussian random perturbations. This model has also been applied in the smoothed analysis of the Perceptron Algorithm by Blum and Dunagan [BD02], of Interior Point Methods by Spielman and Teng [ST03] and Dunagan, Spielman and Teng [DST02]. For a survey of some of these works, we refer the reader to [ST02]. It has been suggested by many that these analyses could be made to have a tighter analogy with practice if the perturbations preserved more properties of their input. For example, it would be reasonable to restrict perturbations to preserve feasibility, infeasibility, or even the condition number of the programs. It is also natural to restrict the perturbations so that zero entries remain zero.

In this paper, we will mainly concern ourselves with discrete problems, in which the natural models of perturbations are not nearly as clear. For graphs, the most natural model of perturbation is probably that obtained by XORing

the adjacency matrix with the adjacency matrix of a random sparse graph. This model is captured by the following definition:

Definition 1. *Let $\bar{G}$ be a graph and $\sigma > 0$. We define the σ-perturbation of $\bar{G}$ to be the graph obtained by converting every edge of $\bar{G}$ into a non-edge with probability σ and every non-edge into an edge with probability σ. We denote this distribution on graphs by $\mathcal{P}(\bar{G}, \sigma)$.*

Unfortunately, there are many purposes for which such perturbations can radically change an input, rendering the model meaningless. For example, it would be pointless to study algorithms testing whether a graph is bipartite or has a ρn-clique under this model because it is highly unlikely that the σ-perturbation of any graph will have either of these properties.

Property preserving perturbations provide a modification of this model in which this study becomes meaningful. Given a property P, and a notion of perturbation, we define a P-preserving perturbation of an object $\bar{X}$ to be a perturbation X of $\bar{X}$ sampled subject to the condition $P(\bar{X}) = P(X)$. For example, if $\bar{G}$ is a graph and G is a P-preserving σ-perturbation of $\bar{G}$, then G has density

$$\frac{\mathbf{Pr}_{G \leftarrow \mathcal{P}(\bar{G},\sigma)} \left[G \text{ and } (P(\bar{G}) = P(G)) \right]}{\mathbf{Pr}_{G \leftarrow \mathcal{P}(\bar{G},\sigma)} \left[P(\bar{G}) = P(G) \right]}.$$

We can then say that an algorithm A has smoothed error probability δ under P-preserving σ-perturbations if

$$\max_{\bar{G}} \mathbf{Pr}_{G \leftarrow \mathcal{P}(\bar{G},\sigma)} \left[A(G) \text{ is incorrect} | P(G) = P(\bar{G}) \right] \leq \delta.$$

Property preserving perturbations are a special case of function preserving perturbations in which the function is binary valued.

Definition 2. *Let f be a function defined on the space of graphs, let $\bar{G}$ be a graph and $\sigma > 0$. We define the f-preserving σ-perturbation of $\bar{G}$ to be the random graph G with density:*

$$\frac{\mathbf{Pr}_{G \leftarrow \mathcal{P}(\bar{G},\sigma)} \left[G \text{ and } (f(\bar{G}) = f(G)) \right]}{\mathbf{Pr}_{G \leftarrow \mathcal{P}(\bar{G},\sigma)} \left[f(\bar{G}) = f(G) \right]}.$$

This function could represent many qualities of a graph. In addition to properties, f could measure numerical quantities such as diameter or conductance. In such cases, it might be more reasonable to merely require the perturbed graph to approximately preserve f.

In the remainder of this paper, we will derive some elementary results on the complexity of graph properties under perturbations that preserve these properties. In particular, we will measure the smoothed error probability of sub-linear time algorithms for these problems. In this sense, we consider a problem closely related to that studied in the field of property testing. In property testing, one

measures the worst-case complexity of Monte Carlo algorithms solving a promise problem of the form: determine whether or not an input has a property given that the input either has the property or is far from those inputs that have the property. For many property testing problems, we find that under perturbations that preserve the same property, the input typically satisfies such a guarantee. Conversely, if one cannot construct a notion of property-preserving perturbations under which inputs typically satisfy such a guarantee, then we feel one should probably not assume such a guarantee is satisfied in practice.

In the following sections, we obtain some simple results on the complexity of testing if graphs have small cliques, bisections, or are bipartite under property-preserving perturbations. We hope stronger results will be obtained by considering perturbations that preserve even more properties of their inputs.

1.2 Comparison with the Semi-random Model

Another approach to interpolating between worst-case and average-case complexity appears in a line of work initiated by Blum and Spencer [BS95]. Blum and Spencer considered the problem of k-coloring k-colorable graphs generated by choosing a random k-colorable graph and allowing an adversary to add edges between color classes. Feige and Kilian [FK98a] extended their results and considered analogous models for finding large cliques and optimal bisections. For the clique problem, a large clique is planted in a random graph, and an adversary is allowed to remove edges outside the clique. Their model for bisection modifies Boppana's model of a random graph with a planted bisection [Bop87] by allowing an adversary to add edges not crossing the bisection and remove edges crossing the bisection. It is easy to show that these models are stronger than the analogous models in which an adversary constructs a graph with a large clique or small bisection and these graphs are then perturbed in a way that preserves the embedded clique or bisection. In Section 3, we show that the graphs produced by ρ-Clique preserving σ-perturbations are close to the graphs produced by this later model, and that we can use the algorithm for Feige and Kilian to produce a fast testing algorithm for these properties.

In contrast, the planted bisection model considered by Feige and Kilian seems to produce rather different graphs than the ρ-Bisection preserving σ-perturbations, and we cannot find a way to use their algorithm to test for small bisections in this model, let alone speed up a tester. The difference is that a ρ-Bisection preserving σ-perturbation may produce a graph with many small bisections of almost exactly the same size, while the model considered by Feige and Kilian produces graphs in which the smallest bisection is significantly smaller than all competitors.

Other work in similar models includes the analysis by Feige and Krauthgamer [FK98b] for bandwidth minimization algorithms and Coja-Oghlan [CO02] for finding sparse induced subgraphs.

1.3 Property Testing

Rubinfeld and Sudan [RS96] defined property testing to be a relaxation of the standard decision problem: rather than designing an algorithm to distinguish between inputs that have and do not have a property, one designs an algorithm to distinguish between those that have and those that are far from having a property. Under this relaxation, many properties can be tested by sub-linear time algorithms that examine random portions of their input. In this paper, we will examine the testers designed by Goldreich, Goldwasser and Ron [GGR98].

Goldreich, Goldwasser and Ron [GGR98] introduced the testing of graph properties. Their results included the development of testers that distinguished between graphs that are bipartite, have size ρn cliques, and size ρn bisections from those graphs that have distance ϵ to those with these properties, where distance is measured by the Hamming distance of adjacency matrices.

Formally speaking, an algorithm A is said to be a property tester for the property P if

1. for all x with property P, $\mathbf{Pr}\left[A(x,\epsilon)=1\right] \geq 2/3$; and
2. for all x of distance at least ϵ from every instance that has property P, $\mathbf{Pr}\left[A(x,\epsilon)=1\right] \leq 1/3$,

under some appropriate measure of distance on inputs (although some testers have one-sided error). A typical property testing algorithm will use a randomized process to choose a small number of facets of x to examine, and then make its decision. For example, a property tester for a graph property may query whether or not certain edges exist in the graph. The quality of a property testing algorithm is measured by its query complexity (the number of queries to the input) and its time complexity.

Since the seminal works of Rubinfeld and Sudan [RS96] and Goldreich, Goldwasser, and Ron [GGR98], property testing has become a very active area of research in which many different types of properties have been examined [GR97, GR98,KR00,Alo01,ADPR00,AKFS99,BR00,GGLR98,Ron01,EKK+98] [PR99,DGL+99,CSZ00,BM98,BM99,CS02,GT01]. In this work, we will restrict our attention to graph properties and geometric properties of point sets.

Following Goldreich, Goldwasser, and Ron [GGR98], we measure the distance between graphs by the Hamming distance between their adjacency matrices. That is, the distance between two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ on n vertices is defined as the fraction of edges on which G_1 and G_2 differ: $|E_1 \cup E_2 - E_1 \cap E_2| / \binom{n}{2}$. The properties considered in [GGR98] include Bipartite, the property of being bipartite; ρ-Clique, the property of having a clique of size at least ρn; and ρ-Bisection, the property of having a bisection crossed by fewer than ρn^2 edges. For these properties, they prove:

Theorem 1 (Goldreich-Goldwasser-Ron). *The properties ρ-Clique and ρ-Bisection have property testing algorithms with query complexity polynomial in $1/\epsilon$ and time complexity $2^{\tilde{O}(1/\epsilon^3)}$, and the property Bipartite has a property testing algorithm with query and time complexities polynomial in $1/\epsilon$.*

We remark that Goldreich and Trevisan [GT01] have shown that every graph property that can be tested by making a number of queries that is independent of the size of the graph, can also be tested by uniformly selecting a subset of vertices and accepting if and only if the induced subgraph has some fixed graph property (which is not necessarily the same as the one being tested).

We now state a lemma that relates the smoothed error probability of a testing algorithm with the probability that the property-preserving perturbation of an input is far from one having the property.

Lemma 1. *Let P be a property and A a testing algorithm for P with query complexity $q(1/\epsilon)$ and time complexity $T(1/\epsilon)$ such that*

$$\mathbf{Pr}\left[A(X) \neq P(X)\right] < 1/3,$$

for all inputs X that either have property P or have distance at least ϵ from those having property P. Then, if $\mathcal{P}(\bar{X}, \sigma)$ is a family of distributions such that for all $\bar{X}$ lacking property P,

$$\mathbf{Pr}_{X \leftarrow \mathcal{P}(\bar{X},\sigma)}\left[X \text{ is } \epsilon\text{-close to } P | P(X) = P(\bar{X})\right] \leq \lambda(\epsilon, \sigma, n),$$

then for all inputs $\bar{X}$,

$$\mathbf{Pr}_{X \leftarrow \mathcal{P}(\bar{X},\sigma)}\left[A(X) \neq P(X) | P(X) = P(\bar{X})\right] < 1/3 + \lambda(\epsilon, \sigma, n).$$

2 Smoothed Error Bound for Graph Property Testers

In this section, we prove that the ρ-Clique, ρ-Bisection and Bipartite property testers of [GGR98] may be viewed as sub-linear-time decision algorithms with low smoothed error probability under the corresponding property-preserving perturbations.

Lemma 2. *Let $\bar{G}$ be a graph on n vertices, let $\rho < 1/8$, and let $\sigma < 1/2$. If G is the ρ-Bisection preserving σ-perturbation of $\bar{G}$, then*

1. *if $\bar{G}$ has a ρ-Bisection, then G has a ρ-Bisection with probability 1, and*
2. *if $\bar{G}$ does not have a ρ-Bisection, then for any $\epsilon < \sigma(1/4 - 2\rho)$*

$$\mathbf{Pr}_{\mathcal{P}(\bar{G},\sigma)}\left[\begin{array}{r} G \text{ is } \epsilon\text{-close to a graph with a } \rho\text{-Bisection} \\ \mid G \text{ does not have a } \rho\text{-Bisection} \end{array}\right] < 2^{-\Omega(n^2)}.$$

Proof. The first part follows from the definition of a ρ-Bisection preserving perturbation.

To prove the second part, we first observe that G is ϵ-close to a graph with a ρ-Bisection if and only if G has a $(\rho + \epsilon)$-Bisection. We express the probability of this event in the property-preserving model as

$$\mathbf{Pr}_{\mathcal{P}(\bar{G},\sigma)}\left[G \text{ has a } (\rho+\epsilon)\text{-Bisection} \middle| G \text{ does not have a } \rho\text{-Bisection}\right] \leq \frac{\mathbf{Pr}_{\mathcal{P}(\bar{G},\sigma)}\left[G \text{ has a } (\rho+\epsilon)\text{-Bisection}\right]}{\mathbf{Pr}_{\mathcal{P}(\bar{G},\sigma)}\left[G \text{ does not have a } \rho\text{-Bisection}\right]}. \quad (1)$$

We now proceed to bound these probabilities. If we flip every edge and non-edge of G with probability σ, then for every partition of the vertices of $\bar{G}$ into two equal-sized sets the expected number of edges crossing this partition in G is at least

$$(1-\sigma)\rho n^2 + \sigma(1/4-\rho)n^2.$$

Applying a Chernoff bound (see for example [MR97, Theorem 4.2]), we find the probability that there are fewer than $(\rho+\epsilon)n^2$ edges crossing this partition is at most

$$e^{-n^2 \frac{(\sigma(1/4-2\rho)-\epsilon)^2)}{\rho+\sigma(1/4-2\rho)}} = 2^{-\Omega(n^2)}.$$

As there are fewer than 2^n partitions, we may plug this inequality into (1) to conclude the proof.

The proofs of the following two lemmas for Bipartite and Clique are similar.

Lemma 3. *Let $\bar{G}$ be a graph of n vertices. If $\epsilon > 0$ and $\epsilon/\rho^2 < \sigma < 1/2$, and if G is the ρ-Clique preserving σ-perturbation of $\bar{G}$, then*

1. *if $\bar{G}$ is has a ρ-Clique, then G has a ρ-Clique with probability 1, and*
2. *if $\bar{G}$ does not have a ρ-Clique, then for any $\epsilon < \sigma(1/4-2\rho)$*

$$\mathbf{Pr}\left[\begin{array}{r} G \text{ is } \epsilon\text{-close to a graph with a } \rho\text{-Clique} \\ \mid G \text{ does not have a } \rho\text{-Clique} \end{array}\right] < 2^{-\Omega(n^2)}.$$

Lemma 4. *Let $\bar{G}$ be a graph of n vertices and let $0 < \epsilon < \sigma/4 < 1/8$. If G is the bipartite-preserving σ-perturbation of $\bar{G}$, then*

1. *if $\bar{G}$ is bipartite, then G is bipartite with probability 1, and*
2. *if $\bar{G}$ is not bipartite, then*

$$\mathbf{Pr}\left[G \text{ is } \epsilon\text{-close to bipartite} | G \text{ is not bipartite}\right] < 2^{-\Omega(n^2)}.$$

Remark 1. Bipartite and Clique differ from Bisection in this model as their natural testers have simple proofs of correctness in the smoothed model. In contrast, we are unaware of a means of proving the correctness of the Bisection tester that does not go through the machinery of [GGR98]. This seems to be related to the fact that we can find exponentially faster testers for Clique in this model.

Using Lemma 1 to combine Theorem 1 with Lemmas 2, 3 and 4, we obtain:

Theorem 2. *Let P be one of Bipartite, ρ-Clique, or ρ-Bisection. There exists an algorithm A that takes as input a graph G, examines poly$(1/\sigma)$ edges of G and runs in time $\tilde{O}(1/\epsilon^3)$ when P is Bipartite, and in $2^{\tilde{O}(1/\epsilon^2)}$ time when P is ρ-Clique or ρ-Bisection such that for every $\bar{G}$, if G is the P-property preserving σ-perturbation of $\bar{G}$, then*

$$\mathbf{Pr}\left[A(G) \neq P(G)\right] < 1/3 + o(1).$$

In the next section, we improve the time complexity of ρ-Clique testing under ρ-Clique preserving σ-perturbations.

3 A Fast Clique Tester

In this section we will consider a tester for ρ-Clique that samples a random set of k vertices and accepts if these vertices contain a $\rho k/2$ clique. In Lemma 5 we prove that this tester rarely accepts a graph without a ρ-Clique under ρ-Clique preserving σ-perturbations. The other lemmas of the section are devoted to adapting the machinery of Feige and Killian [FK98a] to quickly finding the $\rho k/2$ clique when it is present in the graph.

Theorem 3 (Fast Clique Tester). *Let ρ and $\sigma < 1/2$ be constants. There exists an algorithm A that takes as input a graph G, examines the induced subgraph of G on a randomly chosen set of $\frac{8}{\rho\sigma}\log\left(\frac{4}{\rho\sigma}\right)$ vertices of G and runs in time polynomial in $\frac{1}{\rho\sigma}$ such that for every graph $\bar{G}$, if G is the ρ-Clique preserving σ-perturbation of $\bar{G}$, then*

$$\mathbf{Pr}\left[A(G) \neq \rho\text{-}Clique(G)\right] < 1/4 + o(1).$$

In contrast, Goldreich, Goldwasser and Ron [GGR98] prove that the existence of a tester with such worst-case complexity would imply $NP \subseteq BPP$.

Proof. The algorithm A runs the algorithm of Lemma 8 below and accepts if it finds a clique of size at least $\rho k/2$. If $\bar{G}$ does not contain a ρ-Clique, then by Lemma 5 below the probability this algorithm will accept is at most $e^{-8} + o(1) \leq 1/4 + o(1)$.

On the other hand, if $\bar{G}$ does contain a ρ-Clique, We can apply Lemma 8 to show that

$$\begin{aligned}\mathbf{Pr}_{\mathcal{Q}(\bar{G},\sigma)}\left[A(G) \text{ rejects}\right] &= \sum_S w(S)\mathbf{Pr}_{\mathcal{Q}(\bar{G},S,\sigma)}\left[A(G) \text{ rejects}\right] \\ &\leq \sum_S w(S)(1/4 + o(1)) \leq 1/4 + o(1).\end{aligned}$$

The theorem then follows from Lemma 9 below which implies

$$|\mathbf{Pr}_{\mathcal{P}(\bar{G},\sigma)}\left[A(G) \text{ accepts}|\rho\text{-Clique}(G)\right] - \mathbf{Pr}_{\mathcal{Q}(\bar{G},\sigma)}\left[A(G) \text{ accepts}\right]| < o(1).$$

The next lemma states that the tester is unlikely to accept if G does not contain a ρ-Clique.

Lemma 5. *Let $\bar{G}$ be a graph without a ρ-Clique and let G be the ρ-Clique preserving σ-perturbation of G. Let U be a randomly chosen subset of k vertices of G for $k \geq \frac{8}{\rho\sigma}\log\left(\frac{4}{\rho\sigma}\right)$. Then,*

$$\mathbf{Pr}\left[\text{the vertices of } U \text{ contain a } \rho k/2 \text{ clique in } G\right] < e^{-8} + o(1).$$

Proof. We begin by observing that

$$\begin{aligned}
&\mathbf{Pr}_{U,G\leftarrow\mathcal{P}(\bar{G},\sigma)}\left[\begin{array}{r}\text{the vertices of } U \text{ contain a } \rho k/2 \text{ clique in } G\\ \mid G \text{ does not contain a } \rho n \text{ clique}\end{array}\right]\\
&\leq \frac{\mathbf{Pr}_{U,G\leftarrow\mathcal{P}(\bar{G},\sigma)}\left[\text{the vertices of } U \text{ contain a } \rho k/2 \text{ clique in } G\right]}{1-\mathbf{Pr}_{G\leftarrow\mathcal{P}(\bar{G},\sigma)}\left[G \text{ contains a } \rho n \text{ clique}\right]}\\
&\leq \mathbf{Pr}_{U,G\leftarrow\mathcal{P}(\bar{G},\sigma)}\left[\text{the vertices of } U \text{ contain a } \rho k/2 \text{ clique in } G\right]+o(1),
\end{aligned}$$

by Lemma 6.

To bound the last probability, we note that the probability that any particular set of $\rho k/2$ nodes in G is a clique is at most $(1-\sigma)^{\binom{\rho k/2}{2}}$ and that U contains $\binom{k}{\rho k/2}$ sets of $\rho k/2$ nodes, so

$$\begin{aligned}
\mathbf{Pr}_{U,G\leftarrow\mathcal{P}(\bar{G},\sigma)}\left[\begin{array}{l}\text{the vertices of } U \text{ contain}\\ \text{a } \rho k/2 \text{ clique in } G\end{array}\right] &\leq \binom{k}{\rho k/2}(1-\sigma)^{\binom{\rho k/2}{2}}\\
&\leq \left(\frac{2e}{\rho}\right)^{\rho k/2} e^{-\sigma\binom{\rho k/2}{2}}\\
&\leq e^{\frac{\rho k}{2}\left(\ln\left(\frac{2e}{\rho}\right)-\sigma\left(\frac{\rho k-2}{4}\right)\right)}\\
&\leq e^{-\rho k}\leq e^{-8}.
\end{aligned}$$

as $k \geq \frac{8}{\rho\sigma}\log\left(\frac{4}{\rho\sigma}\right)$ and $\sigma < 1$.

Lemma 6. *Let $\bar{G}$ be a graph without a ρn-Clique and let G be the σ-perturbation of G. Then,*

$$\mathbf{Pr}_{G\leftarrow\mathcal{P}(\bar{G},\sigma)}\left[G \textit{ contains a } \rho\textit{-Clique}\right] = 2^{-\Omega(n^2)}.$$

Proof. There are fewer than 2^n sets of ρn nodes, and the probability that any particular such set is a clique in G is at most $(1-\sigma)^{\binom{\rho n}{2}}$.

Lemma 7. *Let $\bar{G}$ be a graph that has a ρ-Clique. Then,*

$$\mathbf{Pr}_{G\leftarrow\mathcal{P}(\bar{G},\sigma)}\left[G \textit{ has at least two } \rho\textit{-Cliques}|G \textit{ has one } \rho\textit{-Clique}\right] \leq 2^{-\Omega(n)}.$$

Proof. By inclusion-exclusion,

$$\begin{aligned}
&\mathbf{Pr}\left[G \text{ has one } \rho\text{-Clique}\right]\\
&\qquad\geq \sum_{|S_1|=\rho n}\mathbf{Pr}\left[K_{S_1}\subseteq G\right] - \sum_{|S_1|=|S_{2|}=\rho n}\mathbf{Pr}\left[K_{S_1}\subseteq G \text{ and } K_{S_2}\subseteq G\right],
\end{aligned}$$

and

$$\mathbf{Pr}\left[G \text{ has at least two } \rho\text{-Cliques}\right] \geq \sum_{|S_1|=|S_{2|}=\rho n}\mathbf{Pr}\left[K_{S_1}\subseteq G \text{ and } K_{S_2}\subseteq G\right].$$

Therefore,

$$\begin{aligned}
&\mathbf{Pr}\left[G \text{ has at least two } \rho\text{-Cliques} | G \text{ has one } \rho\text{-Clique}\right] \\
&\leq \frac{\sum_{|S_1|=|S_2|=\rho n} \mathbf{Pr}\left[K_{S_1} \subseteq G \text{ and } K_{S_2} \subseteq G\right]}{\sum_{|S_1|=\rho n} \mathbf{Pr}\left[K_{S_1} \subseteq G\right] - \sum_{|S_1|=|S_2|=\rho n} \mathbf{Pr}\left[K_{S_1} \subseteq G \text{ and } K_{S_2} \subseteq G\right]} \\
&\leq \max_{|S_1|=\rho n} \frac{\sum_{|S_2|=\rho n} \mathbf{Pr}\left[K_{S_1} \subseteq G \text{ and } K_{S_2} \subseteq G\right]}{\mathbf{Pr}\left[K_{S_1} \subseteq G\right] - \sum_{|S_2|=\rho n} \mathbf{Pr}\left[K_{S_1} \subseteq G \text{ and } K_{S_2} \subseteq G\right]}
\end{aligned}$$

We now prove the lemma by demonstrating that for all $|S_1| = \rho n$,

$$\begin{aligned}
&\frac{\sum_{|S_2|=\rho n} \mathbf{Pr}\left[K_{S_1} \subseteq G \text{ and } K_{S_2} \subseteq G\right]}{\mathbf{Pr}\left[K_{S_1} \subseteq G\right]} \\
&= \sum_{k=1}^{\rho n} \sum_{|U|=|V|=k} \frac{\mathbf{Pr}\left[K_{S_1} \subseteq G \text{ and} K_{S_1 \setminus U \cup V} \subseteq G\right]}{\mathbf{Pr}\left[K_{S_1} \subseteq G\right]} \\
&\leq \sum_{k=1}^{\rho n} \binom{\rho n}{k} \binom{n-\rho n}{k} (1-\sigma)^{k(\rho n - k) + \binom{k}{2}} \\
&= 2^{-\Omega(n)},
\end{aligned}$$

where the last inequality follows from the fact that $k(\rho n - k) + \binom{k}{2}$ is an increasing function in k, and for $k \leq \rho n/2$, the terms in the sum decrease as k increases. In addition, when $k = \rho n/2$, $(1-\sigma)^{k(\rho n-k)+\binom{k}{2}} = 2^{-\Omega(n^2)}$. Therefore, the first term in the sum dominates, and hence the sum is no more than $2^{-\Omega(n)}$.

Feige and Kilian [FK98a] design a polynomial-time algorithm for finding cliques in random graphs with planted cliques which may be modified in a limited fashion by an adversary. A corollary of their work is that if one takes a graph with a large clique and then perturbs the edges not involved in the clique, then with high probability their algorithm will find the large clique. To facilitate the rigorous statement of this corollary and the application of their result to the smoothed model, we introduce the following notation:

Definition 3. *For a graph $\bar{G}$, a subset of its vertices S and σ between 0 and $1/2$, we define $\mathcal{Q}(\bar{G}, S, \sigma)$ to be the distribution on graphs obtained by sampling from $\mathcal{P}(\bar{G}, \sigma)$ and adding edges to create a clique among the nodes in S.*

For a graph $\bar{G}$ and a σ between 0 and $1/2$, we define $\mathcal{Q}(\bar{G}, \sigma)$ to be the distribution obtained by choosing a set S of vertices of size ρn with probability $w(S)$ and then sampling from $\mathcal{Q}(\bar{G}, S, \sigma)$ where

$$w(S) = \frac{\mu(S)}{\sum_{T:|T|=|S|} \mu(T)},$$

and

$$\mu(S) = \prod_{i,j} \sigma^{[(i,j) \notin \bar{G}]} (1-\sigma)^{[(i,j) \in \bar{G}]}.$$

Theorem 4 (Feige-Kilian). *For any positive constant ρ, there is a randomized polynomial time algorithm that with probability $1 - o(1)$ will find a clique of size ρn in a graph G drawn from the distribution $\mathcal{Q}(\bar{G}, S, \sigma)$ where S is a subset of the vertices of $\bar{G}$ of size ρn and $\sigma \geq 2 \ln n / \rho n$.*

From this theorem, we derive

Lemma 8. *Let $\rho > 0$ and let G be drawn from the distribution $\mathcal{Q}(\bar{G}, S, \sigma)$ where S is a subset of the vertices of $\bar{G}$ of size ρn and $1/2 \geq \sigma \geq 2 \ln n / \rho n$. Let U be a random subset of k vertices of G where $k = \min\left(k_0, \frac{8}{\rho\sigma} \log\left(\frac{4}{\rho\sigma}\right)\right)$, where k_0 is some absolute constant. Then, with probability $3/4 - o(1)$ the algorithm of Theorem 4 finds a clique of size at least $\rho k/2$ in the graph induced by G on U.*

Proof. We first note that the probability that U contains fewer than $\rho k/2$ vertices of S is at most

$$e^{-\rho k/8} + o(1) \leq e^{-3} + o(1)$$

as $\log\left(\frac{4}{\rho\sigma}\right) \geq 3$ and $\rho, \sigma < 1$.

Given that there are at least $\rho k/2$ points of S in U, the probability that the algorithm of Theorem 4 fails is at most $1/8$, provided that $\sigma > 2 \log k/(\rho k/2)$, which follows from our setting of $k \geq \frac{8}{\rho\sigma} \log\left(\frac{4}{\rho\sigma}\right)$, and that k is larger than some absolute constant, k_0. Thus, the failure probability is at most $e^{-3} + 1/8 + o(1) \leq 1/4 + o(1)$.

To transfer the result of Lemma 8 to graphs produced by ρ-Clique preserving σ-perturbations of graphs with ρ-Cliques, we show:

Lemma 9. *Let $\bar{G}$ be a graph with a ρ-Clique and $\sigma < 1/2$. Then,*

$$\sum_G |\mathbf{Pr}_{\mathcal{P}(\bar{G},\sigma)}[G | G \text{ has a } \rho\text{-Clique}] - \mathbf{Pr}_{\mathcal{Q}(\bar{G},\sigma)}[G]| < 2^{-\Omega(n)}.$$

Proof. For any graph G, we apply inclusion-exclusion to compute

$$\begin{aligned}
\frac{\mathbf{Pr}_{\mathcal{P}(\bar{G},\sigma)}[G]}{\sum_{S:|S|=\rho n} \mu(S)} &\leq \mathbf{Pr}_{\mathcal{P}(\bar{G},\sigma)}[G | G \text{ contains a } \rho n\text{-Clique}] \\
&\leq \frac{\mathbf{Pr}_{\mathcal{P}(\bar{G},\sigma)}[G]}{\sum_{S:|S|=\rho n} \mu(S) - \sum_{|S_1|=|S_2|=\rho n} \mathbf{Pr}[K_{S_1} \subseteq G \text{ and} K_{S_2} \subseteq G]} \\
&\leq \frac{\mathbf{Pr}_{\mathcal{P}(\bar{G},\sigma)}[G]}{\sum_{S:|S|=\rho n} \mu(S)} \left(1 + 2^{-\Omega(n)}\right),
\end{aligned}$$

by Lemma 7.

On the other hand,

$$\begin{aligned}\mathbf{Pr}_{\mathcal{Q}(\bar{G},\sigma)}[G] &= \sum_{S:K_S\subseteq G,|S|=\rho n} \frac{\mu(S)}{\sum_{|T|=\rho n}\mu(T)}\mathbf{Pr}[G|K_S\subseteq G] \\ &= \sum_{S:K_S\subseteq G,|S|=\rho n} \frac{\mathbf{Pr}_{\mathcal{P}(\bar{G},\sigma)}[G]}{\sum_{|T|=\rho n}\mu(T)} \\ &= (\#\ \rho\text{-Cliques in } G)\,\frac{\mathbf{Pr}_{\mathcal{P}(\bar{G},\sigma)}[G]}{\sum_{|T|=\rho n}\mu(T)}.\end{aligned}$$

We now conclude the proof by observing that if G has no ρn cliques then both probabilities are zero, if G has one ρn clique then the probabilities differ by at most a multiplicative factor of $(1+2^{-\Omega(n)})$, and, by Lemma 7, the probability under $\mathcal{P}(\bar{G},\sigma)$ that there are two ρn cliques is at most $2^{-\Omega(n)}$.

4 Discussion

4.1 Condition Numbers and Instance-Based Complexity

To obtain a finer analysis of algorithms for a problem than that provided by worst-case complexity, one should find a way of distinguishing hard problem instances from easy ones. A natural approach is to find a quantity that may be associated with a problem instance and which is indicative of the difficulty of solving that instance. For example, it is common in Numerical Analysis and Operations Research to bound the running time of an algorithm in terms of a condition number of its input. The condition number is typically defined to be the the reciprocal of the distance of the input to one on which the problem is ill-posed, or the sensitivity of the solution of a problem to slight perturbations of the input.

Thus, one can view the effort to measure the complexity of testing whether or not an input has a property in terms of its distance from having the property if it does not as being very similar. In fact, the perturbation distance used by Czumaj and Sohler [CS01] is precisely the the reciprocal of the condition number of the problem. Moreover, the natural definition of the condition number for a discrete function—the reciprocal of the minimum distance of an input to one on which the function has a different value—is precisely the measure of complexity used in the study of property testing: the larger the condition number the harder the testing.

In fact, in many smoothed analyses [BD02,DST02,ST03], an essential step has been the smoothed analysis of a condition number.

References

[ADPR00] N. Alon, S. Dar, M. Parnas, and D. Ron. Testing of clustering. In *41st Annual Symposium on Foundations of Computer Science*, pages 240–250. IEEE, 2000.

[AKFS99] N. Alon, M. Krivelevich, E. Fischer, and M. Szegedy. Efficient testing of large graphs. In *40th Annual Symposium on Foundations of Computer Science,*, pages 656–666. IEEE, 1999.

[Alo01] N. Alon. Testing subgraphs in large graphs. In IEEE, editor, *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science: proceedings*, pages 434–439, 2001.

[BD02] Avrim Blum and John Dunagan. Smoothed analysis of the perceptron algorithm for linear programming. In *Proceedings of the 13th Annual ACM-SIAM Symposium On Discrete Mathematics (SODA-02)*, pages 905–914. ACM Press, 2002.

[BM98] P. Bose and P. Morin. Testing the quality of manufactured disks and cylinders. In *ISAAC: 9th International Symposium on Algorithms and Computation*, volume 1533 of *Lecture Notes in Computer Science*, pages 129–138, 1998.

[BM99] Prosenjit Bose and Pat Morin. Testing the quality of manufactured balls. In *Workshop on Algorithms and Data Structures*, pages 145–156, 1999.

[Bop87] Ravi Boppana. Eigenvalues and graph bisection: An average-case analysis. In *Proceedings of the 28th Symposium on Foundation of Computer Science*, pages 280–285, 1987.

[BR00] Michael A. Bender and Dana Ron. Testing acyclicity of directed graphs in sublinear time. In *Automata, Languages and Programming*, pages 809–820, 2000.

[BS95] Avrim Blum and Joel Spencer. Coloring random and semi-random k-colorable graphs. *J. Algorithms*, 19(2):204–234, 1995.

[CO02] Amin Coja-Oghlan. Finding sparse induced subgraphs of semirandom graphs. In *Randomization and Approximation Techniques in Computer Science, 2002*, volume 2483 of *Lecture Notes in Computer Science*, pages 139–148. Springer, 2002.

[CS01] A. Czumaj and C. Sohler. Property testing with geometric queries. In *Proceedings of the 9th Annual European Symposium on Algorithms*, volume 2161 of *Lecture Notes in Computer Science*, pages 266–277. Springer-Verlag, 2001.

[CS02] A. Czumaj and C. Sohler. Abstract combinatorial programs and efficient property testers. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 83–92, 2002.

[CSZ00] Artur Czumaj, Christian Sohler, and Martin Ziegler. Property testing in computational geometry. In *European Symposium on Algorithms*, pages 155–166, 2000.

[DGL+99] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky. Improved testing algorithms for monotonocity. In *Proceedings of RANDOM*, pages 97–108, 1999.

[DST02] John Dunagan, Daniel A. Spielman, and Shang-Hua Teng. Smoothed analysis of interior point methods: Condition numbers. available at `http://arxiv.org/abs/cs.DS/0302011`, 2002.

[EKK+98] Funda Ergun, Sampath Kannan, S. Ravi Kumar, Ronitt Rubinfeld, and Mahesh Viswanathan. Spot-checkers. In *ACM Symposium on Theory of Computing*, pages 259–268, 1998.

[FK98a] U. Feige and J. Kilian. Heuristics for finding large independent sets, with applications to coloring semi-random graphs. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 674–683. IEEE, 1998.

[FK98b] Uri Feige and Robert Krauthgamer. Improved performance guarantees for bandwidth minimization heuristics. Unpublished manuscript, 1998.

[GGLR98] Oded Goldreich, Shafi Goldwasser, Eric Lehman, and Dana Ron. Testing monotonicity. In *IEEE Symposium on Foundations of Computer Science*, pages 426–435, 1998.

[GGR98] Oded Goldreich, Shari Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, July 1998.

[GR97] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 406–415, 1997.

[GR98] Oded Goldreich and Dana Ron. A sublinear bipartiteness tester for bounded degree graphs. In ACM, editor, *Proceedings of the thirtieth annual ACM Symposium on Theory of Computing*, pages 289–298. ACM Press, 1998.

[GT01] O. Goldreich and L. Trevisan. Three theorems regarding testing graph properties. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 460–469, 2001.

[KR00] Michael Kearns and Dana Ron. Testing problems with sublearning sample complexity. *J. of Comput. Syst. Sci.*, 61(3):428–456, December 2000.

[MR97] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1997.

[PR99] Michal Parnas and Dana Ron. Testing the diameter of graphs. In *Random Structures and Algorithms*, volume 1671 of *Lecture Notes in Computer Science*, pages 85–96, 1999.

[Ron01] D. Ron. Property testing. In *Handbook on Randomized Computing (Vol. II)*. Kluwer Academic Publishers, 2001.

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, April 1996.

[ST01] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing (STOC '01)*, pages 296–305, 2001.

[ST02] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms. In *Proceedings of the International Congress of Mathematicians*, volume 1, 2002. to appear.

[ST03] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of termination of linear programming algorithms. *Mathematical Programming B*, 2003. to appear.

Approximation Algorithm for Hotlink Assignments in Web Directories

Rachel Matichin and David Peleg*

The Weizmann Institute of Science,
Rehovot, 76100 Israel
{rachelm,peleg}@wisdom.weizmann.ac.il

Abstract. Hotlink assignment concerns the addition of shortcut links to information structures based on linked nodes such as the web. Each node in the structure is associated with a weight representing the frequency that node is accessed by users. To access a node, the user must follow the path leading to it from the root. Introducing additional edges (hotlinks) to the structure may reduce its access cost, taken to be the expected number of steps needed to reach a node from the root. The *hotlink assignment* problem is to find a set of hotlinks achieving the greatest improvement in the access cost. This paper introduces an approximation algorithm for this problem with approximation ratio 2.

1 Introduction

1.1 Motivation

A common approach towards organizing large databases containing diverse information types is based on a hierarchical index to the database according to some classification into categories. Such organizations for the Web, for example, are provided in Yahoo [5] and the Open Directory Service [6]. A user searching for some information item in a hierarchically structured database must traverse a path from the root to the desired node in the classification tree. Typically, the degree of this tree is rather low and subsequently its average depth is high. Moreover, the classification does not take into account the "popularity" of various items, which dictates their access probability by users. This implies that the depth of certain popular items in the classification tree may be high, while certain "unpopular" items may have short access paths. Hence its access cost, taken to be the expected number of steps needed to reach an item from the root, may be high.

As partial remedy, often used in the Web, the tree organization is augmented by "*hotlinks*" added to various nodes of the tree, which lead directly to the most popular items. The selection of the hotlinks to be added should be based on the statistics of visited items in the database. This paper concerns the optimization problem of constructing a set of hotlinks that achieves a maximum improvement in the access cost.

* Supported in part by a grant from the Israel Science Foundation.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 271–280, 2003.
© Springer-Verlag Berlin Heidelberg 2003

More formally, given a rooted directed graph G with root r, representing the database, a *hotlink* is a directed edge that does not belong to the graph. The hotlink *starts* at some node v and *ends* at (or *leads* to) some node u that is a descendant of v. At most one such hotlink is added to each node v. Each node x of G has a weight $\omega(x)$, representing its access frequency, or the proportion of the users visits to that node, compared with the total number of users visits to all nodes. Hence if normalized, $\omega(x)$ can be interpreted as the probability that a user wants to access node x.

The *hotlink assignment* problem, introduced in [1], is formally defined as follows. Denote by $d(G,u)$ the distance from the root r to the leaf u in G. Let $\mathcal{L}(G)$ denote the set of leaves of G. The expected number of steps to reach a leaf in the graph is

$$\mathbb{E}[G,\omega] \;=\; \sum_{u\in\mathcal{L}(G)} d(G,u)\omega(u)\ .$$

Given a graph G and a set of hotlinks S added to the graph, denote by $G\oplus S$ the resulting hotlink-assigned graph. Define the *gain* of the hotlink assignment S on the graph G as

$$g(S,G,\omega) \;=\; \mathbb{E}[G,\omega]-\mathbb{E}[G\oplus S,\omega] = \sum_{u\in\mathcal{L}(G)} \Big(d(G,u)-d(G\oplus S,u)\Big)\omega(u)\ .$$

HOTLINK ASSIGNMENT
Instance: A directed graph $G = (V,E)$, a root node $r \in V$ that can reach every node of the graph, and a positive integer h.
Question: Is there a hotlink assignment S for which $g(S,G,\omega) \geq h$?

Denote by S^* the optimal set of hotlinks, yielding the maximum gain over all possible hotlink sets, i.e., $S^* = argmax_S\{g(S)\}$. Our interest is in the optimization version of the problem, namely, constructing a hotlink assignment S achieving maximum gain $g(S)$.

1.2 Related Work

Most past discussions on the problem concentrated on directed acyclic graphs (DAG's). Moreover, it was assumed that information items with positive probability weights reside only at the leaves of the DAG.

The NP-completeness of the hotlink assignment problem on DAGS is proven in [1] by a reduction from the problem of Exact Cover by 3-Sets. That article also discusses several distribution functions on the leaves, including the uniform, geometric and Zipf distributions, but restricts the discussion to full binary trees. An interesting analogy is presented therein between the hotlink assignment problem on trees and coding theory. A classification tree can be interpreted as a coding of words (associating a move down to the ith child with the letter 'i'). Under this interpretation, every leaf corresponds to a codeword. The addition of a hotlink adds a letter to the alphabet. This provides a lower bound for the problem based

on Shannon's theorem. In particular, in binary trees we have the following. Consider a binary tree T, denote by $H(\omega)$ the entropy of the access distribution on the leaves, and consider a hotlink assignment S. Then

$$\mathbb{E}[T \oplus S, \omega] \;\geq\; \frac{1}{\log 3} \cdot H(\omega) \;=\; \frac{-1}{\log 3} \sum_{u \in \mathcal{L}(G)} \omega(u) \log \omega(u) \;,$$

and in trees of maximal degree Δ,

$$\mathbb{E}[T \oplus S, \omega] \;\geq\; \frac{H(\omega)}{\log(\Delta + 1)} \;.$$

Based on these bounds, an approximation algorithm for a slightly different variant of the hotlink assignment problem on bounded degree trees is presented in [2]. This algorithm approximates not the gain but the access cost $\mathbb{E}[T \oplus S, \omega]$. The access cost guaranteed by this algorithm is no more than $\frac{H(\omega)}{\log(\Delta+1) - (\Delta \log \Delta)/(\Delta+1)} + \frac{\Delta+1}{\Delta}$, hence the approximation ratio achieved by the algorithm is $\frac{\log(\Delta+1)}{\log(\Delta+1) - (\Delta \log \Delta)/(\Delta+1)} + \frac{(\Delta+1)\log(\Delta+1)}{\Delta \cdot H(\omega)}$, which is in general at least $\log(\Delta + 1)$.

A slightly different approach to the problem is studied in [3]. The underlying assumption is that the user has limited a-priori knowledge regarding the structure of the classification tree. Therefore, the user cannot always identify the shortest path to its desired destination. Instead, the user makes "greedy" decisions in every step along the search. The paper proposes a polynomial time algorithm for solving the hotlink assignment problem in that model on trees of logarithmic depth. The solution is also generalized to situations where more than one hotlink per node is allowed. For the case in which the distribution on the leaves is unknown, the paper gives an algorithm guaranteeing (an optimal) logarithmic upper bound on the access cost.

Another recent article [4] discusses an interesting application of hotlink assignments in asymmetric communication protocols for achieving better performance bounds.

1.3 Our Results

We first present a polynomial time algorithm for approximating the hotlink assignment problem on rooted connected directed acyclic graphs. The algorithm uses greedy choices at each iteration, and achieves an approximation ratio of 2. In contrast with [2], our algorithm approximates the achievable gain and not the access cost.

We also show how to generalize the solution to hotlink assignment schemes which:

1. for given function K, allow up to $K(v)$ hotlinks per node,
2. assign positive probability weights to items residing in all the nodes of the graph, and not only the leaves, and
3. allow the graph to have cycles.

The previous approximation algorithm for bounded degree trees presented in [2] cannot be directly compared with the approximation achieved by our greedy algorithm, due to the difference in the optimization parameters. In particular, one can construct instances for which our algorithm will outperform that of [2] and vice versa.

2 Preliminaries

The *gain* of a hotlink set S_2 *relative to* S_1 is defined as the additional gain of S_2 after S_1 has already been added, i.e.,

$$\tilde{g}(S_2 \mid S_1, G, \omega) \;=\; g(S_2, G \oplus S_1, \omega) \;.$$

Let us first establish some basic properties of g and $\tilde{g}$.

Lemma 1. *$d(u,G) \geq d(u,G')$ whenever $E(G) \subseteq E(G')$.*

Proof: For any leaf u in the graph, any path from r to u in G exists also in G', thus the shortest path to u in G exists in G'. ∎

Lemma 2. *$\tilde{g}(S_2 \mid S_1, G, \omega) \;=\; g(S_1 \cup S_2, G, \omega) - g(S_1, G, \omega)$.*

Proof: By the definition of gain,

$$\begin{aligned}
\tilde{g}(S_2 \mid S_1, G, \omega) &= g(S_2, G \oplus S_1, \omega) \\
&= \sum_{u \in \mathcal{L}(G)} \Big(d(G \oplus S_1, u) - d((G \oplus S_1) \oplus S_2, u) \Big) \omega(u) \\
&= \sum_{u \in \mathcal{L}(G)} \big(d(G \oplus S_1, u) - d(G, u) \big) \omega(u) \\
&\qquad + \big(d(G, u) - d((G \oplus S_1) \oplus S_2, u) \big) \omega(u) \\
&= -g(S_1, G, \omega) + g(S_1 \cup S_2, G, \omega). \qquad ∎
\end{aligned}$$

Lemma 3. *$g(S_2 \cup S_1, G, \omega) \leq g(S_1, G, \omega) + g(S_2, G, \omega)$.*

Proof: For $i = 1, 2$, denote by g_i the gain achieved on the path to u by adding the set of hotlinks S_i and let $S_i(u)$ denote the particular subset of S_i used in the improved path to u. Then for any leaf u, the maximum possible gain achievable by the hotlink assignment $S_1 \cup S_2$ is $g_1 + g_2$, by using both the edges $S_1(u) \subseteq S_1$ and $S_2(u) \subseteq S_2$ (we get the exact sum if the union of those two sets of hotlinks are distinct and moreover can be used simultaneously in path leading from the root r to the leaf u). Thus, for every leaf u,

$$d(G,u) - d(G \oplus (S_1 \cup S_2), u) \;\leq\; \big(d(G,u) - d(G \oplus S_1, u) \big) + \big(d(G,u) - d(G \oplus S_2, u) \big)$$

and hence

$$
\begin{aligned}
g(S_2 \cup S_1, G, \omega) &= \sum_{u \in \mathcal{L}(G)} \Big(d(G,u) - d(G \oplus (S_1 \cup S_2, u)) \Big) \omega(u) \\
&\leq \sum_{u \in \mathcal{L}(G)} \Big(\big(d(G,u) - d(G \oplus S_1, u) \big) \\
&\qquad + \big(d(G,u) - d(G \oplus S_2, u) \big) \Big) \omega(u) \\
&= g(S_1, G, \omega) + g(S_2, G, \omega). \qquad \blacksquare
\end{aligned}
$$

Corollary 1. $\tilde{g}(S_2 \mid S_1, G, \omega) \leq g(S_2, G, \omega)$.

3 The Approximation Algorithm

Algorithm A operates as follows.

1. Order the vertices arbitrarily as $z_1, z_2, ...z_n$.
2. Set $G_0 \leftarrow G$.
3. **For** $i = 1$ to n **do:**
   a) Choose for z_i a hotlink assignment to some descendant w in a greedy manner, namely, to w satisfying $g(\langle z_i, w \rangle, G_{i-1}) \geq g(\langle z_i, w' \rangle, G_{i-1})$ for every descendant w' of z_i.
   b) Set $G_i \leftarrow G_{i-1} \oplus \{\langle z_i, w \rangle\}$.

Note that the greedy choice at iteration i of step 3 selects the hotlink which at the *current* state (combined with the hotlinks that the algorithm has already chosen at iterations $1, 2, ..., i-1$) minimizes the expected number of steps to reach the leaves at this point (disregarding the influence it might have on the hotlinks that have not yet been assigned).

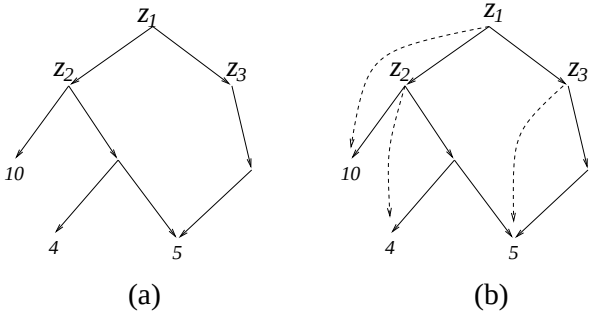


Fig. 1. The optimal hotlink assignment S^*.

An example is presented in Figures 1 and 2. The initial graph is presented in Figure 1(a). The optimal choice of hotlinks for the graph, S^*, is presented in

Figure 1(b). This optimal assignment of links achieves a gain of $g(S^*) = 10 * 1 + 5 * 1 + 4 * 1 = 19$. Figure 2 describes the hotlink assignment S_A produced during the execution of Algorithm A. The three nodes z_1, z_2, z_3 are ordered according to the algorithm's choice. In the first iteration the algorithm chooses an optimal hotlink assignment for z_1 given the original graph in Figure 1(a). In fact, in this situation there are two equal possibilities for this choice, so assume the hotlink chosen is as in Figure 2(a). On the second iteration the algorithm chooses a hotlink for z_2. Here, there is only one optimal choice, as seen in Figure 2(b). After that, no additional hotlink (from z_3 or from any other node) can yield positive gain, so the algorithm terminates with a final assignment as in Figure 2(b). Thus given this ordering, the total gain achieved by the algorithm is $g(S_A, G, \omega) = 10 * 1 + 5 * 1 = 15$.

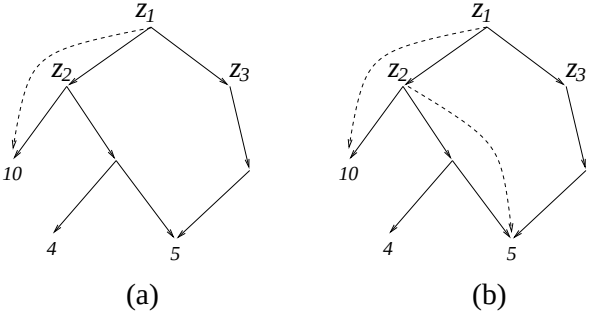


Fig. 2. The algorithm's hotlink assignment S_A.

4 Analysis

Number the vertices in the order chosen by the algorithm, $\langle z_1, z_2, ..., z_n \rangle$. After step i of the algorithm, the first i vertices have already been assigned hotlinks. Denote this set of hotlinks by

$$S_i^A = \langle L_A(z_1), L_A(z_2), ..., L_A(z_i) \rangle \ .$$

Denote the optimal choice of hotlinks by $S^* = \langle L^*(z_1), L^*(z_2), ..., L^*(z_n) \rangle$. Also denote by S_i^* the set of hotlinks *not* including the first i nodes,

$$S_i^* = \langle L^*(z_{i+1}), L^*(z_{i+2}), ..., L^*(z_n) \rangle.$$

Finally, for the ratio proof we have to consider hybrid sets composed of some greedily chosen hotlinks and some optimal ones. Denote the union of two such sets by

$$H_i = S_i^* \cup S_i^A \ .$$

Note that H_i is a complete assignment of hotlinks for the entire graph, i.e., it contains exactly one hotlink for every node in the graph. The sequence

$(H_0, H_1, H_2...)$ captures the process performed by the greedy algorithm, by viewing it as if it starts with the optimal assignment S^* and gradually replaces the optimal hotlinks for $\langle z_1, z_2, ...\rangle$ by the greedy choices. This clearly degrades the quality of the hotlinks, yet our analysis will show that the resulting degradation in gain is not to drastic.

For simplicity, since neither the graph or ω are changed during the process of the algorithm, we denote $d(G \oplus S, u)$ as simply $d(S, u)$ and $g(S, G, \omega)$ as simply $g(S)$. Also relative gain $\tilde{g}(S_1 \mid S_2, G, \omega)$ is denoted by $\tilde{g}(S_1 \mid S_2)$ for short.

We need to argue about the effectiveness of the choices made by the greedy Algorithm A. It is clear that at any iteration i the algorithm takes the currently best hotlink, i.e., the one attaining $\max_L\{\ \tilde{g}(L \mid S_i^A)\ \}$ over all possible hotlinks L. The proof will compare the algorithm assignment versus the optimal one which achieves gain $g(S^*)$. The following two lammas bound the decrease in gain incurred by moving from H_i to H_{i+1}, namely, by replacing the optimal link $L^*(z_{i+1})$ with the greedy link $L_A(z_{i+1})$, and show that this decrease is no greater than what was gained by the link $L_A(z_{i+1})$ in that iteration.

Lemma 4. $\tilde{g}(S^*_{i+1} \mid S_i^A \cup L^*(z_{i+1})) \leq \tilde{g}(S^*_{i+1} \mid S_i^A \cup L_A(z_{i+1})) + \tilde{g}(L_A(z_{i+1}) \mid S_i^A)$.

Proof: Letting $A = \tilde{g}(S^*_{i+1} \mid S_i^A \cup L^*(z_{i+1}))$ and $B = \tilde{g}(S^*_{i+1} \mid S_i^A \cup L_A(z_{i+1}))$, we have to prove that $A - B \leq \tilde{g}(L_A(z_{i+1}) \mid S_i^A)$. By the definition of gain,

$$A \leq \tilde{g}(S^*_{i+1} \mid S_i^A) \ = \ \sum_{u \in \mathcal{L}(G)} \big(d(S_i^A, u) - d(S^*_{i+1} \cup S_i^A, u)\big) \cdot \omega(u) \ ,$$

$$B = \sum_{u \in \mathcal{L}(G)} \big(d(S_i^A \cup L_A(z_{i+1}), u) - d(S^*_{i+1} \cup S_i^A \cup L_A(z_{i+1}), u)\big) \cdot \omega(u)$$

thus we can write,

$$\begin{aligned} A - B &\leq \sum_{u \in \mathcal{L}(G)} \Big(\big(d(S_i^A, u) - d(S_i^A \cup L_A(z_{i+1}), u)\big) \\ &\quad + \big(d(S^*_{i+1} \cup S_i^A \cup L_A(z_{i+1}), u) - d(S^*_{i+1} \cup S_i^A, u)\big) \Big) \omega(u) \\ &\leq \sum_{u \in \mathcal{L}(G)} \big(d(S_i^A, u) - d(S_i^A \cup L_A(z_{i+1}), u)\big)\omega(u) = \tilde{g}(L_A(z_{i+1}) \mid S_i^A), \end{aligned}$$

where the last inequality is due to the fact that by Lemma 1, for any node u,

$$d(S^*_{i+1} \cup S_i^A \cup L_A(z_{i+1}), u) - d(S^*_{i+1} \cup S_i^A, u) \ \leq \ 0. \qquad \blacksquare$$

Lemma 5. $g(H_i) \ \leq \ g(H_{i+1}) + \tilde{g}(L_A(z_{i+1}) \mid S_i^A)$.

Proof: writing $g(H_i)$ and $g(H_{i+1})$ as

$$\begin{aligned} g(H_i) &= g(S_i^A) + \tilde{g}(S_i^* \mid S_i^A), \\ g(H_{i+1}) &= g(S_i^A) + \tilde{g}(L_A(z_{i+1}) \cup S^*_{i+1} \mid S_i^A), \end{aligned}$$

we need to prove that

$$\tilde{g}(S_i^* \mid S_i^A) \;\leq\; \tilde{g}(L_A(z_{i+1}) \cup S_{i+1}^* \mid S_i^A) + \tilde{g}(L_A(z_{i+1}) \mid S_i^A)\ . \tag{1}$$

Note that

$$\tilde{g}(S_i^* \mid S_i^A) \;=\; \tilde{g}(L^*(z_{i+1}) \mid S_i^A) + \tilde{g}(S_{i+1}^* \mid S_i^A \cup L^*(z_{i+1}))\ . \tag{2}$$

and

$$\tilde{g}(L_A(z_{i+1}) \cup S_{i+1}^* \mid S_i^A) \;=\; \tilde{g}(L_A(z_{i+1}) \mid S_i^A) + \tilde{g}(S_{i+1}^* \mid S_i^A \cup L_A(z_{i+1}))\ . \tag{3}$$

Since the link $L_A(z_{i+1})$ was selected by the algorithm as a local optimum,

$$\tilde{g}(L^*(z_{i+1}) \mid S_i^A) \;\leq\; \tilde{g}(L_A(z_{i+1}) \mid S_i^A)\ . \tag{4}$$

Combining equations (1), (2), (3) and (4) with Lemma 4 the claim follows. ▮

Lemma 6. *Algorithm A has an approximation ratio 2, namely, $g(S_A) \;\geq\; g(S^*)/2$.*

Proof: Note that $S^* = H_0$ and $S_A = H_n$. Summing up the inequalities of Lemma 5 for $0 \leq i \leq n-1$ we get

$$g(H_0) \;\leq\; g(H_n) + \sum_{i=0}^{n-1} \tilde{g}(L_A(z_{i+1}) | S_i^A) \;=\; 2 \cdot g(H_n),$$

or, $g(S^*) \leq 2 \cdot g(S_A)$. ▮

Let us next turn to analyzing the time complexity of the algorithm. The algorithm performs n iterations. At each iteration it chooses the best hotlink out of n possible hotlinks at most. The computation of the gain achieved by a single hotlink is polynomial, and thus the time complexity of the entire algorithm is polynomial in n.

To establish the tightness of our analysis for the approximation ratio of Algorithm A, we prove the following lemma.

Lemma 7. *For any $\epsilon > 0$ there exists a graph G and an ordering of the vertices such that the gain of the hotlink assignment S_A returned by Algorithm A is $2-\epsilon$ times smaller than the optimal gain, namely, $g(S_A) \leq g(S^*)/(2-\epsilon)$.*

Proof: Given $\epsilon > 0$, let $d = \lceil 1/\epsilon \rceil$, and construct a graph G with two leaves x and y of weights $\omega(x) = 1/(d+1)$ and $\omega(y) = d/(d+1)$ as in Figure 3. In this graph, the optimal solution S^* is to assign a hotlink from z_1 to y and from z_2 to x, and the resulting optimal gain is $g(S^*) = \omega(y) + (d-1) \cdot \omega(x) = (2d-1)/(d+1)$. However, assuming that the ordering selected for the vertices starts with z_1, z_2 as in the figure, the assignment S_A chosen by the algorithm will consist of a single hotlink leading from z_1 to x, yielding a gain of $g(S_A) = d \cdot \omega(x) = d/(d+1)$. The ratio between the two results is thus $g(S^*)/g(S_A) = (2d-1)/d \geq 2-\epsilon$, and the claim follows. ▮

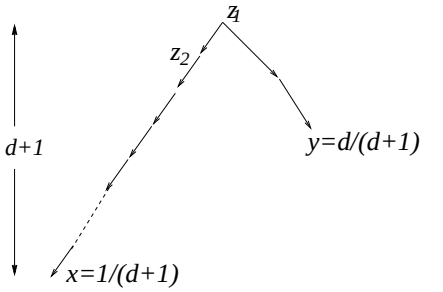


Fig. 3. Construction of the graph G.

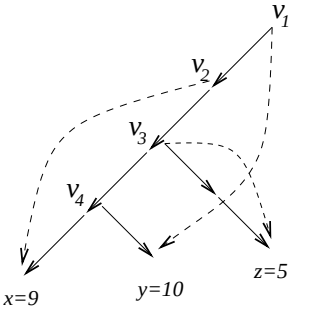


Fig. 4. An optimal hotlink assignment.

5 Generalizations

We conclude with a number of generalizations to the algorithm, removing some of the previous restrictions. To begin with, so far we assumed that at most one link can start at each node. A more general version of the problem allows us to add a number of hotlinks to each node. The input to this general problem includes also a function $K(v)$ specifying the number of hotlinks allowed to start at each node v.

Our algorithm can be generalized into Algorithm $A[K]$ handling this problem as follows: step 3(a) assigns $K(z_i)$ hotlinks instead of just one (in a greedy manner as before).

Lemma 8. *Algorithm $A[K]$ also has approximation ratio 2.*

Proof: Revise the previous definitions as follows. Denote

$$S_i^A = \langle S_A(z_1), S_A(z_2), ..., S_A(z_i) \rangle,$$

where $S_A(z_i)$ is the set of hotlinks assigned to node z_i by the algorithm and $|S_A(z_i)| = K(z_i)$. In the same manner, denote by $\langle S^*(z_1), S^*(z_2), ..., S^*(z_n) \rangle$ the optimal sets of hotlinks for each node. A proof analogous to the one of the previous section for a single hotlink still applies. In particular, Lemma 4 now

states that $g(S^*_{i+1} \mid S^A_i \cup S^*(z_{i+1})) \leq g(S^*_{i+1} \mid S^A_i \cup S_A(z_{i+1})) + g(S_A(z_{i+1})|S^A_i)$, and Lemma 5 now states that $g(H_i) \leq g(H_{i+1}) + g(S_A(z_{i+1}) \mid S^A_i)$. The remainder of the analysis applies as is. ∎

Secondly, in previously discussed models all the relevant data was stored in the leaves of the DAG. A more general model may allow all the nodes of the graph to hold data, with each node having some nonnegative access probability. Our algorithm applies also to this model, and the analysis goes through in a similar manner (noting that it did not in fact use the assumption that data is stored only in the leaves).

Finally, Algorithm A applies without change to arbitrary rooted directed graphs, and not only DAG's, yielding the same approximation ratio. This is true since the analysis did not use the assumption that the graph G is cycle-free.

6 Graphs with No Good Greedy Ordering

In the graphs given in previous examples, there is a "correct" ordering of the vertices, namely, an ordering $z_1, \ldots, z_n$ that, if used by the greedy algorithm, will result in an optimal solution. If this were true for every graph, namely, if every graph G had a "good" ordering (yielding the optimum gain), then a plausible approach for attacking the problem would be to attempt to find such a "good" or "close-to-good" ordering. Unfortunately, this is not true in the general case, meaning that not every graph has a "correct" order of vertices to ensure that Algorithm A will result with optimal solution. Such an example is given in Figure 4 where only v_1, v_2, v_3 have possible hotlinks to choose from. The optimal hotlink assignment is presented. One can easily observe that both v_2, v_3 must appear after v_1 in a "correct" order, as in any other case, the one of them placed before v_1 will choose a hotlink to y. It is also clear that if v_1 appears first in the ordering, then it is more profitable to choose a hotlink from it to v_4 than to y. Hence under any possible ordering, the greedy algorithm will fail to yield optimal gain.

References

1. Bose, P., Czywizowicz, J., Gasieniec, L., Kranakis, E., Krizanc, D., Pelc, A., and Martin, M. V., Strategies for hotlink assignments. *Proc. 11th Symp. on algorithms and computation (ISAAC 2000)*, pp. 23–34.
2. Kranakis, E., Krizanc, D., and Shende, S., Approximating hotlink assignments, *Proc. 12th Symp. on algorithms and computation (ISSAC 2001)*, pp. 756–767.
3. Gerstel, O., Kutten, S., Matichin, R., Peleg, D., Hotlink Enhancement Algorithms for Web Directories, Unpublished manuscript.
4. Bose, P., Krizanc, D., Langerman, S. and Morin, P., Asymmetric communication protocols via hotlink assignments, *Proc. 9th Colloq. on Structural Information and Communication Complexity*, June 2002, pp. 33–39.
5. `http://www.yahoo.com/`.
6. `http://www.dmoz.org/`.

Drawing Graphs with Large Vertices and Thick Edges

Gill Barequet[1], Michael T. Goodrich[2], and Chris Riley[3]

[1] Center for Graphics and Geometric Computing,
Dept. of Computer Science,
The Technion—Israel Institute of Technology,
Haifa 32000, Israel, barequet@cs.technion.ac.il
[2] Dept. of Information and Computer Science,
Univ. of California, Irvine, CA 92697,
goodrich@ics.uci.edu
[3] Center for Algorithm Engineering,
Dept. of Computer Science,
Johns Hopkins University, Baltimore, MD 21218,
chrisr@cs.jhu.edu

Abstract. We consider the problem of representing size information in the edges and vertices of a planar graph. Such information can be used, for example, to depict a network of computers and information traveling through the network. We present an efficient linear-time algorithm which draws edges and vertices of varying 2-dimensional areas to represent the amount of information flowing through them. The algorithm avoids all occlusions of nodes and edges, while still drawing the graph on a compact integer grid.

1 Introduction

An important goal of information visualization is presenting the information hidden in the structure of a graph to a human viewer in the clearest way possible. Most graph drawing algorithms fulfill this by making visually pleasing drawings that minimize the number of crossings, condense the area, ensure approximately uniform edge lengths, and optimize for many other aesthetics [2]. Without these techniques, the graph may appear "cluttered" and confusing, and difficult to study for a human. But in addition to being aesthetically pleasing, a graph drawing may need to convey additional information beyond connectivity of nodes. Our "graphs" are in reality development processes or computer networks or many, many other things. In the example of a network, it is often useful to know the amount of traffic traveling across each edge and through each node, to visualize such network problems as imbalances or Denial-of-Service attacks. The commonly-used graph-drawing algorithms do not handle this sort of additional information and do not have any method for displaying it.

A simple solution that maintains the current drawing of the graph is labeling each edge (or node) with a number corresponding to the volume of information

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 281–293, 2003.
© Springer-Verlag Berlin Heidelberg 2003

passing through (or being generated by or received by). Although this technically is a display of the information, it is nevertheless not fully using the visual element of the display. For example, a user would need to individually examine each edge and its label just to select the maximum. Therefore, we believe that visualizing traffic in a network requires that we modify the representation of the nodes and edges to best indicate levels of that traffic.

Before we describe our approach, we would like to first mention some trivial approaches that require little modification to current techniques. It would be fairly easy, for example, to simply send animated pulses along an edge with density or rate proportional to the data flow. All we need in this case is space for the pulses to be drawn (since, if edges were too close together, their pulses might be indistinguishable). Nevertheless, this solution doesn't differentiate volume well (as short high-volume edges might get missed), it requires a dynamic display, and it is potentially confusing.

Another approach that requires a few algorithmic modifications is introducing a chromatic variation in the edges, similar to that used by weather forecasters in Doppler radar images. The two possible implementations of this involve using several distinct color levels and a corresponding key (which does not allow for much variation), or a continuous spectrum of colors. But edges in most graph drawing are thin, and it is not easy to compare two different edges in the continuous scale (particularly for those who are color-blind or color-deficient, which includes 8% of all men).

Instead, the approach we advocate is to differentiate between nodes and edges of varying volume by drawing them in varying sizes, possibly augmenting such a display with labels if exact values are needed. This approach is inspired by Minard's classic graphic of the march of Napoleon's army in Russia [16, p. 41][1] (see Figure 1), which geometrically illustrates the army's movements while using edge widths to depict its strength. The benefits of width-based drawings include that they easily separate low- and high-volume nodes and edges, and that they can be depicted on any medium. There is an additional challenge of using width to represent edge and vertex weights, however, in that increasing edge and vertex size introduces the possibility of occlusion of vertices or edges. Such occlusion considerations are not present in other graph drawing problems, which usually consider vertices and edges to be drawn as points and curves, respectively. When we allow vertices and edges to take on significant two-dimensional area, especially if they are large enough to stand out, then they may obscure each other, which is unacceptable. We therefore need algorithms for drawing graphs with wide edges and large vertices that avoid edge and vertex occlusions.

1.1 Standard Approaches and Previous Related Work

One way to avoid occlusions when introducing vertex and edge width is to ensure a sufficiently large edge separation and a bounded angular resolution around vertices. Then, one can scale up the entire drawing and increase the width of

[1] Attributed to E.J. Marey, La Méthode Graphique (Paris, 1885), p. 73.

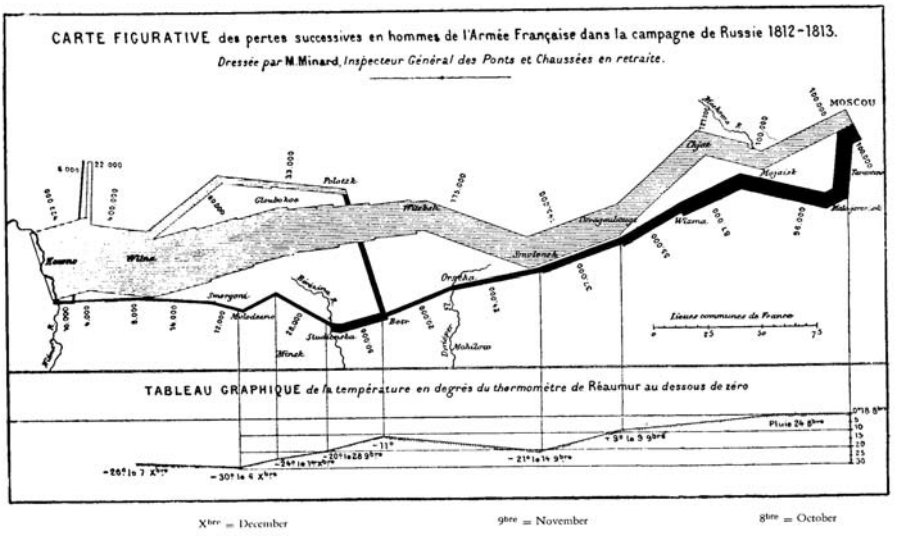


Fig. 1. Image taken from Tufte [16], showing the movements of Napoleon's army in Russia. Edge widths depict army strength, with exact values labeling most edges. Note that this graph has four degree-three vertices and at least 32 edges. Also, two shades are used, with retreating armies shown with solid black edges.

weighted vertices and edges as a proportional fraction of this factor. The easiest approach to perform this scaling is to define a parameter w as the maximum width of any edge, and expand the drawing output from a bounded-angular resolution algorithm to ensure an edge separation of at least $w + 1$. Then edges can be drawn at a weighted proportion of the maximum width w. The problem with this approach is that it produces a drawing with area $\Theta(Aw^2)$, where A is the original (unweighted) drawing area. We would prefer a method without such a quadratic blow-up in area. Note, in addition, that the overall width and height of a drawing made according to this method would be a multiplicative factor of $w + 1$ times the width and height of the drawing with an edge separation of 1. Thus, when such a drawing is compressed to fit on a standard display device, the result would be the same as if we took the original algorithm and simply drew the edges wider within the space already allotted to them (up to a width of $w/(w + 1)$), since it would be compressed $w + 1$ times as much in height and width. Ideally, we would like a weighted graph-drawing algorithm that "shifts" edges and vertices around to make room for edges and vertices of larger widths.

The aesthetics of bounded angular resolution and edge separation have been studied by several researchers (see, e.g., [3,7,9,10,11,12,13,15]). One significant early result is by Malitz and Papakostas [15], which proves that a traditional straight-line drawing of a planar graph with bounded angular resolution can require area exponential in the complexity of the graph. Goodrich and Wagner [11] describe an algorithm for computing a straight-line drawing of a planar graph on

n vertices with at most two bends per edge on an integer grid in $O(n^2)$ area with an asymptotically optimal angular resolution upper bound. An improvement to this, by Cheng et al. [3], reduces the maximum to one bend per edge, but the constants in the area bound increase slightly. Both algorithms are based on a classic algorithm by de Fraysseix, Pach, and Pollack [8], which introduces the "canonical ordering" for drawing vertices of a planar graph used in [11,3] and elsewhere. Their original algorithm produces a planar straight-line drawing of the graph in an $O(n) \times O(n)$ area, but does not bound angular resolution.

A few works dealt with compaction of graphs with vertices of prescribed sizes [1,6,14]. The only work on drawing graphs with "fat" edges, that we are aware of, is that of Duncan et al. [5]. It describes a polynomial-time algorithm for computing, given a graph layout, the thickest possible edges of the graph.

1.2 Our Results

In this paper we give an algorithm to draw a maximally planar graph with a given set of edge traffic amounts. The resulting graph fits in an $O(n + C) \times O(n + C)$ integer grid (C is the total cost of the network, defined below), with vertices centered at grid points. The algorithm draws nodes as solid diamonds, but other shapes such as circles could also be used. Edges are drawn as "pipes" of varying size with a minimum separation of one unit at the base of each edge. There are no bends in the drawing, though edges can leave nodes at various angles. The drawing contains no edge crossings or occlusions of nodes or edges.

One of the main advantages of our algorithm is that it benefits from the disparity between low and high volume levels in the weights of different edges and nodes. Intuitively, our algorithm uses this disparity to take less space for drawing edges and nodes when possible. We use as the upper limit for the traffic on an edge a *capacity* of that edge, and we upper bound the sum of the capacities of adjacent edges as the capacity of a node. We assume that traffic information is supplied as a normalized list of edge thicknesses in the range $[0..w]$, for some parameter w (an edge of width 0 would be considered to have been added to make the graph maximally planar and would not be included in the final drawing). For the graph layout, we will consider edge weights to be integers, though in the rendering stage edges can easily be drawn with noninteger width within the integer space allocated to them (and in fact can be drawn with dynamic values changing over time, as long as they are less than the capacity). Denote the degree of a node v by $d(v)$. Define the thickness or *cost* of an edge e to be $c(e)$, and the size or weight of a node v to be $w(v) = \Sigma c(e)$ for all edges adjacent to v. For edges added to the graph to make it maximally planar, they can be given a cost of 0. Let $C = \Sigma_v w(v) = 2 * \Sigma_e c(e)$ be the total cost of the network. As mentioned above, our algorithm draws a weighted planar graph with edge- and vertex-widths proportional to their weights in an $O(n + C) \times O(n + C)$ integer grid. Thus, the total area is $O(n^2 + C^2)$. Note that, if w denotes the maximum width of an edge in a given graph G, then the area of our drawing of G is never more than $O(n^2w^2)$, for C is $O(nw)$ in a planar graph. Moreover, the area of one of our drawings can be significantly below the corresponding $O(n^2w^2)$ upper

bound for the naive approach. For example, if C is $O(w)$, then the area of our drawing is $O(n^2 + w^2)$, and even if C is $O(n + wn^{0.5})$, then the area is still at most $O(n^2 + nw^2)$.

2 The Algorithm

Suppose we are given a maximally planar graph G with n vertices and integer weights in the range $[0, w]$ assigned to its edges. Our algorithm for drawing G is as follows. Order the vertices of a maximally planar graph $v_1, v_2, \ldots, v_n$ according to their canonical ordering [8]. The following are then satisfied, for all $k \geq 3$:

1. For the graph G_k restricted to the first k vertices in the canonical ordering, G_k is biconnected (internally triconnected), and the cycle C_k of the external vertices of G_k contains (v_1, v_2).
2. The vertex v_{k+1} is in the exterior face of G_{k+1} and has at least two neighbors in G_k all of which are consecutive on $(C_k - (v_1, v_2))$. These are the only neighbors of v_{k+1} in G_k.

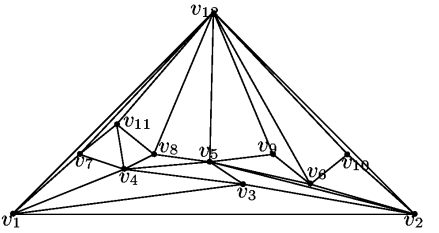


Fig. 2. A sample canonical ordering.

Such an ordering exists for every maximally planar graph and can be constructed in linear time (see, e.g., [4,8]). Figure 2 shows a sample graph with the canonical ordering of its vertices.

Let us define a structure called a *hub* around each vertex (see Figure 3). This is a diamond-shaped area with corners $w(v)+d(v)$ unit spaces above, below, left, and right of the vertex, similar to the *join box* of [11]. The diagonal of each unit square along the perimeter of the hub (see Figure 4) is called a *slot*, and a collection of sequential slots used by a single edge is called a *port*. Each edge is allocated at insertion time a port containing one slot per unit cost (if 0-cost edges are allowed, then the edge is drawn at the boundary between two slots), leaving a free slot between edges.

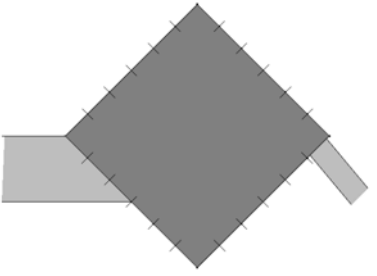


Fig. 3. A sample hub with a pair of edges

In order to show that an edge separation of at least 1 is maintained, we give a few conditions (adapted from invariants in [11]) that must be met for all G_k:

1. The vertices and slot boundaries of G_k are located at lattice points (have integer coordinates).

2. Let $c_1 = v_1, c_2, c_3, \ldots, c_m = v_2$ (for some $m = m(k)$) be the vertices along the exterior cycle C_k of G_k. Then the c_j's are strictly increasing in x.
3. All edges between slots of $c_1, c_2, \ldots, c_m$ have slope $+1$ or -1, with the exception of the edge between v_1 and v_2, which has slope 0.
4. For each $v \notin \{v_1, v_2\}$ in G_k, the slots with the left and right corners as their top boundaries have been used. Also, any slots used in the upper half of the hub are consecutive above either the left or right corner (with a space left in between), except for the slot used by the final edge when a node is *dominated* (see Section 2.2).
5. Each edge is monotone in both x and y.

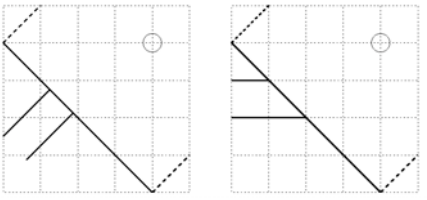


Fig. 4. An edge of width 1 using minimum and maximum perimeter space. Note that if the entry angle were shallower than the right image, the edge would no longer be monotone, since once inside the hub it needs to go up to reach the center.

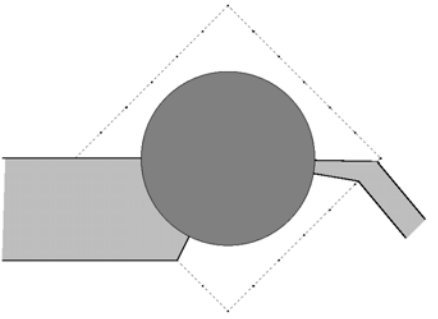


Fig. 5. The hub of Figure 3 drawn with a circular vertex.

2.1 Geometry

There are a few geometric issues with drawing thick edges out from a diamond-shaped box. We are focusing on the drawing of the edges outside the hub, since we intend to draw the entire hub solid as a node in the final graph. The perimeter length allocated to an edge of thickness $t \in \mathbf{Z}$ is actually $t\sqrt{2}$ since it is the diagonal of a square of side length t. This may be necessary, though, as the perimeter space needed by an edge can vary based on the angle it makes with the side of the hub. Thanks to monotonicity of edge segments (condition 5), the allocated length is sufficient to draw the edge, since the angle made between the incoming edge segment and the side of the hub is at least $\pi/4$, meaning the intersection segment in the unit square is of length at most $1/\cos(\pi/4) = \sqrt{2}$ (see Figure 4).

Because of this, we also do not need to concern ourselves with bends in the edges, as we can simply not draw the interior portion, only drawing the segment

between hubs, and drawing it at the correct angle when it leaves the node. If an edge does not need the full space, simply use the center of the allocated port.

The idea of monotonicity is no longer as obvious when we are not drawing the interior portions of the edges. One can extend the edges to the center of the node, and consider the monotonicity of the lines on the boundaries of our edges and ensure monotonicity of these, which we will refer to as the monotonicity of the entire thick edge.

It is also possible to draw the nodes as circular in shape, by using any circle centered within the diamond. This is a simple implementation detail; bend the edges at the segment of the hub, and narrow the edge as it approaches the node. This can be accomplished by bending the sides of the edge differently, pointing each perpendicular to the circle (Figure 5).

The above proves the following lemma:

Lemma 1. *If the five conditions listed above are maintained, then a port containing one slot per integer thickness of an edge is sufficient to draw the edge at its thickness, regardless of its incoming angle, without occluding other adjacent edges.*

2.2 The Construction

We now describe the *incremental* construction of the graph.

First two vertices. Refer to Figure 6. Build the canonical ordering and place the center of node v_1 at the origin of a 2-dimensional x, y graph. Center v_2 at $(x, 0)$ where $x = w(v_1) + d(v_1) + 1 + w(v_2) + d(v_2)$. Our nodes are drawn solid as the entire hub, so this placement of v_2 creates the minimum acceptable separation of one unit between the right corner of v_1 and the left corner of v_2. This graph, G_2, clearly maintains the five conditions (conditions 3 and 4 are trivial with only two nodes).

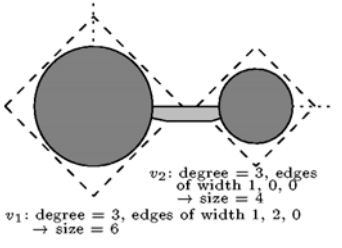


Fig. 6. Sample graph G_2.

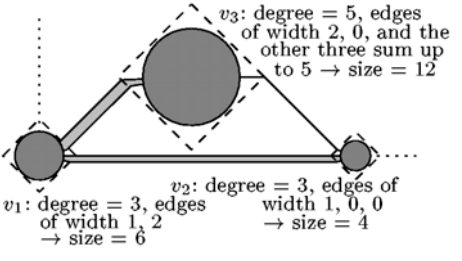


Fig. 7. Sample graph G_3.

Inserting v_3. refer to Figure 7. By the properties of the canonical ordering, v_3 must have edges to v_1 and v_2. Use the lowest slots available on the appropriate segments of v_1, v_2 (upper-right for v_1, upper-left for v_2) and the slots in v_3 whose

top points are the left and right corners. Shift v_2 horizontally to the right to allow the edges to be drawn at the correct slopes and to allow v_3 to be drawn without occluding edge (v_1, v_2). Set v_3 at height $h = 2 * (w(v_3) + d(v_3))$. The top of the edge (v_1, v_2) is at $y = 0$, so the top of v_3 must be at $y = h + 1$ to clear it. The top of v_3 is also the intersection of the lines of slope $+1$ and -1 drawn from the tops of the ports allocated to the edges (v_1, v_3) and (v_2, v_3) on v_1 and v_2, respectively. Since we are dealing with lines of slope ± 1, starting from even integer grid points (as assured for v_2, see below), their intersection is an integer grid point.

We need the intersection of the lines from these two ports to be at height $h + 1$. This requires that their x-coordinates (if extended to the line $y = 0$) be $2h + 2$ units apart. The actual distance necessary between v_1 and v_2 is $(2h + 2) - (2 * (c((v_1, v_3)) + 1)) - (2 * (c((v_2, v_3)) + 1))$. Shift v_2 right one unit less than this (since it is currently one unit to the right).

The case of inserting v_3 should be handled separately because it is the only situation where the top boundary of the initial graph contains edges not of slope ± 1. We will generalize to handle the remaining cases.

Induction. Refer to Figure 8. Assume as an inductive hypothesis that the graph G_k maintains the five conditions and has an edge separation of 1 between all edges. we now need to insert vertex v_{k+1} and its incident edges to G_k. Let $c_l, c_{l+1}, ...c_r$ be the neighbors of v_{k+1} in G_{k+1}. By the properties of the canonical ordering these neighbors are sequential along the outer face of G_k. Before inserting v_{k+1}, we need to make room for it and its edges to be drawn, and to ensure that the five conditions are still maintained for G_{k+1}. In order to do this, we shift the vertices along the exterior cycle C_k to the right. We also need to shift vertices in the interior portion of the graph to preserve planarity and to prevent occlusions. A node u is *dominated* when it is one of the neighbors of v_{k+1} in G_k other than c_l or c_r. A dominated node u has used its last edge (since it is an interior node in G_{k+1} and therefore additional edges would make G_{k+1} nonplanar), and is included in the shifting set of v_{k+1} (see below), so any slots remaining on u can be used to connect to v_{k+1} without creating edge crossings or occlusions in the shifting process. This enables edge (u, v_{k+1}) to select a port on u to maintain monotonicity.

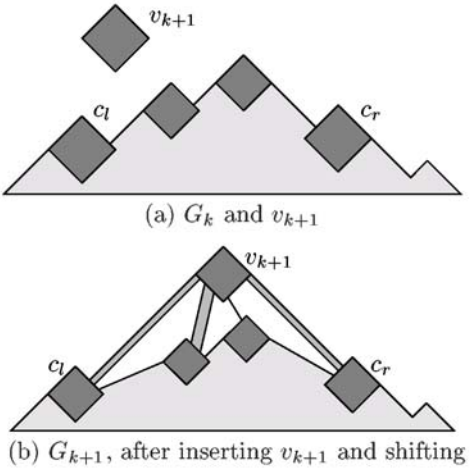


Fig. 8. Induction on the number of nodes.

Shifting sets. The paper by de Fraysseix et al. [8] outlines the concept of shifting sets for each

vertex on the outer cycle C_k of G_k, which designate how to move the interior vertices of the graph. We will use the same concept in our algorithm. The shifting set $M_k(c_i)$ for all c_i $(1 \leq i \leq m)$ on C_k contains the set of nodes to be moved along with c_i to avoid edge crossings and occlusions. Define the M_k's recursively, starting with $M_3(c_1 = v_1) = \{v_1, v_2, v_3\}$, $M_3(c_2 = v_3) = \{v_2, v_3\}$, $M_3(c_3 = v_2) = \{v_2\}$. Then, for the shifting sets used in G_{k+1}, let:

- $M_{k+1}(c_i) = M_k(c_i) \cup \{v_{k+1}\}$ for $i \leq l$;
- $M_{k+1}(v_{k+1}) = M_k(c_{l+1}) \cup \{v_{k+1}\}$;
- $M_{k+1}(c_j) = M_k(c_j)$ for $j \geq r$.

The sets obey the following claims for all k:

1. $c_j \in M_k(c_i)$ if and only if $j \geq i$;
2. $M_k(c_1) \supset M_k(c_2) \supset M_k(c_3) \supset \ldots \supset M_k(c_m)$;
3. For any nonnegative numbers α_i $(1 \leq i \leq m)$, sequentially shifting $M_k(c_i)$ right by α_i maintains planarity,[2] and does not introduce any edge or node occlusions.

The proofs of the first two claims are found in [8]. For the third, it is clearly true for the base case $k = 3$. Consider the graph G_{k+1}, v_{k+1}, and the vertices $c_1, c_2, \ldots, c_m$ along the cycle C_k of the exterior face of G_k. Let us fix shift amounts $\alpha(c_1), \alpha(c_2), \ldots, \alpha(c_l), \alpha(v_{k+1}), \alpha(c_r), \ldots, \alpha(c_m)$ corresponding to the vertices along the cycle C_{k+1}. The graph under the cycle C_k satisfies the condition by induction: set $\alpha(c_{l+1}) = 1 + 2 * (w(v_{k+1}) + d(v_{k+1})) + \alpha(v_{k+1})$ (the sum of the first two terms is the amount c_{l+1} will be shifted when v_{k+1} is inserted, and the last term is how much c_{l+1} and nodes in its shifting set will be shifted because of the shifting of v_{k+1}) and all other interior α's ($\alpha(c_{l+2})$ through $\alpha(c_{r-1})$) to 0, and the exterior α's ($\alpha(c_1), \ldots, \alpha(c_{l+1})$ and $\alpha(c_r), \ldots, \alpha(c_m)$) to their above values. The portion of the graph above C_k, with the exception of the edges (c_l, v_{k+1}) and (c_r, v_{k+1}), is shifted in a single block with v_{k+1}. The edge (c_l, v_{k+1}) cannot be forced to occlude or intersect the next edge, (c_{l+1}, v_{k+1}), since the latter edge can only be pushed farther away, moving along with the former when it shifts. Similarly, (c_{r-1}, v_{k+1}) cannot occlude or intersect (c_r, v_{k+1}) (see Figure 8(b)).

This proves the following lemma:

Lemma 2. *For all G_k, sequentially shifting the nodes in the shifting sets of each node in the exterior cycle of G_k by any nonnegative amount cannot create edge crossings or node or edge occlusions.*

Shifting and placement. Similar to [3], we will shift twice. First, shift $M_k(c_{l+1})$ by the width of node $v_{k+1} + 1$, which is $2 * (w(v_{k+1}) + d(v_{k+1})) + 1$. Also shift $M_k(c_r)$ by the same amount. (To ensure that c_r and c_l are separated by an even amount of units, shift $M_k(c_r)$ by one more unit if necessary.) The intuition behind this is simple. We cannot allow node v_{k+1} to occlude any portion of G_k. Since the

[2] This property of the shifting sets is stronger than what we need. Our algorithm performs only two shifts per iteration.

graph could rise as high in y as half the distance between c_l and c_r in x, placing v_{k+1} at the intersection of the edges of slope ± 1 from these nodes could place it on top of another vertex. Separating c_l and c_r by $2 + 2 * (\text{width/height of } v_{k+1})$ moves v_{k+1} half that much higher, allowing it to clear the graph.

Now that we have sufficiently shifted all nodes in G_k, we can place v_{k+1}. Define l_1 (resp., l_2) as the line of slope $+1$ (resp., -1) from the top of the port of c_l (resp., c_r) allocated to the edge (c_l, v_{k+1}) (resp., (c_r, v_{k+1})). Select the ports of c_l and c_r that maintain condition 4's requirement of minimum separation between edges. If the top corner of v_{k+1} is placed at the intersection of l_1 and l_2, all the edges between v_{k+1} and nodes in C_k can be drawn monotonically in x and y without creating occlusions. Note also that this placement of v_{k+1} assigns the edge (c_l, v_{k+1}) to the port whose top is the left corner of v_{k+1}, and likewise (c_r, v_{k+1}) is assigned to the port at the right corner of v_{k+1}. These edges are clearly monotone. Monotonicity for the new interior edges is ensured by selecting a port from the side of the v_{k+1} facing the target node, and a port from the target node facing v_{k+1}. Since each of the four sides of every node is of size $d(v) + w(v)$, ports can be chosen on arbitrary sides (maintaining condition 4, of course), and sufficient space for the edge is guaranteed. Also, since the edges are at least a distance of 1 apart on v_{k+1}, and their destination ports are all on different nodes each of which are at least a unit apart in x, no occlusions or intersections can be created. By the third detail of the shifting sets, this movement cannot cause edge occlusions or intersections. It remains to show that the graph maintains the five conditions listed above, however. The first is obviously true since everything is shifted by integer values. Likewise the second is true, since v_{k+1} is inserted between c_l and c_r, and each node is shifted at least as much to the right as the node before it, so their ordering remains intact. Since the edges before c_l and after c_r have not been changed (both endpoints of each have been moved by the same amounts), and the edges (c_l, v_{k+1}) and (c_r, v_{k+1}) were inserted at slopes of ± 1, condition 3 is still true. Monotonicity is maintained regardless of any horizontal shifting, so the edges of G_k remain monotone. The outside edges (c_l, v_{k+1}) and (c_r, v_{k+1}) are clearly monotone, and the interior edges were assigned ports on each node to make them monotone.

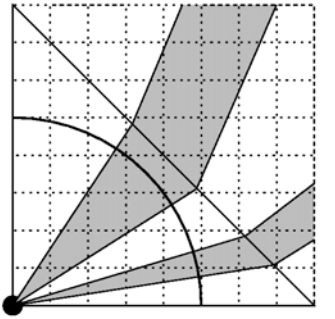


Fig. 9. The upper-right quadrant of a node.

When v_{k+1} is inserted, its left- and rightmost neighbors on C_k are assigned the slots whose tops are at the left and right corner, thus maintaining the first portion of condition 4. The rest is maintained by selecting the correct ports of c_l, c_r, and the interior nodes. Such ports must be available at every node, since each side of a node is large enough to support every edge adjacent to it. Therefore the graph G_{k+1} meets all conditions and has a minimum edge separation of 1.

2.3 Analysis

After inserting all vertices, the graph G still maintains the five conditions, and thus is planar, without crossings or occlusions, and has an edge separation of at least 1. The question of angular resolution is not necessarily relevant, since most or all of the hub area is drawn as a solid node for significance. But if one extended the edges to a point node at the center of the hub, then the boundary lines of the edges have a minimum angular resolution of $O(1/(w(n) + d(n))$ for all nodes (see Figure 9).

We also would like a well-bounded area for the complete drawing of G.

Theorem 1. *The area of the grid necessary to draw the graph is* $O(n + C) \times O(n + C)$*, where* C *is the total cost of the network, defined as* $C = \Sigma_u w(u) = 2 * \Sigma_e c(e)$ *for a given input set of edge costs* $c(e)$ *(and for each node* u*,* $w(u) = \Sigma_{e \in Adj[u]} c(e)$*).*

Proof. Since G is drawn within the convex hull of v_1, v_2, and v_n, the width is equal to the distance between the left corner of v_1 and the right corner of v_2. This initial distance at G_2 is 1 plus the widths of v_1 and v_2. Shifting all v_i for $i \geq 4$ moves v_2 to the right by at most $3+4*(w(v_i)+d(v_i))$, and the insertions of v_1 through v_3 can be upper bounded by this. Therefore the width of the drawing is bounded above by $\sum_{i=1}^{n}(3 + 4 * w(v_i) + 4 * d(v_i)) = 3n + 4C + 8|E|$, where E is the set of edges in the graph. Since in any planar graph $|E| \leq 3n - 6$, the width is bounded above by $27n + 4C$. The resulting drawing is approximately an isosceles triangle with slope ± 1 (approximately since the edges begin below the peak of v_1 and v_2, thus slightly lowering the top of the triangle). The height, therefore, is bounded by $14n + 2C$, except that the nodes v_1 and v_2 actually extend below the graph by half their height, and this height is not previously accounted for as it is outside the triangle. Therefore the bound on the height of the drawing is actually $14n+2C+\max(w(v_1)+d(v_1), w(v_2)+d(v_2))$. The max() term is bounded above by $n + C$, however, and the theorem holds.

For running time analysis, we refer the reader to the $O(n)$ time implementation of the algorithm of de Fraysseix et al. [8] by Chrobak and Payne [4]. This solution can be extended so as to implement our algorithm without changing the asymptotic running-time complexity.

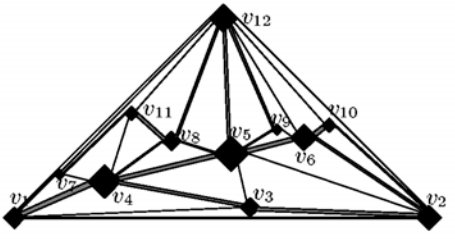


Fig. 10. A sample graph drawn by our method.

See Figure 10 for a sample drawing of a weighted version of Figure 2. The used edge weights and induced vertex sizes are listed in Figure 11.

Edge	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
v_1	-	1	0	5	-	-	1	-	-	-	-	-
v_2	1	-	4	-	0	2	-	-	-	1	-	-
v_3	0	4	-	3	0	-	-	-	-	-	-	-
v_4	5	-	3	-	4	-	0	1	-	-	0	-
v_5	-	0	0	4	-	4	-	1	1	-	-	3
v_6	-	2	-	-	4	-	-	-	0	4	-	-
v_7	1	-	-	0	-	-	-	-	-	-	1	-
v_8	-	-	-	1	1	-	-	-	-	-	3	2
v_9	-	-	-	-	1	0	-	-	-	-	-	2
v_{10}	-	1	-	-	-	4	-	-	-	-	-	-
v_{11}	-	-	-	0	-	-	1	3	-	-	-	-
v_{12}	-	-	-	-	3	-	-	2	2	-	-	-

Vertex	v_1	v_2	v_3	v_4	v_5	v_6
Size	11	13	11	19	20	14
Vertex	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
Size	5	11	6	7	7	10

Fig. 11. Sample graph: edge weights and vertex sizes.

3 Future Work

There are many possibilities for future related work:

- Combine awareness of edge thicknesses with force-directed graph drawing techniques by modifying the forces of nodes and edges according to their individual weights in order to 'make room' for them to be drawn larger.
- Establish an asymptotic lower bound on the area necessary to draw a graph with edge thickness as used in our paper. Node size can be reduced as long as the perimeter is of sufficient length to support all edges with a bounded separation. It is possible such a drawing could be done in $o((n + C)^2)$ area.
- Allow general graphs and edge crossings when necessary, but still use thick edges and large nodes and prevent occlusions, except in edge crossings.
- Combine the algorithms above with graph clustering techniques to represent potentially very large networks. One could add the sizes of nodes and edges clustered together. It could also be useful to represent the amount of information flowing within a cluster node in addition to between the nodes.
- Extend to 3D. The algorithm used here would not extend well, but drawings of graphs in three dimensions with thick edges and large nodes could be useful. Projections of such a graph to 2D would not be aesthetic.
- Study common network traffic patterns to optimize the algorithm based on real world data.

References

1. G. Di Battista, W. Didimo, M. Patrignani, and M. Pizzonia, Orthogonal and quasi-upward drawings with vertices of prescribed size, *Proc. 7th Int. Symp. on Graph Drawing*, *LNCS*, 1731, 297–310, Springer-Verlag, 1999.
2. G. Di Battista, P. Eades, R. Tamassia, and I. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, 1999.

3. C. Cheng, C. Duncan, M.T. Goodrich, and S. Kobourov, Drawing planar graphs with circular arcs, *Discrete & Computational Geometry*, 25:405–418, 2001.
4. M. Chrobak and T. Payne, A linear-time algorithm for drawing planar graphs, *Information Processing Letters*, 54:241–246, 1995.
5. C.A. Duncan, A. Efrat, S.G. Kobourov, and C. Wenk, Drawing with fat edges, *Proc. 9th Int. Symp. on Graph Drawing*, 162–177, 2001.
6. M. Eiglsperger and M. Kaufmann, Fast Compaction for orthogonal drawings with vertices of prescribed size, *Proc. 9th Int. Symp. on Graph Drawing*, 124–138, 2001.
7. M. Formann, T. Hagerup, J. Haralambides, M. Kaufmann, F.T. Leighton, A. Simvonis, E. Welzl, and G. Woeginger, Drawing graphs in the plane with high resolution, *SIAM J. of Computing*, 22:1035–1052, 1993.
8. H. de Fraysseix, J. Pach, and R. Pollack, How to draw a planar graph on a grid, *Combinatorica*, 10:41–51, 1990.
9. E.R. Gansner, E. Koutsofios, S.C. North, and K.P. Vo, A technique for drawing directed graphs, *IEEE Trans. on Software Engineering*, 19:214–230, 1993.
10. A. Garg and R. Tamassia, Planar drawings and angular resolution: Algorithms and bounds, *Proc. 2nd Ann. European Symp. on Algorithms*, *LNCS*, 855, 12–23, Springer-Verlag, 1994.
11. M.T. Goodrich and C. Wagner, A framework for drawing planar graphs with curves and polylines, *Proc. 6th Int. Symp. on Graph Drawing*, 153–166, 1998.
12. G. Kant, Drawing planar graphs using the canonical ordering, *Algorithmica*, 16:4–32, 1996.
13. G.W. Klau and P. Mutzel, Optimal compaction of orthogonal grid drawings, in *Integer Programming and Combinatorial Optimization* (G. Cornuejols, R.E. Burkard, and G.J. Woeginger, eds.), *LNCS*, 1610, 304–319, Springer-Verlag, 1999.
14. G.W. Klau and P. Mutzel, Combining graph labeling and compaction, *Proc. 7th Int. Symp. on Graph Drawing*, *LNCS*, 1731, 27–37, Springer-Verlag, 1999.
15. S. Malitz and A. Papakostas, On the angular resolution of planar graphs, *SIAM J. of Discrete Mathematics*, 7:172–183, 1994.
16. E.R. Tufte, *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, CT, 1983.

Semi-matchings for Bipartite Graphs and Load Balancing

Nicholas J.A. Harvey[1], Richard E. Ladner[2], László Lovász[1], and Tami Tamir[2]

[1] Microsoft Research, Redmond, WA, USA,
{nickhar, lovasz}@microsoft.com
[2] Department of Computer Science and Engineering,
University of Washington, Seattle, WA, USA,
{ladner, tami}@cs.washington.edu

Abstract. We consider the problem of fairly matching the left-hand vertices of a bipartite graph to the right-hand vertices. We refer to this problem as the *semi-matching* problem; it is a relaxation of the known bipartite matching problem. We present a way to evaluate the quality of a given semi-matching and show that, under this measure, an optimal semi-matching balances the load on the right hand vertices with respect to any L_p-norm. In particular, when modeling a job assignment system, an optimal semi-matching achieves the minimal makespan and the minimal flow time for the system.

The problem of finding optimal semi-matchings is a special case of certain scheduling problems for which known solutions exist. However, these known solutions are based on general network optimization algorithms, and are not the most efficient way to solve the optimal semi-matching problem. To compute optimal semi-matchings efficiently, we present and analyze two new algorithms. The first algorithm generalizes the Hungarian method for computing maximum bipartite matchings, while the second, more efficient algorithm is based on a new notion of *cost-reducing paths*. Our experimental results demonstrate that the second algorithm is vastly superior to using known network optimization algorithms to solve the optimal semi-matching problem. Furthermore, this same algorithm can also be used to find maximum bipartite matchings and is shown to be roughly as efficient as the best known algorithms for this goal.

1 Introduction

One of the classical combinatorial optimization problems is finding a maximum matching in a bipartite graph. The bipartite matching problem enjoys numerous practical applications [2, Section 12.2], and many efficient, polynomial time algorithms for computing solutions [8] [12] [14]. Formally, a bipartite graph is a graph $G = (U \cup V, E)$ in which $E \subseteq U \times V$. A matching in G is a set of edges, $M \subseteq E$, such that each vertex in $U \cup V$ is an endpoint of at most one edge in M; that is, each vertex in U is matched with at most one vertex in V and vice-versa.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 294–306, 2003.
© Springer-Verlag Berlin Heidelberg 2003

In this paper we consider a relaxation of the maximum bipartite matching problem. We define a *semi-matching* to be a set of edges, $M \subseteq E$, such that each vertex in U is an endpoint of exactly one edge in M. Clearly a semi-matching does not exist if there are isolated U-vertices, and so we require that each U-vertex in G have degree at least 1. Note that it is trivial to find a semi-matching — simply match each U-vertex with an arbitrary V-neighbor. Our objective is to find semi-matchings that match U-vertices with V-vertices as *fairly* as possible, that is, minimizing the variance of the matching edges at each V-vertex.

Our work is motivated by the following load balancing problem: We are given a set of tasks and a set of machines, each of which can process a subset of the tasks. Each task requires one unit of processing time, and must be assigned to some machine that can process it. The tasks are to be assigned to machines in a manner that minimizes some optimization objective. One possible objective is to minimize the *makespan* of the schedule, which is the maximal number of tasks assigned to any given machine. Another possible goal is to minimize the average completion time, or *flow time*, of the tasks. A third possible goal is to maximize the fairness of the assignment from the machines' point of view, i.e., to minimize the variance of the loads on the machines.

These load balancing problems have received intense study in the online setting, in which tasks arrive and leave over time [4]. In this paper we consider the offline setting, in which all tasks are known in advance. Problems from the online setting may be solved using an offline algorithm if the algorithm's runtime is significantly faster than the tasks' arrival/departure rate, and tasks may be reassigned from one machine to another without expense. In particular, the second algorithm we present can incrementally update an existing assignment after task arrivals or departures.

One example of an online load balancing problem that can be efficiently solved by an offline solution comes from the Microsoft Active Directory system [1], which is a distributed directory service. Corporate deployments of this system commonly connect thousands of servers in geographically distributed branch offices to servers in a central "hub" data center; the branch office servers periodically replicate with the hub servers to maintain database consistency. Partitioning the database according to corporate divisions creates constraints on which hub servers a given branch server may replicate with. Thus, the assignment of branch servers to hub servers for the purpose of replication is a load balancing problem: the branch servers are the "tasks", and the hub servers are the "machines". Since servers are only rarely added or removed, and servers can be efficiently reassigned to replicate with another server, this load balancing problem is amenable to the offline solutions that we present herein.

Load balancing problems of the form described above can be represented as instances of the semi-matching problem as follows: Each task is represented by a vertex $u \in U$, and each machine is represented by a vertex $v \in V$. There is an edge $\{u, v\}$ if task u can be processed by machine v. Any semi-matching in the graph determines an assignment of the tasks to the machines. Furthermore, we

show that a semi-matching that is as fair as possible gives an assignment of tasks to machines that simultaneously minimizes the makespan and the flow time.

The primary contributions of this paper are: (1) the semi-matching model for solving load balancing problems of the form described above, (2) two efficient algorithms for computing optimal semi-matchings, and (3) a new algorithmic approach for the bipartite matching problem. We also discuss in Section 2 representations of the semi-matching problem as network optimization problems, based on known solutions to scheduling problems. Section 3 presents several important properties of optimal semi-matchings. One of these properties provides a necessary and sufficient condition for a semi-matching to be optimal. Specifically, we define a *cost-reducing path*, and show that a semi-matching is optimal if and only if no cost reducing path exists. Sections 4 and 5 present two algorithms for computing optimal semi-matchings; the latter algorithm uses the approach of identifying and removing cost-reducing paths. Finally, Section 6 describes an experimental evaluation of our algorithms against known algorithms for computing optimal semi-matchings and maximum bipartite matchings. Due to space limitations this paper omits proofs for some of the theorems.

2 Preliminaries

Let $G = (U \cup V, E)$ be a simple bipartite graph with U the set of left-hand vertices, V the set of right-hand vertices, and edge set $E \subseteq U \times V$. We denote by n and m the sizes of the left-hand and the right-hand sides of G respectively (i.e., $n = |U|$ and $m = |V|$). Since our work is motivated by a load balancing problem, we often call the U-vertices "tasks" and the V-vertices "machines".

We define a set $M \subseteq E$ to be a *semi-matching* if each vertex $u \in U$ is incident with exactly one edge in M. We assume that all of the vertices in U have degree at least 1 since isolated U-vertices can not participate in the matching. A semi-matching gives an assignment of each task to a machine that can process it.

For $v \in V$, let $deg(v)$ denote the degree of vertex v; in load balancing terms, $deg(v)$ is the number of tasks that machine v is capable of executing. Let $deg_M(v)$ denote the number of edges in M that are incident with v; in load balancing terms, $deg_M(v)$ is the number of tasks assigned to machine v. We frequently refer to $deg_M(v)$ as the *load* on vertex v. Note that if several tasks are assigned to a machine then one task completes its execution after one time unit, the next task after two time units, etc. However, semi-matchings do not specify the order in which the tasks are to be executed.

We define $cost_M(v)$ for a vertex $v \in V$ to be

$$\sum_{i=1}^{deg_M(v)} i = \frac{deg_M(v) \cdot (deg_M(v) + 1)}{2}.$$

This expression gives the total latency experienced by all tasks assigned to machine v. The *total cost* of a semi-matching, M, is defined to be $T(M) =$

$\sum_{i=1}^{m} cost_M(v_i)$. A semi-matching with minimal total cost is called an *optimal semi-matching*. We show in Section 3 that an optimal semi-matching is also optimal with respect to other optimization objectives, such as maximizing the load balance on the machines (by minimizing, for any p, the L_p-norm of the load vector), minimizing the variance of the machines' load, and minimizing the maximum load on any machine.

For a given semi-matching M in G, define an *alternating path* to be a sequence of edges $P = (\{v_1, u_1\}, \{u_1, v_2\}, \ldots, \{u_{k-1}, v_k\})$ with $v_i \in V$, $u_i \in U$, and $\{v_i, u_i\} \in M$ for each i. Without the possibility of confusion, we sometimes treat paths as though they were a sequence of vertices $(v_1, u_1, \ldots, u_{k-1}, v_k)$. The notation $A \oplus B$ denotes the symmetric difference of sets A and B; that is, $A \oplus B = (A \setminus B) \cup (B \setminus A)$. Note that if P is an alternating path relative to a semi-matching M then $P \oplus M$ is also a semi-matching, derived from M by switching matching and non-matching edges along P. If $deg_M(v_1) > deg_M(v_k)+1$ then P is called a *cost-reducing path* relative to M. Cost-reducing paths are so named because switching matching and non-matching edges along P yields a semi-matching $P \oplus M$ whose cost is less than the cost of M. Specifically,

$$T(P \oplus M) = T(M) - (deg_M(v_1) - deg_M(v_k) - 1).$$

2.1 Related Work

The maximum bipartite matching problem is known to be solvable in polynomial time using a reduction from maximum flow [2] [9] or by the Hungarian method [14] [15, Section 5.5]. Push-relabel algorithms are widely considered to be the fastest algorithms in practice for this problem [8].

The load balancing problems we consider in this paper can be represented as restricted cases of *scheduling on unrelated machines*. These scheduling problems specify for each job j and machine i the value $p_{i,j}$, which is the time it takes machine i to process job j. When $p_{i,j} \in \{1, \infty\}\ \forall i, j$, this yields an instance of the semi-matching problem, as described in Section 2.2. In standard scheduling notation [11], this problem is known as $R \mid p_{i,j} \in \{1, \infty\} \mid \sum_j C_j$. Algorithms are known for minimizing the flow time of jobs on unrelated machines [2, Application 12.9] [7] [13]; these algorithms are based on network flow formulations.

The online version of this problem, in which the jobs arrive sequentially and must be assigned upon arrival, has been studied extensively in recent years [3] [5] [6]. A comprehensive survey of the field is given in [4].

2.2 Representation as Known Optimization Problems

The optimal semi-matching problem can be represented as special instances of two well-known optimization problems: weighted assignment and min-cost max-flow. However, Section 6 shows that the performance of the resulting algorithms is inferior to the performance of our algorithms presented in sections 4 and 5.

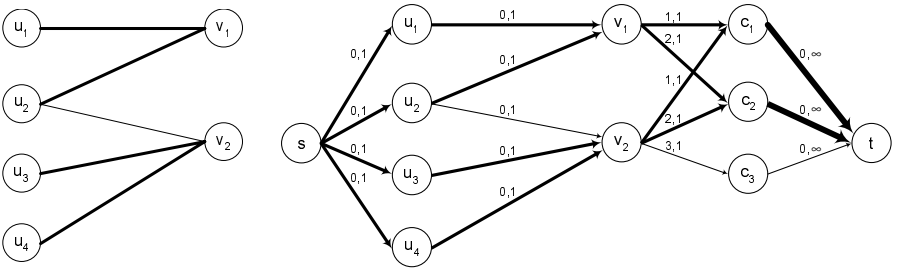


Fig. 1. (a) shows a graph in which the bold edges form an optimal semi-matching. (b) shows the corresponding min-cost max-flow problem. Each edge is labeled with two numbers: a cost, and a capacity constraint. Bold edges carry one unit of flow and doubly-bold edges carry two units of flow.

Recall that the scheduling problem $R \mid\mid \sum_j C_j$, and in particular the case in which $p_{i,j} \in \{1, \infty\}$, can be reduced to a weighted assignment problem [7] [13]. A semi-matching instance can be represented as an $R \mid p_{i,j} \in \{1, \infty\} \mid \sum_j C_j$ instance as follows: Each U-vertex represents a job, and each V-vertex represents a machine. For any job j and machine i, we set $p_{i,j} = 1$ if the edge $\{u_j, v_i\}$ exists, and otherwise $p_{i,j} = \infty$. Clearly, any finite schedule for the scheduling problem determines a feasible semi-matching. In particular, a schedule that minimizes the flow time determines an optimal semi-matching. Thus, algorithms for the weighted assignment problem can solve the optimal semi-matching problem.

The min-cost max-flow problem is one of the most important combinatorial optimization problems; its objective is to find a minimum-cost maximum-flow in a network [2]. Indeed, the weighted assignment problem can be reduced to min-cost max-flow problem. Thus, from the above discussion, it should be clear that a semi-matching problem instance can be recast as a min-cost max-flow problem. We now describe an alternative, more compact, transformation of the optimal semi-matching problem to a min-cost max-flow problem.

Given $G = (U \cup V, E)$, a bipartite graph giving an instance of a semi-matching problem, we show how to construct a network N such that a min-cost max-flow in N determines an optimal semi-matching in G. The network N is constructed from G by adding at most $|U| + 2$ vertices and $2|U| + |E|$ edges (see Figure 1). The additional vertices are a source, s, a sink, t, and a set of "cost centers" $C = \{c_1, \ldots, c_\Delta\}$, where $\Delta \leq |U|$ is the maximal degree of any V-vertex. Edges with cost 0 and capacity 1 connect s to each of the vertices in U. The original edges connecting U and V are directed from U to V and are given cost 0 and capacity 1. For each $v \in V$, v is connected to cost centers $c_1, \ldots, c_{deg(v)}$ with edges of capacity 1 and costs $1, 2, \ldots, deg(v)$ respectively. Edges with cost 0 and infinite capacity connect each of the cost centers to the sink, t.

3 Properties of Optimal Semi-matchings

This section presents various important properties of optimal semi-matchings. Section 3.1 characterizes when a semi-matching is optimal. Section 3.2 states that an optimal semi-matching always contains a maximum matching and discusses various consequences. Section 3.3 states that an optimal semi-matching is also optimal with respect to any L_p-norm and the L_∞-norm.

3.1 Characterization of Optimal Semi-matchings

An important theorem from network flow theory is that a maximum flow has minimum cost if and only if no negative-cost cycle exists [2, Theorem 3.8]. We now prove an analogous result for semi-matchings. In Section 5 we describe the Algorithm $\mathcal{A}_{SM2}$ which is based on this property.

Theorem 1. *A semi-matching M is optimal if and only if no cost-reducing path relative to M exists.*

Proof. Let G be an instance of a semi-matching problem, and let M be a semi-matching in G. Clearly, if M is optimal then no cost-reducing path can exist. We show that a cost-reducing path must exist if M is not optimal.

Let O be an optimal semi-matching in G, chosen such that the symmetric difference $O \oplus M = (O \setminus M) \cup (M \setminus O)$ is minimized. Assume that M is not optimal, implying that M has greater total cost than O: i.e., $T(O) < T(M)$. Recall that $deg_O(v)$ and $deg_M(v)$ denote the number of U-vertices matched with v by O and M respectively. Let G_d be the subgraph of G induced by the edges of $O \oplus M$. Color with green the edges of $O \setminus M$ and with red the edges of $M \setminus O$. Direct the green edges from U to V and the red edges from V to U. We will use the following property of G_d (proof omitted).

Claim 1 *The graph G_d is acyclic, and for every directed path P in G_d from $v_1 \in V$ to $v_2 \in V$, we have $deg_O(v_2) \leq deg_O(v_1)$.*

Both O and M are semi-matchings, implying that $\sum_v deg_O(v) = \sum_v deg_M(v) = |U|$. Since $T(O) < T(M)$, there must exist $v_1 \in V$ such that $deg_M(v_1) > deg_O(v_1)$. Starting from v_1, we build an alternating red-green path, P', as follows. (1) From an arbitrary vertex $v \in V$, if $deg_{M \setminus O}(v) \geq 1$ and $deg_M(v) \geq deg_M(v_1) - 1$, we build P' by following an arbitrary red edge directed out from v. (2) From an arbitrary vertex $u \in U$, we build P' by following the single green edge directed out from u. (3) Otherwise, we stop.

By Claim 1, G_d is acyclic and therefore P' is well-defined and finite. Let $v_2 \in V$ be the final vertex on the path. There are two cases.

(1) $deg_M(v_2) < deg_M(v_1) - 1$: Thus P' is a cost-reducing path relative to M.

(2) $deg_{M \setminus O}(v_2) = 0$. In this case, we know that $deg_M(v_2) < deg_O(v_2)$ since P' arrived at v_2 via a green edge. By Claim 1, we must also have that $deg_O(v_2) \leq deg_O(v_1)$. Finally, recall that v_1 was chosen such that $deg_O(v_1) < deg_M(v_1)$. Combining these three inequities yields: $deg_M(v_2) < deg_O(v_2) \leq deg_O(v_1) < deg_M(v_1)$. This implies that $deg_M(v_2) < deg_M(v_1) - 1$, and so P' is a cost-reducing path relative to M.

Since P' is a cost-reducing path relative to M in both cases, the proof is complete.

3.2 Optimal Semi-matchings Contain Maximum Matchings

In this section, we state, omitting the proof, that every optimal semi-matching must contain a maximum bipartite matching; furthermore, it is a simple process to find these maximum matchings. Thus, the problem of finding optimal semi-matchings indeed generalizes the problem of finding maximum matchings.

Theorem 2. *Let M be an optimal semi-matching in G. Then there exists $S \subseteq M$ such that S is a maximum matching in G.*

We note that the converse of this theorem is not true: Not every maximum matching can be extended to an optimal semi-matching.

Corollary 1. *Let M be an optimal semi-matching in G. Define $f(M)$ to be the number of right-hand vertices in G that are incident with at least one edge in M. Then the size of a maximum matching in G is $f(M)$.*

In particular, if G has a perfect matching and M is an optimal semi-matching in G then M is a perfect matching. Corollary 1 yields a simple algorithm for computing a maximum matching from an optimal semi-matching, M: For each $v \in V$, if $deg_M(v) > 1$, select one arbitrary edge from M that is incident with v.

3.3 Optimality with Respect to L_p- and L_∞-Norm

Let $x_i = deg_M(v_i)$ denote the load on machine i (i.e., the number of tasks assigned to machine i). The L_p-norm of the vector $X = (x_1, \ldots, x_{|V|})$ is $||X||_p = (\sum_i x_i^p)^{1/p}$. The following theorem states that an optimal semi-matching is optimal with respect to the L_p-norm of the vector X for any finite p; in other words, optimal semi-matchings minimize $||X||_p$. (Note that $||X||_1 = |U|$ for all semi-matchings, so all semi-matchings are optimal with respect to the L_1-norm).

Theorem 3. *Let $2 \leq p < \infty$. A semi-matching has optimal total cost if and only if it is optimal with respect to the L_p-norm of its load vector.*

Another important optimization objective in practice is minimizing the maximal load on any machine; this is achieved by minimizing the L_∞-norm of the machines' load vector X. The following theorem states that optimal semi-matchings do minimize the L_∞-norm of X, and thus are an "ultimate" solution that simultaneously minimizes both the variance of the machines' load (from the L_2-norm) and the maximal machine load (given by the L_∞-norm).

Theorem 4. *An optimal semi-matching is also optimal with respect to L_∞.*

The converse of Theorem 4 is not valid; that is, minimizing the L_∞-norm does not imply minimization of other L_p-norms.

4 $\mathcal{A}_{SM1}$: An $O(|U||E|)$ Algorithm for Optimal Semi-matchings

In this section we present our first algorithm, $\mathcal{A}_{SM1}$, for finding an optimal semi-matching. The time complexity of $\mathcal{A}_{SM1}$ is $O(|U||E|)$, which is identical to that of the Hungarian algorithm [14] [15, Section 5.5] for finding maximum bipartite matchings. Indeed, $\mathcal{A}_{SM1}$ is merely a simple modification of the Hungarian algorithm, as we explain below.

The Hungarian algorithm for finding maximum bipartite matchings considers each left-hand vertex u in turn and builds an alternating search tree, rooted at u, in order to find an unmatched right-hand vertex (i.e., a vertex $v \in V$ with $deg_M(v) = 0$). If such a vertex v is found, the matching and non-matching edges along the u-v path are switched so that u and v are no longer unmatched.

Similarly, $\mathcal{A}_{SM1}$ maintains a partial semi-matching M, starting with the empty set. In each iteration, it considers a left-hand vertex u and builds an alternating search tree rooted at u, looking for a right-hand vertex v such that $deg_M(v)$ is as small as possible. To build the tree rooted at u we perform a directed breadth-first search in G starting from u, where edges in M are directed from V to U and edges not in M are directed from U to V. We select in this tree a path P from u to a least loaded V-vertex reachable from u. We increase the size of M by forming $P \oplus M$; that is, we add to the matching the first edge in this path, and switch matching and non-matching edges along the remainder of the path. As a result, u is no longer unmatched and $deg_M(v)$ increases by 1.

We repeat this procedure of building a tree and extending the matching accordingly for all of the vertices in U. Since each iteration matches a vertex in U with a single vertex in V and does not change $deg_M(u)$ for any other $u \in U$, the resulting selection of edges is indeed a semi-matching.

Theorem 5. *Algorithm $\mathcal{A}_{SM1}$ produces an optimal semi-matching.*

Proof. We show that no cost-reducing path is created during the execution of the algorithm. In particular, no cost reducing path exists at the end of the execution; thus, by Theorem 1, the resulting matching is optimal.

Assume the opposite and let $P^* = (v_1, u_1, \ldots, v_{k-1}, u_{k-1}, v_k)$, be the *first* cost-reducing path created by $\mathcal{A}_{SM1}$. Let M be the partial semi-matching after the iteration in which P^* is created. Thus, $deg_M(v_1) > deg_M(v_k) + 1$. Without loss of generality (by taking a sub-path of P^*), we can assume that there exists some x such that $deg_M(v_1) \geq x + 1$, $deg_M(v_i) = x \ \forall i \in \{2, \ldots, k-1\}$, and

$deg_M(v_k) \leq x - 1$. Let u' be the U-vertex added to the assignment during the previous iteration in which the load on v_1 increased. The algorithm gives that v_1 is a least-loaded V-vertex reachable from u'; thus, the search tree built for u' includes only V-vertices with load at least x; thus v_k is not reachable from u'.

Given that the path P^* exists, at some iteration occurring after the one in which u' is added, all the edges (u_i, v_i) of P^* are in the matching. Let u^* be the U-vertex, added after u', whose addition to the assignment creates P^*. The following claims yield a contradiction in the way u^* is assigned.

Claim 2 *When adding u^*, the load on v_k is at most $x - 1$ and v_k is in the tree rooted at u^*.*

Claim 3 *When adding u^*, the load on some vertex with load at least x increases.*

Claims 2 and 3 contradict the execution of $\mathcal{A}_{SM1}$, and therefore P^* cannot exist.

To bound the runtime of $\mathcal{A}_{SM1}$, observe that there are exactly $|U|$ iterations. Each iteration requires at most $O(|E|)$ time to build the alternating search tree and at most $O(\min\{|U|, |V|\})$ time to switch edges along the alternating path. Thus the total time required is at most $O(|U||E|)$.

5 $\mathcal{A}_{SM2}$: An Efficient, Practical Algorithm

We present $\mathcal{A}_{SM2}$, our second algorithm for finding optimal semi-matchings. Our analysis of its runtime gives an upper bound of $O(\min\{|U|^{3/2}, |U||V|\} \cdot |E|)$, which is worse than the bound of $O(|U||E|)$ for algorithm $\mathcal{A}_{SM1}$. However, our analysis for $\mathcal{A}_{SM2}$ is loose; in practice, $\mathcal{A}_{SM2}$ performs much better than $\mathcal{A}_{SM1}$, as our experiments in Section 6 show.

Theorem 1 proves that a semi-matching is optimal if and only if the graph does not contain a cost-reducing path. $\mathcal{A}_{SM2}$ uses that result to find an optimal semi-matching as follows:

Overview of $\mathcal{A}_{SM2}$

```
1  Find an initial semi-matching, M.
2  While there exists a cost-reducing path, P
3     Use P to reduce the cost of M.
```

Since the cost can only be reduced a finite number of times, this algorithm must terminate. Moreover, if the initial assignment is nearly optimal, the algorithm terminates after few iterations.

Finding an Initial Semi-Matching: The first step of algorithm $\mathcal{A}_{SM2}$ is to determine an initial semi-matching, M. Our experiments have shown that the

following greedy algorithm works well in practice. First, the U-vertices are sorted by increasing degree. Each U-vertex is then considered in turn, and assigned to a V-neighbor with least load. In the case of a tie, a V-neighbor with least degree is chosen. The purpose of considering vertices with lower degree earlier is to allow more constrained vertices (i.e., ones with fewer neighbors) to "choose" their matching vertices first. The same rule of choosing the least loaded V-vertex is also commonly used in the online case [3]. However, in the online case it is not possible to sort the U-vertices or to know the degree of the V-vertices in advance.

The total time required to find this initial matching is $O(|E|)$, since every edge is examined exactly once, and the sorting can be done using bucket sort.

Finding Cost-Reducing Paths: The key operation of the $\mathcal{A}_{SM2}$ algorithm is the method for finding cost-reducing paths. As a simple approach, one may determine if a particular vertex $v \in V$ is the ending vertex of a cost-reducing path simply by growing a tree of alternating paths rooted at v. As a better approach, one may determine if *any* $v \in V$ is the ending vertex of a cost-reducing path in $O(|E|)$ time. To do this, simply grow a depth-first search (DFS) forest of alternating paths where each tree root is chosen to be an unused V-vertex with lowest load. To find such a vertex, the V-vertices are maintained sorted by their load in an array of $|U| + 1$ buckets.

Analysis of $\mathcal{A}_{SM2}$: As argued earlier, the initial matching can be found in $O(|E|)$ time. Following this initial step, we iteratively find and remove cost-reducing paths. Identifying a cost-reducing path or lack thereof requires $O(|E|)$ time since it performs a depth-first search over all of G. If a cost-reducing path has been identified, then we switch matching and non-matching edges along that path, requiring $O(\min\{|U|, |V|\}) = O(|E|)$ time. Thus, the runtime of $\mathcal{A}_{SM2}$ is $O(I \cdot |E|)$, where I is the number of iterations needed to achieve optimality.

It remains to determine how many iterations are required. A simple bound of $I = O(|U|^2)$ may be obtained by observing that the worst possible initial matching has cost at most $O(|U|^2)$ and that each iteration reduces the cost by at least 1. The following theorem gives an improved bound.

Theorem 6. *$\mathcal{A}_{SM2}$ requires at most $O(\min\{|U|^{3/2}, |U||V|\})$ iterations.*

Remark 1. For graphs in which the optimal semi-matching cost is $O(|U|)$, the running time of $\mathcal{A}_{SM2}$ is $O(|U||E|)$. This bound holds since Awerbuch et al. [3] show that the cost of the greedy initial assignment is at most $4 \cdot T(M_{OPT})$; thus $\mathcal{A}_{SM2}$ needs at most $O(|U|)$ iterations to achieve optimality.

Practical Considerations: The description of $\mathcal{A}_{SM2}$ given above suggests that each iteration builds a depth-first search forest and finds a single cost-reducing path. In practice, a single DFS forest often contains numerous vertex-disjoint cost-reducing paths. Thus, our implementation repeatedly performs linear-time

Table 1. (a) gives the execution time in seconds of four algorithms for the optimal semi-matching problem, on a variety of graphs with 65,536 vertices. "—" indicates that no results could be recorded since the graph exceeded the memory of our test machine. (b) gives the execution time in seconds of three algorithms for the maximum bipartite matching problem, on a variety of graphs with 524,288 vertices.

Graph	$\mathcal{A}_{SM1}$	$\mathcal{A}_{SM2}$	LEDA	CSA
FewG	1.834	0.337	30.625	1.274
Grid	0.672	0.131	6.850	1.310
Hexa	1.521	0.319	28.349	2.131
Hilo	0.650	0.299	11.141	2.968
ManyG	1.669	0.200	18.388	1.238
Rope	0.269	0.188	7.588	1.330
Zipf	6.134	0.156	—	—
Total	12.749	1.630	>102.941	> 10.251

(a)

Graph	$\mathcal{A}_{SM2}$	BFS	LO
FewG	3.563	15.018	2.085
Grid	0.545	4.182	1.140
Hexa	3.569	13.990	1.755
Hilo	2.942	3.047	6.559
ManyG	3.607	13.640	2.199
Rope	1.308	2.459	1.400
Zipf	1.105	0.375	0.938
Total	16.639	52.711	16.076

(b)

scans of the graph, growing the forest and removing cost-reducing paths. We repeatedly scan the graph until a scan finds no cost-reducing path, indicating that optimality has been achieved.

Our bound of $O(\min\{|U|^{3/2}, |U||V|\})$ iterations is loose: experiments show that much fewer iterations are required in practice. We were able to create "bad" graphs, in which the number of iterations needed is $\Omega(|U|^{3/2})$; however, most of the cost-reducing paths in these graphs are very short, thus each iteration takes roughly constant time. While our bound for $\mathcal{A}_{SM2}$ is worse than our bound for $\mathcal{A}_{SM1}$, we believe that the choice of $\mathcal{A}_{SM2}$ as the best algorithm is justified already by its actual performance, as described in the next section.

Variants of $\mathcal{A}_{SM2}$, in which each iteration seeks a cost-reducing path with some property (such as "maximal difference in load between first and last vertex"), will also result in an optimal semi-matching. It is unknown whether such algorithms yield a better analysis than $\mathcal{A}_{SM2}$, or whether each iteration of such algorithms can be performed quickly in practice.

6 Experimental Evaluation

We implemented a program to execute $\mathcal{A}_{SM1}$, $\mathcal{A}_{SM2}$ and various known algorithms on a variety of "benchmark" input graphs. All input graphs were created by the bipartite graph generators used in [8]. Our simulation program was implemented in C and run on a Compaq Evo D500 machine with a 2.2GHz Pentium 4 CPU and 512MB of RAM.

First, we compared $\mathcal{A}_{SM1}$ and $\mathcal{A}_{SM2}$ with known techniques for computing optimal semi-matchings based on the transformation to the assignment problem.

To solve the assignment problem, we used two available algorithms: CSA [10], and LEDA [16]. For the CSA algorithm, the transformed graph was augmented with additional vertices and edges to satisfy CSA's requirement that a perfect assignment exist[1]. Table 1(a) shows the results of these experiments on graphs with 2^{16} vertices. The Zipf graphs (after being transformed to the assignment problem) exceeded the memory on our test machine, and no reasonable results could be recorded. Table 1(a) reports the elapsed execution time of these algorithms, excluding the time to load the input data. The reported value is the mean over five execution runs, each using a different seed to generate the input graph. These results show that $\mathcal{A}_{SM2}$ is much more efficient than assignment algorithms for the optimal semi-matching problem on a variety of input graphs.

Next, we compared $\mathcal{A}_{SM2}$ with two algorithms for computing maximum bipartite matchings from [8]: BFS, their fastest implementation based on augmenting paths, and LO, their fastest implementation based on the push-relabel method. For this series of experiments, we consider only graphs with 2^{19} vertices. As before, the reported value is the mean of the execution time over five runs; these results are shown in Table 1(b). These results show that $\mathcal{A}_{SM2}$ is roughly as efficient as the best known algorithm for the maximum bipartite matching problem on a variety of input graphs.

References

1. Active Directory. `http://www.microsoft.com/windowsserver2003/technologies`.
2. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.
3. B. Awerbuch, Y. Azar, E. Grove, M. Y. Kao, P. Krishnan, and J. S. Vitter. Load Balancing in the L_p Norm. In *Proceedings of FOCS*, 1995.
4. Y. Azar. On-line Load Balancing. In A. Fiat and G. Woeginger, editors, *Online Algorithms: The State of the Art (LNCS 1442)*, chapter 8. Springer-Verlag, 1998.
5. Y. Azar, A. Z. Broder, and A. R. Karlin. On-line load balancing. *Theoretical Computer Science*, 130(1):73–84, 1994.
6. Y. Azar, J. Naor, and R. Rom. The Competitiveness of On-line Assignments. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1992.
7. J. L. Bruno, E. G. Coffman, and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17:382–387, 1974.
8. B. V. Cherkassky, A. V. Goldberg, P. Martin, J. C. Setubal, and J. Stolfi. Augment or push: a computational study of bipartite matching and unit-capacity flow algorithms. *ACM J. Exp. Algorithmics*, 3(8), 1998.
   Source code available at `http://www.avglab.com/andrew/soft.html`.
9. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, second edition, 2001.

[1] We acknowledge Andrew Goldberg's assistance in finding such a transformation with a linear number of additional vertices and edges.

10. A. Goldberg and R. Kennedy. An efficient cost scaling algorithm for the assignment problem. *Math. Prog.*, 71:153–178, 1995.
Source code available at `http://www.avglab.com/andrew/soft.html`.
11. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete Math*, 5:287–326, 1979.
12. J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Computing*, 2:225–231, 1973.
13. W. A. Horn. Minimizing average flow time with parallel machines. *Operations Research*, 21:846–847, 1973.
14. H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Res. Logist. Quart.*, 2:83–97, 1955.
15. E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Dover, 2001.
16. LEDA. `http://www.algorithmic-solutions.com/`.

The Traveling Salesman Problem for Cubic Graphs

David Eppstein

School of Information & Computer Science
University of California, Irvine
Irvine, CA 92697-3425, USA
eppstein@ics.uci.edu

Abstract. We show how to find a Hamiltonian cycle in a graph of degree at most three with n vertices, in time $\mathcal{O}(2^{n/3}) \approx 1.25992^n$ and linear space. Our algorithm can find the minimum weight Hamiltonian cycle (traveling salesman problem), in the same time bound, and count the number of Hamiltonian cycles in time $\mathcal{O}(2^{3n/8}n^{\mathcal{O}(1)}) \approx 1.29684^n$. We also solve the traveling salesman problem in graphs of degree at most four, by a randomized (Monte Carlo) algorithm with runtime $\mathcal{O}((27/4)^{n/3}) \approx 1.88988^n$. Our algorithms allow the input to specify a set of forced edges which must be part of any generated cycle.

1 Introduction

The traveling salesman problem and the closely related Hamiltonian cycle problem are two of the most fundamental of NP-complete graph problems [5]. However, despite much progress on exponential-time solutions to other graph problems such as chromatic number [2, 3, 6] or maximal independent sets [1, 7, 8], the only worst-case bound known for finding Hamiltonian cycles or traveling salesman tours is that for a simple dynamic program, using time and space $\mathcal{O}(2^n n^{\mathcal{O}(1)})$, that finds Hamiltonian paths with specified endpoints for each induced subgraph of the input graph (D. S. Johnson, personal communication). Therefore, it is of interest to find special cases of the problem that, while still NP-complete, may be solved more quickly than the general problem.

In this paper, we consider one such case: the traveling salesman problem in graphs with maximum degree three. Bounded-degree maximum independent sets had previously been considered [1] but we are unaware of similar work for the traveling salesman problem. More generally, we consider the *forced traveling salesman problem* in which the input is a multigraph G and set of *forced edges* F; the output is a minimum cost Hamiltonian cycle of G, containing all edges of F. A naive branching search that repeatedly adds one edge to a growing path, choosing at each step one of two edges at the path endpoint, and backtracking when the chosen edge leads to a previous vertex, solves this problem in time $\mathcal{O}(2^n)$ and linear space; this is already an improvement over the general graph dynamic programming algorithm. We show that more sophisticated backtracking

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 307–318, 2003.
© Springer-Verlag Berlin Heidelberg 2003

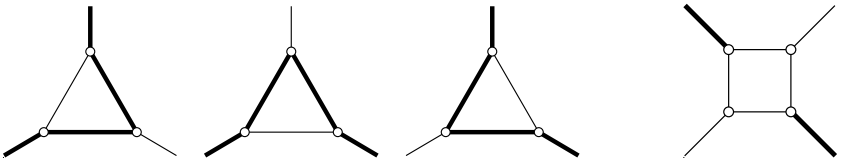


Fig. 1. Left: Case analysis of possible paths a Hamiltonian cycle can take through a triangle. Edges belonging to the Hamiltonian cycle are shown as heavier than the non-cycle edges. Right: Cycle of four unforced edges, with two forced edges adjacent to opposite cycle vertices (step 1(j)).

can solve the forced traveling salesman problem (and therefore also the traveling salesman and Hamiltonian cycle problems) for cubic graphs in time $\mathcal{O}(2^{n/3}) \approx 1.25992^n$ and linear space. We also provide a randomized reduction from degree four graphs to degree three graphs solving the traveling salesman problem in better time than the general case for those graphs. We then consider a weighted counting version of the Hamiltonian cycle problem. Let each edge of G has a weight, and let the weight of a Hamiltonian cycle to be the product of the weights of its edges. We show that the sum of the weights of all Hamiltonian cycles, in graphs with forced edges and maximum degree three, can be found in time $\mathcal{O}(2^{3n/8}n^{\mathcal{O}(1)}) \approx 1.29684^n$. If all weights are one, this sum of cycle weights is exactly the number of Hamiltonian cycles in the graph.

2 The Algorithm and Its Correctness

Our algorithm is based on a simple case-based backtracking technique. Recall that G is a graph with maximum degree 3, while F is a set of edges that must be used in our traveling salesman tour. For simplicity, we describe a version of the algorithm that returns only the *cost* of the optimal tour, or the special value *None* if there is no solution. The tour itself can be reconstructed by keeping track of which branch of the backtracking process led to the returned cost; we omit the details The steps of the algorithm are listed in Table 1. Roughly, our algorithm proceeds in the following stages. Step 1 of the algorithm reduces the size of the input without branching, after which the graph can be assumed to be cubic and triangle-free, with forced edges forming a matching. Step 2 tests for a case in which all unforced edges form disjoint 4-cycles; we can then solve the problem immediately via a minimum spanning tree algorithm. Finally (steps 3-6), we choose an edge to branch on, and divide the solution space into two subspaces, one in which the edge is forced to be in the solution and one in which it is excluded. These two subproblems are solved recursively, and it is our goal to minimize the number of times this recursive branching occurs.

All steps of the algorithm either return or reduce the input graph to one or more smaller graphs that also have maximum degree three, so the algorithm must eventually terminate. To show correctness, each step must preserve the existence and weight of the optimal traveling salesman tour. This is easy to

Table 1. Forced traveling salesman algorithm for graph G and forced edge set F.

1. Repeat the following steps until one of the steps returns or none of them applies:
   a) If G contains a vertex with degree zero or one, return *None*.
   b) If G contains a vertex with degree two, add its incident edges to F.
   c) If F consists of a Hamiltonian cycle, return the cost of this cycle.
   d) If F contains a non-Hamiltonian cycle, return *None*.
   e) If F contains three edges meeting at a vertex, return *None*.
   f) If F contains exactly two edges meeting at some vertex, remove from G that vertex and any other edge incident to it; replace the two edges by a single forced edge connecting their other two endpoints, having as its cost the sum of the costs of the two replaced edges' costs.
   g) If G contains two parallel edges, at least one of which is not in F, and G has more than two vertices, then remove from G whichever of the two edges is unforced and has larger cost.
   h) If G contains a self-loop which is not in F, and G has more than one vertex, remove the self-loop from G.
   i) If G contains a triangle xyz, then for each non-triangle edge e incident to a triangle vertex, increase the cost of e by the cost of the opposite triangle edge. Also, if the triangle edge opposite e belongs to F, add e to F. Remove from G the three triangle edges, and contract the three triangle vertices into a single supervertex.
   j) If G contains a cycle of four unforced edges, two opposite vertices of which are each incident to a forced edge outside the cycle, then add to F all non-cycle edges that are incident to a vertex of the cycle.
2. If $G \setminus F$ forms a collection of disjoint 4-cycles, perform the following steps.
   a) For each 4-cycle C_i in $G \setminus F$, let H_i consist of two opposite edges of C_i, chosen so that the cost of H_i is less than or equal to the cost of $C_i \setminus H_i$.
   b) Let $H = \cup_i H_i$. Then $F \cup H$ is a degree-two spanning subgraph of G, but may not be connected.
   c) Form a graph $G' = (V', E')$, where the vertices of V' consist of the connected components of $F \cup H$. For each set H_i that contains edges from two different components K_j and K_k, draw an edge in E' between the corresponding two vertices, with cost equal to the difference between the costs of C_i and of H_i.
   d) Compute the minimum spanning tree of (G', E').
   e) Return the sum of the costs of $F \cup H$ and of the minimum spanning tree.
3. Choose an edge yz according to the following cases:
   a) If $G \setminus F$ contains a 4-cycle, two vertices of which are adjacent to edges in F, let y be one of the other two vertices of the cycle and let yz be an edge of $G \setminus F$ that does not belong to the cycle.
   b) If there is no such 4-cycle, but F is nonempty, let xy be any edge in F and yz be an adjacent edge in $G \setminus F$.
   c) If F is empty, let yz be any edge in G.
4. Call the algorithm recursively on $G, F \cup \{yz\}$.
5. Call the algorithm recursively on $G \setminus \{yz\}, F$.
6. Return the minimum of the set of at most two numbers returned by the two recursive calls.

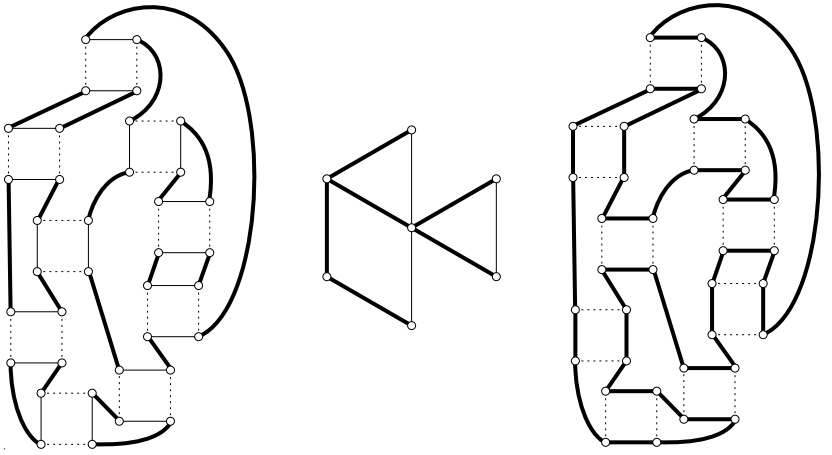


Fig. 2. Step 2 of the traveling salesman algorithm. Left: Graph with forced edges (thick lines), such that the unforced edges form disjoint 4-cycles. In each 4-cycle C_i, the pair H_i of edges with lighter weight is shown as solid, and the heavier two edges are shown dashed. Middle: Graph G', the vertices of which are the connected components of solid edges in the left figure, and the edges of which connect two components that pass through the same 4-cycle. A spanning tree of G' is shown with thick lines. Right: The tour of G corresponding to the spanning tree. The tour includes $C_i \setminus H_i$ when C_i corresponds to a spanning tree edge, and includes H_i otherwise.

verify for most cases of steps 1 and 3–6. Case 1(i) performs a so-called Δ-Y transformation on the graph; case analysis (Figure 1, left) shows that each edge of the contracted triangle participates in a Hamiltonian cycle exactly when the opposite non-triangle edge also participates. Case 1(j) concerns a 4-cycle in G, with edges in F forcing the Hamiltonian cycle to enter or exit on two opposite vertices (Figure 1, right). If a Hamiltonian cycle enters and exits a cycle in G only once, it does so on two adjacent vertices of the cycle, so the 4-cycle of this case is entered and exited twice by every Hamiltonian cycle, and the step's addition of edges to F does not change the set of solutions of the problem.

It remains to prove correctness of step 2 of the algorithm.

Lemma 1. *Suppose that G, F can not be reduced by step 1 of the algorithm described in Table 1, and that $G \setminus F$ forms a collection of disjoint 4-cycles. Then step 2 of the algorithm correctly solves the forced traveling salesman problem in polynomial time for G and F.*

Proof. Let C_i, H_i, H, and G' be as defined in step 2 of the algorithm. Figure 2(left) depicts F as the thick edges, C_i as the thin edges, and H_i and H as the thin solid edges; Figure 2(middle) depicts the corresponding graph G'.

We first show that the weight of the optimal tour T is at least as large as what the algorithm computes. The symmetric difference $T \oplus (F \cup H)$ contains edges only from the 4-cycles C_i. Analysis similar to that for substep 1(j) shows

that, within each 4-cycle C_i, T must contain either the two edges in H_i or the two edges in $C_i \setminus H_i$. Therefore, $T \oplus (F \cup H)$ forms a collection of 4-cycles which is a subset of the 4-cycles in $G \setminus F$ and which corresponds to some subgraph S of G'. Further, due to the way we defined the edge weights in G', the difference between the weights of T and of $F \cup H$ is equal to the weight of S. S must be a connected spanning subgraph of G', for otherwise the vertices in some two components of $F \cup H$ would not be connected to each other in T. Since all edge weights in G' are non-negative, the weight of spanning subgraph S is at least equal to that of the minimum spanning tree of G'.

In the other direction, one can show by induction that, if T' is any spanning tree of G', such as the one shown by the thick edges in Figure 2(middle), and S' is the set of 4-cycles in G corresponding to the edges of T', then $S' \oplus (F \cup H)$ is a Hamiltonian cycle of G with weight equal to that of $F \cup H$ plus the weight of T' (Figure 2(right)). Therefore, the weight of the optimal tour T is at most equal to that of $F \cup H$ plus the weight of the minimum spanning tree of G'.

We have bounded the weight of the traveling salesman tour both above and below by the quantity computed by the algorithm, so the algorithm correctly solves the traveling salesman problem for this class of graphs. □

We summarize our results below.

Theorem 1. *The algorithm described in Table 1 always terminates, and returns the weight of the optimal traveling salesman tour of the input graph G.*

3 Implementation Details

Define a *step* of the algorithm of Table 1 to be a single execution of one of the numbered or lettered items in the algorithm description. As described, each step involves searching for some kind of configuration in the graph, and could therefore take as much as linear time. Although a linear factor is insignificant compared to the exponential time bound of our overall algorithm, it is nevertheless important (and will simplify our bounds) to reduce such factors to the extent possible. As we now show, we can maintain some simple data structures that let us avoid repeatedly searching for configurations in the graph.

Lemma 2. *The algorithm of Table 1 can be implemented in such a way that step 3, and each substep of step 1, take constant time per step.*

Proof. The key observation is that most of these steps and substeps require finding a connected pattern of $\mathcal{O}(1)$ edges in the graph. Since the graph has bounded degree, there can be at most $\mathcal{O}(n)$ matches to any such pattern. We can maintain the set of matches by removing a match from a set whenever one of the graph transformations changes one of its edges, and after each transformation searching within a constant radius of the changed portion of the graph for new matches to add to the set. In this way, finding a matching pattern is a constant time operation (simply pick the first one from the set of known matches), and updating the set of matches is also constant time per operation.

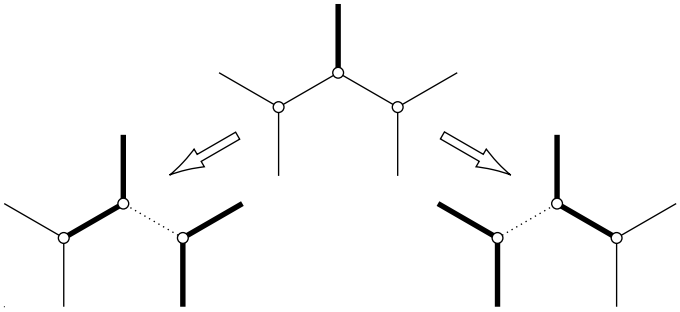


Fig. 3. Result of performing steps 2-5 with no nearby forced edge: one of edges yz and yw becomes forced (shown as thick segments), and the removal of the other edge (shown as dotted) causes two neighboring edges to become forced.

The only two steps for which this technique does not work are 1(c) and 1(d), which each involve finding a cycle of possibly unbounded size in G. However, if a long cycle of forced edges exists, step 1(e) or 1(f) must be applicable to the graph; repeated application of these steps will eventually either discover that the graph is non-Hamiltonian or reduce the cycle to a single self-loop. So we can safely replace 1(c) and 1(d) by steps that search for a one-vertex cycle in F, detect the applicability of the modified steps 1(c) and 1(d) by a finite pattern matching procedure, and use the same technique for maintaining sets of matches described above to solve this pattern matching problem in constant time per step. □

To aid in our analysis, we restrict our implementation so that, when it can choose among several applicable steps, it gives first priority to steps which immediately return (that is, steps 1(a) and 1(c–e), with the modifications to steps 1(c) and 1(d) described in the lemma above), and second priority to step 1(f). The prioritization among the remaining steps is unimportant to our analysis.

4 Analysis

By the results of the previous section, in order to compute an overall time bound for the algorithm outlined in Table 1, we need only estimate the number of steps it performs. Neglecting recursive calls that immediately return, we must count the number of iterations of steps 1(b), 1(f–h), and 3–6.

Lemma 3. *If we prioritize the steps of the algorithm as described in the previous section, the number of iterations of step 1(f) is at most $\mathcal{O}(n)$ plus a number proportional to the number of iterations of the other steps of the algorithm.*

Proof. The algorithm may perform at most $\mathcal{O}(n)$ iterations of step 1(f) prior to executing any other step. After that point, each additional forced edge can cause at most two iterations of step 1(f), merging that edge with previously existing forced edges on either side of it, and each step other than 1(f) creates at most a constant number of new forced edges. □

It remains to count the number of iterations of steps 1(b), 1(g), 1(h), and 3–6. The key idea of the analysis is to bound the number of steps by a recurrence involving a nonstandard measure of the size of a graph G: let $s(G, F) = |V(G)| - |F| - |C|$, where C denotes the set of 4-cycles of G that form connected components of $G \setminus F$. Clearly, $s \leq n$, so a bound on the time complexity of our algorithm in terms of s will lead to a similar bound in terms of n. Equivalently, we can view our analysis as involving a three-parameter recurrence in n, $|F|$, and $|C|$; in recent work [4] we showed that the asymptotic behavior of this type of multivariate recurrence can be analyzed by using weighted combinations of variables to reduce it to a univariate recurrence, similarly to our definition here of s as a combination of n, $|F|$, and $|C|$. Note that step 1(f) leaves s unchanged and the other steps do not increase it.

Lemma 4. *Let a graph G and nonempty forced edge set F be given in which neither an immediate return nor step 1(f) can be performed, and let $s(G, F)$ be as defined above. Then the algorithm of Table 1, within a constant number of steps, reduces the problem to one of the following situations:*

- *a single subproblem G', F', with $s(G', F') \leq s(G, F) - 1$, or*
- *subproblems G_1, F_1 and G_2, F_2, with $s(G_1, F_1), s(G_2, F_2) \leq s(G, F) - 3$, or*
- *subproblems G_1, F_1 and G_2, F_2, with $s(G_1, F_1) \leq s(G, F) - 2$ and $s(G_2, F_2) \leq s(G, F) - 5$.*

Proof. If step 1(b), 1(g), 1(h), or 1(j) applies, the problem is immediately reduced to a single subproblem with more forced edges, and if step 1(i) applies, the number of vertices is reduced. Step 2 provides an immediate return from the algorithm. So, we can restrict our attention to problems in which the algorithm is immediately forced to apply steps 3-6. In such problems, the input must be a simple cubic triangle-free graph, and F must form a matching in this graph, for otherwise one of the earlier steps would apply.

We now analyze cases according to the neighborhood of the edge yz chosen in step 3. To help explain the cases, we let yw denote the third edge of G incident to the same vertex as xy and yz. We also assume that no immediate return is performed within $\mathcal{O}(1)$ steps of the initial problem, for otherwise we would again have reduced the problem to a single smaller subproblem.

- In the first case, corresponding to step 3(a) of the algorithm, yz is adjacent to a 4-cycle in $G \setminus F$ which already is adjacent to two other edges of F. Adding yz to F in the recursive call in step 4 leads to a situation in which step 1(j) applies, adding the fourth adjacent edge of the cycle to F and forming a 4-cycle component of $G \setminus F$. Thus $|F|$ increases by two and $|C|$ increases by one. In step 5, yz is removed from F, following which step 1(b) adds two edges of the 4-cycle to F, step 1(f) contracts these two edges to a single edge, shrinking the 4-cycle to a triangle, and step 1(i) contracts the triangle to a single vertex, so the number of vertices in the graph is decreased by three.
- In the next case, yz is chosen by step 3(b) to be adjacent to forced edge xy, and neither yz nor yw is incident to a second edge in F. If we add yz to

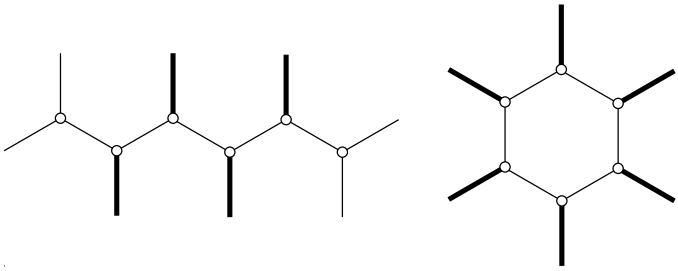


Fig. 4. Chains of two or more vertices each having two adjacent unforced edges. Left: chain terminated by vertices with three unforced edges. Right: cycle of six or more vertices with two unforced edges.

F, an application of step 1(f) removes yw, and another application of step 1(b) adds the two edges adjoining yw to F, so the number of forced edges is increased by three. The subproblem in which we remove yz from F is symmetric. This case and its two subproblems are shown in Figure 3.

- If step 3(b) chooses edge yz, and z or w is incident to a forced edge, then with y it forms part of a chain of two or more vertices, each incident to exactly two unforced edges that connect vertices in the chain. This chain may terminate at vertices with three adjacent unforced edges (Figure 4, left). If it does, a similar analysis to the previous case shows that adding yz to F or removing it from G causes alternating members of the chain to be added to F or removed from G, so that no chain edge is left unforced. In addition, when an edge at the end of the chain is removed from G, two adjacent unforced edges are added to F, so these chains generally lead to a greater reduction in size than the previous case. The smallest reduction happens when the chain consists of exactly two vertices adjacent to forced edges. In this case, one of the two subproblems is formed by adding two new forced edges at the ends of the chain, and removing one edge interior to the chain; it has $s(G_1, F_1) = s(G, F) - 2$. The other subproblem is formed by removing the two edges at the ends of the chain, and adding to F the edge in the middle of the chain and the other unforced edges adjacent to the ends of the chain. None of these other edges can coincide with each other without creating a 4-cycle that would have been treated in the first case of our analysis, so in this case there are five new forced edges and $s(G_2, F_2) = s(G, F) - 5$.
- In the remaining case, step 3(b) chooses an edge belonging to a cycle of unforced edges, each vertex of which is also incident to a forced edge (Figure 4, right). In this case, adding or removing one of the cycle edges causes a chain reaction which alternately adds and removes all cycle edges. This case only arises when the cycle length is five or more, and if it is exactly five then an inconsistency quickly arises causing both recursive calls to return within a constant number of steps. When the cycle length is six or more, both resulting subproblems end up with at least three more forced edges.

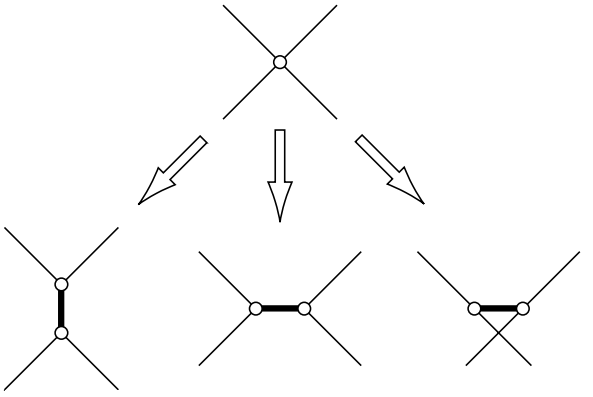


Fig. 5. Reducing degree four vertices to degree three vertices, by randomly splitting vertices and connecting the two sides by a forced edge.

Note that the analysis need not consider choices made by step 3(c) of the algorithm, as F is assumed nonempty; step 3(c) can occur only once and does not contribute to the asymptotic complexity of the algorithm. In all cases, the graph is reduced to subproblems that have sizes bounded as stated in the lemma. □

Theorem 2. *The algorithm of Table 1 solves the forced traveling salesman problem on graphs of degree three in time* $\mathcal{O}(2^{n/3})$.

Proof. The algorithm's correctness has already been discussed. By Lemmas 1, 2, 3, and 4, the time for the algorithm can be bounded within a constant factor by the solution to the recurrence

$$T(s) \leq 1 + \max\{s^{\mathcal{O}(1)}, T(s-1), 2T(s-3), T(s-2) + T(s-5)\}.$$

Standard techniques for linear recurrences give the solution as $T(s) = \mathcal{O}(2^{s/3})$. In any n-vertex cubic graph, s is at most n, so expressed in terms of n this gives a bound of $\mathcal{O}(2^{n/3})$ on the running time of the algorithm. □

5 Degree Four

It is natural to ask to what extent our algorithm can be generalized to higher vertex degrees. We provide a first step in this direction, by describing a randomized (Monte Carlo) algorithm: that is, an algorithm that may produce incorrect results with bounded probability. To describe the algorithm, let f denote the number of degree four vertices in the given graph. The algorithm consists of $(3/2)^f$ repetitions of the following: for each degree four vertex, choose randomly among the three possible partitions of its incoming edges into two sets of two edges; split the vertex into two vertices, with the edges assigned to one or the other vertex according to the partition, and connect the two vertices by a new

forced edge (Figure 5). Once all vertices are split, the graph has maximum degree 3 and we can apply our previous forced TSP algorithm.

It is not hard to see that each such split preserves the traveling salesman tour only when the two tour edges do not belong to the same set of the partition, which happens with probability 2/3; therefore, each repetition of the algorithm has probability $(2/3)^f$ of finding the correct TSP solution. Since there are $(3/2)^f$ repetitions, there is a bounded probability that the overall algorithm finds the correct solution. Each split leaves unchanged the parameter s used in our analysis of the algorithm for cubic graphs, so the time for the algorithm is $\mathcal{O}((3/2)^f 2^{n/3}) = \mathcal{O}((27/4)^{n/3})$. By increasing the number of repetitions the failure probability can be made exponentially small with only a polynomial increase in runtime. We omit the details as our time bound for this case seems unlikely to be optimal.

6 Weighted Counting

Along with NP-complete problems such as finding traveling salesman tours, it is also of interest to solve #P-complete problems such as counting Hamiltonian cycles. More generally, we consider the following *weighted counting* problem: the edges of G are assigned weights from a *commutative semiring*: that is, an algebraic system with commutative and associative multiplication and addition operations, containing an additive identity, and obeying the distributive law of multiplication over addition. For each Hamiltonian cycle in G, we form the product of the weights of the edges in the cycle, and then sum the products for all cycles, to form the *value* of the problem.

The traveling salesman problem itself can be viewed as a special case of this semiring weighted counting problem, for a semiring in which the multiplication operation is the usual real number addition, and the addition operation is real number minimization. The additive identity in this case can be defined to be the non-numeric value $+\infty$. The problem of counting Hamiltonian cycles can also be viewed in this framework, by using the usual real number multiplication and addition operations to form a semiring (with additive identity zero) and assigning unit weight to all edges.

As we show in Table 2, most of the steps of our traveling salesman algorithm can be generalized in a straightforward way to this semiring setting. However, we do not know of a semiring analogue to the minimum spanning tree algorithm described in step 2 of Table 1, and proven correct in Lemma 1 for graphs in which the unforced edges form disjoint 4-cycles. It is tempting to try using the matrix-tree theorem to count spanning trees instead of computing minimum spanning trees, however not every Hamiltonian cycle of the input graph G corresponds to a spanning tree of the derived graph G' used in that step. Omitting the steps related to these 4-cycles gives the simplified algorithm shown in Table 2. We analyze this algorithm in a similar way to the previous one; however in this case we use as the parameter of our analysis the number of unforced edges $U(G)$ in

Table 2. Forced Hamiltonian cycle counting algorithm for graph G, forced edges F.

1. Repeat the following steps until one of the steps returns or none of them applies:
   a) If G contains a vertex with degree zero or one, return zero.
   b) If G contains a vertex with degree two, add its incident edges to F.
   c) If F consists of a Hamiltonian cycle, return the product of edge weights of this cycle.
   d) If F contains a non-Hamiltonian cycle, return zero.
   e) If F contains three edges meeting at a vertex, return zero.
   f) If F contains exactly two edges meeting at some vertex, remove from G that vertex and any other edge incident to it; replace the two edges by a single edge connecting their other two endpoints, having as its weight the product of the weights of the two replaced edges' costs.
   g) If G contains two parallel edges, exactly one of which is in F, and G has more than two vertices, remove the unforced parallel edge from G.
   h) If G contains two parallel edges, neither one of which is in F, and G has more than two vertices, replace the two edges by a single edge having as its weight the sum of the weights of the two edges.
   i) If G contains a self-loop which is not in F, and G has more than one vertex, remove the self-loop from G.
   j) If G contains a triangle xyz, then for each non-triangle edge e incident to a triangle vertex, multiply the weight of e by the weight of the opposite triangle edge. Also, if the triangle edge opposite e belongs to F, add e to F. Remove from G the three triangle edges, and contract the three triangle vertices into a single supervertex.
2. If F is nonempty, let xy be any edge in F and yz be an adjacent edge in $G \setminus F$. Otherwise, if F is empty, let yz be any edge in G.
3. Call the algorithm recursively on $G, F \cup \{yz\}$.
4. Call the algorithm recursively on $G \setminus \{yz\}, F$.
5. Return the sum of the two numbers returned by the two recursive calls.

the graph G. Like $s(G)$, U does not increase at any step of the algorithm; we now show that it decreases by sufficiently large amounts at certain key steps.

Lemma 5. *Let a graph G be given in which neither an immediate return nor step 1(f) can be performed, let F be nonempty, and let $U(G)$ denote the number of unforced edges in G. Then the algorithm of Table 2, within a constant number of steps, reduces the problem to one of the following situations:*

- *a single subproblem G', with $U(G') \leq U(G) - 1$, or*
- *two subproblems G_1 and G_2, with $U(G_1), U(G_2) \leq U(G) - 4$, or*
- *two subproblems G_1 and G_2, with $U(G_1) \leq U(G) - 3$ and $U(G_2) \leq U(G) - 6$.*

We omit the proof, which is similar to that for Lemma 4.

Theorem 3. *For any graph G with maximum degree 3, set F of forced edges in G, and assignment of weights to the edges of G from a commutative semiring, we can compute the semiring sum, over all forced Hamiltonian cycles in G, of the product of weights of the edges in each cycle, in $\mathcal{O}(2^{3n/8})$ semiring operations.*

Proof. By the previous lemma, the number of semiring operations in the algorithm can be bounded within a constant factor by the solution to the recurrence

$$T(u) \leq 1 + \max\{T(u-1), 2T(u-4), T(u-3) + T(u-6)\}.$$

Standard techniques for linear recurrences give the solution as $T(u) = \mathcal{O}(2^{u/4})$. In any n-vertex cubic graph, u is at most $3n/2$, so expressed in terms of n this gives a bound of $\mathcal{O}(2^{3n/8})$ on the number of operations. □

Corollary 1. *We can count the number of Hamiltonian cycles in any cubic graph in time* $\mathcal{O}(2^{3n/8}n^{\mathcal{O}(1)})$.

The extra polynomial factor in this time bound accounts for the time to perform each multiplication and addition of the large numbers involved in the counting algorithm. However, the numbers seem likely to become large only at the higher levels of the recursion tree, while the bulk of the algorithm's time is spent near the leaves of the tree, so perhaps this factor can be removed.

References

1. R. Beigel. Finding maximum independent sets in sparse and general graphs. *Proc. 10th ACM-SIAM Symp. Discrete Algorithms*. pp. S856–S857, January 1999, http://www.eecs.uic.edu/~beigel/papers/MIS-SODA:ps:GZ:
2. J.M. Byskov. Chromatic number in time $\mathcal{O}(2.4023^n)$ using maximal independent sets. Tech. Rep. RS-02-45, BRICS. December 2002
3. D. Eppstein. Small maximal independent sets and faster exact graph coloring. *Proc. 7th Worksh. Algorithms and Data Structures*, pp. 462–470. Springer-Verlag. Lecture Notes in Computer Science 2125, August 2001, arXiv:cs.Ds/0011009.
4. D. Eppstein. Quasiconvex analysis of backtracking algorithms. ACM Computing Research Repository, April 2003, arXiv:cs.DS./0304018.
5. M.R. Garey and D.S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979
6. E.L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters* 5(3):66–67, August 1976
7. J.M. Robson. Algorithms for maximum independent sets. *J. Algorithms* 7(3):425–440, September 1986
8. R.E Tarjan and A. Trojanowski. Finding a maximum independent set. *SIAM J. Comput.* 6(3):537–546, September 1977

Sorting Circular Permutations by Reversal

Andrew Solomon, Paul Sutcliffe, and Raymond Lister

University of Technology, Sydney, Australia
{andrews,psutclif,raymond}@it.uts.edu.au

Abstract. Unsigned circular permutations are used to represent tours in the traveling salesman problem as well as the arrangement of gene loci in circular chromosomes. The minimum number of segment reversals required to transform one circular permutation into another gives some measure of distance between them which is useful when studying the 2-opt local search landscape for the traveling salesman problem, and, when determining the phylogeny of a group of related organisms. Computing this distance is equivalent to sorting by (a minimum number of) reversals. In this paper we show that sorting circular permutations by reversals can be reduced to the same problem for linear reversals, and that it is NP-hard. These results suggest that for most practical purposes any computational tools available for reversal sort of linear permutations will be sufficiently accurate.
These results entail the development of the algebraic machinery for dealing rigorously with circular permutations.

1 Introduction

A circular permutation can be thought of as a necklace with n distinct beads. Rotating and flipping the necklace do not change the object but one necklace may be transformed into any other by cutting it in two places, reversing one segment and rejoining the ends, or a composition of such operations. This paper addresses the problem of finding a minimum length sequence of segment reversals required to transform one circular permutation into another.

Tours in the symmetric traveling salesman problem are precisely circular permutations of the cities. In the context of the traveling salesman problem, segment reversal is called a *2-opt move* and is used to define a combinatorial landscape which is subjected to local search techniques in order to find local minima [11].

Among others, Boese [4] suggests a correlation between the values of local minima and their distance from other local minima – the so called "big valley" hypothesis which informs a number of successful heuristics for traversing the landscape. Boese uses the number of breakpoints (pairs which are adjacent in one permutation, but not the other) as an estimate of reversal distance. Our motivation for the present work is to have a more accurate measure of reversal distance for the purpose of investigating the big valley hypothesis.

Historically, the question of determining reversal distance between circular permutations was first posed in 1982 by Watterson et. al. [14] in the context of

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 319–328, 2003.
© Springer-Verlag Berlin Heidelberg 2003

computational biology. A circular permutation models the arrangement of gene loci around a circular chromosome such as is found in bacterial and mitochondrial DNA. While point mutations occur freqently, the order of gene loci is quite stable over the generations. When the gene order does change, it is usually by segment reversal [5]. Therefore, the reversal distance between two chromosomes is a measure of their evolutionary distance.

Watterson's paper gives the first rudimentary bounds on the reversal distance where it is noted that since each reversal can eliminate at most two breakpoints, half the number of breakpoints is a lower bound on the reversal distance. A simple ratchet algorithm is given to show that for circular permutations of n points, at most n reversals are required to transform one permutation into any other.

This paper sparked a great deal of interest but subsequent investigations focussed on the simpler case of linear chromosomes. In the remainder of this section we review the progress made in the linear case.

1.1 Sorting Linear Permutations by Reversal

As we shall see, the problem of finding a minimum length sequence of reversals transforming one permutation into another is equivalent to sorting a permutation by (a minimum number of) segment reversals. Henceforth we use the initials SBR to refer to this problem.

Kececioglu and Sankoff [9,10] give a greedy algorithm for sorting a permutation σ by reversal bounded above by the number $b(\sigma)$ of breakpoints which is therefore a 2-approximation algorithm. Computing upper and lower bounds on reversal distance, Kececioglu and Sankoff go on to give an algorithm for computing an exact reversal sort for linear permutations.

Bafna and Pevzner [2] improved on these results by formulating a $\frac{7}{4}$-approximation algorithm to sort a permutation by reversals. Along the way they defined the problem of sorting *signed* permutations by reversals, where each point has not only a position, but also an orientation. Signed permutations are arguably more significant biologically as genes have extension as well as position.

Using elaborate graph theoretic constructions, Caprara [5] solved in the affirmative a longstanding conjecture of Kececioglu and Sankoff that sorting by reversals is NP-hard. In contrast Hannenhalli and Pevzner [8] give a polynomial algorithm which sorts signed permutations.

David A. Christie [6] finds a polynomial time $\frac{3}{2}$-approximation algorithm for sorting unsigned permutations by reversals, and this remains the best known approximation factor.

Bounding approximability of SBR, Berman and Karpinski [3] show that it is NP-hard to approximate the reversal length of a linear permutation to a factor better than 1.0008.

1.2 Notational Preliminaries

Formalizing the notion of a circular permutation is a delicate matter and clarity is well served by taking some pains to carefully define the notion of a permutation.

Linear Permutations. Rather than regarding a permutation as a function from a set to itself, we distinguish the set Σ of n objects being permuted, from the ordered set $[n] = \{0, 1, \ldots n-1\}$ of positions in which we place each object. Then a permutation is a bijection $\pi : \Sigma \to [n]$ such that $a\pi$ denotes the position in which we place object $a \in \Sigma$, that is, permutations act on the right. Occasionally, it will be convenient to visualize all the objects of Σ in their positions under π as $(\pi_0\pi_1 \ldots \pi_{n-1})$, which is to say $\pi_i = i\pi^{-1}$.

Fix some permutation $\iota : \Sigma \to [n]$. Then ι defines a canonical ordering of the elements of Σ and ι will be called the *identity* permutation. Then $\Sigma = \{\iota_0, \ldots, \iota_{n-1}\}$. (Identifying the sets $[n]$ and Σ would enable us to revert to the usual notions of permutation and identity.)

A *reversal* $\rho(i, j)$ (with $i < j$ in $[n]$) of a linear permutation is a bijection on the set of positions whose effect can be visualized as reversing the segment from position i to position j, transforming $(\pi_0 \ldots \pi_i\pi_{i+1} \ldots \pi_j \ldots \pi_{n-1})$ into $(\pi_0 \ldots \pi_j\pi_{j-1} \ldots \pi_{i+1}\pi_i\pi_{j+1} \ldots \pi_{n-1})$. Precisely, for $x \in \Sigma$ define

$$\begin{aligned} \rho(i,j) : [n] &\longrightarrow [n] \\ x &\longmapsto \begin{cases} i + j - x & \text{if } i \le x \le j \\ x & \text{otherwise.} \end{cases} \end{aligned}$$

then it is easy to see that $\pi\rho(i, j) = (\pi_0 \ldots \pi_j\pi_{j-1} \ldots \pi_i\pi_{j+1} \ldots \pi_{n-1})$ as required.

Circular Permutations. The notion of circular permutation we are trying to capture is an arrangement of the elements of Σ around the vertices of a regular n-gon subject to the condition that, like a necklace, rotating the n-gon, or flipping it over does not change the circular permutation that it represents.

Arbitrarily, we label the vertices of the n-gon by the elements of $\mathbb{Z}_n$ from 0 at the twelve o'clock position and proceed clockwise up to $n - 1$. A *circular arrangement* of the elements of Σ around the vertices of the n-gon is then a bijection $\pi : \Sigma \to \mathbb{Z}_n$. In a similar way to the treatment of linear permutations, fix an arbitrary circular arrangement $\iota : \Sigma \to \mathbb{Z}_n$ and refer to ι as the *identity* arrangement. For $i \in \mathbb{Z}_n$ define the *elementary rotation* $r : \mathbb{Z}_n \to \mathbb{Z}_n$ by $ir = i \oplus 1$ and *canonical reflection* $s : \mathbb{Z}_n \to \mathbb{Z}_n$ by $is = \ominus i$, where $\oplus$ and $\ominus$ denote addition and negation (or subtraction) in $\mathbb{Z}_n$. For example, $(\pi_0\pi_1\pi_2\pi_3\pi_4)r = (\pi_4\pi_0\pi_1\pi_2\pi_3)$ and $(\pi_0\pi_1\pi_2\pi_3\pi_4)s = (\pi_0\pi_4\pi_3\pi_2\pi_1)$.

The maps r and s generate all $2n$ rotations and reflections of the regular n-gon. Together, these form the *dihedral group* D_n, which has presentation

$$\langle r, s \mid s^2, r^n, rs = sr^{n-1} \rangle. \tag{1}$$

To capture the idea that π, πr and πs all represent the same circular permutation, define a *circular permutation* to be the set πD_n for some circular arrangement π. It is then clear that $\pi D_n = \pi r D_n = \pi s D_n$ as required. Any circular arrangement in πD_n defines a linear permutation by identifying $\mathbb{Z}_n$ with $[n]$. Call such a permutation a *linearization* of πD_n and denote the set of all $2n$ linearizations of πD_n by $\text{lin}(\pi D_n)$.

For $i, j \in \mathbb{Z}_n$, define the *interval* $[i, j]$ to be the set $\{i, i \oplus 1, \ldots, j \ominus 1, j\}$. For example if n is 6 then $[3, 1] = \{3, 4, 5, 0, 1\}$ while $[1, 3] = \{1, 2, 3\}$.

Let $x \in \Sigma$. Then a *circular reversal* $\rho^c(i, j)$ is defined by

$$\begin{aligned} \rho^c(i,j) : \mathbb{Z}_n &\to \mathbb{Z}_n \\ x &\longmapsto \begin{cases} i \oplus j \ominus x & \text{if } x \in [i,j] \\ x & \text{otherwise.} \end{cases} \end{aligned}$$

As an example of the way a circular reversal acts on a circular arrangement, notice that when $n = 6$,

$$(\pi_0\pi_1\pi_2\pi_3\pi_4\pi_5)\rho^c(1,3) = (\pi_0\pi_3\pi_2\pi_1\pi_4\pi_5)$$

and

$$(\pi_0\pi_1\pi_2\pi_3\pi_4\pi_5)\rho^c(4,1) = (\pi_5\pi_4\pi_2\pi_3\pi_1\pi_0).$$

The technical report [12] inspired a number of notational decisions in this section. In particular, the symbols used to denote reversal, arithmetic in $\mathbb{Z}_n$ and intervals appear also in [12].

1.3 Mathematical Preliminaries

In the linear case, the problem one attempts to solve is to find, for two permutations σ and τ a minimum length sequence of reversals $\alpha_1, \ldots, \alpha_k$ such that $\sigma\alpha_1 \ldots \alpha_k = \tau$, however

$$\begin{aligned} \sigma\alpha_1 \ldots \alpha_k = \tau \text{ if and only if} \\ \iota\tau^{-1}\sigma\alpha_1 \ldots \alpha_k = \iota \end{aligned}$$

and since ι is the identity permutation, we see that a minimum length sequence of reversals transforming σ into τ is equivalent to a minimum length sequence of reversals which transforms $\iota\tau^{-1}\sigma$ into ι, which is to say, the sequence of reversals *sorts* $\iota\tau^{-1}\sigma$. The reversal distance between a permutation π and the identity will be called the *reversal length* of the permutation and denoted $l(\pi)$.

In the circular case, the primary problem is to find, given two circular arrangements σ and τ, a minimum length sequence of circular reversals $\alpha_1^c, \ldots, \alpha_k^c$ such that $\sigma\alpha_1^c \ldots \alpha_k^c \in \tau D_n$. Once again, notice that $\sigma\alpha_1^c \ldots \alpha_k^c \in \tau D_n$ if and only if $\iota\tau^{-1}\sigma\alpha_1^c \ldots \alpha_k^c \in \iota D_n$. Regarding ιD_n as the *identity* circular permutation, we see that the sequence $\alpha_1^c, \ldots, \alpha_k^c$ *sorts* the circular arrangement $\iota\tau^{-1}\sigma$.

The reversal distance between a circular arrangement and the identity will be called the *reversal length* of the arrangement.

For the remainder of the paper, fix some n as the size of Σ and let r denote the elementary rotation and s the canonical reflection in D_n. We give some useful facts describing the interaction of circular reversals with the elements of the dihedral group D_n.

Lemma 1.

$$s\rho^c(i,j) = r^{i\oplus j}\rho^c(j\oplus 1, i\ominus 1).$$

Proof. Noting that for any $x \in \mathbb{Z}_n$, $x \in [i,j]$ if and only if $\ominus x \in [\ominus j, \ominus i]$ we have

$$xs\rho^c(i,j) = \begin{cases} i\oplus j\oplus x & \text{if } x \in [\ominus j, \ominus i] \\ \ominus x & \text{otherwise} \end{cases}$$

while

$$\begin{aligned} xr^{i\oplus j}\rho^c(j\oplus 1, i\ominus 1) &= \begin{cases} (j\oplus 1)\oplus(i\ominus 1)\ominus(x\oplus i\oplus j) & \text{if } x\oplus i\oplus j \in [j\oplus 1, i\ominus 1] \\ x\oplus i\oplus j & \text{otherwise} \end{cases} \\ &= \begin{cases} \ominus x & \text{if } x \in [1\ominus i, \ominus 1\ominus j], \text{ subtracting } i\oplus j \text{ everywhere} \\ x\oplus i\oplus j & \text{otherwise} \end{cases} \\ &= \begin{cases} x\oplus i\oplus j & \text{if } x \in [\ominus j, \ominus i] \\ \ominus x & \text{otherwise} \end{cases} \end{aligned}$$

as required.

The reader may easily verify the following equations.

$Eqn-I$	$r\rho^c(i,j) = \rho^c(i\ominus 1, j\ominus 1)r$
$Eqn-II$	$s\rho^c(i,j) = \rho^c(\ominus j, \ominus i)s$
$Eqn-III$	$\rho^c(i,j) = sr^{i\oplus j}\rho^c(j\oplus 1, i\ominus 1) = \rho^c(j\oplus 1, i\ominus 1)sr^{i\oplus j}$
$Eqn-IV$	$\rho^c(i\oplus 1, i) = sr^{2i\oplus 1}$

Eqn-I and Eqn-II ensure that for any $\rho^c(i,j)$ and any $d \in D_n$, $d\rho^c(i,j) = \rho^c(i',j')d$ for some $i', j' \in \mathbb{Z}_n$. Suppose there is a sequence $\alpha_1, \ldots, \alpha_k$ of reversals such that $\sigma\alpha_1\ldots\alpha_k \in \iota D_n$. Then for any $\tau \in \sigma D_n$, $\tau = \sigma d$ so that $\sigma = \tau d^{-1}$ and

$$\begin{aligned} \sigma\alpha_1\ldots\alpha_k &= \tau d^{-1}\alpha_1\ldots\alpha_k \\ &= \tau\beta_1\ldots\beta_k d^{-1} \in \iota D_n \end{aligned}$$

for some reversals $\beta_1, \ldots, \beta_k$, so that $\tau\beta_1\ldots\beta_k \in \iota D_n$. Consequently, τ has length at most k. By symmetry, this shows that any two circular arrangements in the same circular permutation have the same length, so we may speak of the *reversal length of a circular permutation* πD_n and denote it by $l^c(\pi D_n)$.

Proposition 2. *The following table expresses each non-identity element of D_n as a minimum length product of linear reversals.*

Element of D_n	*As reversals*
Orientation preserving elements	
$r^i, i \in \{1, 2\}$	$\rho(0, n-i)\rho(1, n-1)$
$r^i, 2 < i < n-2$	$\rho(0, n-i-1)\rho(n-i, n-1)\rho(0, n-1)$
$r^i, i \in \{n-2, n-1\}$	$\rho(1, n-1)\rho(0, i)$
Orientation reversing elements	
s	$\rho(1, n-1)$
$sr^i, 0 < i < n-2$	$\rho(0, i)\rho(i+1, n-1)$
$sr^i, i \in \{n-2, n-1\}$	$\rho(0, i)$

Proof. To verify the equality of the expressions on the left and right is an easy exercise. The proof that the expressions on the right hand side are of minimum length is tedious and inessential to the development of the remainder of the paper, so we omit it.

2 Reducing Circular SBR to Linear SBR

It is clear that if a sequence $\alpha_1, \ldots, \alpha_k$ sorts a linearization of πD_n then it certainly sorts some circular arrangement of πD_n so that the reversal length of πD_n is bounded above by the minimum reversal length amongst its linearizations.

Theorem 3. *If πD_n can be sorted in m circular reversals, then there is some linearization $\sigma \in \text{lin}(\pi D_n)$ which can be sorted in at most m linear reversals.*

A direct result is that $l^c(\pi D_n)$ is bounded below by the minimum length amongst its reversals so that together with the observation above, we have

Corollary 4. *$l^c(\pi D_n)$ is precisely the minimum value of $l(\sigma)$ for any linearization σ of πD_n.*

Proof (of theorem). By way of a basis for an induction on m, suppose πD_n has reversal length 0. Then $\pi \in \iota D_n$, whence $\iota = \pi t$ for some $t \in D_n$. Consequently, the linearization πt of πD_n is sorted and has a reversal length of 0 as required.

Now suppose πD_n has reversal length m. That is, there is a sequence of circular reversals $\alpha_1^c, \ldots, \alpha_m^c$ such that $\pi\alpha_1^c \ldots \alpha_m^c \in \iota D_n$. Put $\pi_0 = \pi$ and for $1 \leq i \leq m$, set $\pi_i = \pi\alpha_1^c \ldots \alpha_i^c$.

By the inductive hypothesis, there is some linearization $\sigma_1 \in \text{lin}(\pi_1 D_n)$ which is sortable in $m-1$ linear reversals. Say $\gamma_2, \ldots, \gamma_m$ is a sequence of linear reversals sorting σ_1.

We now focus on the relationship between the linear permutation σ_1 and the circular arrangement $\pi_1 = \pi_0\alpha_1 = \pi_0\rho^c(i, j)$ for some $i, j \in \mathbb{Z}_n$. The presentation at (1) shows that an element of the dihedral group may always be written as a rotation, or as a reflection followed by a rotation, giving us only two cases to consider: Case (i) $\sigma_1 = \pi_1 r^k$; Case (ii) $\sigma_1 = \pi_1 s r^k$.

In Case (i) $\sigma_1 = \pi_0\rho^c(i,j)r^k$, and by Eqn-I $\sigma_1 = \pi_0 r^k \rho^c(i \oplus k, j \oplus k)$. There are three subcases to consider: as elements of $\mathbb{Z}$ either (a) $i \oplus k \leq j \oplus k$, (b) $i \oplus k = j \oplus k \oplus 1$, or (c) $i \oplus k > j \oplus k \oplus 1$. In case (a), set $\sigma_0 = \pi_0 r^k$ and $\gamma_1 = \rho(i \oplus k, j \oplus k)$. This gives $\sigma_0\gamma_1 = \sigma_1$ and the sequence $\gamma_1, \gamma_2, \ldots, \gamma_m$ linearly sorts $\sigma_0 = \pi_0 r^k$ as required. In case (b), Eqn-IV gives $\rho^c(i \oplus k, j \oplus k) = sr^{2j \oplus 2k \oplus 1}$ so that

$$\begin{aligned}\sigma_1 &= \pi_0 r^k s r^{2j \oplus 2k \oplus 1} \\ &= \pi_0 s r^{2j \oplus k \oplus 1}\end{aligned}$$

Putting $\sigma_0 = \pi_0 s r^{2j \oplus k \oplus 1}$ and $\gamma_1 = 1_{\mathbb{Z}_n}$, gives the required sequence of linear reversals. In case (c) Eqn-III gives $\rho^c(i \oplus k, j \oplus k) = sr^{i \oplus j \oplus 2k}\rho^c(j \oplus k \oplus 1, i \oplus k \ominus 1)$ so that

$$\begin{aligned}\sigma_1 &= \pi_0 r^k \rho^c(i \oplus k, j \oplus k) \\ &= \pi_0 r^k s r^{i \oplus j \oplus 2k} \rho^c(j \oplus k \oplus 1, i \oplus k \ominus 1) \\ &= \pi_0 s r^{i \oplus j \oplus k} \rho^c(j \oplus k \oplus 1, i \oplus k \ominus 1)\end{aligned}$$

Since $i \oplus k > j \oplus k \oplus 1$, $j \oplus k \oplus 1 \leq i \oplus k \ominus 1$ so that $\rho(j \oplus k \oplus 1, i \oplus k \ominus 1)$ is a linear reversal. Putting $\sigma_0 = \pi_0 s r^{i \oplus j \oplus k}$, and $\gamma_1 = \rho(j \oplus k \oplus 1, i \oplus k \ominus 1)$ then ensures that the sequence $\gamma_1, \ldots, \gamma_m$ sorts σ_0 linearly as required.

In Case (ii)

$$\begin{aligned}\sigma_1 &= \pi_1 s r^k \\ &= \pi_0 \rho^c(i,j) s r^k \\ &= \pi_0 s \rho^c(\ominus j, \ominus i) r^k \\ &= \pi_0 s r^k \rho^c(k \ominus j, k \ominus i).\end{aligned}$$

As above, there are three subcases to consider: as elements of $\mathbb{Z}$ either (a) $k \ominus j \leq k \ominus i$, (b) $k \ominus j = k \ominus i \oplus 1$, or (c) $k \ominus j > k \ominus i \oplus 1$. In case (a) put $\sigma_0 = \pi_0 s r^k$ and $\gamma_1 = \rho(k \ominus j, k \ominus i)$ and $\gamma_1, \ldots, \gamma_m$ is the required sequence of linear reversals which sorts σ_0. In case (b), Eqn-IV gives $\rho^c(k \ominus j, k \ominus i) = sr^{2k \ominus 2i \oplus 1}$ so that

$$\begin{aligned}\sigma_1 &= \pi_0 s r^k s r^{2k \ominus 2i \oplus 1} \\ &= \pi_0 r^{k \ominus 2i \oplus 1}\end{aligned}$$

Putting $\sigma_0 = \pi_0 r^{k \ominus 2i \oplus 1}$ and $\gamma_1 = 1_{\mathbb{Z}_n}$, gives the required sequence of linear reversals. Finally, in case (c) Eqn-III gives $\rho^c(k \ominus j, k \ominus i) = sr^{2k \ominus i \ominus j}\rho^c(k \ominus i \oplus 1, k \ominus j \ominus 1)$ so that

$$\begin{aligned}\sigma_1 &= \pi_0 s r^k \rho^c(i \oplus k, j \oplus k) \\ &= \pi_0 s r^k s r^{2k \ominus i \ominus j} \rho^c(k \ominus i \oplus 1, k \ominus j \ominus 1) \\ &= \pi_0 r^{k \ominus i \ominus j} \rho^c(k \ominus i \oplus 1, k \ominus j \ominus 1)\end{aligned}$$

Since $k \ominus j > k \ominus i \oplus 1$, $k \ominus i \oplus 1 \leq k \ominus j \ominus 1$ so that $\rho(k \ominus i \oplus 1, k \ominus j \ominus 1)$ is a linear reversal. Putting $\sigma_0 = \pi_0 r^{k \ominus i \ominus j}$, and $\gamma_1 = \rho(k \ominus i \oplus 1, k \ominus j \ominus 1)$ then ensures that the sequence $\gamma_1, \ldots, \gamma_m$ sorts σ_0 linearly as required. □

In summary, we see that given an algorithm $\mathcal{L}$ to solve the minimum length SBR problem for linear permutations, in order to solve SBR for some circular permutation πD_n, we need only apply $\mathcal{L}$ to each of the $2n$ linearizations of πD_n, and take the shortest solution.

3 Circular Sort by Reversals Is NP-Hard

By recourse to a result of Berman and Karpinski [3] on the inapproximability of linear SBR, we show that circular SBR is NP-hard. The core of our proof is the following

Lemma 5. *Let $\alpha_1, \ldots, \alpha_m$ be a sequence of circular reversals taking σ to an element of ιD_n. Then there is a sequence $\beta_1, \ldots, \beta_k$ of linear reversals such that $k \leq m$ and $\sigma\beta_1 \ldots \beta_k \in \iota D_n$.*

Proof. Proceed by induction on m. The $m = 0$ case is trivial. If α_1 is a linear reversal, put $\beta_1 = \alpha_1$ and appeal to the inductive hypothesis with the permutation $\sigma\beta_1$.

Therefore we may assume that α_1 is not a linear reversal. That is: $\alpha_1 = \rho^c(x, y)$ with $x > y$. There are two cases: (i) $\alpha_1 = \rho^c(i+1, i)$; and (ii) $\alpha_1 = \rho^c(i, j)$ with $i > j + 1$.

In case (i) Eqn-IV gives $\alpha_1^c = sr^{2i \oplus 1}$. By use of Eqn-I and Eqn-II

$$\begin{aligned} \sigma\alpha_1 \ldots \alpha_m &= \sigma sr^{2i \oplus 1} \alpha_2 \ldots \alpha_m \\ &= \sigma\alpha_2' \ldots \alpha_m' sr^{2i \oplus 1} \end{aligned}$$

so that $\sigma\alpha_2' \ldots \alpha_m' \in \iota D_n$ and we are finished by appeal to the inductive hypothesis.

In case (ii) Eqn-III gives $\alpha_1 = sr^{i \oplus j}\rho(j \oplus 1, i \ominus 1)$ and $i > j + 1$ ensures $j \oplus 1 \leq i \ominus 1$. By Eqn-III we have $\alpha_1 = \rho(j \oplus 1, i \ominus 1)sr^{i \oplus j}$.

Therefore

$$\begin{aligned} \sigma\alpha_1 &= \sigma\rho(j \oplus 1, i \ominus 1)sr^{i \oplus j} \alpha_2 \ldots \alpha_m \\ &= \sigma\rho(j \oplus 1, i \ominus 1)\alpha_2' \ldots \alpha_m' sr^{i \oplus j} \in \iota D_n \end{aligned}$$

so that setting $\beta_1 = \rho(j \oplus 1, i \ominus 1)$, $\sigma\beta_1$ is circularly sorted in $m - 1$ circular reversals, which completes the proof by appeal to the inductive hypothesis.

As an immediate consequence of Lemma 5 and Proposition 2 we have

Proposition 6. *For any linear permutation σ,*

$$l^c(\sigma D_n) \leq l(\sigma) \leq l^c(\sigma D_n) + 3.\square$$

Theorem 7 (Restatement of Theorem 6 in [3]). *For any positive ϵ_1, ϵ_2 it is NP-hard to distinguish linear permutations with $2240k$ breakpoints that have length below $(1236 + \epsilon_1)k$ from those whose length is above $(1237 - \epsilon_2)k$.*

In particular, setting $k = 4m$ and bounding ϵ_1, ϵ_2 we have

Corollary 8. *For $0 < \epsilon_1, \epsilon_2 < \frac{1}{10}$ it is NP-hard to distinguish between linear permutations with $2440 \times 4m$ breakpoints that have length below $l = (1236+\epsilon_1)4m$ and those with length above $u = (1237 - \epsilon_2)4m$. Note that*

$$u - l = 4m - (\epsilon_1 + \epsilon_2)4m > 3\frac{1}{5}m > 3.$$

Finally, we are in a position to prove

Theorem 9. *The problem of computing the reversal length of a circular permutation is NP-hard.*

Proof. We show that the problem of estimating the length of a linear permutation with precision determined by Corollary 8 can be reduced in constant time to the problem of computing the reversal length of the associated circular permutation. Consequently the latter problem must be NP-hard.

To estimate the length of a linear permutation σ, compute the reversal length $l^c(\sigma D_n)$ of the corresponding circular permutation. The reversal length of σ is then approximated by Proposition 6.

With l and u defined as in Corollary 8, let σ be a permutation whose reversal length $l(\sigma)$ is either below l or above u. We show that $l(\sigma) < l$ if and only if $l^c(\sigma D_n) < l$. The forward direction is immediate from the statement of Proposition 6. For the reverse direction, if $l^c(\sigma D_n) < l$ then $l^c(\sigma D_n) + 3 < u$ since we defined l and u to be at least 3 apart. Since $l^c(\sigma D_n) + 3$ is an upper bound on $l(\sigma)$, we have that $l(\sigma) < u$, whence by definition of σ, $l(\sigma) < l$.

4 Conclusion

We showed that determining a reversal sort for circular permutations can be reduced to finding a minimum length sort amongst its $2n$ linearizations (Theorem 3).

Using an inapproximability result on linear SBR, it is shown that determining reversal distance between circular permutations is NP-hard (Theorem 9).

In practical terms, to approximate reversal length for a circular permutation it is sufficient to compute it for one of its linearizations using any of the programs already developed for this purpose (for example [10], [2]). This estimate will be accurate to within three reversals (Proposition 6) and NP-hardness of SBR for circular permutations assures us that using tools for linear permutations is likely to be as efficient as developing specific algorithms for circular permutations.

In case reversal lengths in a given situation are so small that an error margin of three is significant, Bafna and Pevzner's theorem [2, Theorem 5] concerning

the expected reversal length of a random permutation suggests that n will also be small. Therefore it may well be feasible to compute the length of the $2n$ linearizations for an exact result. This will be the subject of a future experimental investigation.

References

1. David A. Bader, Bernard M. E. Moret, Mi Yan, *A linear time algorithm for computing inversion distance between signed permutations with an experimental study*, Journal of Computational Biology, Volume 8, Number 5, 2001 pp. 483–491.
2. V. Bafna and P. A. Pevzner, *Genome rearrangements and sorting by reversals. SIAM Journal on Computing*, **25** (1996), 272–289.
3. P. Berman, M. Karpinski, *On some tighter inapproximability results (extended abstract)*, in "Automata, languages and programming (Prague, 1999)", Lecture Notes in Comput. Sci., **1644**, pp. 200–209, Springer, Berlin, 1999.
4. K. D. Boese, *Cost Versus Distance In the Traveling Salesman Problem*, Technical Report CSD-950018, UCLA Computer Science Department, May 1995.
5. Alberto Caprara, *Sorting Permutations by Reversals and Eulerian Cycle Decompositions* SIAM Journal on Discrete Mathematics, Volume 12, Number 1 (1999) pp. 91–110.
6. David A. Christie, *A 3/2-approximation algorithm for sorting by reversals*, in Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 244–252, San Francisco, California, 25–27 January 1998.
7. Michael R. Garey and David S. Johnson, "Computers and Intractability", W. H. Freeman, New York, 1979.
8. S. Hannenhalli, P. A. Pevzner, *Transforming cabbage into turnip: a polynomial algorithm for sorting signed permutations by reversals*, Journal of ACM, 46, 1–27, 1999.
9. John Kececioglu and David Sankoff, *Efficient bounds for oriented chromosome-inversion distance* Proceedings of the 5th Symposium on Combinatorial Pattern Matching, Springer-Verlag Lecture Notes in Computer Science 807, 307–325, 1994.
10. John Kececioglu and David Sankoff, *Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement* Algorithmica **13**, 180–210, 1995.
11. S. Lin and B. Kernighan, *An efficient heuristic for the traveling salesman problem*, Operations Research, 21(2):498–516, 1973.
12. J. Meidanis, M. E. M. T. Walter and Z. Dias, *Reversal distance of signed circular chromosomes*, Technical Report IC-00-23 (December 2000), Instituto de Computação, Universidade Estadual de Campinas, `http://www.ic.unicamp.br/ic-tr-ftp/2000/Abstracts.html`
13. S. Micali and V. Vazirani, *An* $O(\sqrt{|V|}|E|)$ *algorithm for finding maximum matchings in general gaphs*, Proceedings of the 21st Symposium on Foundations of Computer Science, 17–27, 1980, (cited in [10]).
14. G. Watterson, W. Ewens, T. Hall and A. Morgan, *The chromosome inversion problem*, J. Theor. Biol. **99** (1982), 1–7.

An Improved Bound on Boolean Matrix Multiplication for Highly Clustered Data

Leszek Gąsieniec[1] and Andrzej Lingas[2]

[1] Department of Computer Science, University of Liverpool, Peach Street, L69 7ZF, UK. leszek@csc.liv.ac.uk.
[2] Department of Computer Science, Lund University, 22100 Lund. Andrzej.Lingas@cs.lth.se. Fax +46 46 13 10 21

Abstract. We consider the problem of computing the product of two $n \times n$ Boolean matrices A and B.

For two $0-1$ strings $s = s_1 s_2 ....s_m$ and $u = u_1 u_2 ...u_m$, an extended Hamming distance, $eh(s,u)$, between the strings, is defined by a recursive equation $eh(s,u) = eh(s_{l+1}...s_m, u_{l+1}...u_m) + (s_1 + u_1 \mod 2)$, where l is the maximum number, s.t., $s_j = s_1$ and $u_j = u_1$ for $j = 1, ..., l$. For any $n \times n$ Boolean matrix C, let G_C be a complete weighted graph on the rows of C, where the weight of an edge between two rows is equal to its extended Hamming distance. Next, let $MWT(C)$ be the weight of a minimum weight spanning tree of G_C. We show that the product of A and B as well as the so called witnesses of the product can be computed in time $\tilde{O}(n(n + \min\{MWT(A), MWT(B^t)\}))$ [1].

Since the extended Hamming distance between two strings never exceeds the standard Hamming distance between them, our result subsumes an earlier similar result on the Boolean matrix product in terms of the Hamming distance due to Björklund and Lingas [4]. We also observe that $\min\{MWT(A), MWT(B^t)\} = O(\min\{r_A, r_B\})$, where r_A and r_B reflect the minimum number of rectangles required to cover 1s in A and B, respectively. Hence, our result also generalizes the recent upper bound on the Boolean matrix product in terms of r_A and r_B, due to Lingas [12].

1 Introduction

Since Strassen published his first sub-cubic algorithm for the arithmetic matrix multiplication [1], a lot of work in this area has been done. The best asymptotic upper bound on the number of arithmetic operations necessary to multiply two $n \times n$ matrices is presently $O(n^{2.376})$ due to Coppersmith and Winograd [7]. Since Boolean matrix multiplication is trivially reducible to arithmetic $0-1$-matrix multiplication [1], the same asymptotic upper bound holds in the Boolean case.

If an entry with indices $i,\ j$ of the Boolean product of two Boolean matrices A and B is equal to 1 then any index k such that $A[i,k]$ and $B[k,j]$ are equal to 1 is a *witness* to this. More recently, Alon and Naor [2] and Galil and Margalit

[1] $\tilde{O}(f(n))$ means $O(f(n)poly - \log n)$ and B^t stands for the transposed matrix B.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 329–339, 2003.
© Springer-Verlag Berlin Heidelberg 2003

[8] have shown that the witnesses for the Boolean matrix product of two $n \times n$ Boolean matrices (i.e., for all its nonzero entries) can be computed in time $\tilde{O}(n^{2.376})$ by repeatedly applying the aforementioned algorithm of Coppersmith and Winograd for arithmetic matrix multiplication [7].

Unfortunately, the aforementioned substantially sub-cubic algorithms for the arithmetic matrix multiplication are based on algebraic approaches difficult to implement.

In [14], Schnorr and Subramanian have shown that the Boolean product of two $n \times n$ random Boolean matrices can be determined by a simple combinatorial algorithm with the high probability in time $\tilde{O}(n^2)$. Consequently, they raised the question of whether or not there exist a substantially sub-cubic combinatorial algorithm for the Boolean matrix multiplication.

Unfortunately, the aforementioned question problem seems to be very hard. During the last two decades no essential progress as for upper time bounds in terms of n could be reported (the fastest known combinatorial algorithm for this problem is due to Bash et al. [3] and runs in time $O(n^3/\log^2 n)$). For this reason and because of the practical and theoretical importance of Boolean matrix multiplication, it seems of interest to investigate special cases of structured and random matrices and derive partial results, even if they are not too complicated (e.g., [4,14]). (For comparison, the vast literature on sorting includes several papers on sorting presorted files and the area of parameterized complexity of combinatorial problems rapidly expands.) It might happen that a combination of such partial results could eventually lead to a substantially sub-cubic combinatorial algorithm for Boolean matrix multiplication.

In [4], Björklund and Lingas followed the aforementioned suggestion providing a combinatorial algorithm for Boolean matrix multiplication which is substantially sub-cubic in case the rows of the first $n \times n$ matrix or the columns of the second one are highly clustered, i.e., their minimum spanning tree in the Hamming metric has low cost. More exactly, their algorithm runs in time $\tilde{O}(n(n+c))$ where c is the minimum of the costs of the minimum spanning trees for the rows and the columns, respectively, in the Hamming metric. It relies on the fast methods for computing an approximate minimum spanning tree in the L_1 and L_2 metrics given in [9,10].

In a subsequent paper [12], Lingas has taken a geometric approach to Boolean matrix multiplication. He has provided an algorithm for Boolean matrix multiplication whose time complexity is expressed in terms of the minimum numbers r_A, r_B of rectangles sufficient to cover exactly the rectilinear regions formed by the 1-entries in the input matrices A and B. In particular, his algorithm computes the product of A and B, and the witnesses of the product in time $\tilde{O}(n(n + \min\{r_A, r_B\}))$. For a matrix D, let m_D be the minimum of the number of 0-entries and the number of 1-entries in D. Since $r_D = O(m_D)$, Lingas' algorithm runs also in time $\tilde{O}(n(n + \min\{m_A, m_B\}))$.

In this paper, we strengthen and/or generalize the results from [4], [12] in an uniform way. Our key idea is to consider the so called *extended Hamming distance* instead of the standard Hamming distance.

For two $0-1$ strings $s = s_1 s_2 .... s_m$ and $u = u_1 u_2 ... u_m$, the extended Hamming distance, $eh(s,u)$, between the strings, is defined recursively by $eh(s,u) = eh(s_{l+1} ... s_m, u_{l+1} ... u_m) + (s_1 + u_1 \mod 2)$ where l is the maximum number such that $s_j = s_1$ and $u_j = u_1$ for $j = 1, ..., l$.

We show that the computation of $eh(s,u)$ can be reduced to that of standard Hamming distance between appropriately transformed strings s and t. This reduction is of interest in its own rights. In particular it implies that the search for nearest or approximate nearest neighbors and consequently the construction of minimum spanning trees or their approximations under the extended Hamming metric can be in turn reduced to the corresponding searches and constructions under the standard Hamming metric [5,9,10,11]. Hence, the known method of Hamming MST clustering in high dimensional spaces [5] can be enhanced by the more powerful variant of extended MST clustering.

Next we follow the general MST approach from [4]. For an $n \times n$ Boolean matrix C, let G_C be the complete weighted graph on the rows of C where the weight of an edge between two rows is equal to its extended Hamming distance.

Let $MWT(C)$ be the weight of a minimum weight spanning tree of G_C. We show that the product of A and B as well as the so called witnesses of the product can be computed in time $\tilde{O}(n(n + \min\{MWT(A), MWT(B^t)\}))$ [2].

Since the extended Hamming distance between two strings never exceeds the standard Hamming distance between them, our result subsumes the aforementioned result on Boolean matrix product in terms of Hamming distance due to Björklund and Lingas [4]. We also observe that $\min\{MWT(A), MWT(B^t)\})) = O(\min\{r_A, r_B\})$ where r_A and r_B are the minimum number of rectangles necessary to cover 1s in A and B, respectively. Hence, our result also generalizes the aforementioned recent upper time-bound on Boolean matrix product in terms of r_A and r_B due to Lingas [12].

Our paper is structured as follows. The next section is devoted to the extended Hamming distance and its relationship to the standard Hamming distance. In Section 3, known fact on approximating minimum spanning tree in the L_1 and L_2 metrics are used to derive a fast approximation algorithm for minimum spanning tree under the extended Hamming metric. In Section 4, we describe a dynamic data structure for maintaining a set of intervals on a line and supporting queries returning an element (i.e., a witness of non-emptiness) in the union of the intervals. Section 5 presents our algorithm for fast Boolean matrix multiplication for highly clustered data and its analysis. The algorithm relies both on the approximation MST algorithm and the dynamic witness data structure. In Section 6, we show that our algorithm yields also the time-bound $\tilde{O}(n(n + \min\{r_A, r_B\}))$. In Section 7, we observe that in general case our combinatorial algorithm cannot be substantially sub-cubic.

[2] $\tilde{O}(f(n))$ means $O(f(n)poly - \log n)$ and B^t stands for the transposed matrix B.

2 Extended Hamming Distance

For two $0-1$ strings $s = s_1 s_2 .... s_m$ and $u = u_1 u_2 ... u_m$, the extended Hamming distance, $eh(s,u)$, between the strings, is defined recursively by $eh(s,u) = eh(s_{l+1}...s_m, u_{l+1}...u_m) + (s_1 + u_1 \mod 2)$ where l is the maximum number such that $s_j = s_1$ and $u_j = u_1$ for $j = 1, ..., l$. This definition sets a natural division of s and u into a corresponding sequence of blocks $b_s^1, ..., b_s^q$ and $b_u^1, ..., b_u^q$, where q is an appropriate positive integer, $b_s^1 = s_1 s_2 .... s_l$, $b_u^1 = u_1 u_2 ... u_l$ and the remaining blocks follow recursively.

The following observation is obvious.

Lemma 1. *For any two $0-1$ strings of equal length, the extended Hamming distance between them never exceeds their Hamming distance, i.e., the number of positions at which the two strings differ.*

The next lemma shows that the computing of the extended Hamming distance between two strings can be reduced to the computing of the standard Hamming distance between the two strings appropriately transformed. In the lemma, the Hamming distance between two strings s and u of equal length is denoted by $h(s,u)$.

Lemma 2. *There is a simple, linear-time, transformation of any $0-1$ string w into the string $t(w)$ such that for any two $0-1$ strings s and u, $eh(s,u) = \lceil \frac{h(t(s),t(u))}{2} \rceil$.*

Proof. For any $0-1$ string w, the transformation $t(w)$ is a slight modification of transformation $\bar{t}(w)$ which is defined as follows. Let $w = p \cdot w'$, where p is a non-empty prefix of w formed by a sequence of 0s (1s) and followed by 1 (0) in w. For $p = 0^{|p|}$ we define $\bar{t}(p) = 01 \cdot (00)^{|p|-1}$, and when $s = 1^{|p|}$ we define $\bar{t}(p) = 10 \cdot (00)^{|p|-1}$. Further we define $\bar{t}(w) = \bar{t}(p) \cdot \bar{t}(w')$. Now the transformation $t(w)$ is obtained by a change (if needed) of a second bit of $\bar{t}(w)$ by 0. This operation is performed only if 0 is the first symbol in original string w. Note that each symbol in w has been replaced by two symbols in $t(w)$.

We show now that for any two $0-1$ strings s and u, $eh(s,u) = \lceil \frac{h(t(s),t(u))}{2} \rceil$.

Recall the block decomposition $b_u^1, ..., b_u^q$ and $b_v^1, ..., b_v^q$ of u and v implied by the recursive definition of the extended Hamming distance between u and v. The comparison of blocks in pairs (b_u^i, b_v^i), for $i = 1, .., q$ and their contribution to $eh(u,v)$ is performed by comparison of their counterparts $t(b_u^i)$ and $t(b_v^i)$. Let $P_u(i) = b_u^1 \cdot ... \cdot b_u^i$ and $P_v(i) = b_v^1 \cdot ... \cdot b_v^i$.

The following observation holds.

Assume that $eh(P_u(i), P_v(i)) = k$. If blocks $b_u^i = b_v^i$ then $h(t(P_u(i)), t(P_v(i))) = 2k$. Otherwise, i.e., if $b_u^i \neq b_v^i$ then $h(t(P_u(i)), t(P_v(i))) = 2k - 1$.

The proof of the observation is done by induction. Note that if $b_u^1 = b_v^1$ both $eh(b_u^1, b_v^1) = 0$ and $h(t(b_u^1), t(b_v^1)) = 0$. On the other hand, when $b_u^1 \neq b_v^1$ then $eh(b_u^1, b_v^1) = 1$ and $h(t(b_u^1), t(b_v^1)) = 2 \cdot 1 - 1 = 1$ (thanks to a slight difference between transformation $\bar{t}$ and t). And the basis step is completed.

Initially, we use an inductive assumption that $eh(P_u(i), P_v(i)) = k$, $b_u^i = b_v^i$, and $h(t(P_u(i)), t(P_v(i))) = 2k$. Two cases are possible in respect to the content of b_u^{i+1} and b_v^{i+1}. If $b_u^{i+1} = b_v^{i+1}$ (the change occurred in u and v on the same position) then $eh(P_u(i+1), P_v(i+1)) = k$, $h(t(P_u(i+1)), t(P_v(i+1))) = 2k$ and we end up in the situation when the lastly compared blocks are the same. Alternatively if $b_u^{i+1} \neq b_v^{i+1}$ (the change occurred only in one string either in u or v) then $eh(P_u(i+1), P_v(i+1)) = k+1$, $h(t(P_u(i+1)), t(P_v(i+1))) = 2k+1 = 2(k+1)-1$ (the Hamming distance is increased by a single 1 occurring on one of the first two positions of either $t(b_u^{i+1})$ or $t(b_v^{i+1})$) and we end up in the situation when the lastly compared blocks are different.

Assume now that $eh(P_u(i), P_v(i)) = k$, $b_u^i \neq b_v^i$, and $h(t(P_u(i)), t(P_v(i))) = 2k-1$. Two cases are possible in respect to the content of b_u^{i+1} and b_v^{i+1}. If $b_u^{i+1} \neq b_v^{i+1}$ (the change occurred in u and v on the same position) then $eh(P_u(i+1), P_v(i+1)) = k+1$, $h(t(P_u(i+1)), t(P_v(i+1))) = 2k-1+2 = 2(k+1)-1$ and we end up in the situation when the lastly compared blocks are different. Alternatively if $b_u^{i+1} = b_v^{i+1}$ (the change occurred only in one string either in u or v) then $eh(P_u(i+1), P_v(i+1)) = k$, $h(t(P_u(i+1)), t(P_v(i+1))) = 2k-1+1 = 2k$ (the Hamming distance is increased by a single 1 occurring on one of the first two positions of either $t(b_u^{i+1})$ or $t(b_v^{i+1})$) and we obtain the situation when the lastly compared blocks are the same. This completes the proof of the observation.

The thesis of lemma follows from the definition of the transformation t and the observation.

3 Approximate MST in the Extended Hamming Metric

For $c \geq 1$ and a finite set S of points in a metric space, a *c-approximate minimum spanning tree for S* is a spanning tree in the complete weighted graph on S, with edge weights equal to the distances between the endpoints, whose total weight is at most c times the minimum.

In [9] (section 4.3), Indyk and Motwani in particular considered the bichromatic ϵ-approximate closest pair problem for n points in R^d with integer coordinates in $O(1)$ under the L_p metric, $p \in \{1, 2\}$. They showed that there is a dynamic data structure for this problem which supports insertions, deletions and queries in time $O(dn^{1/(1+\epsilon)})$ and requires $O(dn + n^{1+1/(1+\epsilon)})$-time preprocessing. In consequence, by a simulation of Kruskal's algorithm they deduced the following fact.

Fact 1. For $\epsilon > 0$, a $1+\epsilon$-approximate minimum spanning tree for a set of n points in R^d with integer coordinates in $O(1)$ under the L_1 or L_2 metric can be computed by a Monte Carlo algorithm in time $O(dn^{1+1/(1+\epsilon)})$.

In [10] Indyk, Schmidt and Thorup reported even slightly more efficient (by a poly-log factor) reduction of the problem of finding a $1+\epsilon$-approximate minimum spanning tree to the bichromatic ϵ-approximate closest pair problem via an easy simulation of Prim's algorithm.

Note that the L_1 metric for points in R^n with 0, 1-coordinates coincides with the n-dimensional Hamming metric. Hence, Fact 1 immediately yields the following corollary.

Corollary 1. *For $\epsilon > 0$, a $1+\epsilon$-approximate minimum spanning tree for a set of n $0-1$ strings of length d under the Hamming metric can be computed by a Monte Carlo algorithm in time $O(dn^{1+1/(1+\epsilon)})$.*

By combining Lemma 2 with Corollary 1, we easily obtain the following lemma.

Lemma 3. *For $\epsilon > 0$, a $2+\epsilon$-approximate minimum spanning tree for a set of n $0-1$ strings of length d under the extended Hamming metric can be computed by a Monte Carlo algorithm in time $O(dn^{1+1/(1+\epsilon/2)})$.*

4 A Witness Dynamic Data Structure

We consider the problem of building a dynamic data structure that maintains a set of at most m intervals with integer endpoints in $[1, m]$ and supports a query that reports an integer (witness of non-emptiness) in the union of the intervals. We propose the following solution.

The union of the current set S of intervals is represented by the set of marked nodes in a segment tree (see [13]) on $[1, m]$ for which the associated segments form a partition of the union. With each node v in the segment tree, there is associated a balanced search tree $M(v)$ containing all marked descendants of v. Finally, there is a doubly linked list L of marked nodes in the segment tree following their order from the left to the right.

To answer the witness query, one simply lists, e.g., the first integer covered by the segment corresponding to the first node in L.

To insert a segment s in the segment tree, one marks the $O(\log m)$ nodes in the tree for which the associated segments (see [13]) partition s, provided that they are not descendants of already marked nodes. For each newly marked node v, one also inserts v in the search trees $M(u)$ of each ancestor of v in the segment tree as well as one unmarks all marked descendants w of v and removes them from all the search trees on the paths from w to the root of the segment tree. Finally, one identifies in the search trees $M(u)$ associated with ancestors u of v, the closest to left and the closest to right marked nodes v' and v'' respectively, and one appropriately modifies the list L by inserting the links between v' and v, and v and v'', respectively.

To delete a segment s from the segment tree, one determines the $O(\log m)$ nodes in the tree for which the associated segments partition s, and unmarks them and its descendants using the search trees $M(\ )$. One also appropriately removes the unmarked nodes from the lists $M(u)$ of their ancestors u. For each lastly unmarked node v, one finds the nearest neighbors w', w'' to the left and to the right, respectively, in the search trees $M(u)$ of its ancestors u, in order to modify the list L by linking w' with w''.

Time Complexity

Query: Answering any witness query takes $O(1)$ time. One simply finds the first node on the list L and the first number covered by the segment corresponding to this node.

Segment insertion: Marking the $O(\log m)$ nodes takes $O(\log m)$ time (see [13]). Unmarking an arbitrary of the marked descendants of a newly marked node and deleting them from the search trees $M(\ )$ of their ancestors takes $O(\log m)$ time. We charge the insertion of the segment during which the unmarked node has been marked with the cost of potential later unmarking and deleting from the appropriate search trees $M(\ )$. Note that the charge associated with an insertion of a segment increases to $O(\log^2 m \log m)$.

Modification of the list for each marked node takes $O(\log^2 m)$ time. Thus, it requires totally $O(\log^3 m)$ time. We conclude that the amortized cost of segment insertion is $O(\log^3 m)$.

Segment deletion: The determination of the $O(\log m)$ nodes takes $O(\log m)$ time. The unmarking of an arbitrary node v among them and their marked descendants, and its deletion from the search trees $M(\ )$ of their ancestors, as well as an appropriate modification of the list L, fits within the $O(\log^3 m)$ bound on the charge associated with the insertion of the segment that causes marking v.

Theorem 1. *Let C be a sequence of l insertions and deletions of intervals with integer endpoints in $[1, n]$ such that the current set S of intervals never has more than n elements. After $O(n^2 \log n)$-time preprocessing, the union of the current set S can be maintained in total time $O(l \log^3 n)$ such that the witness query can be always answered in time $O(1)$.*

5 Boolean Matrix Multiplication via MST

An $n \times n$ matrix W such that whenever the i, j entry of the product of matrices A and B is 1 then $W[i, j]$ is a witnesses to this is called a witness matrix for the product of A and B.

The idea of our combinatorial algorithm for witnesses of Boolean matrix product is a generalization of that from [4]. First, we compute an approximate spanning tree of the rows of A (or, the columns of B, alternatively) in the extended Hamming metric. Then, we fix a traversal of the tree.

Next, for each pair of consecutive neighboring rows u, s in the traversal, we determine the decomposition into blocks implied by their extended Hamming distance and precompute the set differences between the sets of blocks having 1s in s and u, respectively. Finally, for each column of B, we traverse the tree and implicitly compute the set of witnesses for the $\cdot$ product of the traversed row of A with the column of B from that for previously traversed row of A and the column of B. See Algorithm 1.

Algorithm 1

Input: $n \times n$ Boolean matrices A and B;

Output: A witness matrix W for the Boolean product C of A and B.

1. Compute an $O(\log n)$-approximate minimum weight spanning tree T_A of the graph G_A;
2. Fix a traversal of T_A of length linear in the size of T_A;
3. $i_0 \leftarrow$ the number of the row corresponding to the firstly traversed node of T_A;
4. $i \leftarrow i_0$;
5. **while** traversing the tree T_A **do**
   begin
   $l \leftarrow i$;
   $i \leftarrow$ the number of the row of A corresponding to the next node of T_A on the traversal;
   Compute the block decomposition D for the l-th and i-th row implied by their extended Hamming distance;
   For each pair b of corresponding blocks in D set $rank_j(b)$ to $[b_1, b_2]$ where b_1, b_2 are respectively the first and last rank of 1s in the j-th column of B covered the sub-interval of $[1, n]$ corresponding to b;
   if $l = i_0$ **then**
   begin
   for each pair b of corresponding blocks in D containing 1s on the side the l-th row **do**
   if $rank_j(b)$ is not empty **then** $D1_l \leftarrow D1_l \cup \{rank_j(b)\}$
   end;
   for each pair b of corresponding blocks in D containing 1s and 0s **do**
   begin
   if b contains 1s on the side of the i-th row and $rank_j(b)$ is not empty **then** $D1_{i,l} \leftarrow D1_{i,l} \cup \{rank_j(b)\}$;
   if b contains 1s on the side of the l-th row and $rank_j(b)$ is not empty **then** $D1_{l,i} \leftarrow D1_{l,i} \setminus \{rank_j(b)\}$;
   end
   end
6. **for** $j = 1, ..., n$ **do**
   begin
   $W[i_0, j] \leftarrow$ a witnesses for the i_0-th row of A and the j-th column of B;
   Initialize the witness data structure WD_j on $[1, n]$;
   for each $w \in D1_{i_0}$ **do**
   insert w into WD_j;
   $i \leftarrow i_0$;
   while traversing the tree T_A **do**
   begin
   $l \leftarrow i$;
   $i \leftarrow$ the number of the row of A corresponding to the next node of T_A on the traversal;
   if i has been already visited **then go to** E;
   for each $w \in D1_{i,l}$ **do**
   insert w into WD_j;
   for each $d \in D1_{l,i}$ **do**
   delete w from WD_j;
   witness query WD_j and in case the query returns a witness set $W[i, j]$ to it
   end

E: **end**

Lemma 4. *Algorithm 1 is correct, i.e., it outputs the Boolean product of the Boolean matrices A and B.*

Lemma 5. *Algorithm 1 can be implemented in time $\tilde{O}(n(n+MWT(A)))+t(n)$ where $t(n)$ is the time taken by the construction of the $O(\log n)$-approximate minimum weight spanning tree in step 2.*

Proof. The set $D1_{i_0}$ and the set differences $D1_{i,l}$, $D1_{l,i}$ can be easily computed in time $\tilde{O}(n)$ (e.g., $rank_j(b)$ can be determined in a logarithmic time after $O(n \log n)$ preprocessing for fixed j). This combined with the linear in n length of the traversal T_A implies an $\tilde{O}(n^2)$-time implementation of steps 1 and 5. In step 6, for each column j of the matrix B, we set WD_j to the witness data structure on $[1, n]$ initially. In fact, WD_j could be set to the witness data structure on the interval $[1, m]$ where m is the number of 1s in column j. However, this could increase the total cost of initializations of the data structures WD_j from $O(n^2 \log n)$ (see Theorem 1) to $\Omega(n^3)$ if done naively. The total number l of interval insertions and deletions over WD_j is easily seen to be not greater than $n + MWT(A)$. The term n bounds from above the number of interval insertions corresponding to blocks of 1s in the starting row in the traversal of the tree T_A. On the other hand, by Theorem 1 and straightforward implementations of the sub-steps in which WD_j is not involved, step 6 takes $\tilde{O}(n^2 + l)$ time. Consequently, the overall time-cost of step 6 is $\tilde{O}(n(n + MWT(A)))$.

The transposed product of matrices A and B is equal to the product of the transposed matrix B with the transposed matrix A. Hence, Lemmata 4, 5 yield our main result.

Theorem 2. *Let A, B be two $n \times n$ Boolean matrices. The product of matrices A and B as well as the witnesses for the product can be computed in expected time $\tilde{O}(n(n + \min\{MWT(A), MWT(B^t)\}))$ where B^t stands for the transposed matrix B.*

6 MST in the Extended Hamming Metric versus Minimum Rectangular Covering

For a Boolean matrix D, let r_D be the minimum number of rectangles sufficient to cover exactly the rectilinear region formed by the 1-entries in D. The following fact was proved in [12].

Fact 2. The Boolean product of two $n \times n$ Boolean matrices can be computed in time $\tilde{O}(n(n + \min\{r_A, r_B\}))$.

Lemma 6. *For a Boolean matrix D, the cost of MST for the rows of D or the columns of D in the extended Hamming metric is $O(r_D)$.*

Proof. Consider a minimum rectangular covering C of 1s in D and a spanning tree T of the rows which is just a path going through the nodes corresponding to consecutive rows. The cost of T in the extended Hamming metric is not larger than the number of horizontal edges in the covering C. Thus, it is upper bounded by $2r_D$. We obtain the same upper bound in the case of the MST for columns by considering vertical edges of C instead of the horizontal ones.

By Lemma 6, Theorem 2 subsumes up to a constant Fact 2.

7 Final Remarks

It follows from the existence of the so called Hadamard matrices [6] that there is an infinite sequence of $n_i \times n_i$ matrices A_i, B_i such that the Hamming distance between any pair of rows of A_i or columns of B_i is $\Omega(n_i)$. We can generalize this observation to include the extended Hamming distance as follows.

To start with, assume without loss of generality that n_i is even. Pick half of the columns of A_i such that the Hamming distance between any pair of resulting half-rows of A_i is $\Omega(n_i/2)$. Complement the picked columns with $n_i/2$ specially chosen columns such that in each row of the resulting matrix A'_i there is equal number of 0s and 1s. Now, consider a random permutation of the columns of A'_i resulting in another matrix A''_i. The probability that in any row of the matrix A''_i there is a block of consecutive 1s or 0s of length $c \log n$ is at most $O(n^{2-c})$. By picking $c > 2$, we conclude that there is a permutation of the columns of A'_i such that the extended Hamming distance between any two rows of A''_i is at most $O(\log n)$ times smaller than the Hamming distance between them. On the other hand, the Hamming distance between any pair of rows in A''_i is $\Omega(n_i)$ by the definition of A''_i. It follows that the cost of the minimum spanning tree for the rows of A''_i is $\Omega((n_i)^2/\log^2 n)$ under the extended Hamming distance. Analogously, we can construct a matrix B''_i such that the cost of the minimum spanning tree for the columns of B''_i is $\Omega((n_i)^2/\log^2 n)$. Thus, our combinatorial algorithm for Boolean matrix multiplication presented in Section 5 cannot break the cubic upper bound in the general case substantially. However, in many applications of Boolean matrix multiplication where the rows or columns respectively tend to be more clustered the aforementioned scenario would be unlikely.

Generally, it seems that whenever the rows of A or the columns of B admit a substantially sub-quadratic representation, there might be good chances for computing the Boolean product of A and B combinatorially in substantially sub-cubic time. On the other hand, the absence of such representations might mean that the matrices have some properties of random ones and therefore could admit a substantially sub-cubic combinatorial algorithm for their Boolean product like the random ones [14]. This general idea gives some hope in the search for a combinatorial substantially sub-cubic algorithm for Boolean matrix product.

Acknowledgements. The second author is grateful to Frank Dehne and Rolf Klein for valuable questions on [12] at ISAAC 2002 which inspired this work.

References

1. A.V. Aho, J.E. Hopcroft and J.D. Ullman. The Design and Analysis of Computer Algorithms (Addison-Wesley, Reading, Massachusetts, 1974).
2. N. Alon and M. Naor. Derandomization, Witnesses for Boolean Matrix Multiplication and Construction of Perfect hash functions. Algorithmica 16, pp. 434–449, 1996.
3. J. Basch, S. Khanna and R. Motwani. On Diameter Verification and Boolean Matrix Multiplication. Technical Report, Standford University CS department, 1995.
4. A. Björklund and A. Lingas. Fast Boolean matrix multiplication for highly clustered data. Proc. 7th International Workshop on Algorithms and Data Structures (WADS 2001), Lecture Notes in Computer Science, Springer Verlag.
5. A. Borodin, R. Ostrovsky and Y. Rabani. Subquadratic Approximation Algorithms For Clustering Problems in High Dimensional Spaces. Proceedings of the 31st ACM Symposium on Theory of Computing, 1999.
6. P.J. Cameron. Combinatorics. Cambridge University Press 1994.
7. D. Coppersmith and S. Winograd. Matrix Multiplication via Arithmetic Progressions. J. of Symbolic Computation 9 (1990), pp. 251–280.
8. Z. Galil and O. Margalit. Witnesses for Boolean Matrix Multiplication and Shortest Paths. Journal of Complexity, pp. 417–426, 1993.
9. P. Indyk and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. Proceedings of the 30th ACM Symposium on Theory of Computing, 1998.
10. P. Indyk, S.E. Schmidt, and M. Thorup. On reducing approximate mst to closest pair problems in high dimensions. *Manuscript*, 1999.
11. E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. SIAM J. Comput. 30, 2, pp. 457–474. (Preliminary version in Proc. 30th STOC, 1989.)
12. A. Lingas. A geometric approach to Boolean matrix multiplication. Proc. 13th International Symposium on Algorithms and Computation (ISAAC 2002), Lecture Notes in Computer Science 2518, Springer Verlag, pp. 501–510.
13. K. Mehlhorn. Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry. *EATCS Monographs on Theoretical Computer Science, Springer Verlag, Berlin, 1984.*
14. C.P. Schnorr and C.R. Subramanian. Almost Optimal (on the average) Combinatorial Algorithms for Boolean Matrix Product Witnesses, Computing the Diameter. Randomization and Approximation Techniques in Computer Science. Second International Workshop, RANDOM'98, Lecture Notes in Computer Science 1518, pp. 218–231.

Dynamic Text and Static Pattern Matching

Amihood Amir[1], Gad M. Landau[2], Moshe Lewenstein[1], and Dina Sokol[1]

[1] Bar-Ilan University, Ramat Gan, Israel
{amir,moshe,sokold}@cs.biu.ac.il
[2] University of Haifa, Haifa 31905, Israel
landau@cs.biu.ac.il

Abstract. In this paper, we address a new version of dynamic pattern matching. The *dynamic text and static pattern matching problem* is the problem of finding a static pattern in a text that is continuously being updated. The goal is to report all new occurrences of the pattern in the text after each text update. We present an algorithm for solving the problem, where the text update operation is *changing* the symbol value of a text location. Given a text of length n and a pattern of length m, our algorithm preprocesses the text in time $O(n \log\log m)$, and the pattern in time $O(m\sqrt{\log m})$. The extra space used is $O(n + m\sqrt{\log m})$. Following each text update, the algorithm deletes all prior occurrences of the pattern that no longer match, and reports all new occurrences of the pattern in the text in $O(\log\log m)$ time.

1 Introduction

The *static pattern matching problem* has as its input a given text and pattern and outputs all text locations where the pattern occurs. The first linear time solution was given by Knuth, Morris and Pratt [12] and many more algorithms with different flavors have been developed for this problem since.

Considering the dynamic version of the problem, three possibilities need to be addressed.

1. A static text and dynamic pattern.
2. A dynamic text and a static pattern.
3. Both text and pattern are dynamic.

The static text and dynamic pattern situation is a traditional search in a non-changing database, such as looking up words in a dictionary, phrases is a book, or base sequences in the DNA. This problem is called the *indexing problem*. Efficient solutions to the problem, using suffix trees, were given in [18,14,16]. For a finite fixed alphabet, the algorithms preprocess the text T in time $O(|T|)$. Subsequent queries seeking pattern P in T can be solved in time $O(|P| + tocc)$, where $tocc$ is the number of occurrences of P in T. Farach [5] presented an improved algorithm, acheiving the same time bounds for large alphabets.

Generalizing the indexing problem led to the *dynamic indexing problem* where both the text and pattern are dynamic. This problem is motivated by making queries to a changing text. The problem was considered by [9,7,15,1]. The Sahinalp and Vishkin algorithm [15] achieves the same time bounds as the suffix

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 340–352, 2003.
© Springer-Verlag Berlin Heidelberg 2003

tree algorithm for initial text preprocessing, $O(|T|)$, and for a search query for pattern P, $O(|P| + tocc)$, for bounded fixed alphabets. Changes to the text are either insertion or deletion of a substring S, and each change is performed in time $O(\log^3 |T|+|S|)$. The data structures of Alstrup, Brodal and Rauhe [1] support insertion and deletion of characters in a text, and movement of substrings within the text, in time $O(\log^2 |T| \log\log |T| \log^* |T|)$ per operation. A pattern search in the dynamic text is done in $O(\log |T| \log\log |T| + |P| + tocc)$.

Surprisingly, there is no direct algorithm for the case of a dynamic text and static pattern, as could arise when one is seeking a known and unchanging pattern in data that keeps updating. We were motivated for solving this missing version of dynamic pattern matching by the *two dimensional run-length compressed matching problem* [2]. The dynamic text pattern matching problem is a special case of the 2d run-length compressed matching problem where all pattern rows are *trivial*, i.e., consist of a single repeating symbol. This special case had no efficient solution in [2].

The **Dynamic Text and Static Pattern Matching Problem** is defined as follows:

Input: Text $T = t_1, ..., t_n$, and pattern $P = p_1, ..., p_m$, over alphabet Σ, where $\Sigma = \{1, ..., m\}$.

Preprocessing: Preprocess the text efficiently, allowing the following subsequent operation:

Replacement Operation: $\langle i, \sigma \rangle$, where $1 \leq i \leq n$ and $\sigma \in \Sigma$. The operation sets $t_i = \sigma$.

Output: Initially, report all occurrences of P in T. Following each replacement, report all **new** occurrences of P in T, and discard all **old** occurrences that no longer match.

The solutions of [15,1] can be adapted to solve our problem with the time bounds stated above. However, one would like a more direct and efficient way to answer queries for a static pattern and a text whose length does not change. In this paper we provide a direct answer to the dynamic text and static pattern matching problem, where the text update operation is *changing* the symbol value of a text location. After each change, both the text update and the reporting of new pattern occurrences are performed in only $O(\log\log m)$ time. The text preprocessing is done in $O(n \log\log m)$ time, and the pattern preprocessing is done in $O(m\sqrt{\log m})$ time. The extra space used is $O(n + m\sqrt{\log m})$. We note that the complexity for reporting the new pattern occurrences is not proportional to the number of pattern occurrences found since all new occurrences are reported in a succinct form.

We begin with a high-level description of the algorithm in Section 2, followed by some preliminaries in Section 3. In Sections 4 and 5 we present the detailed explanation of the algorithm. We leave the details of the data structures and proofs for the journal version.

2 Main Idea

2.1 Text Covers

The central theme of our algorithm is the representation of the text in terms of the static pattern. The following definition captures the notion that we desire.

Definition 1 (cover). *Let S and $S' = s'_1 \cdots s'_n$ be strings over alphabet Σ. A* cover *of S by S' is a partition of S, $S = \tau_1\tau_2 \ldots \tau_v$, satisfying -*
(1) substring property: *for each $1 \leq i \leq v$, τ_i is either a substring of S', or a character that does not appear in S'*
(2) maximality property: *for each $1 \leq i < v$, the concatenation of $\tau_i\tau_{i+1}$ is* not *a substring of S'.*

When the context is clear we call a cover of S by S' simply a cover. We also say that τ_h is an *element* of the cover. A cover element τ_h is represented by a triple $[i, j, k]$ where $\tau_h = s'_i \cdots s'_j$, and k, the *index* of the element, is the location in S where the element appears, i.e. $k = \sum_{l=1}^{h-1} |\tau_l| + 1$.

A cover of T by P captures the expression of the text T in terms of the pattern P. We note that a similar notion of a covering was used by Landau and Vishkin [13]. Their cover had the substring property but did not use the maximality notion. The maximality invariant states that each substring in the partition must be maximal in the sense that the concatenation of a substring and its neighbor is *not* a new substring of P. Note that there may be numerous different covers for a given P and T.

2.2 Algorithm Outline

Initially, when the text and pattern are input, any linear time and space pattern matching algorithm, e.g. Knuth-Morris-Pratt [12], will be sufficient for announcing all matches. The challenge of the Dynamic Text and Static Pattern Matching Problem is to find the new pattern occurrences efficiently after each replacement operation. Hence, we focus on the on-line part of the algorithm which consists of the following.

Online Algorithm
1. Delete old matches that are no longer pattern occurrences.
2. Update the data structures for the text.
3. Find new matches.

Deleting the old matches is straightforward as will be described later. The challenge lies in finding the new matches. Clearly, we can perform any linear time string matching algorithm. Moreover, using the ideas of Gu, Farach and Beigel [9], it is possible find the new matches in $O(\log m + pocc)$ time, where *pocc* are the number of pattern occurrences. The main contribution of this paper is the reduction of the time to $O(\log \log m)$ time per change. We accomplish this goal by using the cover of T by P. After each replacement, the cover of T must first be updated to represent the new text. We *split* and then *merge* elements to update the cover.

Once updated, the elements of the cover can be used to find all new pattern occurrences efficiently.

Observation 1 *Due to their maximality, at most one complete element in the cover of T by P can be included in a pattern occurrence.*

It follows from Observation 1 that all new pattern occurrences must begin in one of three elements of the cover, the element containing the replacement, its neighbor immediately to the left, or the one to the left of that. To find all new pattern starts in a given element of the cover, τ_x, it is necessary to check each suffix of τ_x that is also a prefix of P. We use the data structure of [9], the *border tree*, to allow checking many locations at once. In addition, we reduce the number of checks necessary to a constant.

3 Preliminaries

3.1 Definitions

In this section we review some known definitions on string periodicity, which will be used throughout the paper. Given a string $S = s_1 s_2 \dots s_n$, we denote the substring of S, $s_i \dots s_j$, by $S[i:j]$. $S[1:j]$ is a *border* of S if it is both a proper prefix and proper suffix of S. Let x be the length of the longest border of S. S is *periodic*, with period $n - x$, if $x > n/2$. Otherwise, S is *non-periodic*.

A string S is *cyclic* in string π if it is of the form π^k, $k > 1$. A *primitive* string is a string which is not cyclic in any string. Let $S = \pi'\pi^k$, where $|\pi|$ is the period of S and π' is a (possibly empty) suffix of π. S can be expressed as $\pi'\pi^k$ for one unique primitive π. A *chain of occurrences* of S in a string S' is a substring of S' of the form $\pi'\pi^q$ where $q \geq k$.

3.2 Succinct Output

In the online part of the algorithm, we can assume without loss of generality that the text is of size $2m$. This follows from the simple observation that the text T can be partitioned into $2n/m$ overlapping substrings, each of length $2m$, so that every pattern match is contained in one of the substrings. Each replacement operation affects at most m locations to its left. The cover can be divided to allow constant time access to the cover of a given substring of length $2m$.

The following lemma can be easily proven using the properties of string periodicity. The full proof will appear in the journal version of the paper.

Lemma 1. *Let P be a pattern of length m and T a text of length $2m$. All occurrences of P in T can be stored in constant space.*

4 The Algorithm

The algorithm has two stages, the *static stage* and the *dynamic stage*. The static stage, described in Section 4.1, consists of preprocessing data structures and reporting all initial occurrences of P in T.

The *dynamic stage* consists of the processing necessary following each replacement operation. The main idea was described in Section 2. The technical and implementation details are discussed in Sections 4.2 and 5.

%vspace-.2 cm

4.1 The Static Stage

The first step of the static stage is to use any linear time and space pattern matching algorithm, e.g. Knuth-Morris-Pratt [12], to announce all occurrences of the pattern in the original text. Then, several data structures are constructed for the pattern and the text to allow efficient processing in the dynamic stage.

Pattern Preprocessing. Several known data structures are constructed for the static pattern P. Note that since the pattern does not change, these data structures remain the same throughout the algorithm. The purpose of the data structures is to allow the following four queries to be answered efficiently. The first two queries are used in the *text update step* and the second two are used for *finding new matches*. We defer the description of the data structures to the full version of the paper. The query list is sufficient to enable further understanding of the paper.

Query List for Pattern P

Longest Common Prefix Query (LCP): Given two substrings, S' and S'', of P. Is $S' = S''$? If not, output the position of the first mismatch.

$\Rightarrow$ Query Time [13]: $O(1)$.

Substring Concatenation Query: Given two substrings, S' and S'', of P. Is the concatenation $S'S''$ a substring of P? If yes, return a location j in P at which $S'S''$ occurs.

$\Rightarrow$ Query Time [3,6,10,19]: $O(\log\log m)$.

Longest Border Query: Given a substring S' of P, such that $S' = P[i:j]$, what is the longest border of $P[1:j]$ that is a suffix of S'?

$\Rightarrow$ Query Time [9]: $O(\log\log m)$.

Range Maximum Prefix Query: Given a range of suffixes of the pattern P, $S_i \ldots S_j$. Find the suffix which maximizes the $LCP(S_\ell, P)$ over all $i \leq \ell \leq j$.

$\Rightarrow$ Query Time [8]: $O(1)$.

Text Preprocessing. In this section we describe how to find the cover of T by P for the input text T. Recall that we assume that the alphabet is linearly bounded in m. Thus, it possible to create an array of the distinct characters in P. The initial step in the cover construction is to create an element, τ_i, for each location i of the text. Specifically, for each location, $1 \leq i \leq n$, of the text, we

identify a location of P, say $P[j]$, where t_i appears. We set $j = m + 1$ if t_i does not appear in P, and create $\tau_i = [j, j, i]$. Then, moving from left to right, we attempt to merge elements in the cover using the *substring concatenation query*. The initial cover is stored in a van Emde Boas [17] data structure, sorted by the indices of the elements in the text.

Time Complexity. The algorithm for constructing the cover runs in deterministic $O(n \log\log m)$ time. The amount of extra space used is $O(n)$. Creating an array of the pattern elements takes $O(m)$ time, and identifying the elements of T takes $O(n)$ time. $O(n)$ *substring concatenation queries* are performed, each one takes $O(\log\log m)$ time. The van Emde Boas data structure costs $O(n)$ time and space for its construction [17].

4.2 The Dynamic Stage

In the on-line part of the algorithm, one character at a time is replaced in the text. Following each replacement, the algorithm must delete the old matches that no longer match, update the text cover, and report all new matches of P in T. In this section we describe the first two steps of the dynamic stage. In Section 5 we describe the third step, finding the new matches.

Delete Old Matches. If the pattern occurrences are saved in accordance with Lemma 1 then deleting the old matches is straightforward. If P is non-periodic, we check whether the one or two pattern occurrences are within distance -m of the change. If P is periodic, we truncate the chain(s) according to the position of the change.

Update the Text Cover. Each replacement operation replaces exactly one character in the text. Thus, it affects only a constant number of elements in the cover.

Algorithm: Update the Cover
1. Locate the element in the current cover in which the replacement occurs.
2. Break the element into three parts.
3.Concatenate neighboring elements to restore the maximality property.

Step 1: Locate the desired element. Recall that the partition is stored in a van Emde Boas tree [17] which allows predecessor queries. Let x be the location in T at which the character replacement occurred. Then, the element in the partition in which the replacement occurs will be the $pred(x)$.

Step 2: Break Operation. Let $[i, j, k]$ be an element in the partition which covers the position x at which a replacement occurred. The break operation divides the element $[i, j, k]$ into the following three parts. We assume that the new character is at position q of the pattern. To find the new text character in the pattern we do as described in the algorithm for constructing the cover (Section 4.1).

(1) $[i, i + x - k - 1, k]$, the part of the element $[i, j, k]$ prior to position x.
(2) $[q, q, x]$, position x, the position of the replacement.
(3) $[i + x - k + 1, j, x + 1]$, the part of the element after position x.

Step 3: Restore maximality property. The maximality property is a local property, it holds for each pair of adjacent elements in the cover. As stated in the following lemma, each replacement affects the maximality property of only a constant number of pairs of elements. Thus, to restore the maximality it is necessary to attempt to concatenate a constant number of neighboring elements. This is done using the *substring concatenation query.*

Lemma 2. *Following a replacement and break operation to a cover of T, at most four pairs of elements in the new partition violate the maximality property.*

Time Complexity of Updating the Cover: The van Emde Boas tree implements the operations: insertion (of an element from the universe), deletion, and predecessor, each in $O(\log\log|U|)$ time using $O(|U|)$ space [17]. In our case, since the text is assumed to be of length $2m$, we have $|U| = m$. Thus, the predecessor of x in the cover (Step 1) can be found in $O(\log\log m)$ time. Step 2, the break operation, is done in constant time. Step 3, restoring the maximality property, performs a constant number of substring concatenation queries. These can be done in $O(\log\log m)$ time. Overall, the time complexity for updating the cover is $O(\log\log m)$.

5 Find New Matches

In this section we describe how to find all new pattern occurrences in the text, after a replacement operation is performed. The new matches are extrapolated from the elements in the updated cover.

Any new pattern occurrence must include the position of the replacement. In addition, a pattern occurrence may span at most three elements in the cover (due to the maximality property). Thus, all new pattern starts begin in three elements of the cover, the element containing the replacement, its neighbor immediately to the left, or the one to the left of that. Let the three elements under consideration be labeled τ_x, τ_y, τ_z, in left to right order. The algorithm *Find New Matches* finds all pattern starts in a given element in the text cover, and it is performed separately for each of the three elements, τ_x, τ_y, and τ_z. We describe the algorithm for finding pattern starts in τ_x.

The naive approach would be to check each location of τ_x for a pattern start (e.g. by performing $O(m)$ LCP queries). The time complexity of the naive algorithm is $O(m)$. In the following two subsections we describe two improved algorithms for finding the pattern starts in τ_x. The first algorithm has time $O(\log m)$ and the basic approach comes from [9]. Our algorithm, described in Section 5.2, improves upon this further. Our algorithm also uses the border tree of [9], but we use additional properties of the border groups (defined below) which allow a significant improvement in the time complexity. The total time for announcing all new pattern occurrences is $O(\log\log m)$.

Definition 2 (border groups [4]). *The borders of a given string $S[1:m]$ can be partitioned into $g = O(\log m)$ groups $B_1, B_2, \ldots, B_g$. The groups preserve the left to right ordering of the borders. For each B_i, either $B_i =$*

$\{\pi_i'\pi_i^{k_i}, \ldots, \pi_i'\pi_i^3, \pi_i'\pi_i^2\}$ *or* $B_i = \{\pi_i'\pi_i^{k_i}, \ldots, \pi_i'\pi_i\}$ *where* $k_i \geq 1$ *is maximal,* π_i' *is a proper suffix of* π_i*, and* π_i *is primitive.*[1]

The border groups divide the borders of a string S into disjoint sets, in left to right order. Each group consists of borders that are all (except possibly the rightmost one) periodic with the same period. The groups are constructed as follows. Suppose $\pi'\pi^k$ is the longest border of $S[1:m]$. $\{\pi'\pi^k, \ldots, \pi'\pi^3, \pi'\pi^2\}$ are all added to group B_1. $\pi'\pi$ is added to B_1 if and only if it is *not* periodic. If $\pi'\pi$ is not periodic, it is the last element in B_1, and its longest border begins group B_2. Otherwise, $\pi'\pi$ is periodic, and it is the first element of B_2. This construction continues inductively, until π' is empty and π has no border.

5.1 Algorithm 1: Check Each Border Group

It is possible to use the algorithm of [9] to obtain a $O(\log m)$ time algorithm for finding all new pattern occurrences. The idea is to check all suffixes of τ_x which are prefixes of P. We group together all prefixes that belong to the same border group, and check them in *constant time*. The $O(\log m)$ time bound follows from the fact that there are at most $O(\log m)$ border groups to check. The border groups for any pattern prefix can be retrieved from the border tree of P.

Check one border group for pattern starts. Given a border group, $B_g = \{\pi'\pi^k, \pi'\pi^{k-1}, \ldots\}$, of which some element is a suffix of τ_x, compare π^{k+1} with the text following τ_x (using one or two LCP queries), to see how far right the period π recurs in the text. Depending on the length of the pattern prefix with period π, we locate all pattern starts in τ_x that begin with a border from B_g.

5.2 Algorithm 2: Check $O(1)$ Border Groups

Rather that checking *all* $O(\log m)$ border groups, our algorithm accomplishes the same goal by checking only a constant number of border groups. We use the algorithm for checking one border group to check the leftmost border group in τ_x, and at most one or two additional border groups.

Algorithm: Find New Matches

Input: An element in the cover, $\tau_x = [i, j, k]$.
Output: All starting locations of P in the text between t_k and t_{k+j-i}.

1. Find the longest suffix of τ_x which is a prefix of P. The *longest border query* (Section 4.1) returns the desired location. Let ℓ be the length of the suffix returned by the query.

[1] The definition of Cole and Hariharan [4] includes a third possibility, $B_i = \{\pi_i'\pi_i^{k_i}, \ldots, \pi_i'\pi_i, \pi_i'\}$, when π_i' is the empty string. In the current paper we do not include empty borders.

2. Using the Algorithm Check One Border Group (described in previous section), check the group of $P[1:\ell]$, where ℓ is the length found in Step 1.
3. Choose $O(1)$ remaining border groups and check them using the Algorithm Check One Border Group.

Steps 1 and 2 were explained previously. It remains to describe how to choose the $O(1)$ border groups that will be checked in Step 3.

For ease of exposition we assume that the entire pattern has matched the text (say $\tau_x = P$), rather than some pattern prefix. This assumption does not limit generality since the only operations that we perform use the border tree, and the border tree stores information about each pattern prefix. Another assumption is that the longest border of P is $< m/2$. This is true in our case, since if P were periodic, then all borders with length $> m/2$ would be part of the leftmost border group. We take care of the leftmost border group separately (Step 2), thus all remaining borders will have length $< m/2$.

Thus, the problem that remains is the following. An occurrence of a non-periodic P has been found in the text, and we must find any pattern occurrence which begins in the occurrence of P. Note that there is at most one overlapping pattern occurrence since P is non-periodic. In Section 5.2 we describe some properties of the borders/border groups from Cole and Hariharan [4]. We use these ideas in Section 5.2 to eliminate all but $O(1)$ border groups.

Properties of the Borders. A *pattern instance* is a possible alignment of the pattern with the text, that is, a substring of the text of length m. The pattern instances that interest us begin at the locations of the borders of P. Let $\{x_1, x_2, \ldots\}$ denote the borders of P, with x_1 being the longest border of P. Let X_i be the pattern instance beginning with the border x_i.

Note that $|x_1| < m/2$ and P is non-periodic. Thus, although there may be $O(m)$ pattern instances, only one can be a pattern occurrence. The properties described in this section can be used to isolate a certain substring of the text, overlapping *all* pattern instances, which can match at most three of the overlapping pattern instances. Moreover, it possible to use a *single* mismatch in the text to discover which three pattern instances match this "special" text substring.

The following lemma from Cole and Hariharan [4] relates the overlapping pattern instances of the borders of P.

Definition 3 ([4]). *A* clone set *is a set $Q = \{S_1, S_2, \ldots\}$ of strings, with $S_i = \pi'\pi^{k_i}$, where π' is a proper suffix of primitive π and $k_i \geq 0$.*

Lemma 3. *[4] Let X_a, X_b, X_c, $a < b < c$, be pattern instances of three borders of P, x_a, x_b, x_c, respectively. If the set $\{x_a, x_b, x_c\}$ is not a clone set, then there exists an index d in X_1 with the following properties. The characters in $X_1, X_2, \ldots, X_a$ aligned with $X_1[d]$ are all equal; however, the character aligned with $X_1[d]$ in at least one of X_b and X_c differs from $X_1[d]$. Moreover, $m - |x_a| + 1 \leq d \leq m$, i.e. $X_1[d]$ lies in the suffix x_a of X_1.*

Each border group is a clone set by definition, since every border within a group has the same period. However, it is possible to construct a clone set from elements in two different border groups. The last element in a border group can have the form $\pi'\pi^2$, in which case the borders $\pi'\pi$ and π' will be in (one or two) different border groups. It is not possible to construct a clone set from elements included in more than three distinct border groups. Thus, we can restate the previous lemma in terms of border groups, and a single given border, as follows.

Lemma 4. *Let x_a be a border of P with pattern instance X_a, and let x_a be the rightmost border in its group (definition 2). At most two different pattern instances to the right of X_a can match x_a at the place where they align with the suffix x_a of X_1.*

Let $r = m - |x_1| + 1$. Note that $P[r]$ is the location of the suffix x_1 in P. Since all pattern instances are instances of the same P, an occurrence of a border x_a in some pattern instance below X_a, aligned with $X_a[r]$, corresponds exactly to an occurrence of x_a in P to the left of $P[r]$. The following claim will allow us to easily locate the two pattern instances which are referred to in Lemma 4.

Claim. Let x_a be a border of P, and let x_a be the rightmost border in its group (definition 2). Let $r = m - |x_1| + 1$, where x_1 is the longest border of P. There are at most two occurrences of x_a beginning in the interval $P[r - |x_a|, r]$.

The Final Step. Using ideas from the previous subsection, our algorithm locates a *single mismatch* in the text in constant time. This mismatch is used to eliminate all but at most three pattern instances. Consider the overlapping pattern instances at the mth position of X_1. By Lemma 3, we have an identical alignment of all borders of P at this location. Each x_i is a suffix of all x_j such that $i > j$, since all x_i are prefixes and suffixes of P. Thus, suppose that the algorithm does the following. Beginning with the mth location of X_1, match the text to the pattern borders from right to left. We start with the shortest border, and continue sequentially until a mismatch is encountered. Let x_a be the border immediately below the border with the mismatch. The first mismatch tells two things. First, *all* borders with length longer than $|x_a|$ mismatch the text. In addition, at most two pattern instances with borders shorter than $|x_a|$ match x_a at the location aligned with the suffix x_a of X_1 (Lemma 4).

The algorithm for choosing the $O(1)$ remaining borders is similar to the above description, however, instead of sequentially comparing text characters, we perform a single LCP query to match the suffix x_1 with the text from right to left.

Algorithm: Choose $O(1)$ Borders (Step 3 of Algorithm *Find New Matches*)
A: Match P from *right to left* to the pattern instance of x_1 by performing a single LCP query.
B: Find the longest border that begins following the position of the mismatch found in Step A.

C: Find the $O(1)$ remaining borders referred to in Lemma 4.
D: Check the borders found in Steps B and C using the algorithm for checking one border group.

An LCP query is performed to match the suffix x_1 of X_1, with the text cover from right to left. (Step **A**). The position of the mismatch is found in constant time, and then a *longest border query* is used to find x_a (Step **B**). Once X_a is found, we know that all pattern instances to its left mismatch the text. It remains to find the possibilities to the right of X_a which are referred to in Lemma 4. Claim 5.2 is used for this purpose.

Step C: Let $r = m - |x_1| + 1$. The possible occurrences of x_a in pattern instances to the right of X_a correspond to occurrences of x_a in the interval $P[r - |x_a|, r]$.

By Claim 5.2 there are at most two occurrences of x_a in the specified interval. Since x_a is a pattern prefix, three *range maximum prefix queries* will give the desired result. The first query returns the maximum in the range $[r - |x_a|, r]$. This gives the longest pattern prefix in the specified range. If the length returned by the query is $\geq |x_a|$, then there is an occurrence of x_a prior to position r. Otherwise, there is no occurrence of x_a aligned with $X_a[r]$, and the algorithm is done. If necessary, two more maxima can be found by subdividing the range into two parts, one to the left and one to the right of the maximum.

Step D: The final step is to check each border group, of which there are at most three, using the Algorithm Check One Border Group.

Time Complexity of Algorithm Find New Matches: As shown previously, each step of the algorithm takes either constant time or $O(\log \log m)$ time. Thus, overall, the algorithm has time complexity $O(\log \log m)$.

We summarize the algorithm, including the time and space complexity of each step.
Preprocessing: $O(n \log \log m + m\sqrt{\log m})$ time and $O(n + m\sqrt{\log m})$ space.
On-line algorithm: $O(\log \log m)$ time per replacement.

Pattern Preprocessing: The following data structures are necessary to answer the queries listed in Section 4.1.
(1) The suffix trees for P and the reverse of P: $O(m)$ time/space [5]. The suffix trees must be preprocessed for:
(a) lowest common ancestor queries: $O(m)$ time/space [11],
(b) weighted ancestor queries: $O(m)$ time/space, combined results of [6,10, 19], and
(c) node intersection queries: $O(m\sqrt{\log m})$ time/space [3].
(2) The border tree for P is constructed in $O(m)$ time/space [9], and
(3) a range-maximum prefix array for P is created in $O(m)$ time/space [8].

Text Preprocessing: (Section 4.1)
(1) Construct the cover of T by P: $O(n \log \log m)$ time, $O(n)$ space.
(2) Store the cover in a van Emde Boas data structure: $O(n)$ time/space.

The Dynamic Algorithm: (Sections 4.2,5)
(1) Delete old matches that are no longer pattern occurrences: $O(\log \log m)$ time.

(2) Update the data structures for the text: $O(\log \log m)$ time.
(3) Find new matches: $O(\log \log m)$ time.

6 Conclusion

In this paper we presented an algorithm for the Dynamic Text and Static Pattern Matching Problem, allowing character replacements to be performed on the text. Solving this problem for insertions and deletions in the text remains an interesting open problem. In addition, we would like to extend our algorithm to allow a general alphabet; currently the assumption is that the alphabet is linearly bounded by m. Other directions would be to solve *approximate* pattern matching or *multiple* pattern matching over a dynamic text.

References

1. S. Alstrup, G. S. Brodal, T. Rauhe: Pattern matching in dynamic texts. *Proc. of the Symposium on Discrete Algorithms* (2000) 819–828
2. A. Amir, G. Landau, and D. Sokol: Inplace run-length 2d compressed search. *Theoretical Computer Science* **290**, 3 (2003) 1361–1383
3. A. Buchsbaum, M. Goodrich and J. Westbrook: Range searching over tree cross products. *Proc. of European Symposium of Algorithms* (2000) 120–131
4. R. Cole and R. Hariharan: Tighter upper bounds on the exact complexity of string matching. *SIAM J. on Computing* **26**,3(1997) 803–856
5. Martin Farach: Optimal suffix tree construction with large alphabets. *Proc. of the Symposium on Foundations of Computer Science* (1997) 137–143
6. M. Farach and S. Muthukrishnan: Perfect hashing for strings: formalization and algorithms. *Proc. of Combinatorial Pattern Matching* (1996) 130–140
7. P. Ferragina and R. Grossi: Fast incremental text editing. *Proc. of the Symposium on Discrete Algorithms* (1995) 531–540
8. H.N. Gabow, J. Bentley, and R.E. Tarjan. Scaling and related techniques for geometric problems. *Proc. of the Symposium on Theory of Computing* (1984) 135–143
9. M. Gu, M. Farach, and R. Beigel: An efficient algorithm for dynamic text indexing. *Proc. of the Symposium on Discrete Algorithms* (1994) 697–704
10. T. Hagerup, P.B. Miltersen and R. Pagh: Deterministic dictionaries. *J. of Algorithms* **41** (2000) 69–85
11. D. Harel and R. E. Tarjan: Fast algorithms for finding nearest common ancestors. *SIAM J. on Computing* **13**,2, (1984) 338–355
12. D. Knuth, J. Morris and V. Pratt: Fast pattern matching in strings. *SIAM J. on Computing* **6**,2 (1977) 323–350
13. G.M. Landau and U. Vishkin: Fast string matching with k differences. *Journal of Computer and System Sciences* **37**,1 (1988) 63–78
14. E. M. McCreight: A space-economical suffix tree construction algorithm. *J. of the ACM* **23** (1976) 262–272
15. S. C. Sahinalp and U. Vishkin: Efficient approximate and dynamic matching of patterns using a labeling paradigm. *Proc. of the Symposium on Foundations of Computer Science* (1996) 320–328
16. E. Ukkonen: On-line construction of suffix trees. *Algorithmica* **14** 249–260

17. P. van Emde Boas: An $O(n \log \log n)$ on-line algorithm for the insert-extract min problem. *Technical Report, Department of Computer Science, Cornell University*, Number TR 74-221 (1974)
18. P. Weiner: Linear pattern matching algorithm. *Proc. of the Symposium on Switching and Automata Theory* (1973) 1–11
19. D.E. Willard: Log-logarithmic worst case range queries are possible in space θ(n). *Information Processing Letters* **17** (1983) 81–84

Real Two Dimensional Scaled Matching

Amihood Amir[1⋆], Ayelet Butman[2], Moshe Lewenstein[2], and Ely Porat[2]

[1] Bar-Ilan University
amir@cs.biu.ac.il
[2] Bar-Ilan University
{ayelet,moshe,porately}@cs.biu.ac.il

Abstract. *Scaled Matching* refers to the problem of finding all locations in the text where the pattern, proportionally enlarged according to an *arbitrary real-sized* scale, appears. Scaled matching is an important problem that was originally inspired by Computer Vision.
Finding a *combinatorial* definition that captures the concept of real scaling in discrete images has been a challenge in the pattern matching field. No definition existed that captured the concept of real scaling in discrete images, without assuming an underlying continuous signal, as done in the image processing field. We present a combinatorial definition for real scaled matching that scales images in a pleasing natural manner. W e also present efficient algorithms for real scaled matching. The running time of our algorithm is as follows. If T is a two-dimensional $n \times n$ text array and P is a $m \times m$ pattern array, we find in T all occurrences of P scaled to any real value in time $O(nm^3 + n^2 m \log m)$.

1 Introduction

The original classical string matching problem [7,11] was motivated by text searching. Indeed practically every text editor uses some variant of the Boyer-Moore algorithm [7].
Wide advances in technology, e.g. computer vision, multimedia libraries, and web searches in heterogeneous data, point to a glaring lack of a theory of multidimensional matching [15].
The last decade has seen some progress in this direction. Issues arising from the digitization process were examined by Landau and Vishkin [14]. Once the image is digitized, one wants to search it for various data. A whole body of literature examines the problem of seeking an object in an image.
In reality one seldom expects to find an exact match of the object being sought, henceforth referred to as the *pattern*. Rather, it is interesting to find all text locations that "approximately" match the pattern. The types of differences that make up these "approximations" are:

⋆ Partially supported by ISF grant 282/01. Part of this work was done when the author was at Georgia Tech, College of Computing and supported by NSF grant CCR-01-04494.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 353–364, 2003.
© Springer-Verlag Berlin Heidelberg 2003

1. *Local Errors* - introduced by differences in the digitization process, noise, and occlusion (the pattern partly obscured by another object).
2. *Scale* - size difference between the image in the pattern and the text.
3. *Rotation* - The pattern image appearing in the text in a different angle.

Some early attempts to handle local errors were made in [12]. These results were improved in [5]. The algorithms in [5] heavily depend on the fact that the pattern is a rectangle. In reality this is hardly ever the case. In [4], Amir and Farach show how to deal with local errors in non-rectangular patterns.

The rotation problem has proven quite difficult. There is currently no known asymptotically efficient algorithm for finding all rotated occurrences of a pattern in an image. Fredriksson and Ukkonen [8], give a reasonable definition of rotation in discrete images and introduce a filter for seeking a rotated pattern.

More progress has been made with scaling. In [6] it was shown that all occurrences of a given rectangular pattern in a text can be found in *all discrete scales* in linear time. By discrete scales we mean natural numbers, i.e. the pattern scaled to sizes $1, 2, 3, \ldots$. The algorithm was linear for fixed bounded alphabets, but was not linear for unbounded alphabets. This result was improved in [2].

The above papers dealt with *discrete scales* only. There is some justification for dealing with discrete scales in a combinatorial sense, since it is not clear what is a fraction of a pixel. However, in reality an object may appear in non-discrete scales. It is necessary to, both, define the combinatorial meaning of such scaling, and present efficient algorithms for the problem's solution. A first step in this direction appeared in [1], however that paper was limited to *string* matching with non-discrete scales. There was still no satisfactory rigorous definition of scaling in an "exact matching" sense of combinatorial pattern matching.

In this paper we present a definition for *scaled pattern matching* with arbitrary real scales. The definition is pleasing in a "real-world" sense. We have scaled "lenna" to non-discrete scales by our definition and the results look natural (see Figure 1). This definition was inspired by the idea of digitizing analog signals by sampling, however, it does not assume an underlying continuous function thus stays on the combinatorial pattern matching field. We believe this is the natural way to define combinatorially the meaning of scaling in the signal processing sense.

We believe this definition, that had been sought by researchers in pattern matching since at least 1990, captures *scaling* as it occurs in images, yet has the necessary combinatorial features that allows developing deterministic algorithms and analysing their worst-case complexity. Indeed we present an efficient algorithm for real scaled two dimensional pattern matching.

The running times of our algorithm is as follows. If T is a two-dimensional $n \times n$ text array and P is a $m \times m$ pattern array, we find in T all occurrences of P scaled to any real value in time $O(nm^3 + n^2 m \log m)$.

The main achievements of this paper are pinning down a rigorous combinatorial definition for exact real scaling in images and producing efficient algorithms for scaled matching. The new techniques developed in this paper are analysis of

Fig. 1. An original image, scaled by 1.3 and scaled by 2, using our combinatorial definition of scaling.

the properties of scaled arrays and two-dimensional dictionary matching with a compressed dictionary.

2 Scaled Matching Definition

Definition 1. *Let T be a two-dimensional $n \times n$ array over some finite alphabet Σ.*

1. *The* unit pixels array *for T (T^{1X}) consists of n^2 unit squares, called pixels in the real plane R^2. The corners of the pixel $T[i,j]$ are $(i-1,j-1),(i,j-1),(i-1,j)$, and (i,j). Hence the pixels of T form a regular $n \times n$ array that covers the area between $(0,0),(n,0),(0,n)$, and (n,n). Point $(0,0)$ is the* origin *of the unit pixel array. The* center *of each pixel is the geometric center point of its square location. Each pixel $T[i,j]$ is identified with the value from Σ that the original array T had in that position. We say that the pixel has a color from Σ. See figure 2 for an example of the grid and pixel centers of a 7×7 array.*
2. *Let $r \in \Re$, $r \geq 1$. The* r-ary pixels array *for T (T^{rX}) consists of n^2* r-squares*, each of dimension $r \times r$ whose* origin *is $(0,0)$ and covers the area between $(0,0),(nr,0),(0,nr)$, and (nr,nr). The corners of the pixel $T[i,j]$ are $((i-1)r,(j-1)r),(ir,(j-1)r),((i-1)r,jr)$, and (ir,jr). The* center *of each pixel is the geometric center point of its square location.*

Notation: Let $r \in \Re$. $\|r\|$ denotes the *rounding* of r, i.e. $\|r\| = \begin{cases} \lfloor r \rfloor & \text{if } r - \lfloor r \rfloor < .5; \\ \lceil r \rceil & \text{otherwise.} \end{cases}$
There may be cases where we need to round 0.5 down. For this we denote:
$||\lfloor r \rfloor|| = \begin{cases} \lfloor r \rfloor & \text{if } r - \lfloor r \rfloor \leq .5; \\ \lceil r \rceil & \text{otherwise.} \end{cases}$

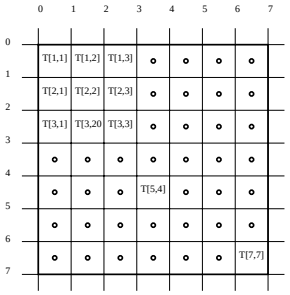


Fig. 2. The grid and pixel centers of a unit pixel array for a 7×7 array.

Definition 2. *Let T be an $n \times n$ text array and P be an $m \times m$ pattern array over alphabet Σ. Let $r \in \Re,\ 1 \leq r \leq \frac{n}{m}$.*
We say that there is an occurrence of P scaled to r *at text location $[i, j]$ if the following condition holds:*
Let T^{1X} be the unit pixels array of T and P^{rX} be the r-ary pixel arrays of P. Translate P^{rX} onto T^{1X} in a manner that the origin of P^{rX} coincides with location $(i-1, j-1)$ of T^{1X}. Every center of a pixel in T^{1X} which is within the area covered by $(i-1, j-1), (i-1, j-1+mr), (i-1+mr, j-1)$ and $(i-1+mr, j-1+mr)$ has the same color as the r-square of P^{rX} in which it falls.
The colors of the centers of the pixels in T^{1X} which are within the area covered by $(i-1, j-1), (i-1, j-1+mr), (i-1+mr, j-1)$ and $(i-1+mr, j-1+mr)$ define a $\|mr\| \times \|mr\|$ array over Σ. This array is denoted by P^r and called P scaled to r.

It is possible to find all scaled occurrences of an $m \times m$ pattern in an $n \times n$ text in time $O(n^2m^2)$. Such an algorithm, while not trivial, is nonetheless achievable with known techniques. We present below an $O(nm^3 + n^2 m \log m)$ algorithm. The efficiency of the algorithm results from the properties of scaling. The scaling definition needs to accommodate a conflict between two notions, the continuous (represented by the real-number scale), and the discrete (represented by the array representation of the images). Understanding, and properly using, the shift from the continuous to the discrete and back are key to the efficiency of our algorithms. To this effect we need the following functions.

Definition 3. *Let k be a discrete length of a pattern prefix in any dimension, i.e. the number of consecutive rows starting from the pattern's beginning, or the length of a row prefix. Let $r \in \Re$ be a scale, and let N be the natural numbers. We define the function $D : N \times \Re \to N$ as follows: $D(k, r) = \|kr\|$.*

We would like to define an "inverse" function $D^{-1} : N \times N \to \Re$ with the property $D^{-1}(k, D(k, r)) = r$. However, that is not possible since D is not injective. Claim 2, which follows from the definition, below tells us that for a fixed k there is a structure to the real numbers r that are mapped to the same element $D(k, r)$, namely, they form an interval $[r_1, r_2)$.

Claim. Let $r_1, r_2 \in \Re,\ k \in N$ such that $D(k, r_1) = D(k, r_2)$ and let $r \in \Re,\ r_1 < r < r_2$. Then $D(k, r) = D(k, r_1)$.

Definition 4. *Let $k, \ell \in N$. Define*
$L^{-1}(k,\ell) = \begin{cases} 1 & \text{if } k = \ell; \\ \frac{(\ell-0.5)}{k} & \text{otherwise.} \end{cases}$
and $R^{-1}(k,\ell) = \frac{(\ell+0.5)}{k}$.

It is easy to see that $L^{-1}(k,\ell) = \min\{r \in \Re | D(k,r) = \ell\}$ and that $R^{-1}(k,\ell) = \min\{r \in \Re | D(k,r) = \ell + 1\}$.
The L^{-1} and R^{-1} functions are designed to give a range of scales whereby a pattern sub-range of length k may scale to a sub-range of scale ℓ. The following claim follows from the definition.

Claim. Let P be an $m \times m$ pattern and T an $n \times n$ text. Let $k \leq m$ and $\ell \leq n$, and let $[L^{-1}(k,\ell), R^{-1}(k,\ell))$ be the range of scales defined by L^{-1} and R^{-1}. Then the difference in number of rows (or number of columns) between P^{r_1} and P^{r_2}, for any two $r_1, r_2 \in [L^{-1}(k,\ell), R^{-1}(k,\ell))$ can not exceed $m + 2$.

3 The Number of Different Scaled Patterns

We utilize various properties of the scaled patterns to aid our search. One of the difficulties presented are that it is possible to have several values $r_1, r_2, ..., r_k \in \Re$ for which $P^{r_1}, P^{r_2}, ..., P^{r_k}$ are different matrices yet all have the same dimensions. See Figure 3. The following claim limits the overall number of different possible matrices that represent scaled occurrences of a given pattern matrix P.

Claim. Let P be an $m \times m$ pattern over finite alphabet Σ. Then there are $O(nm)$ different matrices representing the occurrences of P scaled to all $r \in \Re,\ 1 \leq r \leq \frac{n}{m}$.

Proof. There are $n - m$ different possible sizes of matrices representing a scaled P whose maximum size is $n \times n$. By lemma 1 below, each one of the $n - m$ different matrix sizes has at most m possibilities of matrices representing scaled versions of P, for a total of $O(nm)$. □

Lemma 1. *Let $m \times m$ pattern P be scaled to size $\ell \times \ell,\ \ell \geq m$. Then there are k different intervals, $[a_1, a_2), [a_2, a_3), ..., [a_k, a_{k+1}),\ k \leq m, 1 \leq a_1 < a_2 < \cdots < a_{k+1}$ for which the following hold:*
1. *$P^{r_1} = P^{r_2}$, if $r_1, r_2 \in [a_i, a_{i+1})$, $1 \leq i \leq k$.*
2. *$P^{r_1} \neq P^{r_2}$, if r_1 and r_2 are in different intervals.*
3. *P^r is an $\ell \times \ell$ matrix iff $r \in [a_1, a_{k+1})$.*

Proof. Omitted for space reasons.

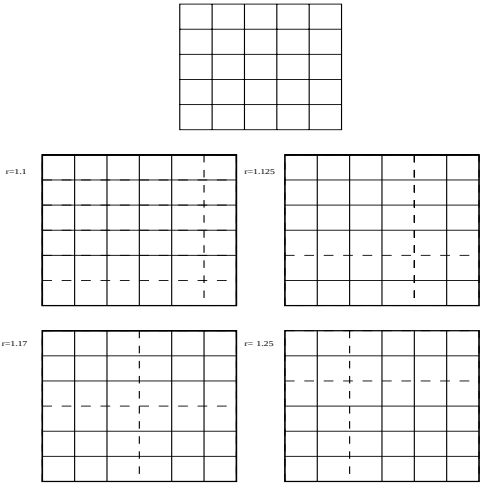


Fig. 3. The 5×5 pattern scaled to $1.1, 1.125, 1.17$ and 1.25 produces a 6×6 pattern. In each of these cases some row and some column needs to be repeated. The dashed grid line indicates the repeating row or column (both rows or columns on the two sides of the dashed grid line are equal).

4 The Scaled Matching Algorithm's Idea

The naive, straightforward idea is to construct a dictionary of the $O(nm)$ different possible scaled occurrences of P and use a two-dimensional dictionary matching algorithm (e.g. [3,10,9]) that can scan the text in linear time and find all dictionary occurrences. The trouble with this idea is that the different matrices in the dictionary range in sizes from m^2 to n^2 which will make the dictionary size $O(n^3m)$, which we are not willing to pay.
Our idea, then, is to keep the dictionary in a compressed form. The compression we use is *run-length* of the rows.

Definition 5. *Let $S = \sigma_1\sigma_2 \cdots \sigma_n$ be a string over some alphabet Σ. The* run-length representation *of string S is the string $S' = \sigma_1'^{r_1}\sigma_2'^{r_2} \cdots \sigma_k'^{r_k}$ such that: (1) $\sigma_i' \neq \sigma_{i+1}'$ for $1 \leq i < k$; and (2) S can be described as concatenation of the symbol σ_1' repeated r_1 times, the symbol σ_2' repeated r_2 times, ..., and the symbol σ_k' repeated r_k times.*

We denote by $S'^{\Sigma} = \sigma_1'\sigma_2' \cdots \sigma_k'$, the symbol part *of S', and by $c(S)$, the vector of natural numbers $r_1, r_2, ..., r_k$, the* run-length part *of S'. We say that the* number of runs of S is k *and denote it by $|c(S)|$. For j, $1 \leq j \leq k$ denote* $\mathsf{prefixsum}(S, j) = \sum_{i=1}^{j} r_i$.

Since every scaled occurrence has at most m different rows, each repeating a certain number of times, and each row is of run-length at most m, then we can encode the information of every pattern scaled occurrence in space $O(m^2)$. Now our dictionary is of size $O(nm^3)$. The construction details are left for the

journal version. The challenge is to perform dictionary matching using a compressed dictionary. We show in Section 5 that such a search can be done in time $O(n^2 m \log m)$.

The idea behind the text searching is as follows: For every text location $[i, j]$, we assume that there is a pattern scaled occurrence beginning at that location. Subsequently, we handle every one of the pattern m rows separately (in time $O(\log m)$ for each row). For every row, we establish the number of times this row repeats in the text. This allows us to narrow down the range of possible scales for this occurrences and compare the row to an appropriately scaled pattern row from the dictionary. The high-level description of the text searching algorithm appears below. We denote the ℓth row of P by P_ℓ.

Scaled Text Scanning (high level)

For every text location $[i, j]$ do:
  Set $k \leftarrow i$.
  For pattern row $\ell = 1$ to m do:

1. Establish the number of times the subrow of T starting at location $[k, j]$ and whose run-length equals the run-length of P_ℓ repeats in the text.
2. If this number of repetitions is incompatible with the numbers for rows $1, ..., \ell - 1$ then halt – no scaled occurrence possible at $[i, j]$.
3. Binary search all the dictionary rows P_ℓ in the appropriate scale compared to the run-length $c(P_\ell)$ subrow starting at $T[k, j]$. If no match then halt – no scaled occurrence at $[i, j]$.
   update the possible range of scales and k.

  EndFor
EndFor

end Scaled Text Scanning

5 Scaled Search Algorithm Implementation

We describe the details of efficiently computing the high-level steps of Section 4.

5.1 Computing Text Subrow Repetition

Consider a given text location $[i, j]$. We are interested in ascertaining whether there exists a $r \in \Re$ for which P^r occurs starting at location $[i, j]$. If such an r exists, the first pattern row starts at location $[i, j]$ and repeats $\|r\|$ times. Since we do not know the scale r, we work backwards. If we know the number of repetitions of the first subrow, we will be able to derive $\|r\|$. In [6] a method was presented that preprocesses an $n \times n$ text matrix in time $O(n^2)$ and subsequently allows answering every subrow repetition query in constant time. A *subrow repetition query* is defined as follows: Given an $n \times n$ matrix T,
Input: Location $[i, j]$ and natural numbers k, ℓ.

Output: Decide whether the substring $T[i,j], T[i,j+1], ..., T[i,j+\ell-1]$ repeats k consecutive times starting at column j in rows $i, i+1, ..., i+k-1$ of T. Formally, decide whether for every natural number $y \in \{0, ..., \ell-1\}$ it is true that $T[i+x, j+y] = T[i, j+y], \quad x = 1, ..., k-1$.

The Scale Range Computation – Overview. Every text subrow gives us *horizontal* scaling information, by analysing how many times each symbol repeats in the subrow, and *vertical* scaling information, by computing how many times the subrow repeats until it changes. Note that both the horizontal and vertical scaling information are exact up until the last run-length symbol and the last unique row. Both may repeat in the text for longer than the scale. However, assuming that there is either a row or a column of run-length at least 3, the "inner part" of the run-length places some limitations on the scale range.
The strategy of our algorithm is as follows. For text location $[i,j]$, we compute the range induced by the horizontal scale of the first row, and update the range by the vertical scale of the first row, then move on to the next distinct row, and continue until all pattern rows are accounted for. This means we handle $O(m)$ distinct rows per text location $[i,j]$. The horizontal scale calculations are a constant number of numeric operations per row. The vertical scale computation utilizes the subrow repetition queries. However, not knowing a-priori how many times a row repeats, we do a binary search on the values induced by the horizontal scale range. Claim 2 guarantees that the range of these values can not exceed $m+2$.

The Scale Range Computation – Details Terminology: We denoted the rows of pattern P by $P_1, P_2, ..., P_m$. We can conceptualize the pattern as a string with every row a symbol. Consider the run-length representation of this string, $(P'^{\Sigma}[1])^{c(P)[1]}(P'^{\Sigma}[2])^{c(P)[2]} \cdots (P'^{\Sigma}[k])^{c(P)[k]}$. We call this presentation the *row run-length presentation of the pattern*. We will calculate the subrow length for the rows grouped according to the row run-length representation.
Notation: Denote by $T_{i,j}$ the subrow $T[i,j], T[i,j+1], ..., T[i,n]$.
The details of the scale range computation follow:
For every location $[i_0, j_0]$ in the text calculate the scale range in the following manner:
Initialization:

Set the text row variable tr to i_0
Initialize the pattern row pr to 1.
Initialize the pattern run-length row rr to 1.
Initialize the scale range to $[1, \frac{n}{m})$.

Assume that the scale range $[a,b)$ has been computed so far, and the algorithm is now at text row tr, pattern row pr, and pattern run-length row rr.
Update Scale Range:
Distinguish between three cases:
1. The run-length of pattern row P_{pr} is 1.
2. The run-length of pattern row P_{pr} is 2.
3. The run-length of pattern row P_{pr} is 3 or more.

Each of the three cases is handled somewhat differently: At any stage of the algorithm if an intersection is empty, the algorithm is halted for location $[i_0, j_0]$ – no scaled occurrence. We omit the details of the simple cases 1 and 2 for space reasons and present the interesting case, case 3.

Case 3:

Let the new values of $[a, b)$ be the intersections of the current $[a, b)$ and $[L^{-1}(s, s'), R^{-1}(s, s'))$ where $s = \text{prefixsum}(P_{pr}, |c(P_{pr})| - 1), s' = \text{prefixsum}(T_{tr,j_0}, |c(P_{pr})| - 1)$

Using a "binary search on $[a, b)$" (see Section 5.1) determine the number k of row repetitions.

$tr \leftarrow tr + k$; $pr \leftarrow pr + c(P)[rr]$; $rr \leftarrow rr + 1$.

Check if text subrow is equal to pattern row (see Section 5.2).

end Case 3

We do not want to explicitly check that all symbols in the text match the pattern, since that could take time $O(m)$. So we just make sure the text row repeats a sufficient amount of times. In Section 5.2 we show how to compare the text and pattern rows quickly.

Implementation Details: The prefixsum computations and several computations of the L^{-1} and R^{-1} functions are performed on a run-length compression of the text and pattern, whereas the subrow repetition queries are performed on the uncompressed text and pattern. An initial preprocessing stage will create compressed text and pattern, with double pointers of locations in the compressed and uncompressed text and pattern. All such preprocessing can be done in linear time and space using standard techniques.

Time: The running time for every text location is $O(mt)$, where t is the time it takes to do the binary search. We will see in the next section that the binary search is done in time $O(\log m)$. Thus the total search time is $O(n^2 m \log m)$.

Binary Search on $[a, b)$. $[a, b)$ is an interval of real numbers, thus it is impossible to do "binary search" on it. When we say "binary search" on $[a, b)$ we mean the discrete number of rows that P_{pr} repeats scaled to r for $r \in [a, b)$. This number is at most $\lfloor \frac{n}{m} \rfloor$, and we will show that we can actually limit it to $O(m)$. In any event, we have a finite set of natural numbers to search.

The values we need to search are when a block of k equal subrows of length ℓ occur starting in text location $[tr, j_0]$. It is easy to see that D is non-decreasing in both variables. Therefore, consider a (k, ℓ) block to be *valid* if the k subrows of length ℓ are all equal. The monotonicity of k in ℓ guarantees that if a (k, ℓ) block is not valid, no greater block with a greater ℓ (or k) is valid. If a (k, ℓ) block is valid, all smaller blocks are valid. Thus, it is possible to do a binary search on k or ℓ to find the largest k that gives a valid block.

The only information still needed is how to compute the k and ℓ from interval $[a, b)$. Note that the D function computes from a number of repetitions in P and a given real scale r, the number of times those repetitions will scale to r. It seems like the function we need, and it actually does the job for computing the ℓ. However, we have a problem with the k. The definition of D assumes that the repetitions start at the *beginning* of the pattern on the pixel array, and the rounding is done at the end. Now, however, we find ourselves in the middle of

the pattern. This means that the pattern rows we are interested in may start in the middle of a pixel in the text pixel array. The D function would assume they start at the beginning of a pixel in the text pixel array and provide an answer that may be incorrect by 1. An example can be seen in Figure 4.

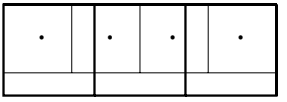


Fig. 4. Assume the pattern's first row is the three symbols *abc*. Consider $P^{1\frac{1}{3}}$. $D(1, 1\frac{1}{3}) = 1$. Yet, the second symbol extends over **two** pixel centers.

Definition 6. *Let k be a discrete length of a sub-pattern in any dimension, starting from location i in the pattern. Let $r \in \Re$ be a scale, and let N be the natural numbers. We define the function $D' : N \times N \times \Re \to N$ as follows: $D'(i, k, r) = D(i + k - 1, r) - D(i - 1, r)$. Since D' is a straightforward generalization of D ($D(k, r) = D'(1, k, r)$), and to avoid unnecessary notation, we will refer to both functions as D.*

It is easy to see that $D(i, k, r)$ is indeed the number of discrete rows or columns that k in position i scales to by r.

Binary Search on $[a, b)$.

Perform a binary search on the values $\ell \in \{D(m, a), ..., \lfloor\lfloor mb \rfloor\rfloor\}$.

For each value ℓ compute the two values k_1, k_2 for which to perform subrow repetition queries starting at location $[tr, j_0]$, with k_1 or k_2 repetitions of length ℓ as follows:

Let $x = L^{-1}(m, \ell), \quad y = R^{-1}(m, \ell)$.

$k_1 = D(pr, c(P)[rr], x)$ and $k_2 = D(pr, c(P)[rr], y)$.

If both subrow repetition queries are positive, this is a valid case. The ℓ value should be increased.

If both subrow repetition queries are negative, this is an invalid case. The ℓ value should be decreased.

If the k_1 subrow repetition query is positive and k_2 is negative the search is done with $k = k_1$.

Correctness: It follows from claims 5.1 and 5.1 that the binary search indeed finds the correct number of repetitions. Their proofs are technical and are omitted for lack of space.

Claim. The binary search algorithm considers all possibilities of k and ℓ.

Claim. Let $r \in \Re$ and assume there is an occurrence of P^r starting at location $[i, j]$ of T. Then the $c(P)[rr]$ rows of the pattern starting at pattern row pr are scaled to the value k discovered by our algorithm.

Time: A subrow repetition query takes constant time. The number of queries we perform is logarithmic in the range of the binary search. By Claim 2, if we know for any number of rows and columns in the pattern, to exactly how many rows or columns they scale, the range is then $O(m)$. Note that in all cases except for Case 1, Claim 2 holds. Even in Case 1, the claim does not hold only for cases where the first pattern rows are all of run-length 1. This is a very trivial pattern and all its scaled occurrences can be easily detected by other easy means. Every other case has at least a row or a column with run-length greater than 1. Without loss of generality we may assume that there is a row r_0 of run-length greater than 1 (otherwise we rotate the text and pattern by 90^o). We consider the pattern P' consisting of rows $P_{r_0}, P_{r_0+1}, ..., P_m$. We compute the ranges of all possible scales of P' for every text location, as described above, and then eliminate all locations where rows $P_1, ..., P_{r_0-1}$ do not scale appropriately. By this scheme there is never a binary search on range greater than $m + 2$. Thus the total time for the binary search is $O(\log m)$.

5.2 Efficient Subrow Comparison

At this stage we know, for every text location, the number of repetitions of every row. But we do not know if the repeating text subrow is, indeed, the appropriate scaled pattern row, for any pattern row whose run-length exceeds two. Checking every row element by element would add a factor of m to our complexity, which we want to avoid. It would be helpful if we could compare entire subrows in constant time. We are interested in a query of the form below.

Given a string S of length n over *ordered* alphabet Σ, a *substring comparison query* is defined as follows.

Input: Locations $i, j, \quad 1 \leq i, j, \leq n$ and length ℓ.

Output: Decide whether the substring $S[i], S[i+1], ..., S[i+\ell-1]$ is lexicographically larger, smaller or equal to substring $S[j], S[j+1], ..., S[j+\ell-1]$.

In [13] a method was presented that preprocesses a string of length n in time $O(n \log \sigma)$, where $\sigma = \min\{m, |\Sigma|\}$, and subsequently allows answering *longest common prefix* queries in constant time. A *Longest Common Prefix* query is defined as follows:

Input: Locations $i, j, \quad 1 \leq i, j, \leq n$.

Output: The length of the longest common prefix of the substrings $S[i], S[i+1], ..., S[n]$ and $S[j], S[j+1], ..., S[n]$, i.e. the number m for which $S[i+\ell] = S[j+\ell], \quad \ell = 0, ..., m-1$ and $S[i+m] \neq S[j+m]$ or one of $i+m, j+m$ is greater than n.

Using the Landau and Vishkin method, we can also answer the substring comparison query in constant time, following a $O(n \log \sigma)$-time preprocessing, in the following way.

Given locations $i, j, \quad 1 \leq i, j, \leq n$ and length ℓ, find the length k of the longest common prefix of $S[i], S[i+1], ..., S[n]$ and $S[j], S[j+1], ..., S[n]$. If $k \geq \ell$ then the two substrings $S[i], S[i+1], ..., S[i+\ell-1]$ and $S[j], S[j+1], ..., S[j+\ell-1]$ are equal. Otherwise, compare $S[i+k]$ and $S[j+k]$. If $S[i+k] > S[j+k]$ then the substring $S[i], S[i+1], ..., S[i+\ell-1]$ is lexicographically larger than

$S[j], S[j+1], ..., S[j+\ell-1]$, and if $S[i+k] < S[j+k]$ then the substring $S[i], S[i+1], ..., S[i+\ell-1]$ is lexicographically smaller than $S[j], S[j+1], ..., S[j+\ell-1]$. Our algorithm will do a binary search on a precomputed run-length compress dictionary of all scaled possibilities on the row. While the symbol part of the run-length can indeed be checked in such a dictionary, the numerical part is more problematic. The difficulty is that our text subrow is not wholly given in run-length compressed form. The first and last symbols of the run-length compressed row may occur in the text more times than in the scaled pattern. The details of verifying that the numerical part of the run-length of the pattern row matches the run-length of the text subrow are left for the journal version.

References

1. A. Amir, A. Butman, and M. Lewenstein. Real scaled matching. *Information Processing Letters*, 70 (4) : 185–190, 1999.
2. A. Amir and G. Calinescu. Alphabet independent and dictionary scaled matching. *Proc. 7th Annual Symposium on Combinatorial Pattern Matching (CPM 96)*, pages 320–334, 1996.
3. A. Amir and M. Farach. Two dimensional dictionary matching. *Information Processing Letters*, 44: 233–239, 1992.
4. A. Amir and M. Farach. Efficient 2-dimensional approximate matching of half-rectangular figures. *Information and Computation*, 118(1):1–11, April 1995.
5. A. Amir and G. Landau. Fast parallel and serial multidimensional approximate array matching. *Theoretical Computer Science*, 81:97–115, 1991.
6. A. Amir, G.M. Landau, and U. Vishkin. Efficient Pattern matching with scaling. *Journal of Algorithms*, 13(1):2–32, 1992.
7. R.S. Boyer and J.S. Moore. A fast string searching algorithm. *Comm. ACM*, 20:762–772, 1977.
8. K. Fredriksson and E. Ukkonen. A rotation invariant filter for two-dimensional string matching. In *Proc. 9th Annual Symposium on Combinatorial Pattern Matching (CPM 98)*, pages 118–125. Springer, LNCS 1448, 1998.
9. R. Giancarlo and R. Grossi. On the construction of classes of suffix trees for Square matrices: Algorithms and applications. *Information and Computation*, 130(2):151–182, 1996.
10. R.M. Idury and A.A Schäffer. Multiple matching of rectangular Patterns. Proc. *25th ACM STOC*, pages 81–89, 1993.
11. D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast Pattern matching in strings. *SIAM J. Comp.*, 6:323–350, 1977.
12. K. Krithivansan and R. Sitalakshmi. Efficient two dimensional Pattern matching in the presence of errors. *Information Sciences*, 13:169–184, 1987.
13. G. M. Landau and U. Vishkin. Efficient string matching with k mismatches. *Theoretical Computer Science*, 43:239–249, 1986.
14. G. M. Landau and U. Vishkin. Pattern matching in a digitized image. *Algorithmica*, 12 (3/4):375–408, 1994.
15. A. Pentland. Invited talk. NSF Institutional Infrastructure Workshop, 1992.

Proximity Structures for Geometric Graphs

Sanjiv Kapoor and Xiang-Yang Li

Department of Computer Science, Illinois Institute of Technology
10 W. 31st Street, Chicago, IL 60616, USA
kapoor@iit.edu,xli@cs.iit.edu

Abstract. In this paper we study proximity structures like Delauney triangulations based on geometric graphs, i.e. graphs which are subgraphs of the complete geometric graph. Given an arbitrary geometric graph G, we define several restricted Voronoi diagrams, restricted Delaunay triangulations, relative neighborhood graphs, Gabriel graphs and then study their complexities when G is a general geometric graph or G is some special graph derived from the application area of wireless networks. Besides being of fundamental interest these structures have applications in topology control for wireless networks.

1 Introduction

Given a set S of two dimensional points, many geometric proximity structures were defined for various applications, such as the Delaunay triangulation [1,2, 3], the Voronoi Diagram [2,3], the Gabriel graph (GG) [4,5], and the relative neighborhood graph (RNG) [6,7,8]. These diagrams are defined with respect to a geometric neighborhood. For example an edge uv is in GG if and only if the circle with uv as a diameter, denoted by $disk(u, v)$, is empty of any other points of S inside. An edge is in RNG if and only if the lune defined by this edge is empty. The *lune* defined by edge uv, denoted by $lune(u, v)$, is the intersection of two disks centered at u and v with radius $\|uv\|$. Obviously, RNG is a subgraph of GG, which is a subgraph of the Delaunay triangulation. Since Delaunay triangulation is planar, all these three structures are planar and have at most $O(n)$ edges.

All these structures are defined solely on the given point set and can be viewed as defined on the complete geometric graph topology. Recently, Li *et al.* [9], motivated by constructing distributed protocols for network routing in mobile networks, extended these definitions to account for the edge structures in the unit disk graph. The unit disk graph is used for topology control and power efficient topology construction for wireless ad hoc networks. In wireless ad hoc networks, nodes can directly communicate with all nodes within its transmission range, which is often normalized to one unit. For a unit disk graph G, [9] defined the k-localized Delaunay graph as follows. A triangle $\triangle uvw$ formed by edges in G is a k-localized Delaunay triangle if its circumcircle is empty of nodes which are within k hops of u, or v, or w. The k-localized Delaunay graph $LDel^k$ contains all k-localized Delaunay triangles and all Gabriel graph edges on G. In [9] it is shown that $LDel^k$ is a planar graph for $k \geq 2$ and $LDel^1$ has thickness 2.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 365–376, 2003.
© Springer-Verlag Berlin Heidelberg 2003

However, graphs representing communication links are rarely so completely specified as the unit disk graph. We thus consider the general structure of arbitrary graphs defined by points in the plane, *geometric graphs*, i.e., its edges are straightline segment connecting the endpoints. For example, for wireless communications, different nodes may have different transmission radius. Consequently, two nodes can communicate directly if they are within the transmission range of each other, i.e., there is a communication link between these two nodes. The graph formed by all such communication links is different from the traditional disk graph, in which two nodes are connected by a straight edge if the two corresponding disks centered at these two nodes intersect. And for wireless communications, two nodes sometimes cannot communicate directly even though they are within the transmission range of each other, due to the blocking of the signal by some barrier. As another example, paths may be required to be found in visibility graphs defined amongst polygonal obstacles in the plane. Traditional proximity structures are often defined based solely on the information of points. We consider the effect on these proximity structures biased by the changed neighborhood created by the topology of geometric graphs. The use of these proximity structures to reduce the complexity of the underlying graph while still retaining connectivity or path properties of the original graph is an interesting issue for research.

In this paper we first present several new proximity structures, based on a given geometric graph $G = (V, E)$. We show relationships between these structures and bounds on their sizes. Most of our definitions are for undirected graph, but can be extended to directed graphs also. Let $N_G^k(u)$ be all nodes that are within k hops of node u in G. We define the zero-edge oriented localized Delaunay graph on graph G, denoted by $LDel_0(G)$. This consists of all edges $uv \in E$ such that there is a circle passing through u and v, which contains no other point w inside the circle. The one-edge oriented k-localized Delaunay graph on G, denoted by $LDel_1^k(G)$, consists of all edges $uv \in E$ such that there is a circle passing through u and v, which contains no point $w \in N_G^k(u) \cup N_G^k(v)$ inside. Finally, the two-edge oriented k-localized Delaunay neighborhood graph on G, denoted by $LDel_2^k(G)$, consists of all edges $uv \in E$ such that there is a circle passing through u and v, which contains no point $w \in N_G^k(u) \cap N_G^k(v)$ inside.

These definitions are extended in the natural way to Gabriel Graphs and the relative neighborhood graphs. Define the k-localized Voronoi region of a vertex v as the set of points p such that v is the closest vertex to p among v and all nodes w such that $w \in N_G^k(u)$. The union of all such region is called the one-edge oriented k-localized Voronoi diagram, denoted by $LVor_G^k(V)$. We show that the localized Voronoi diagram and Delaunay triangulation are dual of each other: given an edge $uv \in G$, uv is in the one-edge k-localized Delaunay triangulation iff their corresponding Voronoi regions in k-localized Voronoi diagram share a common boundary.

We study the edge complexity of the proximity diagrams. Given a geometric graph G, we show that the one-edge oriented Delaunay graph, $LDel_1^k(G)$ has at most $O(n^{5/3})$ edges; and the one-edge oriented Gabriel graph has at most

$O(n^{3/2})$ edges. Notice that the zero-edge oriented structures defined so far always have at most $O(n)$ edges due to the planar property. However, the two-edge oriented structures could have $O(n^2)$ edges. When the graph G is the communication graph MG derived from the wireless networks, we show that the two-edge oriented Gabriel graph has at most $O(n^{5/3} \log \frac{r_{\max}}{r_{\min}})$ edges, where $r_{\max}$ and $r_{\min}$ are the maximum and minimum transmission range respectively. In addition, we show that all one-edge oriented localized structures on MG have thickness $1 + 2\log_2 \frac{r_{\max}}{r_{\min}}$. We also study some conditions under which the proposed structures are planar graphs.

The remaining of the paper is organized as follows. We define the generalized Delaunay triangulation and Voronoi diagram on general geometry graphs and study their duality and the edge complexity in Section 2. We further extend this ideas to the relative neighborhood graph, and Gabriel graph in Section 3. We study their properties when the geometry graph is derived from wireless communications in Section 4. We conclude our paper in Section 5.

2 Generalized Delaunay Triangulation, Voronoi Diagram

Voronoi diagram and Delaunay triangulation have been widely used in many areas. A triangulation of V is a *Delaunay triangulation*, denoted by $Del(V)$, if the circumcircle of each of its triangles does not contain any other vertices of V in its interior. The *Voronoi region*, denoted by $Vor(p)$, of a vertex p in V is a collection of two dimensional points such that every point is closer to p than to any other vertex of V. The *Voronoi diagram* for V is the union of all Voronoi regions $Vor(p)$, where $p \in V$. The Delaunay triangulation $Del(V)$ is also the dual of the Voronoi diagram: two vertices p and q are connected in $Del(V)$ if and only if $Vor(p)$ and $Vor(q)$ share a common boundary.

2.1 Definitions

In this section, we extend the Voronoi region and the Delaunay triangulation from being defined on a point set to being defined on a geometric graph.

The zero-edge oriented localized Delaunay graph on a geometry graph $G = (V, E)$, denoted by $LDel_0(G)$, consists of all edges $uv \in E$ such that there is a circle passing through u and v, containing no other point w inside the circle. Obviously, $LDel_0(G) = Del \cap G$. The one-edge oriented k-localized Delaunay graph on G, denoted by $LDel_1^k(G)$, consists of all edges $uv \in E$ such that there is a circle passing through u and v, which contains no point $w \in N_G^k(u) \cup N_G^k(v)$ inside. The two-edge oriented k-localized Delaunay neighborhood graph on G, denoted by $LDel_2^k(G)$, consists of all edges $uv \in E$ such that there is a circle passing through u and v, containing no point $w \in N_G^k(u) \cap N_G^k(v)$ inside. Notice $LDel_i^{k+1}(G) \subseteq LDel_i^k(G)$ for $i = 1, 2$, and $LDel_0(G) \subseteq LDel_1^k(G) \subseteq LGG_2^k(G)$.

Let line l_{vw} be the perpendicular bisector of segment vw and let h_{vw} denote the half-space partitioned by l_{vw}, containing the vertex v. Then it is well-known

that the Voronoi region $Vor(v) = \bigcap_{w \in V} h_{vw} = \bigcap_{vw \in Del(V)} h_{vw}$. Given a geometry graph G, the k-localized Voronoi region of a vertex $v \in V$, denoted by $LVor_G^k(v)$, is the intersection of all half-spaces h_{vw} such that $w \in N_G^k(v)$, i.e.,

$$LVor_G^k(v) = \bigcap_{w \in N_G^k(v)} h_{vw} = \{x \mid \|x - v\| \leq \|x - w\|, \forall w \in N_G^k(v)\}.$$

2.2 Duality

Let γ be a function mapping every vertex of V to a polygonal region, which could be unbounded; δ be some simple graph on V. Then functions γ and δ are *dual* of each other, denoted by $\gamma \perp \delta$, if we have: given any edge $uv \in G$, $\gamma(u)$ and $\gamma(v)$ share a common boundary segment iff vertices u and v are connected in δ. It is well-known that $Vor \perp Del$ for any point set V.

Theorem 1. *For any geometry graph G, $LVor_G^k \perp LDel_1^k(G)$.*

Proof. Given any edge $uv \in G$, if $LVor_G^k(u)$ and $LVor_G^k(v)$ share some common boundary segment then the shared common boundary must be on the perpendicular bisector l_{uv} of segment uv. Figure 1 (a) illustrates the proof that follows. Consider any point x on the shared segment of $LVor_G^k(u)$ and $LVor_G^k(v)$. For any vertex $w \in N_G^k(u)$, $\|x - u\| \leq \|x - w\|$. It implies that w is not inside the disk centered at x with radius $\|x - u\|$. Similarly, for any vertex $y \in N_G^k(v)$, $\|x - v\| \leq \|x - y\|$. It implies that y is not inside the disk centered at x with radius $\|x - v\| = \|x - u\|$. Therefore, there is a disk (centered at x) passing through vertices u, v that does not contain any vertex from $N_G^k(u) \cup N_G^k(v)$ inside. Thus, $uv \in LDel_1^k(G)$.

Consider any edge uv from $LDel_1^k(G)$. Then there is a disk passing through u, v that is empty of $N_G^k(u) \cup N_G^k(v)$. Let $B(x, \|x - u\|)$ be such disk. Then for any $w \in N_G^k(v)$, we have $\|x - u\| \leq \|x - w\|$. It implies that $x \in LVor_G^k(u)$. Similarly, $x \in LVor_G^k(v)$. Due to the presence of the edge uv in G, we know that $LVor_G^k(u)$ and $LVor_G^k(v)$ are on different sides of the bisector l_{uv}. By definition of the one-edge localized Voronoi region, we know that $LVor_G^k(u)$ and $LVor_G^k(v)$ share a common boundary segment containing point x. □

2.3 Edge Complexity

It is well-known that the Delaunay triangulation has at most $3n - 6$ edges for a two-dimensional point set from its planarity. Thus, all structures that are zero-edge oriented have at most $O(n)$ edges. However, it is easy to construct a geometry graph such that all other structures introduced so far are not planar graphs. Thus, it is not obvious how many edges each of these new structures have. Recently, there had been some studies on the complexity of these geometry structures on unit disk graphs. Li *et al.* [9] proved that the (one-edge oriented) local Delaunay triangulation on the unit disk graph has $O(n)$ edges. In this

section, we will further the study of the complexity of these structures when a more general geometry graph G is given.

We first give an upper bound on the number of edges of $LDel_1^k(G)$ on a general geometry graph G. To do so, we first review the following theorem proved in [10] (Theorem 11 from Chapter 4).

Theorem 2. *[10] A $K_{s,t}$-free graph G with n vertices has size at most*

$$\frac{1}{2}(s-1)^{1/t}n^{2-1/t} + \frac{1}{2}(t-1)n$$

Theorem 3. *Graph $LDel_1^k(G)$ has no more than $O(n^{\frac{5}{3}})$ edges.*

PROOF. We prove that $LDel_1^k(G)$ has no $K_{3,3}$ subgraph. For the sake of contradiction, assume that $LDel_1^k(G)$ has a $K_{3,3}$ subgraph composed of six vertices u_1, u_2, u_3, v_1, v_2, and v_3. Nodes u_i and v_j are connected for $i = 1, 2, 3$ and $j = 1, 2, 3$. Notice that the subgraph $K_{3,3}$ is not a planar graph. Without loss of generality, we assume that edges u_1v_2 and u_2v_1 intersect. Then u_1, u_2, v_1, and v_2 form a convex hull $u_1u_2v_2v_1$. Notice that we have assumed that there are no four vertices co-circular. From the pigeonhole principal, either $\angle u_1u_2v_2 + \angle u_1v_1v_2 > \pi$ or $\angle u_2v_2v_1 + \angle u_2u_1v_1 > \pi$. Assume that $\angle u_1u_2v_2 + \angle u_1v_1v_2 > \pi$. Then any circle passing through u_1 and v_2 either contains u_2 or v_1. It is a contradiction to the existence of edge u_2v_2 or u_1v_1 in $LDel_1^k(G)$. From Theorem 2, $LDel_1^k(G)$ has no more than $2^{-2/3}n^{5/3} + n = O(n^{\frac{5}{3}})$ edges. □

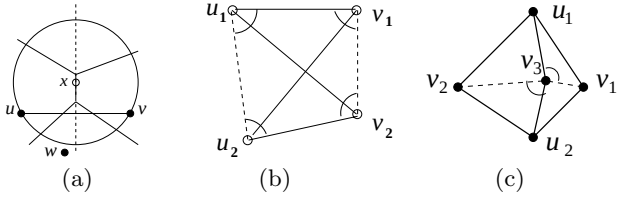


Fig. 1. (a): $LDel_1^k(G)$ and $LVor_G^k$ are dual. (b): No subgraph $k_{2,2}$ with crossing edges exists in $LDel_1^1(G)$. (c): $LGG_1(G)$ does not have $K_{2,3}$ subgraph.

The above theorem is true only if the points are in a general position, i.e., no four points are co-circular. The proof of the above theorem implies that $LDel_1^k(G)$ does not contain the structure of a crossing C_4 as a subgraph. Generally, we would like to know what is the tight upper bound on the number of edges any geometry graph that is free of a crossing C_4. The above theorem implies that there are at most $O(n^{\frac{5}{3}})$ edges. Out conjecture is that there is only $O(n)$ edges. Notice that the two-edge oriented k-localized structure could have $O(n^2)$ edges, e.g., when G is a bipartite graph.

3 Geometric RNG and GG

We next extend this idea to the relative neighborhood graph and the Gabriel graph to any geometry graph.

3.1 Definitions

The zero-edge oriented localized relative neighborhood graph on a geometry graph $G = (V, E)$, denoted by $LRNG_0(G)$, consists of all edges $uv \in E$ such that there is no point w inside $lune(u, v)$. The one-edge oriented k-localized relative neighborhood graph on graph G, denoted by $LRNG_1^k(G)$, consists of all edges $uv \in E$ such that there is no point $w \in N_G^k(u) \cup N_G^k(v)$ inside $lune(u, v)$. The two-edge oriented k-localized relative neighborhood graph on graph G, denoted by $LRNG_2^k(G)$, consists of all edges $uv \in E$ such that there is no point $w \in N_G^k(u) \cap N_G^k(v)$ inside $lune(u, v)$.

Obviously, $LRNG_i^{k+1}(G) \subseteq LRNG_i^k(G)$ for $i = 1, 2$, and $RNG \cap G = LRNG_0(G) \subseteq LRNG_1^k(G) \subseteq LRNG_2^k(G)$. Similarly, we can define localized Gabriel graphs $LGG_0(G)$, $LGG_1^k(G)$, and $LGG_2^k(G)$ using $disk(u, v)$ instead of $lune(u, v)$. Then, $GG \cap G = LGG_0(G) \subseteq LGG_1^k(G) \subseteq LGG_2^k(G)$, $LGG_i^{k+1}(G) \subseteq LGG_i^k(G)$, and $LRNG_i^k(G) \subseteq LGG_i^k(G) \subseteq LDel_i^k(G)$ for $i = 0, 1, 2$.

3.2 Edge Complexity

Theorem 3 implies that graphs $LGG_1^k(G)$ and $LRNG_1^k(G)$ also have no more than $O(n^{\frac{5}{3}})$ edges due to $LRNG_1^k(G) \subseteq LGG_1^k(G) \subseteq LDel_1^k(G)$. We have

Theorem 4. *Graph $LGG_1^k(G)$ has at most $O(n^{\frac{3}{2}})$ edges.*

Proof. We first prove that $LGG_1^k(G)$ has no $K_{2,3}$ subgraph. Assume that $LGG_1^k(G)$ has a $K_{2,3}$ subgraph composed of five vertices u_1, u_2, v_1, v_2, and v_3. Nodes u_i and v_j are connected for $i = 1, 2$ and $j = 1, 2, 3$. Then similar to Theorem 3, we know that there are no intersections among these edges. It implies that four vertices u_1, u_2, v_1, and v_2 form a convex hull $u_1v_1u_2v_2$. There are two cases: node v_3 is inside the convex hull; it is outside of the convex hull. When node v_3 is outside of the convex hull, we can rename the vertices. Thus, generally, we can assume that node v_3 is inside the convex hull $u_1v_1u_2v_2$. See Figure 1. Then one of the angles among $\angle u_1v_3v_2$, $\angle u_2v_3v_2$, $\angle u_2v_3v_1$, and $\angle u_1v_3v_1$ is at least $\pi/2$. It implies that one of the disks using u_1v_1, v_1u_2, u_2v_2, or v_2u_1 as diameter contains node v_3. It is a contradiction to their existence in $LGG_1^k(G)$.

It was shown that a graph without a $K_{r,s}$ subgraph has edges at most $n^{2-\frac{1}{r}}$ where $r \leq s$. Thus, $LGG_1^k(G)$ has at most $\frac{\sqrt{2}}{2}n^{3/2} + n/2 = O(n^{\frac{3}{2}})$ edges. □

The proof of the upper bounds of the number of edges in local Delaunay triangulation and other relatives is based on the general graph structure. We expect a tight bound by using more geometric properties of the structures.

3.3 Planarity

It was proved that $RNG(V)$, $GG(V)$, and $Del(V)$ are planar graphs. Li *et al.* [9] recently showed that $LDel_1(G)$[1] on UDG is not a planar graph, but $LDel_1^k(G)$ on UDG is always a planar graph for any integer $k > 1$. The following lemma presents a sufficient condition such that all localized structures $LDel_1^k(G)$ are planar graphs for any integer $k > 1$.

Lemma 1. *Assume that the geometry graph G is such that, given any two intersected edges uv and xy, at least one of the four edges of the convex hull of u, v, x, and y is in G. Then all localized structures $LRNG_1^k(G)$, $LGG_1^k(G)$, and $LDel_1^k(G)$ are planar graphs for any integer $k > 1$.*

Proof. We only have to prove that $LDel_1^k(G)$ is a planar graph if G satisfies the condition and $k > 1$. Consider any two intersected edges uv and xy. Without loss of generality, assume that four vertices u, x, v, and y are placed clockwise and the edge $ux \in G$. See Figure 2 (a) for an illustration.

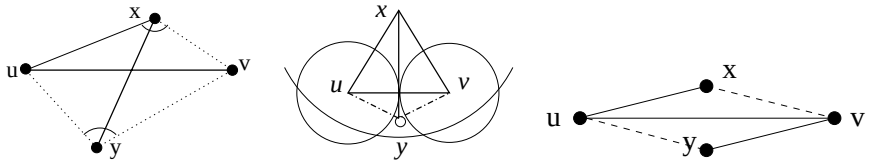


Fig. 2. (a): Either xy or uv does not belong to $LDel_1^k(G)$, for $k \geq 2$. (b): $LGG_1^1(DG)$ and $LDel_1^1(DG)$ are not planar graphs. (c): Here $\angle uxv + \angle uyv > \pi$.

From the pigeonhole principle, either $\angle uxv + \angle vyu \geq \pi$ or $\angle yux + \angle yvx \geq \pi$. Assume that $\angle uxv + \angle vyu \geq \pi$. Then any circle passing through edge uv must contain x or y or both. Notice that both x and y are from $N_G^2(u)$. It implies that edge uv cannot be in $LDel_1^k(G)$ for any $k > 1$. □

The condition specified in Lemma 1 is satisfied by most practical geometry graphs such as the unit disk graph, the disk graph. Here a graph $G = (V, E)$ is disk graph, denoted by DG, if there is a two-dimensional disk $d(u)$ (with radius r_u) for each vertex u such that an edge $uv \in E$ iff $d(u)$ and $d(v)$ intersect.

Theorem 5. *$LRNG_1^k(DG)$, $LGG_1^k(DG)$, and $LDel_1^k(DG)$ are planar, $\forall k > 1$.*

Proof. Given a disk graph DG, assume that we have two intersected edges uv and xy. See Figure 2 (a) for an illustration. We will show that one of the edges on the convex hull exists in the disk graph.

For the sake of contradiction, assume that all four edges are not in the disk graph. Then $\|ux\| > r_u + r_x$, $\|xv\| > r_v + r_x$, $\|vy\| > r_v + r_y$, and $\|uy\| > r_u + r_y$. From triangle inequality, $\|ux\| + \|vy\| < \|uv\| + \|xy\|$, $\|uy\| + \|vx\| < \|uv\| + \|xy\|$. Thus, $\|uv\| + \|xy\| > r_u + r_v + r_x + r_y$. The existences of edges uv and xy imply

that $\|uv\| \leq r_u + r_v$, and $\|xy\| \leq r_x + r_y$, which contradicts the previous bound. Thus, one of the four edges is in G if G is a disk graph, which, together with lemma 1 finishes the proof. □

Figure 2 (b) gives an example such that structures $LGG_1^1(DG)$, and $LDel_1^1(DG)$ are not planar graphs. Here node x has the largest disk and node y has the smallest and $\pi/3 < \angle xuy = \angle xvy < \pi/2$, and $\angle uxv < \pi/3$. Thus, edges xu, xv, xy and uv are preserved in $LGG_1^1(DG)$ and $LDel_1^1(DG)$.

Theorem 6. *$LRNG_1^1(DG)$ is planar.*

Proof. Assume that there are two intersected edges xy and uv in $LRNG_1^k(DG)$. Similar to the proofs in Theorem 5, we can actually show that there are two adjacent edges of the convex hull $uxvy$ existing in the disk graph. W.l.o.g, assume that xu and xv are in the disk graph. If $\angle uxv > \pi/3$, edge uv cannot belong to $LRNG_1^k(DG)$. Otherwise, one of the angles $\angle xuv$ and $\angle xvu$ is larger than $\pi/3$, which implies that edge xy cannot belong to $LRNG_1^k(DG)$. We have contradictions in both cases. Thus, no pair of edges intersect in $LRNG_1^k(DG)$. □

Notice that, the conditions specified in Lemma 1 are not satisfied by some other interesting geometry graphs, such as mutually-inclusion communication graph defined later for wireless ad hoc networks.

3.4 Minimum Spanning Tree

Unfortunately, the zero-edge oriented or one-edge oriented localized structures may be disconnected. The right figure of Figure 2 illustrates such an example, in which edge uv is removed in any zero-edge or one-edge oriented localized structures. Therefore, they do not always contain the minimum spanning tree of graph G.

Lemma 2. *Assume that, given any edge uv, the lune(u, v) is either empty of $N_G^1(u) \cup N_G^1(v)$ or it contains a vertex w such that wu and wv are edges of G, then $MST_G(V) \subseteq LRNG_1^1(G)$.*

Assume that, given any edge uv, either (1) disk(u, v) is empty of $N_G^1(u) \cup N_G^1(v)$ or (2) lune(u, v) contains a vertex w such that wu and wv are edges of G. Then $MST_G(V) \subseteq GG_1^1(G)$.

Assume that, given any edge uv, either (1) there is a disk passing through uv and empty of $N_G^1(u) \cup N_G^1(v)$ or (2) lune(u, v) contains a vertex w such that wu and wv are edges of G. Then $MST_G(V) \subseteq LDel_1^1(G)$.

The proof is simple and omitted. Similarly, it is easy to show that all two-edge oriented k-localized structures do contain the Euclidean minimum spanning tree as a subgraph. As we will show later that, these structures have sub-quadratic number of edges for some special communication graphs derived from wireless as hoc networks. This makes a fast distributed computation of the minimum spanning tree possible. Notice that, it is well-known [11] that the optimal time and communication complexity of computing MST_G in a distributed manner is proportional to $O(n)$ and $O(m + n \log n)$ respectively.

4 Structures on Graphs from Wireless Ad Hoc Networks

In wireless ad hoc networks, there are some special geometry graphs. Consider a set of wireless device distributed in a two-dimensional plane. Assume each point u has a fixed transmission range r_u. A mutual inclusion graph, denoted by MG hereafter, used for ack-based communication in wireless ad hoc networks, has an edge uv if and only if $\|uv\| \leq \min(r_u, r_v)$. In [9], Li *et al.* showed that the one-edge oriented k-Localized Delaunay graph $LDel^k_{1,UDG}$ has only a linear number of edges. Moreover, they showed that it can be constructed using only $O(n)$ total messages in wireless ad hoc communications model, i.e., assuming that a message sent by a node can be received by all nodes within its transmission range.

4.1 Complexity of $LRNG^k_1(MG)$, $LGG^k_1(MG)$, $LDel^k_1(MG)$

For simplicity, we first study their complexities when the transmission radius of all nodes is within a constant factor of each other. Since for general graph G, the one-edge oriented localized Gabriel graph has at most $O(n^{3/2})$ edges, thus all structures $LRNG^k_1(MG)$ and $LGG^k_1(MG)$ have also at most $O(n^{3/2})$ edges. Additionally, $LDel^k_1(MG)$ has at most $O(n^{5/3})$ edges.

Here we will show a stronger result. Let $r_{\min}$ be the smallest transmission range; $r_{\max}$ be the maximum transmission range of all nodes.

Theorem 7. *The structure $LGG^k_1(MG)$ has thickness 2 if $r_{\max} \leq \sqrt{2} r_{\min}$.*

Proof. First of all, it is easy to show that all edges with length at most $r_{\min}$ belongs to the Gabriel graph of the unit disk graph defined over all nodes with transmission range $r_{\min}$. Thus, the number of all such edges is at most $3n - 6$ since the Gabriel graph over any unit disk graph is planar. We then show that the number of edges with length larger than $r_{\min}$ also forms a planar graph.

Assume, for contradiction, there are two edges uv and xy that intersect. Here $r_{\min} < \|uv\| \leq r_{\max} \leq \sqrt{2} r_{\min}$, so does $\|xy\|$. See Figure 2 (a) for illustration. We then show that one of the four edges of xu, uy, yv and vx has length at most $r_{\min}$. Assume that all such four edges have length larger than $r_{\min}$. W.l.o.g, assume that $\angle uxv + \angle uyv \geq \pi$ and the angle $\angle uxv \geq \pi/2$. Then $\|uv\|^2 = \|ux\|^2 + \|xv\|^2 - 2\|ux\| \cdot \|xv\| \cdot \cos(\angle uxv) > 2r^2_{\min}$. Thus $\|uv\| > \sqrt{2} r_{\min}$, which is a contradiction. Thus, we know that one of the two edges ux and xv has length at most $r_{\min}$. Assume that $\|ux\| \leq r_{\min}$. Thus link ux belongs to the original communication graph. Consequently, in the original communication graph, node x is inside $disk(u, v)$ and has an edge xu to node u, which is a contradiction to the existence of edge uv in graph $LGG^k_1(G)$. □

Since $LGG^k_1(MG)$ contains $LRNG^k_1(MG)$ as a subgraph, graph $LRNG^k_1(MG)$ also has thickness 2 when $r_{\max} \leq \sqrt{2} r_{\min}$. Li *et al.* [9] proved that the localized Delaunay triangulation $LDel^k_1(G)$ is a planar graph if G is a unit disk graph and $k \geq 2$. Similarly, we have

Theorem 8. *If $k \geq 2$ and $r_{\max} \leq \sqrt{2} r_{\min}$, then $LDel^k_1(MG)$ has thickness 2.*

By a simple bucketing of the edges into the following buckets: $(0, r_{\min}]$, $(r_{\min}, \sqrt{2}r_{\min}]$, $\cdots$, $(\sqrt{2}^i r_{\min}, \sqrt{2}^{i+1} r_{\min}]$, $\cdots$, $(\sqrt{2}^{t-1} r_{\min}, \sqrt{2}^t r_{\min}]$, it is easy to prove the following theorem. Here $\sqrt{2}^t r_{\min} \geq r_{\max}$ and $\sqrt{2}^{t-1} r_{\min} < r_{\max}$.

Theorem 9. *Let $\beta = r_{\max}/r_{\min}$. Then $LRNG_1^k(MG)$ and $LGG_1^k(MG)$ have thickness $1 + 2\log_2 \beta$ and $LDel_1^k(MG)$ has thickness $1 + 2\log_2 \beta$, if $k \geq 2$.*

4.2 Complexity of $LRNG_2^k(MG)$, $LGG_2^k(MG)$, and $LDel_2^k(MG)$

We study the structure $LGG_2(MG)$ when the transmission radius of all nodes is within a constant factor of each other. Assume the minimum transmission range is r and the maximum transmission range is βr, where β is a constant.

First of all, all edges in $LGG_2^k(MG)$ with length at most r form a planar graph since they are in the Gabriel graph over a unit disk graph (each node with transmission range r). Thus, the number of edges with length at most r is at most $3n$. We then study the edges with length larger than r but less than βr. We prove that the number of edges with length $\in (r, \sqrt{2}r]$ is at most $O(n^{5/3})$.

Lemma 3. *The number of edges in $LGG_2^k(MG)$ with length between r and $\sqrt{2}r$ is at most $O(n^{5/3})$, where G is the mutually-inclusion communication graph defined over a set nodes whose transmission radius is at least r and at most $\sqrt{2}r$.*

Proof. We prove that the crossing circle C_4 is a forbidden subgraph. Assume that there is a crossing $C_4 = xvuy$ formed by crossing edges uv and xy. Obviously, all such nodes have transmission range at least r.

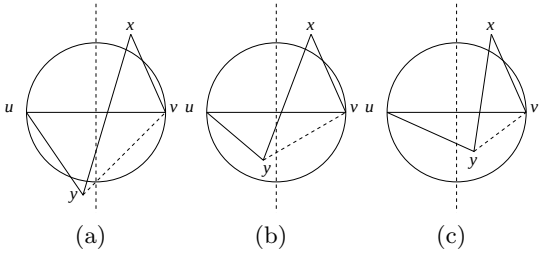


Fig. 3. Crossing C_4 is a forbidden subgraph.

We first prove that both x and y cannot be outside of $disk(u, v)$. Suppose that happens. W.l.o.g., assume that the midpoint of uv is on the same side of xy as u (Figure 4.2 (a)). Then $\angle xvy > \pi/2$.

For both cases, if $\|vy\| \leq r$, then the edge vy is in the original mutual communication graph since all nodes have transmission range at least r. Since $\angle xvy > \pi/2$ edge xy cannot be in the Gabriel graph. If $\|vy\| > r$, together with $\|xv\| \geq r$ and $\angle xvy > \pi/2$, we have $\|xy\| > \sqrt{2}r$, which is a contradiction to the fact that we only consider edges with length $\leq \sqrt{2}r$.

Then we know that at least one of x or y or both is inside $disk(u, v)$. Assume that y is inside. There are two cases here: (b) y is on the same side of bisector of segment uv as u; (c) y is on the same side of bisector of segment uv as v.

Case (b) is impossible since $\|uy\| < \frac{\sqrt{2}}{2}\|uv\| < \frac{\sqrt{2}}{2}\sqrt{2}r = r$, which is a contradiction to the fact that we only consider edges with length between r and $\sqrt{2}r$.

In case (c), similarly we have $\|vy\| < r$, which implies the existence of edge vy in the original mutual communication graph. This, together with existence of edge uy, is a contradiction to the existence of edge uv in the Gabriel graph.

Notice in Theorem 3, we showed that if a graph is $k_{3,3}$ free then it is free of crossing C_4. This finishes the proof. □

By bucketing edges into $1 + 2\log_2 \beta$ buckets, we have

Theorem 10. *The number of edges in $LGG_2^k(G)$ is at most $O(n^{5/3}\log_2 \beta)$, where $\beta = r_{\max}/r_{\min}$.*

Conjecture 1. At most $O(n)$ edges in $LGG_2^k(MG)$ have length $\in (r, \sqrt{2}r]$.

5 Conclusion

In this paper we proposed several new proximity structures on general geometric graphs and studied their complexities for both general geometric graph and some special geometric graphs. We summarize the results about the edge complexities of the structures we have discussed as follows. Here $C_\beta = 1 + 2\log_2 \beta$ and $\beta = r_{\max}/r_{\min}$. The complexities with a star mark is true only when $k \geq 2$.

Table 1. Upper bounds of the edge numbers.

	G	DG	MG
$LDel_1^k(G)$	$O(n^{5/3})$	$\Theta(n)$ *	$O(C_\beta \cdot n)$ *
$LDel_2^k(G)$	$\Theta(n^2)$		$O(C_\beta \cdot n^{5/3})$
$LGG_1^k(G)$	$O(n^{3/2})$	$\Theta(n)$ *	$O(C_\beta \cdot n)$
$LGG_2^k(G)$	$\Theta(n^2)$		$O(C_\beta \cdot n^{5/3})$
$LRNG_1^k(G)$	$O(n^{3/2})$	$\Theta(n)$	$O(C_\beta \cdot n)$
$LRNG_2^k(G)$	$\Theta(n^2)$		$O(C_\beta \cdot n^{5/3})$

Notice that one way to study the complexity of these geometry structures is from the point view of forbidden subgraphs. Although the complexity of general graph with forbidden structure is well-studied, little is known about the complexity of the geometry graph with some forbidden structure. We indirectly showed that any geometry graph on n points with forbidden crossing C_4 has at most $O(n^{5/3})$ edges. To the best of our knowledge, this is the currently best known upper bound. However, it is unlikely that we can achieve this upper bound. We summarize some open questions we have discussed in this paper as follows.

1. What are the tight bounds on the sizes of $LDel_1^k(G)$, $LDel_2^k(G)$, $LGG_1^k(G)$, $LGG_2^k(G)$, etc.? We can also consider the case when G is some special graph such as a disk graph DG, a mutually-inclusion graph MG etc.
2. What is the size of a geometric graph, free of crossing C_4. We know that it is at most $O(n^{5/3})$ for graph of n vertices.
3. How to construct the proximity structures defined in the paper efficiently. For the UDG, Li *et al.* [9] previously gave an asymptotically optimal method to construct $LDel_1^k(UDG)$.
4. Is the graph $LDel_2^k(G)$ a spanner? This question would be interest also for some special graphs like the disk graph or the mutually-inclusion graph. Notice that it was known [5] that GG and RNG are not length spanners. Thus localized Gabriel graph and relative neighborhood graphs are not spanners.

References

1. Edelsbrunner, H.: Algorithms in Combinatorial Geometry. Springer-Verlag (1987)
2. Fortune, S.: Voronoi diagrams and delaunay triangulations. In: F. K. Hwang and D.-Z. Du, editors, Computing in Euclidean Geometry, World Scientific, (1992) 193–233
3. Preparata, F.P., Shamos, M.I.: Computational Geometry: an Introduction. Springer-Verlag (1985)
4. Gabriel, K., Sokal, R.: A new statistical approach to geographic variation analysis. Systematic Zoology **18** (1969) 259–278
5. Bose, P., Devroye, L., Evans, W., Kirkpatrick, D.: On the spanning ratio of gabriel graphs and beta-skeletons. In: Proceedings of the Latin American Theoretical Infocomatics (LATIN). (2002)
6. Jaromczyk, J.W., Kowaluk, M.: Constructing the relative neighborhood graph in three-dimensional euclidean space. Discrete Applied Mathematics (1991) 181–192
7. Jaromczyk, J., Toussaint, G.: Relative neighborhood graphs and their relatives. Proceedings of IEEE **80** (1992) 1502–1517
8. Supowit, K.J.: The relative neighborhood graph, with an application to minimum spanning trees. Journal of Associate Computing Machine (1983)
9. Li, X.Y., Calinescu, G., Wan, P.J.: Distributed construction of planar spanner and routing for ad hoc wireless networks. In: 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM). Volume 3. (2002)
10. Bollobás, B.: Extremal Graph Theory. Academic Press (1978)
11. Faloutsos, M., Molle, M.: Creating optimal distributed algorithms for minimum spanning trees. Technical Report CSRI-327 (also in WDAG '95) (1995)
12. Alzoubi, K., Wan, P.J., Frieder, O.: Message-optimal connected-dominating-set construction for routing in mobile ad hoc networks. In: 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'02). (2002)

The Zigzag Path of a Pseudo-Triangulation

Oswin Aichholzer[1], Günter Rote[2⋆], Bettina Speckmann[3], and Ileana Streinu[4⋆⋆]

[1] Institute for Software Technology, Graz University of Technology, oaich@ist.tugraz.at
[2] Institute of Computer Science, FU Berlin, rote@inf.fu-berlin.de
[3] Institute for Theoretical Computer Science, ETH Zürich, speckman@inf.ethz.ch
[4] Department of Computer Science, Smith College, streinu@cs.smith.edu

Abstract. We define the zigzag path of a pseudo-triangulation, a concept generalizing the path of a triangulation of a point set. The pseudo-triangulation zigzag path allows us to use divide-and-conquer type of approaches for suitable (i.e., decomposable) problems on pseudo-triangulations. For this we provide an algorithm that enumerates all pseudo-triangulation zigzag paths (of all pseudo-triangulations of a given point set with respect to a given line) in $O(n^2)$ time per path and $O(n^2)$ space, where n is the number of points. We illustrate applications of our scheme which include a novel algorithm to count the number of pseudo-triangulations of a point set.

1 Introduction

Pseudo-triangulations, unlike triangulations, only recently emerged as a promising data structure with a variety of applications. They were originally introduced in the context of visibility complexes [15] and ray shooting [8,12], but in the last few years they also found application in robot arm motion planning [18], kinetic collision detection [1,13], and guarding [17]. In particular the so-called minimum or pointed pseudo-triangulations introduced by Streinu [18] exhibit many fascinating properties that initiated a growing interest in their geometric and combinatorial nature.

There exist already several algorithms to enumerate pseudo-triangulations of sets of n points. Bespamyatnikh [5], extending his work on enumerating triangulations [6], defines a lexicographical order on pseudo-triangulations which he uses to enumerate pseudo-triangulations in $O(\log n)$ time per pseudo-triangulation. Brönnimann et al. [7] implemented an ad-hoc technique of Pocchiola based on a greedy strategy for generating edges of pseudo-triangulations. Unfortunately the time complexity of this algorithm is not known, but it requires $O(n^2)$ space. A third possibility is to apply some vertex enumeration algorithm to the polytope of pseudo-triangulations developed in [14,16]. For example, Motzkin's double description method or the reverse-search technique of Avis and Fukuda [4] are two methods for vertex enumeration which have been implemented [3,11].

⋆ Research partly supported by the Deutsche Forschungsgemeinschaft (DFG) under grant RO 2338/2-1.
⋆⋆ Research supported by NSF grant CCR-0105507.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 377–388, 2003.
© Springer-Verlag Berlin Heidelberg 2003

We propose a different scheme for solving counting and optimization problems for pseudo-triangulations, inspired by an analogous approach developed for triangulations. The "path of a triangulation" was introduced by Aichholzer [2] in order to count the triangulations of a planar point set in a divide-and-conquer like manner. This concept can be used to attack any decomposable problem on triangulations. Dumitrescu et al. [9] provided an algorithm that enumerates all triangulation paths (of all triangulations of a given point set with respect to a given line) in $O(n^3 \log n)$ time per path and $O(n)$ space.

In this paper we describe a meaningful extension of the path concept to pseudo-triangulations. We first recall some definitions concerning pseudo-triangulations and also formalize the notion of a decomposable problem. In Section 4 and 5 we then develop the definition of the zigzag path of a pseudo-triangulation, which retains all of the useful properties of a triangulation path. Finally in Section 6 we show how to generate all pseudo-triangulation zigzag paths in $O(n^2)$ time per path (at the expense of $O(n^2)$ space and preprocessing time).

The path concept can be generalized to arbitrary (i.e., not necessarily pointed) pseudo-triangulations. However, in this extended abstract we concentrate on the results pertaining to pointed pseudo-triangulations. The extension to general pseudo-triangulations can be found in the journal version of this paper.

2 Pseudo-Triangulations

We consider a simple planar polygon P and a point set $S \subseteq P$, $|S| = n$, which contains all vertices of P but may also contain additional *inner points*. We will assume throughout that S is in general position, i.e., it contains no three collinear points. We will refer to the pair (S, P) as a *point set S in a polygon P*, or shorter as *pointgon*. We denote the boundary of P by ∂P.

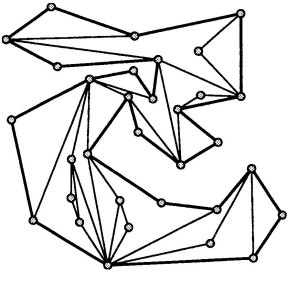


Fig. 1. A pointed pseudo-triangulation of a pointgon.

A *pseudo-triangle* is a planar polygon that has exactly three convex vertices, called *corners*, with internal angles less than π. A *pseudo-triangulation* T of a pointgon (S, P) is a partition of the interior of P into pseudo-triangles whose vertex set is exactly S (see Fig. 1) . A vertex p in a pseudo-triangulation T of (S, P) is *pointed* if there is one region incident to p (either a pseudo-triangle or the outer face) whose angle at p is greater than π. A pseudo-triangulation T of (S, P) is called *pointed* if each point $p \in S$ is pointed. A pseudo-triangulation for a point set S corresponds to the case where P is the convex hull of S.

Proposition 1 (Streinu [18]) *Every non-crossing pointed set of edges in a pointgon (S, P) can be extended to a pointed pseudo-triangulation of (S, P).*

3 Decomposable Problems and Divide-and-Conquer

We are interested in certain types of optimization or counting problems for the set of pseudo-triangulations for a point set S. We associate with each pseudo-triangulation a *zigzag path*, which decomposes the convex hull of S into several parts on which the problem can be solved recursively. Our approach can be summarized as follows:

1. Enumerate all zigzag paths α.
2. For each α:
3. Use α to split the problem into several pieces.
4. Solve each subproblem recursively.
5. Combine the solutions of the subproblems.
6. Combine the solutions for all zigzag paths into the solution for the original problem.

The main contribution of this paper is a proper definition of a zigzag path and an algorithm for enumerating zigzag paths, in order to carry out step 1 of this procedure.

The problem that we want to solve must have a certain decomposable structure in order to be amenable to this approach. This structure can be described by a commutative *semiring* $(H, \oplus, \otimes)$ with two associative and commutative operations $\oplus$ and $\otimes$ which satisfy the distributive law:

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$$

We assume that an "objective function" $f(T)$ for a pseudo-triangulation T can be computed as the $\otimes$-product of $f(t)$ for the individual pseudo-triangles $t \in T$, where $f(t) \in H$ is some function that is determined individually for each pseudo-triangle. We use the $\oplus$ operation to accumulate the values of all pseudo-triangulations into the quantity in which we are finally interested in. The task is to calculate

$$\tilde{f}(\mathcal{T}) := \bigoplus_{T \in \mathcal{T}} f(T) = \bigoplus_{T \in \mathcal{T}} \bigotimes_{t \in T} f(t) \tag{1}$$

over some set $\mathcal{T}$ of pseudo-triangulations T.

Now if we can generate all zigzag paths, then we can easily count the number of pseudo-triangulations as follow: $(H, \oplus, \otimes) = (\mathbb{N}, +, \cdot)$, with $f(t) \equiv 1$ for every pseudo-triangle t. We can also optimize various quantities over the set of pseudo-triangulations, for example the smallest angle, or the sum of the edge lengths. In the first case, we take $(H, \oplus, \otimes) = (\mathbb{R}, \max, \min)$, and $f(t) =$ the smallest angle in t. In the second case, we take $(H, \oplus, \otimes) = (\mathbb{R}, \min, +)$, and $f(t) =$ the perimeter of t. Here we count the length of the interior edges twice, but since the total length of the boundary edges is constant, this is equivalent to optimizing the total length.

As mentioned before, one can of course solve these problems, and more general optimization problems, by enumerating all pseudo-triangulations by one of the methods mentioned in the introduction, evaluating $f(T)$ for each pseudo-triangulation T, and taking the $\oplus$-sum. However, our divide-and-conquer procedure is usually several orders of magnitude faster than this trivial approach.

4 The Zigzag Path

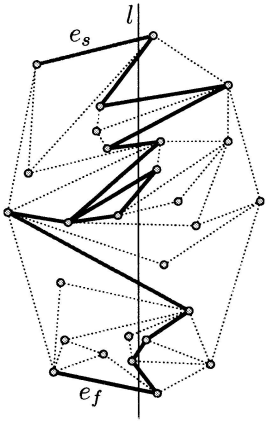


Fig. 2. The pseudo-triangulation zigzag path.

Assume that we are given a pseudo-triangulation T of a pointgon (S, P). We have to choose a *cut segment* l that connects two boundary points of P through the interior of P but avoids all points in S. For simplicity we will assume throughout the paper that l is vertical. The endpoints of l lie on two edges of P, the *start edge* e_s on the top and the *final edge* e_f on the bottom. Let $E = \{e_1 = e_s, e_2, \cdots, e_k = e_f\}$ be the set of edges of T that are crossed by l, ordered from top to bottom according to their intersection with l. Consider a pair of consecutive edges e_i and e_{i+1} in E. We say that the pair (e_i, e_{i+1}) *leans* to the left or to the right, respectively, according to the location of the intersection of the lines through e_i and e_{i+1} with respect to l. Since two edges of a common pseudo-triangle are never parallel, this direction is always well-defined. If (e_{i-1}, e_i) and (e_i, e_{i+1}) lean in different directions, the edge e_i is called a *signpost* (see Fig. 3.a–b). The starting and ending edges e_s and e_f are also considered to be signposts.

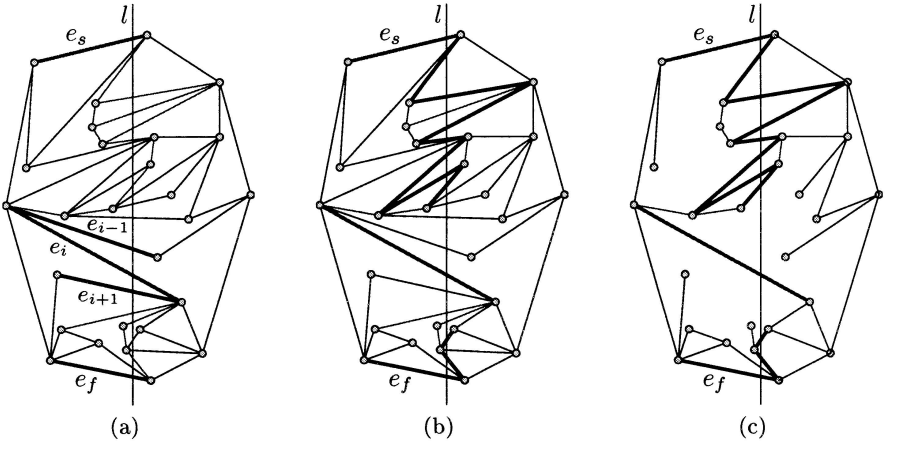


Fig. 3. Constructing the zigzag path of a pseudo-triangulation. (a) A pseudo-triangulation cut by a segment l — the pair (e_i, e_{i+1}) leans to the right. (b) The signposts. (c) Removing edges that are cut by l but are not signposts.

We define the *zigzag path* $\alpha_l(T)$ of a pseudo-triangulation T with respect to a cut segment l as follows: We remove all edges of E that are not signposts. Let

P^* denote the resulting set of polygons, see Figure 3.c. We now construct $\alpha_l(T)$ by joining adjacent signposts along the boundary of their common face in P^* according to their lean, i.e., if two adjacent signposts lean to the left, then we connect them via the edges of their common polygon that lie to the left of l, see Fig. 2. Note that a vertex can appear on the path several times.

Before stating a couple of characteristic properties of the zigzag path, we introduce some terminology. Consider a pseudo-triangle $t \in T$ which is cut by l in precisely two edges e and f. Let l^+ denote the side of l to which e and f lean. Then the part of t that lies in l^+ is a pseudo-triangle. t has one corner v in l^+, which is called a *turning point*. v is connected to e and f via two x-monotone chains, whose vertices (excluding v) are called the *monotone vertices*. In other words, a monotone vertex of a path has one edge incident from the right and one edge incident from the left.

Lemma 1. *The zigzag path of a pseudo-triangulation T has the following properties:*

1. *It starts at e_s, ends at e_f, and contains no edge twice. Its intersections with l are ordered along l.*
2. (Empty Pseudo-Triangle Property) *The area bounded by the path between two consecutive signposts and the line l is an empty pseudo-triangle, i.e., it contains no points of S in its interior.*
3. *All vertices of the path which are monotone vertices of an empty pseudo-triangle in Property 2 are pointed in T.*

Proof. Property 1 is true by construction. Properties 2 and 3 can be proved inductively by successive elimination of edges e which are not signposts. Each removal will merge two adjacent pseudo-triangles into one. Let e' and e'' be e's neighboring intersecting edges with l. Suppose that (e', e) and (e, e'') lean in the same direction, say, to the left. Let t_1 and t_2 be the pseudo-triangles on the left side of l to which (e', e) and (e, e'') belong, respectively. The left endpoint of e must be a corner (turning point) of t_1 or t_2 (or both), because it cannot be incident to two angles larger than π.

Thus, if we remove e, t_1 and t_2 will merge into a single pseudo-triangle, which is empty. All its monotone vertices were already monotone vertices on the side chains of t_1 or t_2; hence, by induction, they are pointed in T. □

Lemma 2. *The zigzag path of a pseudo-triangulation T is the unique chain of edges α in T which satisfies Properties 1–3 of Lemma 1.*

Here, a chain of edges is taken in the graph-theoretic sense, as a walk (or path) in the graph.

Proof. The proof is based on the following easy observation, see Figure 4.

Proposition 2 *Let t be a pseudo-triangle on one side of l, with a segment of l forming a side of t. The other two sides of t are formed by edges of T. Suppose that t contains no points of S in its interior and all monotone vertices of t are*

pointed in T. Let e' and e'' denote the two edges of T on the boundary of t which intersect l. Then any edge e of T which intersects the interior of t intersects l. Moreover, any two of these edges (including e' and e'') lean in the same direction as e' and e''.

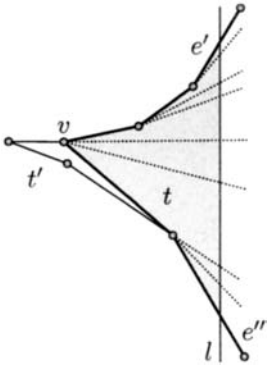


Fig. 4. The pseudo-triangle t in Prop. 2. The dotted lines are some possible locations for the edges e. t' is an alternative pseudo-triangle in the proof of Lemma 2.

Now, to prove the Lemma 2, let us consider two successive intersections e' and e'' of the chain α with l and the pseudo-triangle t formed between them. By Proposition 2, all edges of T intersecting l between e' and e'' lean in the same direction. Hence there can not be a signpost of T between e' and e'' which implies that every signpost is part of the path α.

Let us consider three successive crossings e', e'', e''' between α and l. Between two crossings, α forms a pseudo-triangle with l; hence the two crossing edges lean to the side on which this piece of α lies. Since α crosses from one side of l to the other side at each crossing, the pairs (e', e'') and (e'', e''') lean in different directions. Let $e'' = e_i$ in the ordered set of intersections of T with l. Proposition 2 implies that (e_{i-1}, e_i) leans on the same side as (e', e'') and (e_i, e_{i+1}) leans on the same side as (e'', e'''). Hence e_i is a signpost of T.

At this point we have established that the intersections of α with l are exactly the signposts of T. We still have to check that α bounds a unique pseudo-triangle between two signposts. Let t be the pseudo-triangle between two signposts e' and e'' in the zigzag path $\alpha_l(T)$, and let v be its turning point. Suppose, for the sake of deriving a contradiction, that α bounds a different pseudo-triangle t' between e' and e''. Since t bounds the face in T^* obtained by removing all crossing edges between e' and e'' from T and since α does not contain these edges, we must have $t \subset t'$. Because t' has no interior vertices, it must have all vertices of t on its boundary. If v is the turning point of t', then $t' = t$. So let us assume w.l.o.g. that v lies on the upper chain of t', see Figure 4. Then the lower side chain of t starts with an edge going from v into the interior of t' and ends at e''. This initial edge contradicts Proposition 2 applied to t'. □

The properties of Lemma 1 allow us to define a pseudo-triangulation path of a pointgon without reference to a particular pseudo-triangulation.

Definition 1 (Zigzag Path of a pointgon) *Let (S, P) be a pointgon and let l be a cut segment. A* pseudo-triangulation zigzag path *of (S, P) with respect to l is a non-crossing path in P using vertices of S with the following properties:*

1. *It starts at e_s and ends at e_f. Its intersections with l are ordered along l.*
2. (Empty Pseudo-Triangle Property) *The area bounded by the path between two consecutive intersections with l and the line l is an empty pseudo-triangle.*
3. *The path respects the pointedness property at S, i.e., every vertex of S is pointed in $\alpha \cup P$.*

We denote by $\Pi_l(S, P)$ the set of all paths for a pointgon (S, P) with respect to a line l, i.e.,

$$\Pi_l(S, P) = \{\, \alpha_l(T(S, P)) \mid T \text{ is a pointed pseudo-triangulation of } (S, P) \,\}.$$

Lemma 3. *Let α be a path for (S, P) with respect to the cut segment l.*

1. *$P \cup \alpha$ can be extended to a pointed pseudo-triangulation of (S, P) .*
2. *Let T be any pointed pseudo-triangulation of (S, P) which contains α. Then α is the zigzag path of T with respect to l. The intersections of α with l are the signposts of T.*

5 Making Progress – Trivial Paths

A zigzag path α for a pointgon (S, P) that runs completely along the boundary of P does not cut P into pieces and we will not make any progress by using α. But we will see that the only case where we cannot continue is in fact a single pseudo-triangle without interior points. Then clearly, there is only the "trivial" pseudo-triangulation and we can solve the problem directly.

For a set S of points in the plane a direction d is *feasible* if no line spanned by two points of S is parallel to d. A feasible line is a line with a feasible direction.

Theorem 1. *If δP of a pointgon (S, P) contains at least 4 convex vertices or if (S, P) has at least one inner point, then for each given feasible direction there exists a line l such that all path in $\alpha_l(P)$ are non-trivial.*

Proof. (Sketch) Any trivial path α is a part of δP, i.e., there are no signposts between start and final edge. By Definition 1 two signpost are always connected via exactly one turning point which implies that if the part of δP in consideration contains two convex corners no trivial path can be part of it.

W.l.o.g. let the given orientation of l be vertical. We will use l as a sweep-line for P, moving from left to right. We consider any convex corner of δP, any inner point of (S, P), as well as the left- and rightmost point of any side-chain of δP as an event. There are five different types of events: (1) A corner c of δP, such that after the sweep line passes through c the two incident side chains of δP form a wedge opening to the right. (2) Two of these wedges coalesce at a vertex. (3) A wedge is 'split' by a vertex of δP into two wedges. (4) One of the side chains of a wedge swept by l ends in a convex corner of δP. (5) An inner point of (S, P). A careful case analysis (full details can be found in the journal version) shows that during the sweep there always occurs a position for l such that any path with respect to l and P is non-trivial. □

6 Generating Pseudo-Triangulation Zigzag Paths

We will construct the zigzag paths incrementally, edge by edge, starting from the start edge e_s. In each stage, there may be several possibilities to continue the path. All these possibilities are explored in a backtracking tree. The important point of our construction is that one can never get stuck. There is always at least one way to continue. This means that the total work of the algorithm can be bounded in terms of the number of paths generated. This is in sharp contrast to the zigzag path of a triangulation, which cannot be generated in this way without backtracking [2].

Definition 2 (Partial path of a pointgon) *A* partial path α *of a pointgon* (S, P) *with respect to a line* l *is a noncrossing chain starting with* e_s *with the following properties.*

1. *The intersections of* α *with* l *are ordered from top to bottom on* l
2. *The path respects the pointedness property at every vertex of* S*, i.e., every vertex of* $P \cup \alpha_l(S, P)$ *is pointed.*
3. *The area bounded by the path between two consecutive intersections with* l *and the line* l *is an empty pseudo-triangle.*
4. *If we extend the last segment of* α *until it hits* l*, the area bounded by this extension, the line* l*, the path from the last intersection with* l *to the end of* α *is an empty pseudo-triangle. (If the last edge of* α *moves away from* l*, then this last segment is not included in this pseudo-triangle. In particular, if the last edge intersects* l*, the pseudo-triangle degenerates into a line segment and the condition is trivially fulfilled.)*

For a partial path α^* we define the *lower opposite wedge* as follows: we extend the last edge of α^* across l to the opposite side of the current endpoint of α^* until it hits δP. The area in P below this line and on the opposite side of l is the *lower opposite wedge* (the shaded region in Figure 5a).

Lemma 4. *A partial zigzag path* α *can be extended to a complete zigzag path if and only if the lower opposite wedge contains a point of* S*.*

Proof. Suppose that such a point exists. We will construct an extension for α, without destroying the pointedness of any vertex. W.l.o.g., assume that α ends on the right side of l in the point a. α may partition P into several regions. We look at the region R which contains a and the left endpoint b of e_f, see Figure 5.b. The desired extension of α must pass through R. If the angle at a in R is bigger than π, then we walk along the boundary of R away from l to the next point a' where the angle in R is less than π, see Figure 5.a. (This is done to maintain pointedness at a.) If the angle at a in R is smaller than π, we set $a' = a$. Similarly we construct a point b' by starting at b and walking away from l to the first small angle.

Now we take the following path β from a' to b': Start at a', follow the boundary of R to a, follow the extension of the last edge towards l, follow l to its intersection with the lower edge e_f, follow e_f to its left endpoint b, and continue

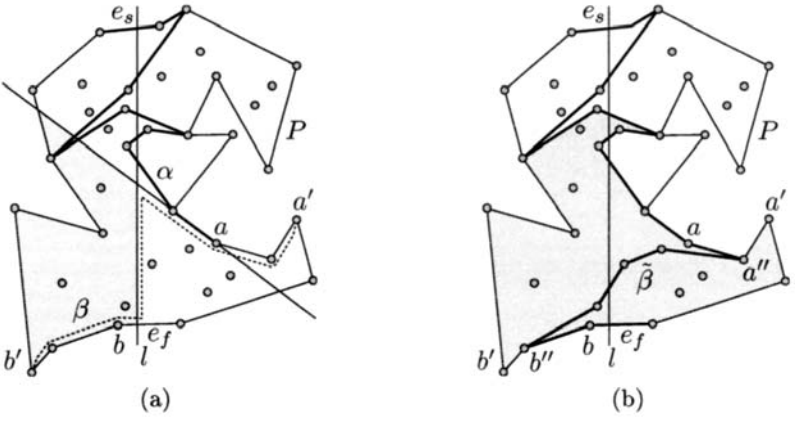


Fig. 5. (a) The lower opposite wedge of a partial zigzag path α and the path β in the proof of Lemma 4. (b) The region R (shaded) and the extension of α.

to b'. The path β runs in P and does not intersect α. Now we take the shortest path $\tilde{\beta}$ homotopic to β. In other words, we consider β as a string and pull it taut, regarding the points of S as obstacles, see Figure 5.b. The path $\tilde{\beta}$ may share some initial part of the boundary of R between a' and a with β, and it will split off at some vertex a''. Similarly we can define such a point b'' towards the end of $\tilde{\beta}$. The path from a to a'', from there to b'' via $\tilde{\beta}$, and from there to b and e_f extends α to a zigzag path. Since the additional edges come from a geodesic path between two convex vertices, pointedness is maintained.

On the other hand, suppose that the lower opposite wedge is empty. Then the extension of the last edge hits the lower edge e_f in an interior point, and the lower opposite wedge is a triangle. Clearly, the path cannot be extended by an edge which leads to a point on the other side of l without violating Property 3 of Definition 2. If α is extended without crossing l, this makes the lower opposite wedge smaller, and hence there is no way to complete the zigzag path. □

Note that the construction in the above proof is only carried out for the purposes of the proof; it is not performed by our algorithm.

Now, if we have a partial path satisfying the condition of Lemma 4, we have to find all edges that may be used to extend the path. We will show that this can be done in $O(n)$ time, after some preprocessing of the point set which takes $O(n^2)$ time and storage. In the preprocessing phase we compute and store the circular order of the edges from each point to all other points of S in $O(n^2)$ time [10]. At this stage, we can already eliminate edges which do not lie inside$P$.

The next edge which is added to a partial path must fulfill Properties 2 (pointedness) and 3 (empty area) of Definition 2, the non-empty opposite wedge condition of Lemma 4, and it must not cross the previous edges of the path.

Let a be the endpoint of α^* and assume w.l.o.g. that it lies on the right side of l. Take a line through the last edge of α^* and rotate it counterclockwise around

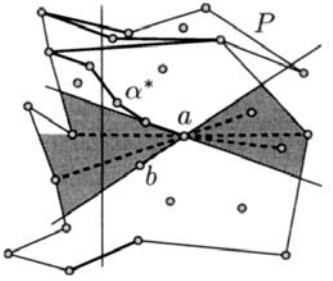


Fig. 6. The possible continuations of a partial path.

a until it hits the first point b on the right side of l. All points that are hit by this line and that are visible from a (including b) are candidates for the next point that satisfy the empty area condition, see Figure 6. If the last edge has moved away from l, then this holds for points on both sides of l. Otherwise, the next point must either be b or on the opposite side of l.

This set of continuation points depends only on a and on the last edge of α^*, and hence it can be determined in the preprocessing phase. Similarly the condition of Lemma 4 can be checked beforehand and edges which violate the condition are eliminated. The only conditions which have to be checked dynamically are the pointedness and non-crossing conditions.

Pointedness is easy to maintain: For each vertex a of S we store the unique angle between two incident edges which is larger than α. If a new edge incident to a is inserted, we see whether it falls into the wedge of the big angle, and if so, we either updated the big angle or we reject the edge because it destroys pointedness, in constant time. During the generation of all paths in the enumeration tree, edges are removed in the reverse order as they were inserted, so it is easy to maintain the big angle in stack-like manner.

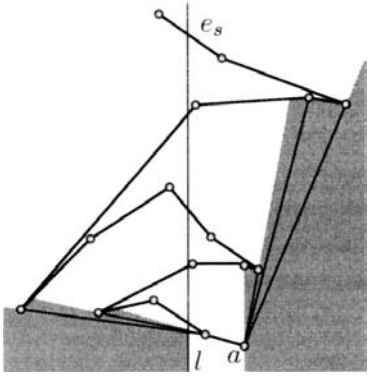


Fig. 7. The visibility polygon.

Now we still have to check that the new edge does not cross the partial path α^*. We show that we can do this, for all possible continuation edges from the endpoint a, in linear time.

We can easily check whether any edge intersects α^* if we know the *visibility polygon* from a with respect to α^*, see Figure 7. The visibility polygon is stored as a sequence of consecutive angular intervals together with an identification which edge of α^* is first first hit by a ray from a in that interval. We will show below in Lemma 7 how to exploit the special structure of the path to compute the desired visibility polygon in $O(n)$ time in an easy way.

Lemma 5. *For a given partial path all possible edges which extend it to a legal partial path satisfying the condition of Lemma 4 can be found in $O(n)$ time.*

Proof. For the last edge of the partial path leading to the endpoint a, we have already precomputed the set of possible extension edges for which the following conditions are maintained: the empty pseudo-triangle condition (Property 3 of Definition 2), the non-empty opposite wedge condition of Lemma 4, and the edge lies inside P. This list of $O(n)$ candidate edges is given in cyclic order. We

compute the visibility polygon of a with respect to α^* in $O(n)$ time, by Lemma 7, and we merge the candidate edges into the cyclic order of the visibility polygon, checking for each edge whether it intersects α^* in constant time.

As mentioned above, pointedness can also be checked in constant time for each edge. □

We will now sketch how to construct the (relevant part of) the visibility polygon in an easy way. Suppose that the current endpoint a is on the right of l and let us concentrate on the possible continuation edges to the right of a (moving further away from l). In this case we are only interested in the part of the visibility polygon that lies to the right of a.

Lemma 6. *Suppose a is on the right side of l and let r be a ray which emanates from a to the right (away from l). Let e_i be the first edge of α^* which is hit by r. Then all other edges of α^* which are hit by r come before e_i on α^*.*

Proof. (Sketch.) This is based on the fact that each of the pseudo-triangles formed by l and the parts of α^* right of l consist of two x-monotone chains from l to the right, meeting at a corner vertex, and that the intersections of α^* with l occur in the correct order (Property 1 of Definition 2). □

It follows that we can simply compute the right part of the visibility polygon by scanning the edges of α^* in reverse order, starting at a. The edges which are scanned so far will cover some angular region Q around a starting at the vertical upward direction. This part of the visibility polygon is already a correct part of the final visibility polygon. We only have to wait until some edge of α^* appears behind the already seen edges at the right edge of Q, and extend Q accordingly.

The same arguments apply to possible continuation edges to the left of a. Such an edge can only cross α^* if it crosses l. For the part of the visibility polygon that lies to the left of l, the above arguments can be applied. Thus we have:

Lemma 7. *The part of the visibility polygon of a with respect to α^* which lies to the right of a or to the left of l can be computed in $O(n)$ time.*

We can now enumerate all zigzag paths by scanning the enumeration tree. Note that the path is not automatically complete when it reaches an endpoint of the final edge e_f, but only when the edge e_f itself is inserted into the path. (Lemma 4 also holds when the partial path ends at an endpoint of e_f. In this case the continuation is always guaranteed.)

Theorem 2. *For a pointgon (S, P) and a line l we can enumerate the set $\alpha_l(S, P)$ of pseudo-triangulation zigzag paths in time $O(n^2 + n^2|\alpha_l(S, P)|)$ and space $O(n^2)$.*

Of course, this space bound does not include the space which is necessary to store all paths.

Proof. The enumeration tree has $|\alpha_l(S, P)|$ leaves. Since a zigzag path has length $O(n)$, being a noncrossing set of edges, the enumeration tree has depth $O(n)$,

and hence $O(n|\alpha_l(S,P)|)$ nodes. By Lemma 5, we spend $O(n)$ per node. The $O(n^2)$ preprocessing time was already mentioned. The $O(n^2)$ space includes the space for storing all cyclic orders and the stack of large angles for each point. □

Note that the time bound is overly pessimistic. In practice, the tree can be expected to be "bushy" and have only $O(|\alpha_l(S,P)|)$ nodes.

References

1. P. K. Agarwal, J. Basch, L. J. Guibas, J. Hershberger, and L. Zhang. Deformable free space tilings for kinetic collision detection. In B. R. Donald, K. Lynch, and D. Rus (eds.), *Algorithmic and Computational Robotics: New Directions (Proc. 5th Workshop Algorithmic Found. Robotics)*, pages 83–96. A. K. Peters, 2001.
2. O. Aichholzer. The Path of a Triangulation. In *Proc. 15th ACM Symp. Computational Geometry*, pages 14–23, 1999.
3. D. Avis. lrslib Software: Reverse search algorithm for vertex enumeration/convex hull problems. `http://cgm.cs.mcgill.ca/~avis/C/lrs.html`
4. D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Appl. Math.*, 65:21–46, 1996.
5. S. Bespamyatnikh. Enumerating Pseudo-Triangulations in the Plane. In *Proc. 14th Canad. Conf. Comp. Geom.*, pages 162–166, 2002.
6. S. Bespamyatnikh. An efficient algorithm for enumeration of triangulations. *Comp. Geom., Theory Appl.*, 23(3):271–279, 2002.
7. H. Brönnimann, L. Kettner, M. Pocchiola, and J. Snoeyink. Counting and enumerating pseudo-triangulations with the greedy flip algorithm. Manuscript, 2001.
8. Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas J. Guibas, John Hershberger, Micha Sharir, and Jack Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12:54–68, 1994.
9. A. Dumitrescu, B. Gärtner, S. Pedroni, and E. Welzl. Enumerating triangulation paths. *Comp. Geom., Theory Appl.*, 20:3–12, 2001.
10. H. Edelsbrunner and J. O'Rourke and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. In *SIAM J. Comput.*, 15:341–363, 1986.
11. K. Fukuda. Software: cdd and cddplus. `http://www.cs.mcgill.ca/~fukuda/soft/cdd_home/cdd.html`
12. Michael Goodrich and Roberto Tamassia, Dynamic ray shooting and shortest paths in planar subdivisions via balanced geodesic triangulations. *J. Algorithms* 23:51–73, 1997.
13. D. Kirkpatrick, J. Snoeyink, and B. Speckmann. Kinetic collision detection for simple polygons. *Intern. Journal Comp. Geom. Appl.*, 12(1 & 2):3–27, 2002.
14. David Orden, Francisco Santos The polytope of non-crossing graphs on a planar point set. Manuscript, February 2003, arXiv:math.CO/0302126.
15. M. Pocchiola and G. Vegter. Topologically sweeping visibility complexes via pseudo-triangulations. *Discrete Comp. Geom.*, 16:419–453, 1996.
16. G. Rote, F. Santos, and I. Streinu. Expansive motions and the polytope of pointed pseudo-triangulations. Manuscript, FU-Berlin, September 2001.
17. B. Speckmann and C. Tóth. Allocating Vertex π-guards in Simple Polygons via Pseudo-Triangulations. In *Proc. 14th Symp. on Discr. Algor.*, pages 109–118, 2003.
18. I. Streinu. A combinatorial approach to planar non-colliding robot arm motion planning. In *Proc. 41st FOCS*, pages 443–453, 2000.

Alternating Paths along Orthogonal Segments

Csaba D. Tóth

Department of Computer Science
University of California at Santa Barbara, CA 93106, USA,
toth@cs.ucsb.edu

Abstract. It was shown recently that the segment endpoint visibility graph $\mathrm{Vis}(S)$ of any set S of n disjoint line segments in the plane admits an alternating path of length $\Theta(\log n)$, and this bound is best possible apart from a constant factor. This paper focuses on the variant of the problem where S is a set of n disjoint *axis-parallel* line segments. We show that the length of a longest alternating path in the worst case is $\Theta(\sqrt{n})$. We also present an $O(n^{2.5})$ time algorithm to find an alternating path of length $\Omega(\sqrt{n})$. Finally, we consider sets of axis-parallel segments where the extensions of no two segments meet in the free space $\mathbb{E}^2 \setminus \bigcup S$, and show that in that case all the segments can be included in a common alternating path.

1 Introduction

Given a set S of disjoint line segments in the plane, an *alternating path* is a simple polygonal path $p = (v_1\, v_2,\, \ldots,\, v_k)$ such that $v_{2i-1}v_{2i} \in S$ for $i = 1, \ldots, \lfloor k/2 \rfloor$ and $v_{2i}v_{2i+1}$ does not cross any segment of S for $i = 1, \ldots, \lfloor (k-1)/2 \rfloor$.

A sets of disjoint segments do not always admit an alternating Hamiltonian path [17]. Hoffmann and Tóth [8] proved recently, answering a question of Urrutia [18,19] and Bose [3], that for any set S of n disjoint line segments in the plane, there is an alternating path that traverses at least $\lceil \log_2(n+2) \rceil - 1$ segments of S, and this bound is best possible apart from a constant factor.

The upper bound construction [18,8] (where there is no alternating path is longer than $O(\log n)$) consists of a set S of line segments such that every segment $s \in S$ has two endpoints on the convex hull $\mathrm{conv}(\bigcup S)$, and therefore any alternating path containing segments from both sides of s must contain s as well. In that construction n segments have $\Omega(n)$ distinct orientations.

If the segments have only two distinct orientations, or equivalently, if every segment is axis-parallel, then we obtain a tight bound of $\Theta(\sqrt{n})$ on the maximum length of an alternating path that any set of n disjoint line segments admits:

Theorem 1. *(i) For any n disjoint axis-parallel segments in the plane, there is an alternating path containing at least $\sqrt{n/2}$ of them.*

(ii) For any $n \in N$, there are n disjoint axis-parallel segments such that the longest alternating path contains at most $O(\sqrt{n})$ of them.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 389–400, 2003.
© Springer-Verlag Berlin Heidelberg 2003

The problem of of finding long alternating path along disjoint segments is related to three computational geometry problems: (i) To the Euclidean TSP problem where an agent A aims to visit a large number of line segments (each of them at most once), and by *visiting* we mean that A must traverse the segments from one endpoint to the other and between two segments A follows a straight line segment that does not cross any of the obstacles. (ii) It is also related to Ramsey type results in geometric graphs [10,5] where we want to find a large subset of segments that admits a Hamiltonian alternating path, but we also want to make sure that the portions of the path between segments does not cross any other segment obstacles. Finally it is related to visibility problems [11,18] because the alternating path is also an alternating path in the segment endpoint visibility graph [13] where the planar embedding of the path is simple (i.e., it has no self-crossings).

We consider also a special type of segment sets where there is no point in the free space of the segments which is the extension of two segments of S. (This was shown to be equivalent to the condition that the convex partitioning of the free space with a minimum number of cells is unique [15].) We call *protruded* a set of segments with this property. Any set of disjoint segments can be transformed into a *protruded* set by a simple *protruding* procedure: that is, by extending every segment one-by-one until it hits another (possibly already extended) segment or a sufficiently large bounding box.

A protruded set of segments, in general, does not admit a Hamiltonian alternating path if the segments can have many distinct orientations (see, e.g., the example of [17]). If all the input segments are axis-parallel, however, the following holds true.

Theorem 2. *For any protruded set S of disjoint axis-parallel line segments in the plane, there is a (Hamiltonian) alternating path through all segments of S.*

Using this theorem and the *protruding* procedure, we can answer a question of Mitchell about *1-2-alternating paths* for the case of axis-parallel segments. A 1-2-alternating path for S is a polygonal path $p = (v_1\, v_2, \ldots, v_k)$ such that $v_{3i-2}v_{3i-1} \in S$ and $v_{3i} \in \mathbb{E}^2 \setminus \bigcup S$ for $i = 1, \ldots, \lfloor k/3 \rfloor$; and neither $v_{3i-1}v_{3i}$ nor $v_{3i}v_{3i+1}$ crosses any segment of S for $i = 1, \ldots, \lfloor (k-1)/3 \rfloor$.

Theorem 3. *For any set S of disjoint axis-parallel segments in the plane, there is a 1-2-alternating path through all segments of S.*

2 Lower Bound Proof and Algorithm

2.1 Partial Order on Horizontal Segments

We may assume that at least $n/2$ segments of S are horizontal. We let H, $H \subseteq S$, be the set of horizontal segments and denote the left and right endpoint of every $s_i \in H$ by a_i and b_i respectively.

For two horizontal segments $s_1 = a_1b_1$ and $s_2 = a_2b_2$, we say that s_2 *cover*s s_1 (in sign, $s_1 \prec s_2$) if the x- and y-coordinate, resp., of a_1 is smaller or equal than

that of b_2. The relation $\prec$ induces a partial order on the horizontal segments: $s < t$ if and only if there is a sequence $(s = s_0, s_1, s_2, \ldots, s_r = t)$ in H such that s_i covers s_{i+1}, for $i = 0, 1, 2, \ldots, r-1$. Similar partial orders were previously used in [14] and [16].

By Dilworth's theorem [4], there is either (i) a chain or (ii) an anti-chain of size $\sqrt{n/2}$ with respect to $<$. In either case, we show that all segments in the chain or anti-chain can be interlaced into a common alternating path. Here we briefly prove the properties we need for two consecutive elements of a chain and for two elements of an anti-chain.

Proposition 1. *If $a_1b_1 < a_2b_2$ and there is no segment $t \in H$ with $a_1b_1 < t < a_2b_2$ then the line segment a_1b_2 is monotone increasing and does not cross any horizontal segment of S.*

Proof. Assume that $a_1b_1 < a_2b_2$ and there is no segment t with $a_1b_1 < t < a_2b_2$. Necessarily, $a_1b_2 \prec a_1b_2$, which implies that that a_1b_2 is monotone increasing. If the monotone increasing a_1b_2 crosses a segment t, that would imply $a_1b_1 \prec t \prec a_2b_2$ by definition, which would in turn contradict our initial assumption. □

Proposition 2. *If $a_1b_1 \not\prec a_2b_2$, $a_1b_1 \not\succ a_2b_2$, and the x-coordinate of a_1 is smaller than that of a_2, then there is a monotone decreasing curve from b_1 to a_2 that does not cross any horizontal segment of S.*

Proof. We first construct a monotone decreasing path η starting from b_1 and not crossing any horizontal segment of S, and then we show that η reaches a_2. Let $v_1w_1 = a_1b_1$ and put $i = 1$. We start drawing η from $b_1 = w_i$. Let η descend vertically from w_i until it hits either the horizontal line through a_2b_2 or a horizontal segment of $S \setminus \{a_2b_2\}$. If it hits the line through a_2b_2 then let η continue horizontally to b_1, and we are done. If, however, it hits a horizontal segment $v_{i+1}w_{i+1}$ then η continues along $v_{i+1}w_{i+1}$ until it reaches w_{i+1} and we repeat the procedure with $i := i + 1$.

Now suppose that η reaches the vertical line through a_2 but does not reach the horizontal line through a_2 (and so does not reach a_2 either). This means that there is a sequence $\{(v_iw_i) : i = 1 \ldots r\}$ of horizontal segments in S such that $a_1b_1 = v_1w_1 \succ v_2w_2 \succ \ldots \succ v_rw_r \succ a_2b_2$. That is, $a_1b_1 > a_2b_2$, a contradiction.

2.2 Expanding Operation

The segments of a chain or an anti-chain delivered by Dilworth's theorem do not form an alternating path right away. We describe below two recursive algorithms (for the case of a chain and an anti-chain) to build an alternating path passing through all the segments of the chain or anti-chain. Both algorithms start with an initial polygonal path which is not necessarily alternating. Both use a simple operation, which we call `Expand`, such that if a path has one common point with a segment then it modifies the path locally so as to include that segment. The same operation `Expand` was already used to construct a Hamiltonian circuit in [9]. In this Subsection, we define `Expand` and state a new and simple property that is particularly useful for our purposes.

Definition 1. *Let $v_1v_2v_3$ be simple polygonal path that does not cross any segment from S. The* convex arc $\mathrm{carc}(v_1, v_2, v_3)$ *with respect to S is the shortest polygonal path from v_1 to v_3 such that there is no segment endpoint in the interior of the closed polygonal curve* $\mathrm{carc}(v_1, v_2, v_3) \cup (v_3v_2v_1)$.

If v_1, v_2, and v_3 are not collinear, then $\mathrm{carc}(v_1, v_2, v_3) \cup v_3v_2v_1$ is a *pseudo-triangle* where all internal vertices of $\mathrm{carc}(v_1, v_2, v_3)$ are reflex.

Proposition 3. *If wa is monotone increasing, av is vertical and $90° \leq \angle vaw < 180°$ then every segment of the path* $\mathrm{carc}(v, a, w)$ *is also monotone increasing and* $\mathrm{carc}(v, a, w)$ *contains the right endpoints of horizontal segments of S and lower endpoints of vertical segments of S. (See Fig. 1)* □

Analogous statements for *wa* monotone *decreasing* and for *av horizontal* also hold (we do not list all analogues here, although we shall refer to all four variants as Proposition 3). All four variants require that $90° \leq \angle vaw < 180°$.

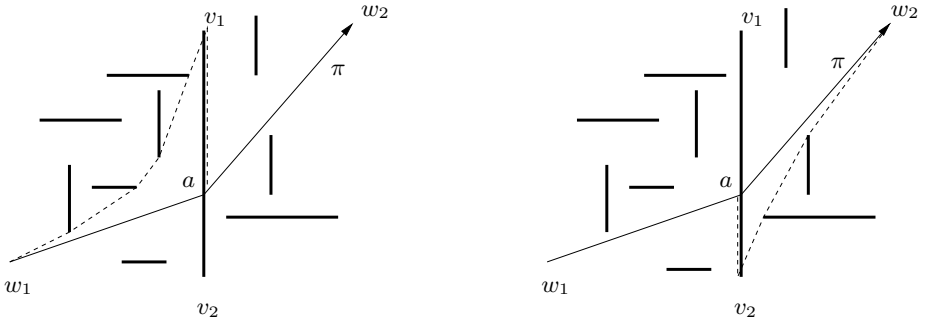


Fig. 1. `Expand`$(\pi, av_1, -)$ on the left and `Expand`$(\pi, av_2, +)$ on the right.

The operation `Expand` replaces a segment of a polygonal path by a path:

Operation 1 *Expand*(π, av, u) *(see Fig. 1).*
Input: a directed polygonal path π; a segment av such that $a \in \pi$ and $v \notin \pi$; and an orientation $u \in \{-, +\}$.
Operation: Let a^- and a^+ be the vertices of π preceding and following a. Obtain π' from π by replacing the edge aa^u of π by the path $(av)\mathrm{carc}(v, a, a^u)$.
Output: π'.

2.3 Alternating Path Obtained from a Chain

Let $s_1, s_2, \ldots, s_r$ be a sequence of r segments of H such that for every $i = 1, \ldots r-1$ we have $s_i \prec s_{i+1}$ and there is no $t \in H$ such that $s_i \prec t \prec s_{i+1}$.

We start out with an initial polygonal path γ through $s_1, s_2, \ldots, s_r$ which is not necessarily alternating but partitions the bounding box B of S into two

parts: Let γ_0 be a vertical segment from the lower side of B to b_1, let $\gamma_i = a_i b_{i+1}$ for $i = 1, 2, \ldots, r-1$, and let γ_r be a vertical segment from b_r to the upper side of B. Our initial path is $\gamma = \gamma_0\,(b_1 a_1)\,\gamma_1\,(b_2 a_2)\,\gamma_2\,\ldots\,(b_r a_r)\,\gamma_r$. According to Proposition 1, this initial γ does not cross any horizontal segment of S, but it might cross vertical segments.

Proposition 4. *The initial γ crosses every vertical segment of S at most once.*

Proof. By construction, only segments γ_i, $\gamma_i \subset \gamma$, can cross a vertical segment $t \in S$. The initial path γ is y-monotone increasing (the y-coordinate of a point moving along γ is monotone increasing). If γ crosses t twice, then one crossing traverses t from right to left. But this is impossible, because every right-to-left portion of γ lies along a horizontal segment of S which is disjoint from t. □

A segment which is neither crossed by γ nor lying on γ is strictly on its left or on its right side. We modify the path γ by calling recursively the operation `Expand` in two phases: The first phase eliminates all crossings with segments of S, the second phase proceeds to an alternating path. During both phases, we maintain five invariants:

1. Every vertex of γ is an endpoint of a segment from S.
2. Every portion of γ which does not lie along a segment of S is monotone increasing;
3. γ is a simple polygonal path;
4. If a segment $t \in S$ lies on the left (right) of γ and has a common point with γ, then $t \cap \gamma$ is the right or lower (left or upper) endpoint of t;
5. If γ crosses a segment $t \in S$, then t is vertical and there is a unique intersection point $t \cap \gamma$, which is not a vertex of γ.

In the first phase, our goal is to obtain a simple polygonal path from γ that does not cross any line segment of S. Let C denote the set of segments of S crossed by γ. We repeat the following step until C is empty: Consider the first segment $v_1 v_2 \in S$ crossed by γ such that v_1 is the upper endpoint of $v_1 v_2$, and let $a = v_1 v_2 \cap \gamma$. We modify γ by two operations: $\gamma := \texttt{Expand}(\gamma, av_1, -)$ and then $\gamma := \texttt{Expand}(\gamma, av_2, +)$ such that we form the convex arcs $\mathrm{carc}(a^-, a, v_1)$ and $\mathrm{carc}(a^+, a, _2)$ with respect to the set $S \setminus C$. As a result of the two `Expand` operations, a is not a vertex of γ, and the set C of segments crossed by γ is strictly decreases. Note that by invariant 3, whenever we append a path $\mathrm{carc}(vaw)$ to γ, we have $90° \leq \angle vaw < 180°$. This, by Proposition 3, assures that invariant 2 and 4 hold. The other three invariants hold by construction.

In the second phase, we replace invariant 5 by a stronger condition:

5.' The path γ does not cross any segment of S.

We expand recursively γ into an alternating path from s_1 to s_r. In the sight of the five invariants we only have to worry about segments which has one endpoint on γ, but which do not lie along γ. Let a be the first vertex along γ such that $ab = t \in S$ but $b \notin \gamma$. We modify γ to include t and visit the endpoint b as well:

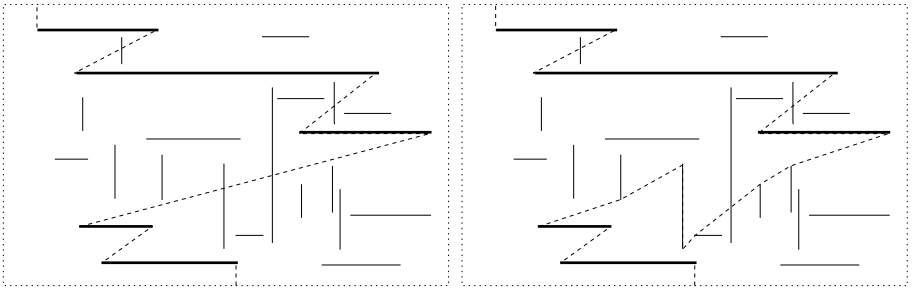


Fig. 2. The path γ in initial form (left), and after the first step of phase 1 (right).

- If ab is vertical and lies on the left side of γ, or if ab is horizontal and lies on the right side of γ, then apply $\texttt{Expand}(\gamma, ab, -)$.
- If ab is vertical and lies on the right side of γ, or if ab is horizontal and lies on the left side of γ, then apply $\texttt{Expand}(\gamma, ab, +)$.

We have chosen the orientation u of every call of operation `Expand` such that $90° \leq \angle baa^u < 180°$. Therefore, by Proposition 4, invariants 1–5' hold true. If every segment of S that intersect γ actually lies along γ, then γ is an alternating path (after removing the first and last edges γ_0 and γ_r). Since $s_1, s_2, \ldots, s_r$ still lie along γ, it contains at least $\sqrt{n/2}$ segments of S.

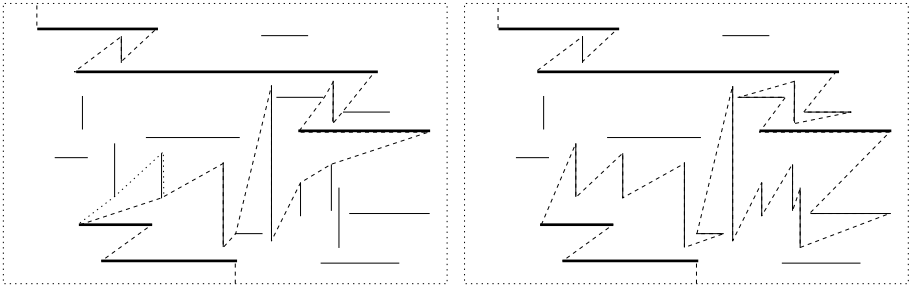


Fig. 3. γ at the end of phase 1 (left), and the output alternating path (right).

2.4 Alternating Path Obtained from an Anti-chain

Assume that there is an anti-chain A of size $r \geq \sqrt{n/2}$ in H. Since any two segments in A are separated by a vertical line, there is a linear left-to-right order among the elements of A. Consider the r segments of an anti-chain $A = \{s_1, s_2, \ldots, s_r\} \subset H$ labeled according to this order (Fig. 4).

By Proposition 2, there is a monotone decreasing path η_i between b_i and a_{i+1} for every $i = 1, 2, \ldots, r-1$. We can also construct two descending paths η_0 and η_r from the bounding box B of S to a_1 and to b_r resp. (e.g., η_0 can connect s_1 and an artificial horizontal segment outside the bounding box B).

For every η_i, $i = 0, 1, \ldots, r$, let γ_i be the shortest polygonal path between the two endpoints of η_i such that it does not cross any horizontal segment of S and γ_i is homothetic to η_i. (This can be modeled by placing a rubber band along the path η_i and letting it contract while its endpoints stay pinned down with the constraint that it cannot cross any segment of H.) Notice that every vertex of γ_i is an endpoint of a horizontal segment and every γ_i is monotone decreasing.

The remainder of the proof is similar to the argument in Subsection 2.3. We consider an initial path $\gamma = \gamma_0(a_1b_1)\gamma_1 \ldots (a_rb_r)\gamma_r$ which satisfies five invariants. The invariants are essentially the same as for the case of a chain except that now Invariant 2 states that every portion of γ which does not lie along a segment of S is monotone *decreasing*.

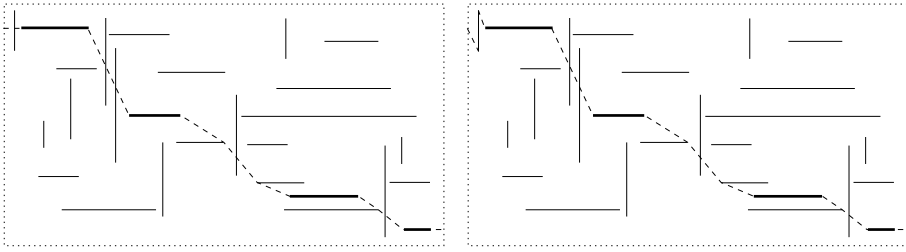


Fig. 4. The initial paths γ (left), after fist step of phase 1 (right).

We can again apply operations `Expand` in two phases: first eliminating all crossings with vertical segments and then preceding to an alternating path. The only difference compared to the previous subsection is that in every operation $\texttt{Expand}(\gamma, a, u)$ we use the opposite of the direction $u \in \{-, +\}$ used previously. This ensures that the angles $\angle baa^u$ are always in the range $[90°, 180°)$. Thus H contains either a chain or an anti-chain of size $\sqrt{n/2}$, we can interlace them into a common alternating path. This completes the proof of Theorem 1.

2.5 Complexity

The bottleneck in our algorithm is the use of Dilworth's theorem. The currently known best algorithm to find a chain or an anti-chain of size $\sqrt{n}$ in an n element partially ordered set is based on a reduction [6] to a bipartite matching problem for which one can apply the Hopcroft–Karp algorithm [7]. Given a partial order on n elements and m comparable pair the algorithm runs in $m\sqrt{n}$ time. In our case $m = O(n^2)$, and so this part of our algorithm takes $O(n^{2.5})$ time.

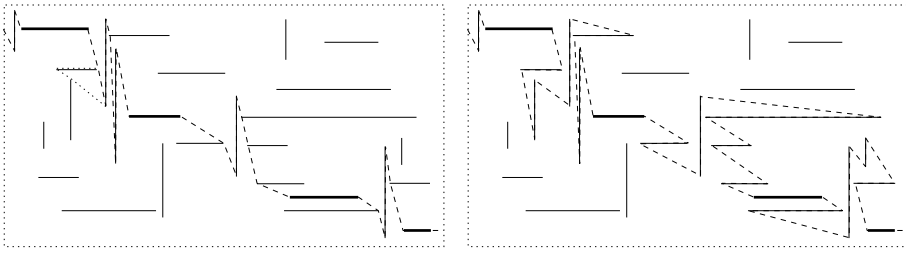


Fig. 5. γ at the end of phase 1 (left) and the resulting alternating path (right).

(Benczúr et al. [1] proposed an nh time algorithm where h is the number of *directly comparable pairs* $s_1 < s_2$ such that there is no t, $s_1 < t < s_2$. This does not help us, since in our case possibly $h = \Theta(n^2)$.)

The task of finding a shortest path homothetic to η_i, $i = 1, 2, \ldots r-1$ can be completed in a total of $O(n \log n)$ time because it can be reduced to determining convex arcs along η_i due to the fact that the resulting path is monotone. The best known general algorithm for this problem requires $O(n \log^2 n)$ time [2].

We can compute a convex arc of length k in output sensitive $O(k \log n)$ time using a triangulation of the free space of the line segments conformal to the path γ (by adding artificial vertices at the crossing points of vertical segments and γ if necessary). Since the set of vertices of γ is always increases (if a vertex a is included into γ at some point then a will be part of the final alternating path), all `Expand` operations can be completed in $O(n \log n)$ time.

3 Upper Bound Construction

First we prove Theorem 1 (ii) for every $n = \frac{4}{3}(4^k - 1)$, $k \in N$.

Lemma 1. *For every $k \in N$ there is a set of $n = \frac{4}{3}(4^k - 1)$ disjoint axis-parallel line segments such that the length of the longest alternating path is $\sqrt{12(n+1)} - 4 = 4(2^k - 1)$*

Proof. We describe a construction S_k of $\frac{4}{3}(4^k - 1)$ disjoint axis-parallel line segments recursively for every $k \in N$. S_1 is a set of four disjoint line segments placed along the four sides of a square. For $k > 1$, we obtain S_k as follows. Consider a disk D_k and a square Q_k such that they have a common center of symmetry and both $D_k \setminus Q_k$ and $Q_k \setminus D_k$ are non-empty. S_k consists of four chords of D_k along the four sides of Q_k and four copies of S_{k-1} in the four components of $D_k \setminus Q_k$ (see Fig. 6). We call the four segments along sides of Q_k the *principal segments* of S_k.

By construction, $|S_1| = 4$ and $S_k = 4 + 4|S_{k-1}|$, so $|S_k| = 4 + 4^2 + \ldots + 4^k = \frac{4}{3}(4^k - 1)$. We also remark that the construction S_k contain a total of $4^{\ell-1}$ copies of the construction S_ℓ for every ℓ, $1 \leq \ell \leq k$.

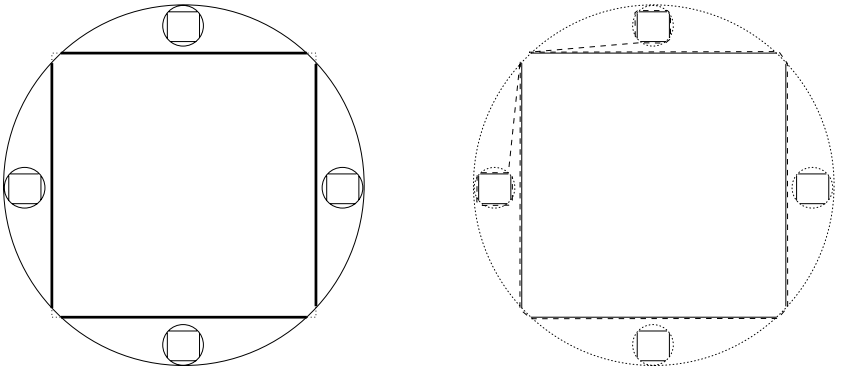


Fig. 6. S_2 (left) and a longest alternating path on S_2 (right).

It rests to see that the longest alternating path includes at most $4(2^k - 1)$ segments. We prove by induction on ℓ that an alternating path can contain the principle segments of at most $2^{k-\ell}$ copies of S_ℓ, $1 \leq \ell \leq k$, within S_k. Since every copy has only four principle segments, this totals to $4(1+2+\ldots+2^{k-1}) = 4(2^k-1)$ segments.

For the ease of the induction argument, we actually prove a stronger statement: An alternating path α has at most $2^{k-\ell}$ disjoint maximal subpaths such that each subpath uses only segments from one copy of S_ℓ $1 \leq \ell \leq k$, in S_k. The statement clearly holds for $\ell = k$.

Assuming that the statement holds for all ℓ', $\ell < \ell' \leq k$, we argue about ℓ. Let C be a copy of $S_{\ell+1}$ and let α_C be a maximal subpath of α within C. Observe that if α_C contains segments from a copy of S_ℓ in C but is also contains segments from another copy of S_ℓ in C, then α_C must include the principle segments of C which block the each copy of S_ℓ from the rest of C. Therefore if α_C contains segments from a copy of S_ℓ, then at least one endpoint of α_C must be in that copy. Consequently, α_C has at most two maximal subpaths such that each uses segment exclusively from one copy of S_ℓ within C. □

For values n, $\frac{4}{3}(4^{k-1}-1) < n < \frac{4}{3}(4^k-1)$, we can give similar but unbalanced constructions: Let us assume that $n = 4 + m_1 + m_2 + m_3 + m_4$ such that $m_i \leq \frac{4}{3}(4^{k-1} - 1)$ for $i = 1, 2, 3, 4$. We place four segments along the chords of D along the four sides of Q. Then in the four components of $D \setminus Q$, we place copies of construction with m_i, $i = 1, 2, 3, 4$, segments respectively.

4 Protruded Orthogonal Segments

In this section, we prove Theorem 2 and give an $O(n \log n)$ time algorithm that constructs an alternating path along all segments of S.

Let B be the (axis-parallel) bounding box of S. We compute the (unique) convex partitioning of the free space $B \setminus \bigcup S$ in $O(n \log n)$ time. This can be

done by extending sequentially every segment until it hits another segment or the boundary of B. Thus we partition B into $n+1$ rectangular faces. Consider a face F of the partition. A corner v of F is either a corner of B or a point where the extension of a segment ab beyond its endpoint a hits another segment or the boundary of B. In this latter case, the vertex a lies on the boundary of F, because S is protruded. So we can say that the corner $v(F)$ corresponds to the segment endpoint a.

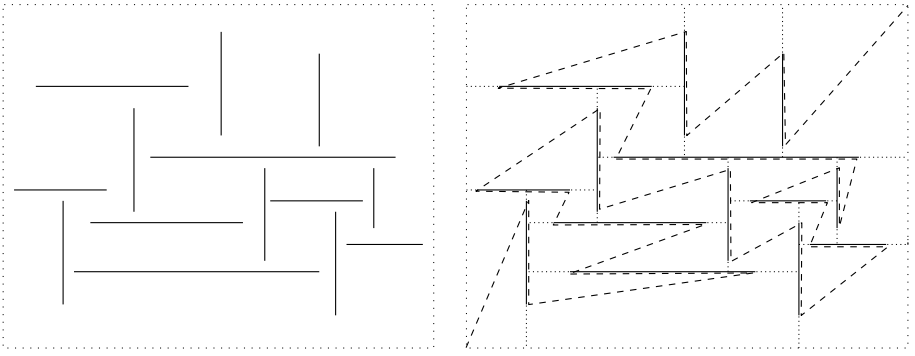


Fig. 7. A protruded set of 14 segments (left) and our alternating path (right).

We are ready to construct the alternating path α though S: Denote by b_0 the lower left corner of B and set $i = 0$. We start drawing an alternating path from b_i. Let F_i be the face whose lower left corner is b_i. If F_i is not the upper right face of the partition then let a_{i+1} denote the segment endpoint corresponding to the upper right corner of F_i where $a_{i+1}b_{i+1} \in S$. Append the segments b_ia_{i+1} and $a_{i+1}b_{i+1}$ to the path α and put $i := i + 1$.

Observe that if a_{i+1} corresponds to the upper right corner of a face, then a_{i+1} is an upper endpoint of a vertical segment or the right endpoint of a horizontal segment. Therefore, the other endpoint b_{i+1} of the segment corresponds to the lower left corner of a face F_{i+1}. This assures that our algorithm ends only if α reaches the upper right corner c of B, which does not correspond to any segment endpoint.

In order to prove that the alternating path α visits all n segments, it is enough to show that α traverses all $n+1$ faces of the partition. For this, we observe that α traverses a face F_i only if it has already traversed every face whose lower left corner has smaller x- or y-coordinate than that of F_i. Since the lower left corner of every face has smaller x- or y-coordinate than that of the face F_c incident to c, this implies that α traverses all the faces before it reaches c

5 Concluding Remarks

We have shown that the longest alternating path in a set of n disjoint axis-parallel line segments in the plane includes $\Omega(\sqrt{n})$ segments and this bound is best possible. Our proof is based on Dilworth's theorem and computation of convex arcs among polygonal obstacles. We close paper with a couple of open questions.

- What is the complexity of finding the longest alternating path (for axis-parallel and for generic segments)?
- Is there always an $\Omega(\sqrt{n})$ long alternating path if the segments have a constant number of directions? (Our upper bound construction readily generalizes, but our lower bound algorithm does not.)
- Is there a faster algorithm to find a chain or anti-chain of size $\sqrt{n/2}$ than the one using a detour via Hopcroft-Karp algorithm?
- Is there always a 1-2-alternating path through all segments of any protruded set of disjoint line segments?

References

1. Benczúr A.A., Förster J., Király. Z..: Dilworth's Theorem and its application for path systems of a cycle – implementation and analysis. In: *Proc. 7th European Symp. on Algorithms (Prague, 1999)*, LNCS vol. 1643, Springer-Verlag, Berlin, 498–509.
2. Bespamyatnikh S.: Computing homotopic shortest paths in the plane. In: *Proc. 14th ACM-SIAM Symp. Discrete Algorithms (Baltimore, MD, 2003)*, 609–617.
3. Demaine E.D., O'Rourke J.: Open Problems from CCCG'99. In: *Proc. 11th Canadian Conf. on Comput. Geom. (Vancouver, BC, 1999).*
4. Dilworth R.: A decomposition theorem for partially ordered sets. *Ann. of Maths.* **51** (1950), 161–166.
5. Dumitrescu A., Tóth G.: Ramsey-type results for unions of comparability graphs. *Graphs and Combinatorics* **18** (2002), 245–251.
6. Ford, Jr., L. R., Fulkerson, D. R.: *Flows in Networks.* Princeton University Press, Princeton, NJ, 1962.
7. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM J. Cornput.* **2** (1973), 225–231.
8. Hoffmann M., Tóth Cs.D.: Alternating paths through disjoint line segments. *Inform. Proc Letts.* (to appear).
9. Hoffmann M., Tóth Cs.D.: Segment endpoint visibility graphs are Hamiltonian. *Comput. Geom. Theory Appl.* **26** (1) (2003).
10. Larman D.G., Matoušek J., Pach J., Törőcsik J.: A Ramsey-type result for planar convex sets. *Bulletin of the London Mathematical Society* **26** (1994), 132–136.
11. O'Rourke J.: Visibility. In: *Handbook of Discrete and Computational Geometry (J. E. Goodman and J. O'Rourke, eds.)*, CRC Press, 1997, chap. 25, pp. 467–480.
12. O'Rourke J., Rippel J.: Two segment classes with Hamiltonian visibility graphs. *Comput. Geom. Theory Appl.* **4** (1994), 209–218.
13. M. H. Overmars and E. Welzl, New methods for computing visibility graphs. In: *Proc. 4th ACM Symp. Comput. Geom. (Urbana-Champaign, IL, 1988)*, 164–171.

14. Tamassia R., Tollis I.G.: A unified approach to visibility representations of planar graphs. *Discrete Comput. Geom.* **1** (1986), 321–341.
15. Tóth Cs.D.: Illumination in the presence of opaque line segments in the plane. *Comput. Geom. Theory Appl.* **21** (2002), 193–204.
16. Tóth G.: Note on geometric graphs. *J. Combin. Theory, Ser. A* **89** (2000), 126–132.
17. Urabe M., Watanabe M.: On a counterexample to a conjecture of Mirzaian. *Comput. Geom. Theory Appl.* **2** (1992), 51–53.
18. J. Urrutia J.: Algunos problemas abiertos (in Spanish). In: *Actas de los IX Encuentros de Geometría Computacional (Girona, 2001)*.
19. Urrutia J.: Open problems in computational geometry. In: *Proc. 5th Latin Amer. Symp. Theoret. Inf. (Cancún, 2002)*, LNCS vol. 2286, Springer-Verlag, pp. 4–11.

Improved Approximation Algorithms for the Quality of Service Steiner Tree Problem

Marek Karpinski[1], Ion I. Măndoiu[2], Alexander Olshevsky[3], and Alexander Zelikovsky[4]

[1] Department of Computer Science, University of Bonn, Bonn 53117, Germany
marek@theory.cs.uni-bonn.de
[2] Electrical and Computer Engineering Department, University of California at San Diego, La Jolla, CA 92093-0114
mandoiu@cs.ucsd.edu
[3] Department of Electrical Engineering, Georgia Institute of Technology, Atlanta, GA 30332
gte864k@prism.gatech.edu
[4] Computer Science Department, Georgia State University, Atlanta, GA 30303
alexz@cs.gsu.edu

Abstract. The Quality of Service Steiner Tree Problem is a generalization of the Steiner problem which appears in the context of multimedia multicast and network design. In this generalization, each node possesses a rate and the cost of an edge with length l in a Steiner tree T connecting the non-zero rate nodes is $l \cdot r_e$, where r_e is the maximum rate in the component of $T - \{e\}$ that does not contain the source. The best previously known approximation ratios for this problem (based on the best known approximation factor of 1.549 for the Steiner tree problem in networks) are 2.066 for the case of two non-zero rates and 4.211 for the case of unbounded number of rates. We give better approximation algorithms with ratios of 1.960 and 3.802, respectively. When the minimum spanning tree heuristic is used for finding approximate Steiner trees, then the previously best known approximation ratios of 2.667 for two non-zero rates and 5.542 for unbounded number of rates are reduced to 2.414 and 4.311, respectively.

1 Introduction

The Quality of Service Steiner Tree (QoSST) problem appears in two different contexts: multimedia distribution for users with different bitrate requests [7] and the general design of interconnection networks with different grade of service requests [6]. The problem was formulated as a natural generalization of the Steiner problem under the names "Multi-Tier Steiner Tree Problem" [8] and "Grade of Service Steiner Tree Problem" [13]. More recently, the problem has been considered by [5,7] in the context of multimedia distribution. This problem generalizes the Steiner tree problem in that each node possesses a rate and the cost of a link is not constant but depends both on the cost per unit of transmission bandwidth and the maximum rate routed through the link.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 401–411, 2003.
© Springer-Verlag Berlin Heidelberg 2003

Formally, the QoSST problem can be stated as follows (see [5]). Let $G = (V, E, l, r)$ be a graph with two functions, $l : E \to R^+$ representing the length of each edge, and $r : V \to R^+$ representing the rate of each node. Let $\{r_0 = 0, r_1, r_2, \ldots r_N\}$ be the range of r and S_i be the set of all nodes with rate r_i. The Quality of Service Steiner Tree Problem asks for a minimum cost subtree T of G spanning a given source node s and nodes in $\bigcup_{i \geq 1} S_i$, all of which are referred to as *terminals*. The cost of an edge e in T is $\text{cost}(e) = l(e)r_e$, where r_e, called the *rate of edge* e, is the maximum rate in the component of $T - \{e\}$ that does not contain the source. Note that the nodes in S_0, i.e., zero rate nodes, do not require to be connected to the source s but may serve as Steiner points for the output tree T.

The QoSST problem is equivalent to the Grade of Service Steiner Tree Problem (GOSST) [13], which has a slightly different formulation. In GOSST there is no source node and edge rates r_e should be assigned such that the minimum edge rate on the tree path from a terminal with rate r_i to a terminal with rate r_j is at least $min(r_i, r_j)$. It is not difficult to see that these two formulations are equivalent. Indeed, an instance of QoSST can be transformed into an instance of GOSST by assigning the highest rate to the source. The cost of an edge will remain the same, since each edge e in a tree T will be on the path from the source to the node of the highest rate in the component of $T - \{e\}$ that does not contain the source. Conversely, an instance of GOSST can be transformed into a QoSST by giving source status to any node with the highest rate.

The problem was studied before in several contexts. Current et al. [6] gave an integer programming formulation for the problem and proposed a heuristic algorithm for its solution. Some results for the case of few rates were obtained in [1] and [2]. Specifically, [2] (see also [13]) suggested an algorithm for the case of two non-zero rates with approximation ratio of $\frac{4}{3}\alpha \approx 2.065$, where $\alpha \approx 1.549$ is the best approximation ratio of an algorithm for the Steiner tree problem. Recently, [5] gave the first constant-factor approximation algorithm for an unbounded number of rates. They achieved an approximation ratio of $e\alpha \approx 4.211$.

In this paper we give algorithms with improved approximation factors. Our algorithms have an approximation ratio of 1.960 when there are two non-zero rates and an approximation ratio of 3.802 when there is an unbounded number of rates. The improvement comes from the reuse of higher rate edges in establishing connectivity for lower rate nodes. We give the first analysis of the gain resulting from such reuse, critically relying on approximation algorithms for computing k-restricted Steiner trees. To improve solution quality, we use different Steiner tree algorithms at different stages of the computation. In particular, we use both the Steiner tree algorithm from [11] which has the currently best approximation ratio and the algorithm from [10] which has the currently best approximation ratio among Steiner tree algorithms producing 3-restricted trees.

Table 1 summarizes the results of this paper. It presents previously known approximation ratios using various Steiner tree algorithms and the approximation ratios produced by our method utilizing the same algorithms. Note that along with the best approximation ratios resulting from the use of the loss-contracting

Table 1. Runtime and approximation ratios of previously known algorithms and of the algorithms given in this paper. In the runtime, n and m denote the number of nodes and edges in the original graph $G = (V, E)$, respectively.

Algorithm	LCA [11]		RNS[10]		BR [14,3]		MST [12]	
runtime	polynomial		polynomial		$O(n^3)$ [15]		$O(n \log n + m)$ [9]	
#rates	2	any	2	any	2	any	2	any
previous ratio	2.066 + ϵ	4.211 + ϵ	2.222 + ϵ	4.531 + ϵ	2.444	4.934	2.667	5.44
our ratio	1.960+ϵ	3.802 +ϵ	2.059 +ϵ	3.802 + ϵ	2.237	4.059	2.414	4.311

algorithm from [11], Table 1 also gives approximation ratios resulting from the use of the algorithm in [10] and the more practical algorithms in [3,12,14].

The rest of the paper is organized as follows. In next section, we tighten the analysis given in [4] for the k-restricted Steiner ratio. In Section 3, we introduce the so called β-convex Steiner tree approximation algorithms and tighten their performance bounds. We give approximation algorithms for QoSST problem with two non-zero rates and unbounded number of rates in Sections 4 and 5, respectively, and conclude in Section 6.

2 A Tighter Analysis of the k-Restricted Steiner Ratio

In this section, we tighten the analysis given in [4] for the k-restricted Steiner ratio. The tightened results will be used later to prove the approximation ratio of our algorithms. The exposition begins with a claim from [4] which encapsulates several of the proofs provided in that paper. This claim is then used in a manner slightly different from [4] to arrive at a stronger result.

We begin by introducing some definitions. A Steiner tree is called *full* if every terminal is a leaf. A Steiner tree can be decomposed into components which are full by breaking the tree up at the non-leaf terminals. A Steiner tree is called k-restricted if every full component has at most k terminals. Let us denote the length of the optimum k-restricted Steiner tree as opt_k and length of the optimum unrestricted Steiner tree as opt. By duplicating nodes and introducing zero length edges, it can be assumed that a Steiner tree T is a complete binary tree (see Figure 1). Furthermore, we may assume that the leftmost and rightmost terminals form a diametrical pair of terminals. The leftmost and rightmost terminals will be called *extreme terminals*, and the edges on the path between them will be called *extreme edges*.

Let the k-restricted Steiner ratio ρ_k be $\rho_k = sup\frac{opt_k}{opt}$, where the supremum is taken over all instances of the Steiner tree problem. It has been shown in [4] that $\rho_k = \frac{(r+1)2^r+s}{r2^r+s}$ where r and s are obtained from the decomposition $k = 2^r + s$, $0 \leq s < 2^r$.

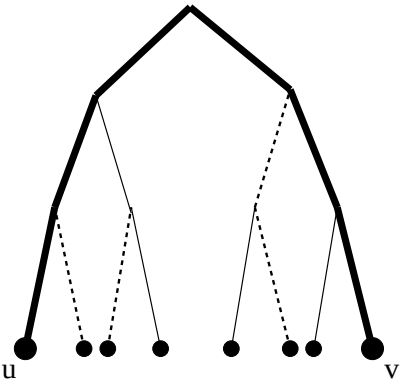


Fig. 1. Optimal Steiner tree T represented as a complete binary tree. Extreme terminals u and v form a diametrical pair of terminals, extreme edges (the path between u and v) are shown thicker. Each L_i represents the total length of a collection of paths (e.g., dashed paths) connecting internal nodes of T to non-extreme terminals via non-extreme edges.

Lemma 1. *[4][1] Given a Steiner tree T, there exist k-restricted Steiner trees T_i, $i = 1, 2, \ldots, r2^r + s$ such that $l(T_i) = l(T) + L_i$, where each L_i represents the total length of a collection of paths connecting internal nodes of T to non-extreme terminals via non-extreme edges in such a way that each non-extreme edge of T is counted at most 2^r times in the sum $L_1 + L_2 + \cdots + L_{r2^r+s}$.*

We now use Lemma 1 to produce a tighter bound on the length of the optimal k-restricted Steiner tree.

Theorem 1. *For every full Steiner tree T, $opt_k \leq \rho_k(l-D)+D$, where $l = l(T)$ is the length of T and $D = D(T)$ is the length of the longest path in T.*

Proof. Lemma 1 implies that $L_1 + L_2 + \cdots + L_{r2^r+s} \leq 2^r(l - D)$. From this it follows that there exists L_m such that $L_m \leq \frac{2^r}{r2^r+s}(l-D)$. Since $l(T_m) = l+L_m$, it follows that $l(T_m) \leq l + \frac{2^r}{r2^r+s}(l - D)$. Therefore,

$$\begin{aligned} opt_k &\leq l(T_m) \\ &\leq l + \frac{2^r}{r2^r + s}(l - D) \\ &= \left(1 + \frac{2^r}{r2^r + s}\right)(l - D) + D \\ &= \rho_k(l - D) + D \end{aligned}$$

We now strengthen this theorem to the case of partitioned trees.

[1] The claim in [4] is stated for an optimum Steiner tree T, but optimality is not needed in the proof.

Corollary 1. *For every Steiner tree T partitioned into edge-disjoint full components T^i,*

$$opt_k \leq \sum_i \left(\rho_k(l(T^i) - D(T^i)) + D(T^i)\right)$$

Proof. Let opt_k^i be the length of the optimal k-restricted tree for the full component T^i. Then,

$$opt_k \leq \sum_i opt_k^i \leq \sum_i \left(\rho_k(l(T^i) - D(T^i)) + D(T^i)\right)$$

3 β-Convex Steiner Tree Approximation Algorithms

In this section we introduce β-convex Steiner tree approximation algorithms and show tighter upper bounds on their output when applied to the QoSST problem.

Definition 1. *An α-approximation Steiner tree algorithm A is called* β-convex *if the length of the tree it produces, $l(A)$, is upper bounded by a linear combination of optimal k-restricted Steiner trees, i.e.,*

$$l(A) \leq \sum_{i=2}^{m} \lambda_i opt_i$$

and the approximation ratio is equal to

$$\alpha = \sum_{i=2}^{m} \lambda_i \rho_i$$

where $\lambda_i \geq 0$, $i = 2, \ldots, m$ and

$$\beta = \sum_{i=2}^{m} \lambda_i$$

The algorithms from [12,3,10] are β-convex, while the currently best approximation algorithm from [11] is not known to be β-convex.

Given a β-convex α-approximation algorithm A, it follows from Theorem 1 that

$$l(A) \leq \sum_i \lambda_i opt_i \leq \sum_i \lambda_i \rho_i (opt - D) + \beta D = \alpha(opt - D) + \beta D \qquad (1)$$

Let OPT be the optimum cost QoSST tree T, and let t_i be the length of rate r_i edges in T. Then,

$$cost(OPT) = \sum_{i=1}^{N} r_i t_i$$

Below we formulate the main property that makes β-convex Steiner tree approximation algorithms useful for QoSST approximation.

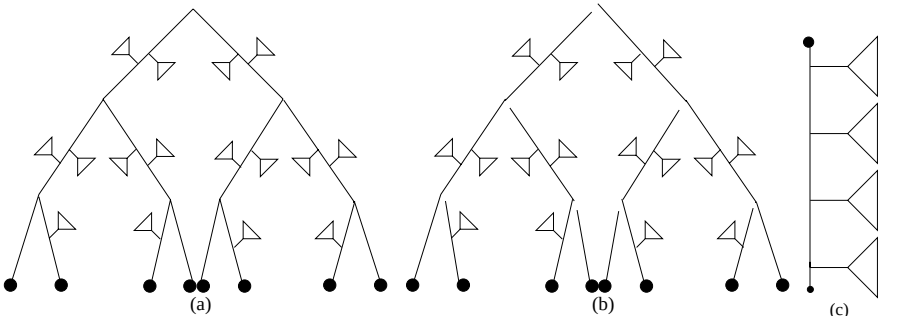


Fig. 2. (a) The subtree OPT_k of the optimal QoS Steiner tree OPT induced by edges of rate r_i, $i \geq k$. Edges of rate greater than r_k (shown as solid lines) form a Steiner tree for $s \cup S_{k+1} \cup \ldots S_N$ (filled circles); attached triangles represent edges of rate r_k. (b) Partition of OPT_k into edge-disjoint connected components OPT_k^i each containing a single terminal of rate r_i, $i > k$. (c) A connected component OPT_k^i which consists of a path D_k^i containing all edges of rate r_i, $i > k$, and attached Steiner trees containing edges of rate r_k.

Lemma 2. *Given an instance of the QoSST problem, let T_k be the Steiner tree computed for s and all nodes of rate r_k by a β-convex α-approximation Steiner tree algorithm after collapsing all nodes of rate strictly higher than r_k into the source s and treating all nodes of rate lower than r_k as Steiner points. Then,*

$$cost(T_k) \leq \alpha r_k t_k + \beta(r_k t_{k+1} + r_k t_{k+2} + \cdots + r_k t_N)$$

Proof. We can visualize the subtree OPT_k of the optimal QoS Steiner tree OPT induced by edges of rate r_i, $i \geq k$ as in Figure 2(a), with nodes of rate r_k forming subtrees attached to the tree OPT_{k+1} that spans all nodes with rate higher than r_k and the source. We break OPT_{k+1} into edge disjoint paths connecting each terminal with an appropriately chosen non-extreme node as illustrated in Figure 2(b). A proof that this kind of decomposition is always possible can be found in [14]. We then consider each such path along with all nodes of rate r_k that are attached to it. This results in a decomposition of OPT_k into edge-disjoint connected components OPT_k^i, where each component consists of a path $D_k^i = OPT_k^i \cap OPT_{k+1}$ and attached Steiner trees with edges of rate r_k (see Figure 2(c)). Furthermore, note that the total length of D_k^i's is $l(OPT_{k+1}) = t_{k+1} + t_{k+2} + \cdots + t_N$.

Now we decompose the tree T_k along these full components OPT_k^i and by Corollary 1 we get:

$$\begin{aligned} l(T_k) &\leq \sum_i \left[\alpha(l(OPT_k^i) - D_k^i) + \beta D_k^i\right] \\ &= \alpha t_k + \beta(t_{k+1} + t_{k+2} + \cdots + t_N) \end{aligned}$$

The lemma follows by multiplying the last inequality by r_k.

Input: Graph $G = (V, E, l)$ with two nonzero rates $r_1 < r_2$, source s, terminal sets S_1 of rate r_1 and S_2 of rate r_2, Steiner tree α_1-approximation algorithm A_1 and a β-convex α_2-approximation algorithm A_2
Output: Low cost QoSST spanning all terminals

1. Compute an approximate Steiner tree ST1 for $s \bigcup S_1 \bigcup S_2$ using algorithm A_1
2. Compute an approximate Steiner tree T_2 for $s \bigcup S_2$ (treating all other points as Steiner points) using algorithm A_1. Next, contract T_2 into the source s and compute the approximate Steiner tree T_1 for s and remaining rate r_1 points using algorithm A_2. Let ST2 be $T_1 \bigcup T_2$
3. Output the minimum cost tree among ST1 and ST2

Fig. 3. QoSST approximation algorithm for two non-zero rates

4 QoSST Approximation Algorithm for Two Non-zero Rates

In this section we give a generic approximation algorithm for the QoSST Steiner tree problem with two non-zero rates (see Figure 3) and analyze its approximation ratio.

Recall that an edge e has rate r_i if the largest rate of a node in the component of $T - \{e\}$ that does not contain the source is r_i. Let the optimal Steiner tree in G have cost $opt = r_1 t_1 + r_2 t_2$, with t_1 being the total length of the edges of rate r_1 and t_2 being the total length of the edges of rate r_2. Let α_1 be the approximation ratio of algorithm A_1 and let α_2 be the approximation ratio of the β-convex algorithm A_2. Then, the following theorem holds:

Theorem 2. *The approximation ratio of the algorithm from Figure 3 is*

$$\max\left\{\alpha_2, \quad \max_r \ \alpha_1 \frac{\alpha_1 - (\alpha_2 - \beta)r}{\beta r^2 + \alpha_1 - \alpha_2 r}\right\}$$

Proof. We can bound the cost of ST1 by $cost(ST1) \leq \alpha_1 r_2 (t_1 + t_2)$. To obtain a bound on the cost of ST2 note that $cost(T_2) \leq \alpha_1 r_2 t_2$, and that, by Lemma 2, $cost(T_1) \leq \alpha_2 r_1 t_1 + \beta r_1 t_2$.

Thus, the following two bounds for the costs of $ST1$ and $ST2$ follow:

$$cost(ST1) \leq \alpha_1 r_2 t_1 + \alpha_1 r_2 t_2$$
$$cost(ST2) \leq \alpha_1 r_2 t_2 + \alpha_2 r_1 t_1 + \beta r_1 t_2$$

We distinguish between the following two cases:
Case 1: If $\beta r_1 - (\alpha_2 - \alpha_1) r_2 \leq 0$, then $cost(ST2) \leq \alpha_2 (r_2 t_2 + r_1 t_1) = \alpha_2 opt$.
Case 2: If $\beta r_1 - (\alpha_2 - \alpha_1) r_2 \geq 0$, then let

$$x_1 = \frac{\beta r_1^2 + (\alpha_1 - \alpha_2) r_1 r_2}{\alpha_1 r_2 (\alpha_1 r_2 - \alpha_2 r_1 + \beta r_1)}$$
$$x_2 = \frac{r_2 - r_1}{\alpha_1 r_2 - \alpha_2 r_1 + \beta r_1}$$

It is easy to check that

$$x_1 cost(ST1) + x_2 cost(ST2) \leq opt$$

which implies that

$$Approx \leq \frac{1}{x_1 + x_2} opt$$

In turn, this simplifies to

$$Approx \leq \alpha_1 \frac{\alpha_1 - (\alpha_2 - \beta)r}{\beta r^2 - \alpha_2 r + \alpha_1} opt$$

where $r = \frac{r_1}{r_2}$.

We can use Theorem 2 to obtain numerical bounds on the approximation ratios of our solution. Using $\alpha_1 = 1 + \ln 3/2$ for the algorithm from [11], $\alpha_2 = 5/3$ for the algorithm from [10], $\alpha_1 = \alpha_2 = 11/6$ for the algorithm from [3], and $\alpha_1 = \alpha_2 = 2$ for the MST heuristic, and $\beta \to 1$ for all of the above algorithms, we maximize the expression in Theorem 2 to obtain the following theorem.

Theorem 3. *If the algorithm from [11] is used as A_1 and the algorithm from [10] is used as A_2, then the approximation ratio of the QoSST algorithm in Figure 3 is* 1.960*. If the algorithm from [3] is used in place of both A_1 and A_2, then the ratio is 2.237. If the MST heuristic is used in place of both A_1 and A_2, then the ratio is 2.414.*

5 Approximation Algorithm for QoSST with Unbounded Number of Rates

In this section, we propose an algorithm for the case of a graph with arbitrarily many non-zero rates $r_1 < r_2 < \cdots < r_N$. Our algorithm is a modification of the algorithm in [5]. A description of the algorithm is given in Figure 4. As in [5], node rates are rounded up to the closest power of some number a starting with a^y, where y is picked uniformly at random between 0 and 1. In other words, we round up node rates to numbers in the set $\{a^y, a^{y+1}, a^{y+2}, \ldots\}$. The only difference is that we *contract* each approximate Steiner tree, $Approx_k$, constructed over nodes of rounded rate a^{y+k}, instead of simply taking their union as in [5]. This allows contracted edges to be reused at zero cost by Steiner trees connecting lower rate nodes. The following analysis of this improvement shows that it decreases the approximation ratio from 4.211 to 3.802.

Let T_{opt} be the optimal QoS Steiner tree, and let t_i be the total length of the edges of T_{opt} with rates rounded to a^{y+i}. First, we prove the following technical lemma:

Lemma 3. *Let S be the cost of T_{opt} after rounding node rates as in Figure 4, i.e., $S = \sum_{i=0}^{n} t_i a^{y+i}$. Then,*

$$S \leq \frac{a-1}{\ln(a)} cost(T_{opt})$$

Input: Graph $G = (V, E, l)$, source s, sets S_i of terminals with rate r_i, positive number a, and α-approximation β-convex Steiner tree algorithm
Output: Low cost QoSST spanning all terminals

1. Pick y uniformly at random between 0 and 1. Round up each rate to the closest power of some number a starting with a^y, i.e. round up to numbers in the set $\{a^y, a^{y+1}, a^{y+2}, \ldots\}$. Form new terminal sets S_i' which are unions of terminal sets with rates rounded to the same number r_i'
2. $Approx \leftarrow \emptyset$
3. Repeat until all terminals are contracted into the source s:
   Find an α-approximate Steiner tree $Approx_i$ spanning $s \bigcup S_i'$
   $Approx \leftarrow Approx \cup Approx_i$
   Contract $Approx_i$ into source s
4. Output $Approx$

Fig. 4. Approximation algorithm for multirate QoSST

Proof. First, note that an edge e used at rate r in T_{opt} will be used at the rate a^{y+m}, where m is the smallest integer i such that a^{y+i} is no less than r. Indeed, e is used at rate r in T_{opt} if and only if the maximum rate of a node connecting to the source via e is r, and every such node will be rounded to a^{y+m}. Next,[2] let $r = a^{x+m}$. If $x \leq y$ then the rounded up cost is a^{y-x} times the original cost; otherwise, if $x > y$, is a^{y+1-x} times the original cost. Hence, the expected factor by which the cost of each edge increases is

$$\int_0^x a^{y+x-1} dy + \int_x^1 a^{y-x} dy = \frac{a-1}{\ln a}$$

By linearity of expectation, the expected cost after rounding of T_{opt} is

$$S \leq \frac{a-1}{\ln a} cost(T_{opt})$$

Theorem 4. *The approximation ratio of the algorithm given in Figure 4 is*

$$\min_a \left((\alpha - \beta) \frac{a-1}{\ln a} + \beta \frac{a}{\ln a} \right)$$

Proof. Let $Approx_k$ be the tree added when considering rate r_k. Then, by Lemma 2,

$$cost(Approx_k) \leq \alpha a^{y+k} t_k + \beta a^{y+k+1} t_{k+1} + \beta a^{y+k+2} t_{k+2} + \cdots + \beta a^{y+n} t_n$$

[2] Our proof follows the proof of Lemma 4 in [5]

where n is the total number of rates after rounding. Thus, we obtain the following upper bound on the total cost of our approximate solution.

$$
\begin{aligned}
cost(Approx) \le\ & \alpha t_1 a^y + \beta t_2 a^y + \beta t_3 a^y + \cdots + \beta t_{n-1} a^y + \beta t_n a^y \\
& + \alpha t_2 a^{y+1} + \beta t_3 a^{y+1} + \cdots + \beta t_{n-1} a^{y+1} + \beta t_n a^{y+1} \\
& \ddots \\
& + \alpha t_{n-1} a^{y+n-1} + \beta t_n a^{y+n-1} \\
& + \alpha t_n a^{y+n} \\
= (\alpha-\beta) S + \beta \times & \begin{pmatrix}
t_1 a^y + t_2 a^y + t_3 a^y + \cdots + t_{n-1} a^y + t_n a^y \\
+ t_2 a^{y+1} + t_3 a^{y+1} + \cdots + t_{n-1} a^{y+1} + t_n a^{y+1} \\
\ddots \\
+ t_{n-1} a^{y+n-1} + t_n a^{y+n-1} \\
+ t_n a^{y+n}
\end{pmatrix} \\
\le (\alpha-\beta) S + \beta \times & \begin{pmatrix}
\vdots & & & & \\
t_1 a^{y-n+1} & \vdots & & & \\
t_1 a^{y-n+2} & + t_2 a^{y-n+2} & \vdots & & \\
t_1 a^{y-n+3} & + t_2 a^{y-n+3} & + t_3 a^{y-n+3} & & \\
\vdots & \vdots & \vdots & \ddots \vdots & \\
t_1 a^{y-1} & + t_2 a^{y-1} & + t_3 a^{y-1} & + \cdots + t_{n-1} a^{y-1} & \vdots \\
t_1 a^{y} & + t_2 a^{y} & + t_3 a^{y} & + \cdots + t_{n-1} a^{y} & + t_n a^y \\
& + t_2 a^{y+1} & + t_3 a^{y+1} & + \cdots + t_{n-1} a^{y+1} & + t_n a^{y+1} \\
& & \ddots & & \\
& & & + t_{n-1} a^{y+n-1} & + t_n a^{y+n-1} \\
& & & & + t_n a^{y+n}
\end{pmatrix} \\
\le (\alpha-\beta) S + \beta S & \left(1 + \frac{1}{a} + \frac{1}{a^2} + \cdots\right) \\
\le (\alpha-\beta) & \frac{a-1}{\ln a} cost(T_{opt}) + \beta \frac{a}{\ln a} cost(T_{opt})
\end{aligned}
$$

where the last inequality follows from Lemma 3.

Numerically, we obtain approximation ratios of 3.802, 4.059, respectively 4.311, when the α-approximation β-convex Steiner tree algorithm used in Figure 4 is the algorithm in [10], [3], respectively the MST heuristic.

Remark. The algorithm in Figure 4 can be easily derandomized using the same techniques as in [5]

6 Conclusions and Open Problems

In this paper we have considered a generalization of the Steiner problem in which each node possesses a rate and the cost of an edge with length l in a Steiner

tree T connecting the terminals is $l \cdot r_e$, where r_e is the maximum rate in the component of $T - \{e\}$ that does not contain the source. We have given improved approximation algorithms finding trees with a cost at most 1.960 (respectively 3.802) times the minimum cost for the case of two (respectively unbounded number of) non-zero rates. Our improvement is based on the analysis of the gain resulting from the reuse of higher rate edges in the connectivity of the lower rate edges. An interesting open question is to extend this analysis to the case of three non-zero rates. The best known approximation factor for this case, is $\alpha(5 + 4\sqrt{2})/7 \approx 2.358$ [2,13].

References

1. A. Balakrishnan, T.L. Magnanti, P. Mirchandani, *Modeling and Heuristic Worst-Case Performance Analysis of the Two-Level Network Design Problem*, Management Science, **40**: 846–867, (1994)
2. A. Balakrishnan, T.L. Magnanti, P. Mirchandani, *Heuristics, LPs, and Trees on Trees: Network Design Analyses*, Operations Research, **44**: 478–496, (1996)
3. P. Berman and V. Ramaiyer, *Improved Approximations for the Steiner Tree Problem*, Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA 1992), 325–334.
4. A. Borchers and D.Z. Du, *The k-Steiner Ratio in Graphs*, SIAM Journal on Computing, **26**:857–869, (1997)
5. M. Charikar, J. Naor, and B. Schieber, *Resource Optimization in QoS Multicast Routing of Real-Time Multimedia*, Proceedings of the 19th Annual IEEE INFOCOM, (2000)
6. J.R. Current, C.S. Revelle, and J.L.Cohon, *The Hierarchical Network Design Problem*, European Journal of Operations Research, **27**: 57–66, (1986)
7. N. Maxemchuk, *Video Distribution on Multicast Networks*, IEEE Journal on Selected Issues in Communications **15**:357–372 (1997)
8. P. Mirchandani, *The Multi-Tier Tree Problem*, INFORMS Journal on Computing, **8**: 202–218, (1996)
9. K. Mehlhorn, *A faster approximation algorithm for the Steiner problem in graphs*, Information Processing Letters **27**: 125–128, (1988)
10. H. Promel and A. Steger, *A New Approximation Algorithm for the Steiner Tree Problem with Performance Ratio* $\frac{5}{3}$, Journal of Algorithms, **36**: 89–101,(2000)
11. G. Robins and A. Zelikovsky, *Improved Steiner Tree Approximation in Graphs*, Proc. of ACM/SIAM Symposium on Discrete Algorithms (SODA 2000), 770–779.
12. H. Takahashi and A. Matsuyama, *An Approximate Solution for the Steiner Problem in Graphs*, Math. Japonica, **6:** 573–577, (1980)
13. G. Xue, G.-H. Lin, D.-Z. Du, *Grade of Service Steiner Minimum Trees in the Euclidean Plane*, Algorithmica, **31**: 479–500, (2001)
14. A. Zelikovsky, *An 11/6-approximation algorithm for the network Steiner problem*, Algorithmica **9**: 463–470, (1993)
15. A. Zelikovsky, *A faster approximation algorithm for the Steiner tree problem in graphs*, Information Processing Letters **46**: 79–83, (1993)

Chips on Wafers
(Extended Abstract)

Mattias Andersson[1], Joachim Gudmundsson[2⋆], and Christos Levcopoulos[1]

[1] Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden.
mattias@cs.lth.se, christos@cs.lth.se
[2] Department of Mathematics and Computing Science, TU Eindhoven 5600 MB Eindhoven, The Netherlands. h.j.gudmundsson@tue.nl

Abstract. A set of rectangles $\mathcal{S}$ is said to be *grid packed* if there exists a rectangular grid (not necessarily regular) such that every rectangle lies in the grid and there is at most one rectangle of $\mathcal{S}$ in each cell. The area of a grid packing is the area of a minimal bounding box that contains all the rectangles in the grid packing. We present an approximation algorithm that given a set $\mathcal{S}$ of rectangles and a real constant $\varepsilon > 0$ produces a grid packing of $\mathcal{S}$ whose area is at most $(1 + \varepsilon)$ times larger than an optimal packing in polynomial time. If ε is chosen large enough the running time of the algorithm will be linear. We also study several interesting variants, for example the smallest area grid packing containing at least $k \leq n$ rectangles, and given a region $\mathcal{A}$ grid pack as many rectangles as possible within $\mathcal{A}$. Apart from the approximation algorithms we present several hardness results.

1 Introduction

In the VLSI wafer industry it is nowadays possible that multiple projects share a single fabrication matrix (the wafer); this permits fabrication costs to be shared among the participants. No a priori constraints are placed on either the size of the chips nor on the aspect ratio of their side lengths (except the maximum size of the outer bounding box). After fabrication, in order to free the separate chips for delivery to each participant, they must be cut from the wafer. A diamond saw slices the wafer into single chips. However cuts can only be made all the way across the bounding box, i.e., all chips must be placed within a grid. A grid is a pattern of horizontal and vertical lines (not necessarily evenly spaced) forming rectangles in the plane. There are some practical constraints, for example, the distance between two parallel cuts cannot be infinitely small, since machines with a finite resolution must be programmed with each cut pattern. Although some of these constraints may simplify the problem we will not consider them in this paper. This application leads us to define grid packing as follows.

Definition 1. *A set of rectangles $\mathcal{S}$ is said to be* grid packed *if there exists a rectangular grid such that every rectangle lies in the grid and there is at most*

⋆ Supported by The Netherlands Organisation for Scientific Research (NWO).

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 412–423, 2003.
© Springer-Verlag Berlin Heidelberg 2003

one rectangle of $\mathcal{S}$ in each cell, as illustrated in Fig. 1. The area of a grid packing is the area of a minimal bounding box that contains all the rectangles in the grid packing.

There are several cases in the chip design industry where it is favorable to produce a good grid packing. Some practitioners have therefore asked for algorithms solving such problems [6]. In practice one is sometimes given a set of chips and one wants to minimize the number of wafers used. A naive simulated-annealing rectangle placement [6] with n rectangles might be packed on n separate wafers, as a worst case. This means, fabricating n wafers, wasting all except one chip on each wafer. So the number of recovered chips per cut pattern should be maximized. The general problem considered in this paper is defined as follows. Note that the input rectangles are allowed to be rotated.

Problem 1. [**Minimum area grid packing (MAGP)**] Given a set $\mathcal{S}$ of rectangles find a grid packing of $\mathcal{S}$ that minimizes the total area of the grid.

We also consider several interesting variants of the problem, for example:

Problem 2. [**Minimum area k-grid packing (MAkGP)**] Given a set of n rectangles and an integer $k \leq n$ compute a minimum area grid packing containing at least k rectangles.

Problem 3. [**Minimum area grid packing with bounded aspect ratio (MAGPAR)**] Given a set $\mathcal{S}$ of rectangles and a real number $\mathcal{R}$, compute a minimum area grid packing whose bounding box aspect ratio is at most $\mathcal{R}$.

Problem 4. [**Maximum wafer packing (MWP)**] Given a set of rectangles $\mathcal{S}$ and a rectangular region $\mathcal{A}$ compute a grid packing of $\mathcal{S}' \subseteq \mathcal{S}$ on $\mathcal{A}$ of maximal size.

Problem 5. [**Minimum number of wafers (MNWP)**] A plate is a prespecified rectangular region. Given a set of rectangles $\mathcal{S}$ compute a grid packing of $\mathcal{S}$ onto a minimal number of plates.

A problem that is similar to grid packing is the tabular formatting problem. In the most basic tabular formatting problem one is given a set of rectangular entries and the aim is to construct a table containing the entries in a given order for each row and column. Shin *et al.* [8] suggested three different objective functions that evaluate the quality of a tabular layout: minimal diameter, minimal area, and minimal white space. Therefore we also consider the following problem:

Problem 6. [**Minimum diameter grid packing (MDGP)**] Given a set $\mathcal{S}$ of rectangles find a grid packing of $\mathcal{S}$ that minimizes the diameter of the grid.

A problem more similar to ours, with the exception that the rectangles cannot be rotated, was considered by Beach [3] under the name "Random Pack". Beach showed that Random Pack is strongly NP-hard, and so it partly corresponds to our NP-hardness result in Theorem 4.

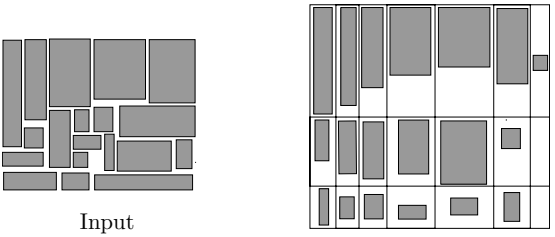


Fig. 1. Input is a set of rectangles $\mathcal{S}$. Output a grid packing of $\mathcal{S}$

Weighted variants of these problems are also studied. In the weighted case each rectangle $r \in \mathcal{S}$ also has a weight, denoted $w(r)$, associated to it.

Our main result is a polynomial time $(1+\varepsilon)$-approximation scheme (PTAS) for the MAGP-problem and some of its variants (Problems 2,3 and 6). For Problems 4 and 5 we present polynomial time approximation algorithms with approximation factor $(1-\varepsilon)$ and $\lfloor 2(1+\varepsilon)\rfloor^2$ respectively. Surprisingly, if the value of ε is a large enough constant the approximation algorithms will run in linear time.

The approximation algorithms all build upon the same ideas. The main idea is that for every possible grid $\mathcal{G}$ there exists a grid $\mathcal{G}'$ that can be uniquely coded using only $\mathcal{O}(\log n)$ bits such that the area of $\mathcal{G}'$ is at most a factor $(1+\varepsilon)$ larger than the area of $\mathcal{G}$. Now, let $\mathcal{F}$ be the family of these grids that can be uniquely coded using $\mathcal{O}(\log n)$ bits. It trivially follows that there are only a polynomial number of grids in $\mathcal{F}$. Hence, every grid $\mathcal{G}'$ in $\mathcal{F}$ can be generated and tested. The test is performed by computing a maximal packing of $\mathcal{S}$ into $\mathcal{G}'$, which in turn is done by transforming the problem into an instance for the max-flow problem, i.e, given a directed graph with a capacity function for each edge, find the maximum flow through the graph. To obtain a PTAS one uses $\mathcal{O}(\log_{1+\varepsilon} n)$ bits for coding a grid in $\mathcal{F}$. In a similar way only $\log n/2$ bits are used to obtain a linear time approximation algorithm, however the approximation factor will be quite large in this case. More details about the family of grids, called the family of (α, β, γ)-*restricted* grids are given in Section 3 together with two important properties. Then, in Section 4 we show how a grid is tested by reducing the problem to a min-cost max-flow problem. The main theorem is also stated in this section. In Section 5 we consider variants of the MAGP-problem, and finally, in Section 6 we show some hardness results.

1.1 Approximability Preserving Simplifications

We will assume that width, height and weight of each rectangle $r \in \mathcal{S}$ is between $[1, n^c]$, for some constant c. We argue that this assumption can be made without loss of generality with respect to our approximation results. The approximation algorithm that is to be presented in the next section, produces a grid on an area of size a constant factor times the area of an optimal solution. Let n^c denote

the length of the longest side of any rectangle in $\mathcal{S}$. For simplicity we scale the weights such that the rectangle with the largest weight has weight n^c. Since the area is approximated we may assume that the shortest side of any input rectangle has length at least 1. If not, the side is expanded such that it has length 1. Note that the size of any solution will expand by at most n which is a factor $(1 + n^{1-c})$ larger than an optimal solution and therefore negligible for the approximation algorithms we consider in this paper. Hence, we may assume that the sides of the input rectangles have length within the interval $[1, n^c]$ (we could also add the absolute error $1/n^{c-1}$ to the approximation factor).

Similarly, we may assume that all rectangles have weight in the interval $[1, n^c]$. Let $\max_{r \in \mathcal{S}} w(r) = n^c$; it holds that all rectangles with weight less than 1 can be discarded since their combined weight sums up to at most n, which is at most n^{c-1} of the weight of an optimal solution and hence negligible.

2 The Approximation Algorithm

The approximation algorithm is simple and straight-forward, therefore we here give the global structure. The two non-trivial steps, lines 10 an 11, will be described in detail in Sections 3 and 4 respectively. The last step, PackIntoGrid, is obtained by slightly modifying the procedure TestGrid. As input to the algorithm we will be given a set $\mathcal{S}$ of n rectangles and a real value $\varepsilon' > 0$.

Algorithm GridPack($\mathcal{S}, \varepsilon'$)

1. $bestVal \leftarrow \infty$, $\alpha = \beta = \sqrt{1+\varepsilon'}$, $\gamma = \frac{1}{\sqrt{1+\varepsilon'}-1}$
2. **for** each $1 \leq i, j \leq \log_\alpha n^c$ **do**
3. $\quad \mathcal{S}_{i,j} \leftarrow \emptyset$
4. **for** each $r \in \mathcal{S}$ **do**
5. $\quad i \leftarrow \lceil \log_\alpha \text{ width}(r) \rceil$
6. $\quad j \leftarrow \lceil \log_\alpha \text{ height}(r) \rceil$
7. $\quad \mathcal{S}_{i,j} \leftarrow \mathcal{S}_{i,j} \cup \{r\}$
8. **end**
9. **for** $k \leftarrow 1$ **to** $n^{2cf(\alpha,\beta,\gamma)}$ **do**
10. $\quad \mathcal{G} \leftarrow$ GenerateGrid$(\alpha, \beta, \gamma, k)$
11. $\quad val \leftarrow$ TestGrid$(\mathcal{G}, \mathcal{S}, \alpha, \beta, \gamma)$
12. $\quad$ **if** $val < bestVal$ **then**
13. $\quad\quad bestVal \leftarrow val$ and $bestGrid \leftarrow \mathcal{G}$
14. **end**
15. **Output** PackIntoGrid$(\mathcal{S}, bestGrid)$

The initialization is performed on lines 1 to 3. On lines 4-8, the rectangles are partitioned into groups in such a way that a rectangle $r \in \mathcal{S}$ belongs to $\mathcal{S}_{i,j}$ if and only if the width of r is between α^{i-1} and α^i, and the height of r is between α^{j-1} and α^j. Lines 1-8 obviously run in linear time. Next, a sequence of grids $\mathcal{G}$ are produced in a loop of lines 9-14. They are the members of the so-called

family of (α, β, γ)-restricted grids which is described in Section 3. (It consists of $n^{(2c \cdot f(\alpha,\beta,\gamma))}$ grids, where $f(\alpha, \beta, \gamma) = (\log(\alpha\beta\gamma))/(\log\alpha \log\beta)$.)

The generated grid is tested and the weight of an approximative grid packing of $\mathcal{S}$ into the grid $\mathcal{G}$ is computed. If the grid packing is better than the previously tested grids then $\mathcal{G}$ is saved as the best grid tested so far. Finally, when all grids in the family of (α, β, γ)-restricted grids have been generated and tested a call to PackIntoGrid performs a grid packing of $\mathcal{S}$ into the best grid found. This procedure is a simple modification of the TestGrid-step and it is briefly described in Theorem 1 and Lemma 2.

3 The Family of (α, β, γ)-Restricted Grids

The aim of this section is to define the family $\mathcal{F}$ of (α, β, γ)-restricted grids and prove two properties about $\mathcal{F}$. Before the properties can be stated we need the following definition. A grid G_1 is said to *include* a grid G_2 if every possible set of rectangles that can be grid packed into G_2 also can be grid packed into G_1. $\mathcal{F}$ has the following two properties.

1. For every grid G (with at most n rows and n columns) there exists a grid $\mathcal{G} \in \mathcal{F}$ that includes G and whose width and height is at most a factor $\left(\frac{\alpha^2\beta\gamma}{\alpha\gamma-1}\right)$ times larger than the width and height of G, and
2. $\#\mathcal{F} \leq n^{(2c \cdot f(\alpha,\beta,\gamma))}$ where $f(\alpha, \beta, \gamma) = \frac{\log(\alpha\beta\gamma)}{\log\alpha \log\beta}$.

The definition of a (α, β, γ)-restricted grid is somewhat complicated, therefore we choose to describe this step by step.

A trivial observation is that two grid-packings are equivalent if the one can be transformed to the other by exchanging the order of rows and/or the columns. Hence we may assume that the columns are ordered with respect to decreasing width from left to right and that the rows are ordered with respect to decreasing height from top to bottom. This ordering will be assumed throughout the paper. The proofs of the observations and corollaries in this section can be found in the full version.

3.1 Size of a (α, β, γ)-Restricted Grid

Consider an arbitrary grid G and let α be a real constant greater than 1. An *α-restricted* grid is a grid where the width and height of each cell in the grid is an integral power of α (multiple of α^i for some integer i).

Observation 1 *For any grid G there exists an α-restricted grid $\mathcal{G}$ that includes G and whose width and height is at most a factor α longer than the width and height of G.*

Let G be a α-restricted grid. If the number of columns/rows of each size is an integral power of β then G is a (α, β)-restricted grid.

Observation 2 *For any α-restricted grid G there exists a (α, β)-restricted grid $\mathcal{G}$ that includes G and whose width and height is at most a factor β greater than the width and height of G.*

The columns/rows in an α-restricted grid of width/height α^i are said to have column/row *size* i. A grid G is said to be γ-*monotone* if the number of columns (rows) of size i is greater than $1/\gamma > 0$ times the number of columns (rows) of size $i+1$ for every i.

Observation 3 *Let γ be a constant greater than or equal to 1 such that $\gamma = \beta^k$ for some integer $k \geq 0$. For any (α, β)-restricted grid G there exists a γ-monotone (α, β)-restricted grid $\mathcal{G}$ ((α, β, γ)-restricted grid for short) that includes G and whose width and height is at most a factor $(\gamma\alpha)/(\gamma\alpha - 1)$ greater than the width and height of G.*

Putting together the observations immediately gives the following corollary and, hence, $\mathcal{F}$ is shown to have Property 1.

Corollary 1. *For any grid G there exists a (α, β, γ)-restricted grid $\mathcal{G}$ that includes G and whose width and height is at most a factor $\left(\frac{\alpha^2\beta\gamma}{\alpha\gamma - 1}\right)$ greater than the width and height of G.*

Most often we do not need the actual grid, instead we are interested in the number of cells in the grid of a certain size. That is, the grid $\mathcal{G}$ is represented by a $[1..\log_\alpha n^c, 1..\log_\alpha n^c]$ integer matrix, where $\mathcal{G}[i,j]$ stores the number of cells in $\mathcal{G}$ of width α^i and, height α^j. We call this a matrix representation of a grid.

3.2 Size of the Family of (α, β, γ)-Restricted Grids

In this section it will be shown that the second property holds for the family $\mathcal{F}$ of (α, β, γ)-restricted grids, i.e., the number of grids that are members of $\mathcal{F}$ is at most $n^{(2c \cdot f(\alpha,\beta,\gamma))}$. It will be a constructive proof where it will be shown that every grid in $\mathcal{F}$ can be coded uniquely with $2(\log_\alpha n^c(1 + \log_\beta \gamma) + \log_\beta n^c)$ bits. Hence, producing all words of length $2(\log_\alpha n^c(1 + \log_\beta \gamma) + \log_\beta n^c)$ will also generate all members of $\mathcal{F}$.

Assume that we are given a member $f \in \mathcal{F}$ and that f has c_i columns of size i, $1 \leq i \leq \log_\alpha n^c$, and r_j rows of size j, $1 \leq j \leq \log_\alpha n^c$. Recall that r_j and c_i are integral powers of β.

The idea of the scheme is as follows, see also algorithm CodingRows below. The bit string, denoted S, is built incrementally. Consider a generic step of the algorithm. Assume that the bit string, denoted S_{j+1} has been built for all the row sizes greater than j and that the number of rows of size $(j+1)$ is $\beta^{(\#Rows)}$. Initially $S_{\log_\alpha n^c}$ is the empty string and $\#Rows = 0$. Consider the row size j. We will have two cases, either $r_j \leq (\beta^{\#Rows}/\gamma)$ or $r_j > (\beta^{\#Rows}/\gamma)$. In the first case, add '1' to S_{j+1} to obtain S_j. In the latter case, when $r_j > (\beta^{\#Rows}/\gamma)$, add $(\#Rows - (\log_\beta r_j - \log_\beta \gamma))$ zeros followed by a '1' to S_{j+1} to obtain S_j. Decrease the value of j and continue the process until $j = 0$, and hence, $S_0 = S$.

The algorithm above describes how to generate the rows, the coding for the columns is done in the same way. The following observation proves that property 2 holds for the family of (α, β, γ)-restricted grids. The proof is omitted in this extended abstract.

Observation 4 *The length of S is* $2(\log_\alpha n^c(1 + \log_\beta \gamma) + \log_\beta n)$.

3.3 GenerateGrid

On line 10 in algorithm GridPack the procedure GenerateGrid is called with the parameters α, β, γ and k, where k is a positive integer $\leq n^{(2c \cdot f(\alpha,\beta,\gamma))}$ and hence its binary representation, denoted $B(k)$, has length $2(\log_\alpha n^c(1+\log_\beta \gamma)+ \log_\beta n)$. Note that for simplicity we chose to generate all bit strings of length at most $2(\log_\alpha n^c(1 + \log_\beta \gamma) + \log_\beta n)$ and then decide in the procedure GenerateGrid if it corresponds to a (α, β, γ)-restricted grid or not. Alternatively, one could generate only valid bit strings. We obtain the following corollary:

Corollary 2. *Given a bit string B of length* $2(\log_\alpha n^c(1 + \log_\beta \gamma) + \log_\beta n)$ *one can in time $\mathcal{O}(\log^2 n)$ construct the unique matrix representation of the corresponding (α, β, γ)-restricted grid, or decide that there is no corresponding (α, β, γ)-restricted grid.*

4 Testing a (α, β, γ)-Restricted Grid

In the previous section we showed a simple method to generate all possible (α, β, γ)-restricted grids. For the approximation algorithm, shown in Section 2, to be efficient we need a way to pack a maximal number of rectangles of $\mathcal{S}$ into the grid. As input we are given a matrix representation of a (α, β, γ)-restricted grid $\mathcal{G}$, and a set $\mathcal{S}$ of n rectangles partitioned into groups $\mathcal{S}_{i,j}$ depending on their width and height. Let $\mathcal{C}_{p,q}$ denote the number of cells in $\mathcal{G}$ that have width α^p and height α^q. We will give an exact algorithm for the problem by reformulating it as a max-flow problem. The problem could also be solved by reformulating it as a matching problem but in the next section we will show that the max-flow formulation can be extended to the weighted case. The max-flow problem is as follows. Given a directed graph $G(V, E)$ with capacity function $u(e)$ for each edge e in E. Find the maximum flow f through G.

The flow network $\mathcal{F}_{\mathcal{S},\mathcal{G}}$ corresponding to a grid $\mathcal{G}$ and a set of rectangles $\mathcal{S}$ contains four levels, numbered from top-to-bottom, and will be constructed level-by-level. The number of nodes on level i, $1 \leq i \leq 4$, is denoted m_i.

Level 1. At the top level there is one node, the source node $\nu^{(1)}$.

Level 2. At level 2 there are $\log_\alpha^2 n^c$ nodes. A node $\nu_{i,j}^{(2)}$ at level 2 represents the group $\mathcal{S}_{i,j}$. For each node $\nu_{i,j}^{(2)}$, there is a directed edge from $\nu^{(1)}$ to $\nu_{i,j}^{(2)}$. The capacity of this edge is equal to the number of rectangles in $\mathcal{S}$ that belongs to $\mathcal{S}_{i,j}$.

Level 3. At level 3 there are also $\log_\alpha^2 n^c$ nodes. A node $\nu_{p,q}^{(3)}$ on level 3 represents the set of cells in $\mathcal{C}_{p,q}$. For each node $\nu_{p,q}^{(3)}$ there is a directed edge from node $\nu_{i,j}^{(2)}$ to node $\nu_{p,q}^{(3)}$ if and only if $p \geq i$ and $q \geq j$ (or $q \geq i$ and $p \geq j$), i.e., if a rectangle in $\mathcal{S}_{i,j}$ can be packed into a cell in $C_{p,q}$. All the edges from level 2 to level 3 have capacity n.

Level 4. The bottom level only contains one node, the sink $\nu^{(4)}$. For every node $\nu_{p,q}^{(3)}$ on level 3 there is a directed edge from $\nu_{p,q}^{(3)}$ to $\nu^{(4)}$. The capacity of this edge is equal to the number of cells in $\mathcal{G}$ that belongs to $\mathcal{C}_{p,q}$.

The following observations are straight-forward.

Observation 5 *Every optimal packing of rectangles into a grid G is maximal w.r.t. the number of rectangles.*

Proof. Let P' be a maximal packing and let P be the optimal packing that is closest to P'. If P does not equal P' then there must exist a cell s that is empty in P but not in P'. Denote by r the rectangle in cell s of P'. We now have two simple cases, either r is in P' or not. If r is included in P then we could move r to s and we would get an optimal packing that is closer to P', hence a contradiction. If r is not in P then P cannot be an optimal solution since r can be added to P and a better solution is obtained.

Observation 6 *The maximal grid packing of $\mathcal{S}$ into $\mathcal{G}$ has size k if and only if the max flow in the flow network is k.*

Proof. Every unit flow corresponds to a rectangle and since the flow is k it implies that every rectangle in $\mathcal{S}$ has been matched to a cell. It is easily seen that a rectangle can only be matched to a cell it can fit into, hence if the flow is k there exists a grid packing of size k of $\mathcal{S}$ into $\mathcal{G}$.

The 'only if' statement is obvious since if there exists a grid packing there must exist a k-matching between the rectangles and the cells, and if this is true there must be a maximal flow of value k.

In 1998 Goldberg and Rao [5] presented an algorithm for the maximum flow problem with running time $\mathcal{O}(N^{2/3}M \log(N^2/M) \log U)$. If we apply their algorithm to the flow network we obtain the following lemma.

Lemma 1. *Given a matrix representation of an (α, β, γ)-restricted grid $\mathcal{G}$ and a set $\mathcal{S}$ of n rectangles partitioned into the groups $\mathcal{S}_{i,j}$ w.r.t. their width and height. (1) The size of an optimal packing of $\mathcal{S}$ in $\mathcal{G}$ can be computed in time $\mathcal{O}(\log^{19/3} n)$. (2) An optimal packing can be computed in time $\mathcal{O}(\log^{19/3} n + k)$, where k is the number of rectangles in the optimal grid packing of $\mathcal{S}$ in $\mathcal{G}$.*

As a result we obtain the first grid packing theorem.

Theorem 1. *Algorithm* GridPack *is a $(1+\varepsilon)$-approximation algorithm for the MAGP-problem with running time $\mathcal{O}(n^{f(\sqrt{1+\varepsilon})} \log^{19/3} n + n)$, where $f(\chi) = \frac{2c \log(\chi/(\sqrt{\chi}-1))}{\log^2 \chi}$).*

Proof. Corollary 1 guarantees the approximation ratio. The time-complexity of the approximation algorithm is dominated by testing if the set of rectangles can be grid packed into a grid $\mathcal{G}$. The total time for this step is the number of tested grids times the time to test one grid, hence, $\mathcal{O}(n^{f(\chi)} \log^{19/3} n + n)$.

The final packing is easily computed from the max-flow and hence can be done in time linear with respect to the number of rectangles in the packing.

Even though the expression for the running time in the above theorem looks somewhat complicated it is not hard to see that by choosing the value of ε appropriately we obtain that, algorithm GridPack is a PTAS for the MAGP-problem, and if ε is set to be a large constant GridPack produces a grid packing that is within a constant factor of the optimal in linear time.

5 Applications and Extensions

The approximation algorithm presented above can be extended and generalized to variants of the basic grid packing problem. Below we show how to extend the algorithm to these variants.

By performing some small modifications to the procedure TestGrid we will obtain the following corollary from Theorem 1. Let $f(\chi) = \frac{2c \log(\chi/(\sqrt{\chi}-1))}{\log^2 \chi})$.

Corollary 3. *Algorithm* GridPack *is a t-approximation algorithm with time complexity $\mathcal{O}(n^{f(\sqrt{1+\varepsilon})} \log^{19/3} n + n)$, where:*

- $t = (1+\varepsilon)$ *for the problems MAkGP, MAGPAR and MDGP,*
- $t = (1-\varepsilon)$ *for the MWP-problem, and*
- $t = ((\lfloor 2(1+\varepsilon) \rfloor)^2)$ *for the MNWP-problem.*

5.1 Extending the Test Algorithm

The problems considered below are weighted variants of the grid packing problem, hence every rectangle $r \in \mathcal{S}$ also has a weight $w(r)$. To be able to apply the above algorithm to the weighted case we need a way to pack a maximal number (weight) of rectangles in $\mathcal{S}$ into the grid, or at least approximate the weight of a maximal packing. As input we are given a matrix representation of a (α, β, γ)-restricted grid $\mathcal{G}$, and a set $\mathcal{S}$ of n rectangles partitioned into groups $\mathcal{S}_{i,j}$ depending on their width and height. We will give a τ-approximation algorithm for the problem by reformulating it as a min-cost max-flow problem, where $\tau > 1$ is a given parameter. Hence the call to TestGrid in algorithm GridPack will now include a fifth parameter τ.

A min-cost max-flow problem in a directed graph where each edge e has a capacity $c(e)$ and a cost $d(e)$ is to find the maximum flow f with a minimum cost, where the cost of a flow f is $\sum_{e \in E} d(e) \cdot f(e)$.

The network that will be constructed next contains four levels, numbered from top-to-bottom, and will be constructed level-by-level. A rectangle r in $\mathcal{S}$ belongs to weight class W_k if and only if $w(r)$ is between τ^{k-1} and τ^k. The number of nodes on level i, $1 \leq i \leq 5$, is denoted m_i and we set $W = \sum_{r \in \mathcal{S}} w(r)$.

Level 1. At the top level there is one node, the source node $\nu^{(1)}$.

Level 2. At level 2 there are $\log_\alpha^2 n \cdot \log_\tau n$ nodes. A node $\nu^{(2)}_{i,j,k}$ at level 2 represents the class of rectangles in the group $\mathcal{S}_{i,j}$ and weight class W_k. For each node $\nu^{(2)}_{i,j,k}$, there is a directed edge from $\nu^{(1)}$ to $\nu^{(2)}$. The capacity of this edge is equal to the number of rectangles in $\mathcal{S}$ that belongs to $\mathcal{S}_{i,j}$ and weight class W_k. The cost per unit flow is $W - \tau^{k-1}$, i.e., a rectangle with greater weight is "cheaper" to send than a rectangle with low weight.

Level 3. At level 3 there are $\log_\alpha^2 n^c$ nodes, one for each set $\mathcal{C}_{p,q}$ of cells. Hence, a node $\nu^{(3)}_{p,q}$ on level 3 represents the set of cells in $\mathcal{C}_{p,q}$. For each node $\nu^{(3)}_{p,q}$ there is a directed edge from node $\nu^{(3)}_{k,i,j}$ to node $\nu^{(3)}_{p,q}$ if and only if $p \geq i$ and $q \geq j$, i.e., the rectangles in $\mathcal{S}_{i,j}$ can be packed into the cells in $C_{p,q}$. All the edges from level 2 to level 3 have capacity n and zero cost.

Level 4. The bottom level only contains one node, the sink $\nu^{(4)}$. For every node $\nu^{(3)}_{p,q}$ on level 3 there is a directed edge from $\nu^{(3)}_{p,q}$ to $\nu^{(4)}$ with cost 0 and capacity $|\mathcal{C}_{p,q}|$.

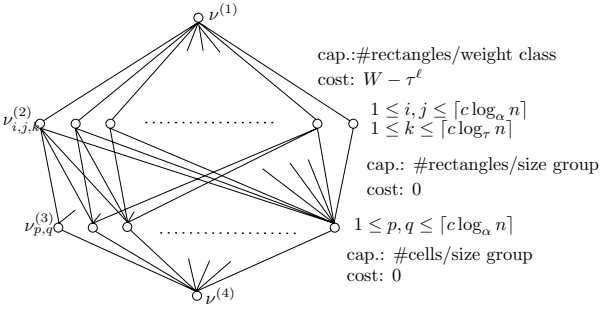


Fig. 2. Illustrating the flow network given as input to the min-cost max-flow algorithm.

We need the following observation before we can state the main lemma.

Observation 7 *If the min-cost max-flow of $\mathcal{F}_{\mathcal{S},\mathcal{G}}$ is $(W - w)$ then the maximal grid packing of $\mathcal{S}$ into $\mathcal{G}$ has weight at most τw.*

Proof. The min-cost max-flow of cost $(W - w)$ is equivalent to a max cost max flow of cost w. It is not hard to see that if there exists a max flow of max cost w then there exists a maximum matching between the rectangles in $\mathcal{S}$ and the cells of $\mathcal{G}$ of total weight w. Since the weight of each rectangle is rounded down such that its weight is divisible by τ it holds that the weight of a maximal packing is at most τw.

We summarize this section by stating the following lemma.

Lemma 2. *Given a matrix representation of an (α, β, γ)-restricted grid $\mathcal{G}$ and a set $\mathcal{S}$ of n rectangles partitioned into the groups $S_{i,j}$ w.r.t. their width and height. (1) The cost of a τ-approximate packing of $\mathcal{S}$ onto G can be computed in time $\mathcal{O}(\log^9 n)$. (2) A τ-approximate packing can be computed in time $\mathcal{O}(\log^9 n + k)$, where k is the number of rectangles in the grid packing.*

Proof. According to Observation 7 the solution to the min-cost max-flow problem on $\mathcal{F}_{\mathcal{S},\mathcal{G}}$ will give an approximate solution to the maximal grid packing problem of $\mathcal{S}$ into $\mathcal{G}$ that is at most a factor τ of the optimal.

Let N be the number of vertices in the network; M, the number of edges; U, the maximum value of an edge capacity; and C, the maximum value of an edge cost. The values of the parameters in the above transformed instance can be bounded as follows: $N = \mathcal{O}(\log_\alpha^2 n \cdot \log_\gamma n)$, $M = \mathcal{O}(\log_\alpha^4 n \cdot \log_\gamma n)$, $C \leq n$ and, finally, $U_f = \mathcal{O}(n^{c+1})$. Plugging in the values in the algorithm by Ahuja *et al.* [1] gives that the max-flow min-cost problem can be solved in time $\mathcal{O}(\log^9 n)$.

Finally, if we are not satisfied with an estimated cost but actually want a packing then we just pair up the rectangles with the matching cells. This is trivially done in time linear with respect to the number of rectangles in the packing.

5.2 Weighted Variants

In this section we consider two weighted variants of the grid packing problem, i.e., every rectangle $r \in \mathcal{S}$ also has a weight $w(r)$. Both results follows from Corollaries 1-2 and Lemma 2. Recall that $f(\chi) = \frac{2c \log(\chi/(\sqrt{\chi}-1))}{\log^2 \chi}$.

Problem 7. [**Minimum area weighted grid packing(MAwGP)**] Given a set of n rectangles and a real number w compute a minimum area grid packing containing a set of rectangles of $\mathcal{S}$ of total weight at least w.

Theorem 2. *Given a set $\mathcal{S}$ of n rectangles and a real value $\varepsilon > 0$, one can compute a grid packing containing rectangles of total weight at least w whose area is at most $(1+\varepsilon)$ greater than the area of a minimum area (τw)-grid packing in time $\mathcal{O}(n^{f(\sqrt{1+\varepsilon})} \cdot \log^9 n + n)$.*

Problem 8. [**Maximum weight wafer packing (MwWP)**] Given a set of rectangles $\mathcal{S}$ and a rectangular region $\mathcal{A}$ compute a grid packing of $\mathcal{S}' \subseteq \mathcal{S}$ onto $\mathcal{A}$ of maximum weight.

Let $\kappa(\mathcal{S}, \mathcal{A})$ denote the maximum weight grid packing of $\mathcal{S}$ onto $\mathcal{A}$. Let $\mathcal{A}'$ be a rectangular region such that the sides of $\mathcal{A}$ is $(1 + \varepsilon)$ longer than the sides of $\mathcal{A}'$.

Theorem 3. *Given a set $\mathcal{S}$ of n rectangles, a rectangular region $\mathcal{A}$ and a real value $\varepsilon > 0$, one can compute a grid packing of $\mathcal{S}$ onto $\mathcal{A}$ whose weight is at least $\kappa(\mathcal{S}, \mathcal{A}')/\tau$ in time $\mathcal{O}(n^{f(1+\varepsilon)} \cdot \log^9 n + n)$.*

The approximation algorithm can be generalized to d dimensions, such that the resulting grid packing has sides that are at most $(1+\varepsilon)$ times longer than the length of the sides of an optimal grid packing. The running time is $\mathcal{O}(dn \log n)$ for sorting the d-dimensional rectangles into bins. The size of the coding $\mathcal{O}(d \log n)$, which implies that the number of grids is $n^{dc \cdot f(\alpha,\beta,\gamma)}$. The flow network will have $\mathcal{O}(\log^{d+1} n)$ nodes and $\mathcal{O}(\log^{2d+1} n)$ arcs, which implies that the running time of the max-flow algorithm will be $\mathcal{O}(\log^{3d+3} n)$.

6 Hardness Results

In this section we show that several of the problems considered are NP-hard. Due to lack of space we will in this extended abstract only state the results without any proofs. We start with an interesting observation:

Observation 8 *There exists a set $\mathcal{S}$ such that every optimal grid packing of $\mathcal{S}$ on a wafer (rectangular region) $\mathcal{A}$ induces a grid with $\Omega(n^2)$ cells.*

Next we consider the hardness of several variants of the MAGP-problem.

Theorem 4. *(1) The MWP-problem is NP-hard, (2) the MAGPAR-problem is NP-hard, and (3) the MNWP-problem cannot be approximated within a factor of $3/2 - \varepsilon$ for any $\varepsilon > 0$, unless $P = NP$.*

Acknowledgements. We thank Esther and Glenn Jennings for introducing us to the problem. We would also like to thank Bernd Gärtner for pointing us to "The table layout problem" [2,3,8,9].

References

1. R. K. Ahuja, A. V. Goldberg, J. B. Orlin and R. E. Tarjan. Finding minimum-cost flows by double scaling. Tech. rep. CS-TR-164-88. Department of computer science. Princeton University, Princeton, NJ,. 1988.
2. R. J. Anderson and S. Sobti. The Table Layout Problem. In proceedings of Symposium on Computational Geometry, pp. 115–123, 1999.
3. R.J. Beach. Setting tables and illustrations with style. PhD thesis, Dept. of computer science, University of Waterloo, Canada, 1985.
4. M. R. Garey and D. S. Johnson. Computers and Intractability: A guide to the theory of NP-completeness, W. H. Freeman and Company, San Francisco, 1979.
5. A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. Journal of the ACM, 45(5):783–797, 1998.
6. G. Jennings. Personal communication, 2002.
7. C. H. Papadimitriou and K. Steiglitz. Combinatorial Optimization; Algorithms and Complexity. Dover Publications, Mineola, USA, 1998
8. K. Shin, K. Kobayashi and A. Suzuki. TAFEL MUSIK, formatting algorithm of tables. In proceedings of Principles of Document Processing, 1994.
9. X. Wang and D. Wood. Tabular formatting problems. In proceedings of Principles of Document Processing, 1996.

A Model for Analyzing Black-Box Optimization

Vinhthuy Phan[1], Steven Skiena[2⋆], and Pavel Sumazin[3]

[1] Computer Science, SUNY at Stony Brook, Stony Brook, NY 11794, USA, phan@cs.sunysb.edu
[2] Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794, USA, skiena@cs.sunysb.edu.
[3] Computer Science, Portland State University, P.O. Box 751, Portland, Oregon 97207, ps@cs.pdx.edu.

1 Introduction

The design of heuristics for NP-hard problems is perhaps the most active area of research in the theory of combinatorial algorithms. However, practitioners more often resort to local-improvement heuristics such as gradient-descent search, simulated annealing, tabu search, or genetic algorithms. Properly implemented, local-improvement heuristics can lead to short, efficient programs that yield reasonable solutions. Designers of efficient local-improvement heuristics must make several crucial decisions, including the choice of neighborhood and heuristic for the problem at hand. We are interested in developing a general methodology for predicting the quality of local-neighborhood operators and heuristics, given a time budget and a solution evaluation function.

In this paper, we introduce a neighborhood-dependent model for the landscape of combinatorial optimization problems, and analyze the behavior and expected performance of black-box search heuristics on this model. Our model is simple enough to reason formally about, yet it is both intuitive and supported by experimental evidence. *Black-box* local-search heuristics are problem-independent and require only (1) a local-neighborhood operator that takes an arbitrary element s of the solution space S and generates nearby elements of S, and (2) a cost function which evaluates the quality of a given element s.

We model the solution space S associated with a particular minimization problem instance as a random directed graph $G = (V, E)$. We measure solution quality in ordinal terms, namely we rank the solutions from 0 to $N-1$ in order of increasing cost, breaking ties arbitrarily, so that $\text{rank}(i) < \text{rank}(j)$ implies $\text{cost}(i) \leq \text{cost}(j)$. Each of the N vertices in V represents a solution $s \in S$, and we say that vertex v_i corresponds to the solution of rank i. Each vertex v has an out-degree of k, with edges drawn from v to every vertex representing a neighbor of s. We analyze the case where neighbors are chosen uniformly at random from a window of up to $2c$ vertices, centered at v and consisting of the vertices ranked immediately higher and lower than v. The goal of our search, starting from vertex v_{N-1}, is to reach the lowest ranked vertex possible.

⋆ Supported in part by NSF Grant CCR-9625669 and ONR Award N00149710589.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 424–438, 2003.
© Springer-Verlag Berlin Heidelberg 2003

In summary, problem instances are parameterized by their solution space size N, local neighborhood size k, and rank edge span c. Although this model is simple, it is sufficiently rich to study such questions as the impact of neighborhood size and structure on search performance. Indeed, we [16] are building a general combinatorial optimization engine based on local search, and developed the theory in this paper to guide our implementation.

This model has proved to be very helpful in enabling us to reason about the relevant search algorithms in a rigorous way. For example, analysis of the model suggests that the advantage of gradient descent heuristics lies in their ability to isolate a subset of high-scoring solutions and to search within that subset. There is a cost for finding this subset, and repeated application of hill-climbing from different initial solutions multiplies this overhead. Thus certain forms of backtracking which avoid this penalty are preferred under our theory.

In this paper, we analyze our model to establish the following results:

- Local search cannot perform better than random sampling for unrestricted random local neighborhood operators (i.e. $c = N$), as suggested by Wolpert and Macready's [20] *no free lunch theorem*. The no-free-lunch theorem states that any heuristic that (1) does not extract information about the cost function, and (2) evaluates new solutions at the same rate is expected to perform no better than random sampling. However, we show that local search outperforms random sampling if the neighborhood has some structure, i.e. $c < N$.
- We analyze four different gradual-descent heuristics, known in the literature as hill-climbing or as the standard-algorithm variants. These include greedy hill climbing (GHC), random hill climbing (RHC), reverse greedy hill climbing (RGHC) and combined hill climbing (CHC). See Section 4 for definitions. Our results, summarized in the table below, give the expected number of search operations and solution ranks for both small neighborhoods ($k < \lg(N/c)$) and large neighborhoods ($k > \lg(kN/c)$). Note that ϵ_1, ϵ_2, and ϵ_3 are all small constants.

	$k < \lg N/c$		$k \geq \lg kN/c$	
Algorithm	Operations	Expected Rank	Operations	Expected Rank
RHC	2^{k+1}	$N - c2^{k-1}$	$2(N-c)/c + ek$	$c/(ek + \epsilon_2)$
GHC	$k2^k$	$N - c2^k(k+2)/k - c/k$	$k(N-c)/c + ek$	$c/(ek + \epsilon_1)$
RGHC	$k2^k$	$N - c(2^{k+1} - 1)/k$	$k^2(N+c)/2c + ek^2/2$	$2c/(k + \epsilon_3)$
CHC	2^{k+1}	$N - c2^{k-1}$	$2(N-c)/c + ek$	$c/(ek + \epsilon_1)$

- We analyze five different backtracking heuristics including greedy backtracking (GBT), reverse greedy backtracking (RGBT), random backtracking (RBT) and a combined hill climbing and backtracking heuristic (CBT). See Section 4 for definitions. These heuristics backtrack to higher ranked neighbors when they reach a sink. We compare the performance of these heuristics when allowed a total of x backward moves against repeated runs of the random hill climbing heuristic (RHC) (allowed x' repetitions). A brief analysis of the variable-depth search algorithm first proposed by Kernighan and Lin [12] is given in Section 5.2.

	$k < \lg(N/c)/x$		$k \geq \lg N/c$, $m = 2(N-c)/c$	
Algorithm	Operations	Expected Rank	Operations	Expected Rank
RBT	$2^{k+1}(x+1)$	$N - c(2^{k-1}(x+1) - x/2)$	$m + ek(x+1) + 2x$	$c/(ek(\frac{2}{3}x+1))$
GBT	$2^{k+1} + kx2^k$	$N - c2^{k-1} - xc(2^k(1-\frac{2}{k}) - \frac{1}{2})$	$\frac{mk}{2} + ek(x+1) + kx$	$c/(ek(\frac{kx}{k+2}+1))$
RGBT	$2^{k+1} + kx2^k$	$N - c2^{k-1} - (xc/2)(2^{k+1}-1)$	$m + x(ek^2+1)$	$2c/(k(x+1))$
CBT	$2^{k+1}(x+1)$	$N - c(2^{k-1}(x+1) - x/2)$	$m + ek(x+1) + kx$	$c/(ek(\frac{kx}{k+2}+1))$
RHC	$x'2^{k+1}$	$N - 2^k(c+1)$	$x'(m+ek)$	$c/(ekx')$

– A fundamental problem facing any general optimization strategy is making effective use of the full time budget. Hill climbing heuristics make rapid initial progress toward improved solutions, but terminate quickly and thus do not use large amounts of search time effectively.
We use our model to investigate efficient time allocation policies. When the search-time budget is sufficiently small, random sampling is the most effective heuristic. As search times and local neighborhood sizes increase, hill climbing heuristics with increasing amount of backtracking become more effective. We identify the best heuristic as a function of search-time budget and local neighborhood size, with our results summarized below:

Time Budget	$k < \lg(N/c)/x$	$\lg(N/c)/x \leq k < \lg N/c$	$\lg N/c \leq k$
$t < 4(N-c)/c$	random sampling	random sampling	random sampling
$4(N-c)/c \leq t < 4(N-c)/c + x$	random sampling	random sampling	backtracking
$4(N-c)/c + x \leq t$	random sampling	backtracking	backtracking

Assuming that t total number of operations and x number of backtracking moves are allowed, this table presents the best heuristic choice for different ranges of k.
– We evaluate the basic validity of our abstract model through experimentation. Using the black-box combinatorial-optimization system described in [16], we evaluate the analyzed heuristics on instances of five classical combinatorial problems (traveling salesman, shortest common superstring, max-cut, maximum satisfiability, and vertex cover).
We compare the observed times and solution qualities with predictions of our model, and demonstrate the effect of neighborhood choice on the performance of RHC. To a great extent, the gross predictions of our theory are confirmed. Where our theory breaks down (most clearly, on the performance of greedy backtracking), clear avenues for further research are suggested – specifically, the study of undirected neighborhoods and non-uniform edge probability distributions.
Finally, to validate our analytical results we evaluate the performance of the proposed heuristics on random graphs generated according to the model.

Organization

We begin this paper with a brief survey of related work on local-improvement heuristics in Section 2. We then describe our random graph model for heuristic search in Section 3. The local search and backtracking procedures we analyze are described in Section 4. We analyze local search heuristics for arbitrarily wide

search neighborhoods ($c = N$) and narrow search neighborhoods ($c < N$). We conclude in Section 6 with representative experimental results.

2 Previous Work

Local search heuristics such as simulated annealing, tabu search, and genetic algorithms have been extensively studied. We refer to the relevant chapters of [1] for recent surveys.

Much of the analytical work on these algorithms concerns the rate to which these algorithms, particularly simulated annealing, converge on the global optimum. See [8,14] for representative results. Aldous and Vazirani [3] model the combinatorial search landscape as walks on a tree to analyze the *go with the winners* heuristic, which breaks the search procedure into stages where the best solutions in the previous stage are used as initial solutions for new parallel searches. There has also been interest in analyzing the performance of local search on specific classes of problems, particularly graph bisection in random graph models with planted solutions [5,11]. [5]. Here the goal is to identify problems and input classes where local optimization heuristics ensure good solutions with high probability.

Our model is akin to measures of the *combinatorial dominance* or *ordinal quality* of heuristics, where the dominance number of a heuristic solution s counts the number of solutions which are provably no better than s. Combinatorial dominance and ordinal optimization provide a quality measure which is sometimes orthogonal to approximation ratio [9]. Problem-specific heuristics which achieve high dominance for the traveling salesman problem have been extensively studied [6]. Combinatorial dominance provides a quality measure for heuristics which is sometimes orthogonal to approximation ratio. For example, a 2-factor approximation to metric TSP from the minimum spanning tree heuristic can yield the worst of all $(n-1)!$ tours on certain instances [7].

Finding a local optimum is difficult under certain problem formulations. The class *PLS* [10] consists of local search problems for which local optimality can be verified in polynomial time. Papadimitriou, Schafffer and Yannakakis [15] show that finding a local optimum for some problems in PLS is PSPACE-complete if we insist on a specific initial solution for the local search procedure. Other investigations into the relative difficulty of NP search problems include [2,4]. The properties of the standard algorithm on hypercubes, and its ability to converge on a local optima has been by Tovey [19].

3 A Model of Energy Landscapes

We model the energy landscape for a given problem as a random directed graph $G = (V, E)$, where $|V| = N$, $|E| = k \cdot N$, together with an objective function $f : V \longrightarrow \Re$ that imposes a total order among the vertices, so that $f(v_i) \leq f(v_j)$ for $1 \leq i < j \leq N$. We assume without loss of generality that $f(v_i) = i$. The edge set E is constructed by drawing k edges from $v_i \in V$ to randomly chosen

vertices in range $[v_{i-c}, v_{i+c}]$; the target vertices are chosen with substitution, so that v_i may have more than one edge going to $v_j \in [v_{i-c}, v_{i+c}]$.

We seek to find a vertex of minimum rank while restricting our search algorithms to two operations, either (1) generating an element of V uniformly at random, or (2) traversing an edge (v_1, v_2) of G, where v_1 denotes a previously visited vertex of G. All searches begin at vertex v_{N-1}. We pay unit cost to evaluate the objective function on any vertex. We are interested in the expected performance of search heuristics on such graphs.

The performance of a heuristic depends on both the expected rank of the vertex that the heuristic reports (the solution quality) and the number of objective function evaluations it performs (the search cost). We say that one heuristic *dominates* another if it reports a vertex of the same or better expected rank as the other using fewer expected operations. We say that the heuristic *visits* a vertex if and only if that vertex is in the tree.

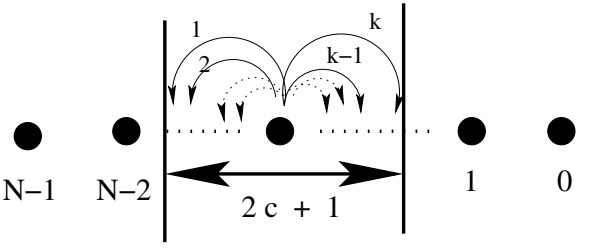


Fig. 1. The landscape model, where each of N vertices has k randomly selected outgoing edges of span at most c. The goal of a search is to reach a vertex of lowest possible rank

The graphs we study are parameterized by three constants: size N, out-degree k, and range c. Each graph has fixed out-degree k, with the vertices adjacent to v drawn uniformly at random from the set of vertices in a window of rank $\pm c$. For simplicity, we allow self-loops and multi-edges in G. See Figure 1 for a cartoon representation of our model.

We note that neighborhood operators such as *swap* and *2-opt* for permutation and subset problems have polynomial size-neighborhoods that are large enough to maintain connectivity. Furthermore, the length of the shortest path between any two permutation or subset solutions is in $O(\lg N)$, implying that N/c is at most logarithmic in the size of the solution space.

To study the shape of our energy landscape, we partition G into two regions by rank: the *transient* region $(v_c, v_{N-1}]$, and the *goal* region $[v_0, v_c]$. The analysis of the behavior of a local-search heuristic depends on a proper understanding of its behavior in these regions.

– The range parameter c governs the size of the goal region and how fast we can hope to descend from the start vertex to the goal region. At least N/c local moves are necessary to enter the goal region.

- Within the transient region of the graph, the probability of a vertex being a sink is $1/2^k$, and hence depends completely on out-degree. Thus if k is sufficiently small relative to N and c, any gradient-descent search is likely to terminate in the transient region and not reach the goal region.
- The probability that a goal region vertex v is a sink depends on c, k, and $rank(v)$. This probability increases rapidly as we move down through the goal region. Thus it becomes progressively harder to make additional progress toward the optimum as we search through the goal region.
- Backtracking heuristics examine solutions in the goal region or its vicinity. RBT will only examine solutions of rank lower than $v_{3c/2}$ for any given backtracking move with high probability.
- Note that proximate vertices (v_i and v_{i+1}) represent similar scoring solutions. However, no short or even long path between them in the graph need exist for sufficiently small k.

4 Search Heuristics

Local search heuristics exploit the neighborhood structure for each solution. *Hill climbing* or *gradient descent* are the simplest local search heuristics. Starting from an initial solution, they proceed by visiting lower-scoring neighboring solutions of the current solution. These heuristics terminate when the vertex is a sink. In this paper, we consider four flavors of hill climbing:

- *Random hill climbing (RHC)* – Candidates are generated as random solutions from the set of neighboring solutions. We assume this is done without repeated selections, by traversing a random permutation of the neighborhood; the heuristic selects the first lower ranked neighbor discovered.
- *Greedy hill climbing (GHC)* – The heuristic selects the lowest ranked solution from the entire set of neighboring solutions. Thus each step of GHC requires examining the entire set of k neighboring solutions.
- *Reverse greedy hill climbing (RGHC)* – The heuristic selects the highest ranked solution from the set of all lower ranked neighbors. Thus it makes the slowest possible gradient descent through the search space. Again, every step of RGHC requires examining the entire set of k neighboring solutions.
- *Combined hill climbing (CHC)* – This hill climbing heuristic knows the value of c for the problem instance, which is sufficient insight to determine whether any solution s lies in the transient or goal regions of the solution space. CHC uses a combination of the random hill climbing and the greedy hill climbing heuristics: RHC-like traversal from an initial solution, and GHC-like traversal after entering the goal region.

We also consider the basic heuristic *random sampling*, which chooses solutions at random from S and visits as many solutions as possible within the allocated time.

Backtracking heuristics emerge from a given sink v_i by visiting some neighbor of inferior rank and continuing the search from there. We assume that backtrack

heuristics remember the lowest ranking vertex encountered on this tour, so the additional search can only improve the quality of the resulting solution. We analyze four different backtracking heuristics based on variants of hill climbing; in all cases the search begins with a RHC search to a sink, followed by a backtracking move. All backtracking moves are made from a sink v to a randomly selected vertex in the neighborhood of v. Each are parameterized by the number of permissible backtrack moves x:

- *Greedy hill climbing with backtracking (GBT)* – Here, after the first backtracking move, downward moves are made to the lowest of the solution's k neighbors. Every step of GBT requires examining the entire set of k neighboring solutions.
- *Random hill climbing with backtracking (RBT)* – Here, all downward moves are made by selecting a random lower ranking neighbor.
- *Reverse-greedy hill climbing with backtracking (RGBT)* – Here, after the first backtracking move, downward moves are made to the highest ranking vertex from the set of lower ranking vertices. Every step of RGBT requires examining the entire set of k neighboring solutions.
- *Combined hill climbing with backtracking (CBT)* – This backtracking heuristic knows the value of c for the problem instance, and switches from a random to a greedy search strategy after reaching a sink of rank $\leq c$. For large k, this strategy is identical to GBT, for small k it will operate like RBT.
- *variable-depth search (VDS)* – This heuristic assumes that a natural order is induced by the neighborhood operator. Beginning at a random solution, it proceeds by creating a solution list of length $n < k$. The list is constructed by choosing the current solution's highest ranked neighbor whose index has not been chosen before. The heuristic chooses the highest ranked solution from this list and recurs, terminating when the first solution in the list is the best solution.

5 Heuristic Analysis

The performance of each hill climbing variant depends on the relationship between $c < N$ and k, and follows in a relatively direct way from the properties listed at the end of Section 3. Proofs are omitted due to lack of space, and will appear in the full version of the paper.

The analysis of the behavior of random sampling is a baseline for our analysis of local search:

Theorem 1. *m iterations of random sampling will find a solution of expected rank $\frac{N}{m+1}$ for $m \ll N$.*

We are interested in estimating the solution quality as well as the cost of heuristics. We must establish whether they reach the goal region, and if so how deep into the lower region of the landscape, namely $[v_0, v_{c/k}]$. We begin by demonstrating that the standard heuristic reaches the goal region if $k > \log(N/c)$.

Lemma 1. *A random sampling graph G will be connected (with high probability) if $k > 2 \ln N$ and will be disconnected (with high probability) if $k < (1/2) \ln N$.*

Lemma 2. *The expected number of steps made by a hill-climbing heuristic before encountering a sink in the transient region is $(\frac{2c+1}{c+1})^k$.*

Corollary 1. *The expected number of neighborhood evaluations made by the standard algorithm is $(\frac{c+1}{2c+1})^k$ if $k < \log \frac{N}{c}$.*

Assuming that $k > \ln N$, we see that this number of steps together with the next lemma implies that hill-climbing heuristics easily get past the transient region $(v_c, v_{N-1}]$.

Lemma 3. *The expected reduction in rank on every move made by each heuristic in the transient region is: (1) $\frac{c}{2}$ for RHC, and (2) $c - \frac{2c}{k+1}$ for GHC.*

Theorem 2. *Let $r = N/c$. Then with high probability:*

- *Random hill climbing reaches the goal region if $k > \lg r + 1$. Otherwise, it will terminate after an expected 2^{k+1} operations and return a solution of expected rank $N - c2^k$.*
- *Greedy hill climbing reaches the goal region if $k > \lg r$. Otherwise, it will terminate after an expected $k2^k$ operations and return a solution of expected rank $N - c2^k(k+2)/k - c/k$.*

Having reached the goal region, RHC and GHC find solutions within $[v_0, v_{c/ek}]$, which contains vertices that are likely to be local optima.

Theorem 3. *The expected rank of the solution found by RHC and GHC is approximately $\frac{c}{ek}$; the expected number of operations is $\frac{2(N-c)}{c} + ek$ and $\frac{k(N-c)}{c} + ek$ respectively.*

5.1 Comparing the Hill Climbing Heuristics and Random Sampling

Random sampling samples vertices in the range $[v_0, v_{N-1}]$, remembering the lowest-ranked vertex encountered. When $c = N$, RHC and GHC essentially perform random sampling until reaching a sink. When $c < N$, local search can make average rank improvements of at most $c/2$ per evaluation, thus random sampling dominates local search in the transient region. We formulate this below, Let t denote the number of operations allocated to a given search:

Theorem 4. *Random sampling dominates local search if $k < \lg(N/c)$ or $t < N/c$.*

5.2 Efficient Time Strategies

Black-box optimization systems are given a time budget, and repeated iterations may be performed if a single search completes before this time is spent. We compare the performance of the repeated random hill climbing, random sampling

and backtracking heuristics introduced in Section 4. We show that any reasonable backtracking heuristic will avoid traversing the transient region multiple times, and thus sample a larger number of vertices in the goal region. If $k \geq \lg(kN/c)$ all backtracking heuristics described in this section are expected to reach a sink in the goal region in $4N/c + ek$ operations.

Lemma 4. *RBT allowing x backtracking steps provides a solution of equal rank to $\lceil \frac{x3^k}{2^k+3^k} \rceil$ independent hill climbing runs in the goal region.*

If $k < \lg(N/c)/x$ the backtracking heuristics can expect to reach a vertex in the goal region. If $\lg(N/c)/x \leq k < \lg(N/c)$ the backtracking heuristics will reach a vertex in the goal region, but will sample fewer vertices there since some backtracking moves will be used to get out of local optima in the transient region. Reverse greedy will reach the goal region if $k > \frac{\lg(kN/c)}{x}$.

Lemma 5. *RBT examines solutions of rank strictly smaller than $\frac{3}{2}c$ with probability $e^{-1.5x/k}$.*

Lemma 6. *GBT is expected to examine solutions of rank smaller than $c(1 + \frac{e+1}{k(e-1)})$.*

Our model predicts that GBT and CBT dominate RBT, however GBT underperforms in practice as demonstrated in Section 6. This is because our model assumes independent neighborhoods while our experiments where ran using symmetric neighborhoods. In our experiments GBT repeatedly returned to the same local optima. Tabu search and variable-depth search are designed to overcome this deficiency. The variable-depth search algorithm performs GBT while ensuring that the same solution is not revisited in a given cycle. Efficient operation of the variable-depth search algorithm depends on the choice of the list length n. Long lists have a higher level of tabu but increase the number of lower-rank solution evaluations.

Theorem 5. *A single iteration of variable-depth search is expected to make $T = \frac{e+1}{4}(k - \frac{k}{n} - 1 + \frac{1}{n})$ operations and find a solution of rank $c(1 + \frac{n(e+1)}{k(e-1)})/(T - \frac{k(N-c)}{c})$.*

5.3 Other Search Heuristics

We have analyzed a variety of basic hill climbing and backtracking search heuristics on a uniform random graph model. Our framework can also provide insight into more sophisticated search heuristics as well:

- *Go with the winners algorithms* – This heuristic breaks the search procedure into l stages, where the best solutions from stage i are used as initial solutions for the m parallel searches done in the $(i+1)$st stage. The heuristic will perform poorly in the transient region whenever $k > \lg N$, since it will use an excessive number of operations until it first reaches the goal region. However, this parallel search is more likely to reach the goal region for small k, making it a good alternative to backtracking for sparse neighborhoods. For $k > \lg N$, parallel searches will be useful within the goal region, but no more so than the random or the greedy backtracking heuristics discussed in this paper.

– *'Lin-Kernighan' Heuristics* – The Lin-Kernighan heuristic for TSP [13] uses a multiple neighborhood operator (k-opting for various values of k) and switches neighborhoods on getting stuck. Heuristics that use more than one neighborhood function are aptly rewarded under our model. Local search outperforms random sampling for bounded neighborhood functions, so the existence of an alternate, independent neighborhood on reaching a 'sink' enables us to continue local search, effectively increasing k for random hill climbing strategies.

We leave more rigorous analysis for these and other heuristics on uniform and non-uniform random graph models as an open problem.

6 Experimental Results

We conducted two classes of experiments to validate our results. The first (and more interesting class) evaluates instances of classical combinatorial optimization problems to establish the extent to which their adjacency graphs resemble the random graphs of our model. The second class constructs the random graphs of our model and performs simulated searches on them, to establish the accuracy of our analytical results and will be discussed in the full paper.

All experiments were conducted on an AMD Athlon 1Ghz, 768Mb RAM PC running Linux. Detailed information about our experimental platform were given in [16]. Software and data sources are available through the DISCROPT web page http://www.cs.sunysb.edu/∼discropt.

6.1 Model Validation

We report experimental results about the observed performance of the most interesting of our heuristics on the following five different combinatorial optimization problems:

Table 1. A comparison of the extrapolated N/c from (1) the number of evaluations required for RHC to outperform random sampling, and (2) the average number of evaluations per neighborhood performed by RHC. Extrapolations are made from the results depicted in Figure 2

source	TSP	SCS	MaxCut	MaxSat	Vertex Cover
break-even point	47	36	158	25	60
neighborhood evaluations	45	40	210	25	90

– *Traveling Salesman* - given an edge-weighted graph, find a shortest tour visiting each vertex exactly once. TSP is well suited to local search heuristics, as move generation and incremental cost evaluation are inexpensive. The representative test case is a complete graph on 280 vertices from TSPlib [17] a280.

Table 2. The performance of the heuristics. Backtracking heuristics were allowed to make ten backtracking moves

	RandHillClimb		GreedHillClimb		RandBackTrack		GreedBackTrack	
Problem	score	σ	score	σ	score	σ	score	σ
TSP	6476.2	231.1	6688.5	584.0	5566.1	309.8	6200.2	318.9
SCS	2339.8	119.7	2436.5	135.1	2092.8	125.8	2239.8	165.4
MaxCut	2184.3	27.8	2119.5	27.1	2173.1	27.0	2131.5	23.3
MaxSat	3882.6	1101.2	4056.7	1060.0	2753.8	1065.3	3085.3	635.5
Vcover	2118.0	11.9	1795.4	454.7	647.5	7.4	648.6	5.5

- *Shortest Common Superstring* - given a set of strings, find a shortest string which contains as substrings each input string. Our test case consists of 135 strings, each of length 50, randomly chosen from 15 concatenated copies of the length-500 prefix of the Book of Genesis.
- *Max Cut* - partition a graph into two sets of vertices which maximize the number of edges between them. Max cut is a subset problem which is well suited to heuristic search. The representative test case is the graph alb5000 from TSPlib [17] with 5000 vertices and 9998 edges.
- *Max Satisfiability* - given a set of weighted Boolean clauses, find an assignment that maximizes the weights of satisfiable clauses. Max sat is a difficult subset problem not particularly well suited to local search. The representative test case is the file jnh308 from [18], a set of 900 randomly weighted clauses of 100 variables.
- *Vertex Cover* - find a smallest set of vertices incident on all edges in a graph. Vertex cover is representative of constrained subset problems, where we seek the smallest subset satisfying a correctness constraint. The changing importance of size and correctness over the course of the search are a significant complications for any heuristic. The representative test case is a randomly generated graph of 1000 vertices and 1998 edges.

Although we limit the discussion to one instance for each problem due to lack of space, we assure the reader that the reported results are representative of the other instances in our test bed. Each experiment was repeated 15 times, and the reported results pertain to mean or median values; standard deviations are given when appropriate.

In practice, each move made by a local search heuristic is less costly than that of random sampling, since moving from a solution to its neighbor may require only a partial solution evaluation, while generating a new random solution always requires a complete solution evaluation. Here, however, we look at the number of operations and do not differentiate between partial and full solution evaluations.

Our analytical results on the abstract model make several predictions about the relative performance of search heuristics. Here we boldly compare these predictions to data on our five real-world problems:

- *Random sampling outperforms hill-climbing for short searches* – Our theory predicts that random sampling should outperform hill-climbing until $\approx N/c$

Table 3. The number of evaluations made by the heuristics (median across the 15 experiments). The backtracking heuristics were allowed to make ten backtracking moves

	RandHillClimb		GreedHillClimb		RandBackTrack		GreedBackTrack	
Problem	actual	predicted	actual	predicted	actual	predicted	actual	predicted
TSP	758067	58635	17655120	937440	3266920	585976	1480500	1171856
SCS	92739	13605	1003995	180900	496577	274591	468836	521711
MaxCut	49950	7685	6915000	467500	119360	75215	6890010	150195
Vcover	8855	1575	341000	38500	21492	15105	24883	30085
MaxSat	537	175	2800	1350	2253	1555	2410	3035

evaluations. This indeed appears to be the case. Table 1 presents the average break-even point (in terms of number of evaluations) between random sampling and random hill-climbing for each of our five problems.

– *Edge length is bounded by c* – Our simple model assumes that all edges are drawn uniformly at random from a span at most c solutions. This is indeed simplistic (and demonstrably false) for many problems. However our results will remain applicable under the reasonable assumption that average edge span is $\theta(c)$.

  Our theory provides two relatively independent ways to estimate N/c for a given problem from observed data, by (1) observing when the number of evaluations per neighborhood for random hill-climbing increases, and (2) observing the break-even point between random sampling and hill-climbing. These events occur when the search heuristics cross from the transient region to the goal region; see Figure 2 for an example.

  We predict that RHC, starting the search with a random initial solution, will quickly descend into the goal region where it will terminate at about $v_{c/ek}$. After x evaluations in the transient region, the algorithm is expected to find a solution of rank $(N - cx)/2$. This solution is expected to be of higher rank than the solution found by random sampling. RHC is expected to find a solution of rank lower than c after N/c solution evaluations, and it is expected to outperform random sampling after $N/c+2$ solutions evaluations. As shown in Table 1, these predictions are in excellent agreement with each other on all problems.

– *RHC and GHC return solutions of similar quality* – Our theory predicts that RHC and GHC yield solutions of similar quality. Table 2 gives the average scores of our heuristics on each of the five problems. Note that GHC does better than RHC only on maxcut.

– *GHC is much more time consuming than RHC* – Our theory predicts that GHC requires roughly $k/2$ times more evaluations than RHC, where $k = n(n-1)/2$ for the swap operator employed in our simulations. Table 3 gives the median run-times of our heuristics on each of the five problems.

– Our theory predicts that a standard algorithm using a neighborhood with a smaller c will terminate with a higher quality solution. Indeed as demonstrated in figure 3, using relatively few operations we can determine c and choose the right local-neighborhood operator for our search.

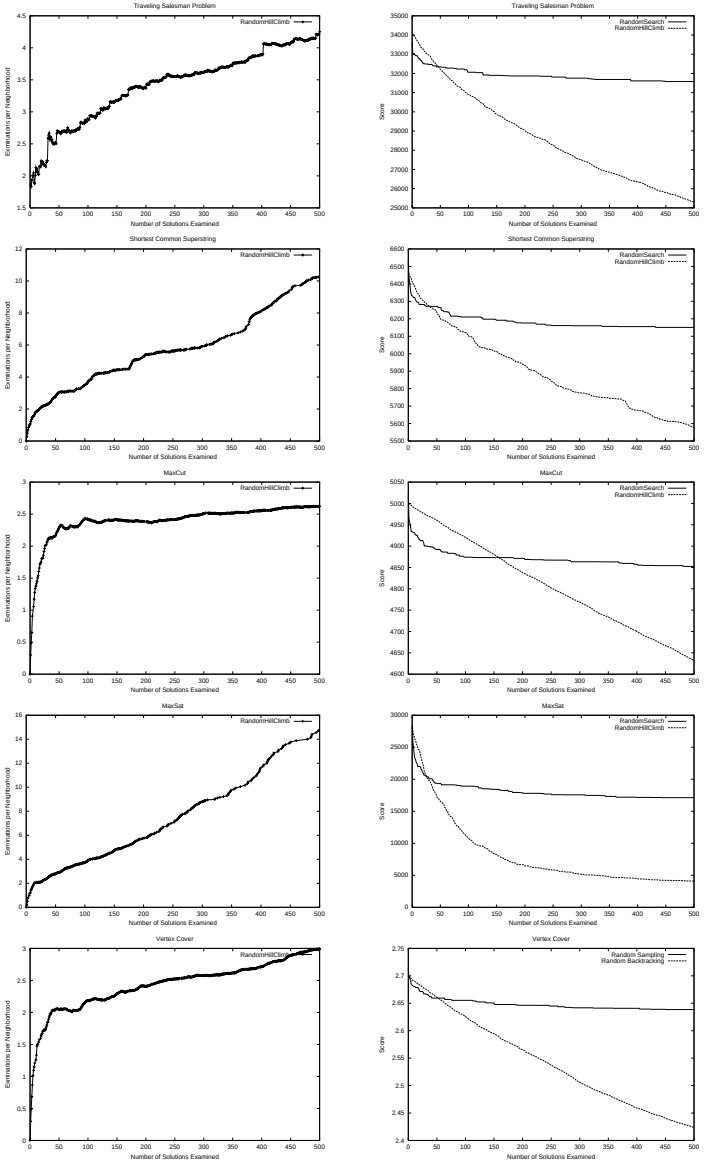
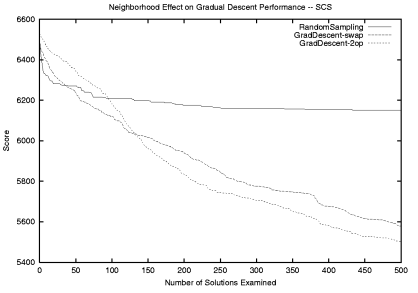


Fig. 2. Representative N/c estimations for the five instances of TSP, SCS, MaxCut, MaxSat, and Vertex Cover. The average number of solution examinations per neighborhood as a function of the total number of solution examinations (left), and The progress of the random sampling and the random hill climbing heuristics using the score of the best solution found as a function of the solutions examined (right)

– *RBT dominates GBT* – Greedy backtracking performs much more poorly than our model predicts. This is because it repeatedly returns to the same



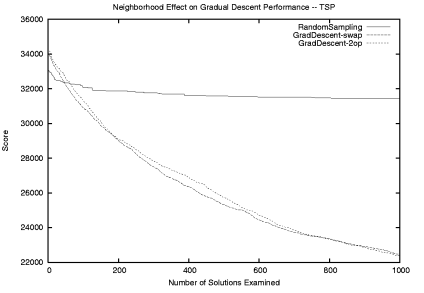


Fig. 3. We compare the behavior of RHC on an instance of SCS (left) and TSP (right) under two local neighborhoods, *swap* and *2opt*. We learn N/c from the point of intersection between RHC and random sampling. For the SCS instance, the 2-opt neighborhood has a smaller c, therefore a standard algorithm will use a greater number of operations to reach the goal region and terminate with a higher quality solution. For the TSP instance, the difference between the RandomSampling-RHC intersection points is to small to make reliable predictions about local-neighborhood-operator quality. We learn about the differences between the neighborhoods using a total of $\sim 3N/c$ evaluations

local optima, a problem which occurs because our model assumed independent neighborhoods, while the *swap* is in fact symmetric. It is just this phenomenon that tabu search is designed to overcome.

– *Limitation of the model* – Table 3 shows that we underestimate the number of operations required for the convergence of the standard algorithm, grossly in a few instances. We believe this is a result of the assumption of uniform distribution of the edges for each vertex. The problem appears to lie at the top of the goal region. For example, we predict that in less than 3 GHC moves, we will get from rank c to rank c/k. In fact, that data suggests that we make many more short moves in this region than predicted, almost as if c has shrank in the goal region. Well within the goal region, the probability that an edge will go down is still quite high. This is unlike what we've predicted. This observation motivates a more elaborate model with a non-uniform distribution of edges.

References

1. E. Aarts and J. K. Lenstra. *The Traveling Salesman Problem: A Case Study.* Wiley, 1997.
2. M. Agrawal, E. Allender, R. Impagliazzo, T. Pitassi, and S. Rudich. Reducing the complexity of reductions. In *ACM Symposium on Theory of Computing*, pages 730–738, 1997.
3. D. Aldous and U. V. Vazirani. "Go with the Winners" algorithms. In *IEEE Symposium on Foundations of Computer Science*, pages 492–501, 1994.
4. P. Beame, S. Cook, J. Edmonds, R. Impagliazzo, and T. Pitassi. The relative complexity of NP search problems. In *ACM Symposium on Theory of Computing*, pages 303–314, 1995.

5. T. Carson and R. Impagliazzo. Hill-climbing finds random planted bisections. In *Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 903–909, 2001.
6. G. Gutin and A. Yeo. TSP heuristics with large domination number. Technical Report PP-1998-13, Odense University, Denmark, Aug. 20, 1998.
7. G. Gutin, A. Yeo, and A. Zverivich. Polynominal restriction approach for the atsp and stsp. In *The Traveling Salesman Problem*, to appear.
8. B. Hajek. Cooling schedules for optimal simulated annealing. *Math. Operations Res.*, 13:311–329, 1988.
9. Y. Ho, R. Sreeniva, and P. Vakili. Ordinal optimization of discrete event dynamic systems. *J. on DEDS*, 2:61–68, 1992.
10. D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? In *Proc. 26th Annual Symp. on Foundations of Computer Science*, pages 39–42, 1985. Also *J. Computer System Sci.*, 37(1), pp. 79-100, 1988.
11. A. Juels. Topics in black box optimization. Ph.D. Thesis, University of California, Berkeley, 1996.
12. B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(1):291–307, 1970.
13. S. Lin and B. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.
14. O. Martin, S. Otto, and E. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5:299–326, 1991.
15. C. H. Papadimitriou, A. Schaffer, and M. Yannakakis. On the complexity of local search. In *Proc. 22nd Annual ACM Symp. on Theory of Computing*, pages 438–445, 1990.
16. V. Phan, P. Sumazin, and S. Skiena. A time-sensitive system for black-box combinatorial optimization. In *4th Workshop on Algorithm Engineering and Experiments*, San Francisco, USA, Jan. 2002.
17. G. Reinelt. *TSPLIB*. University of Heidelberg, www.iwr.uni-heidelberg.de/groups/comopt/ software/TSPLIB95.
18. M. Resende. *Max-Satisfiability Data*. Information Sciences Research Center, AT&T, www.research.att.com/ ∼mgcr.
19. C. A. Tovey. Local improvements on discrete structures. In E. Aarts and J. K. Lenstra, editors, *Local Search and Combinatorial Optimization*, pages 57–89. John Wiley and Sons Ltd., 1997.
20. D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

On the Hausdorff Voronoi Diagram of Point Clusters in the Plane

Evanthia Papadopoulou

IBM TJ Watson Research Center Yorktown Heights, NY 10598, USA
evanthia@watson.ibm.com

Abstract. We study the *Hausdorff Voronoi diagram* of point clusters in the plane and derive a tight combinatorial bound on its structural complexity. We present a plane sweep algorithm for the construction of this diagram improving upon previous results. Motivation for the investigation of this type of Voronoi diagram comes from the problem of computing the *critical area* of a VLSI Layout, a measure reflecting the sensitivity of the design to spot defects during manufacturing.

1 Introduction

Given a set S of point clusters in the plane their *Hausdorff* Voronoi diagram is a subdivision of the plane into regions such that the Hausdorff Voronoi region of a cluster $P \in S$ is the locus of points closer to P, according to the Hausdorff distance[1], than to any other cluster in S. As it was shown in [9], the Hausdorff Voronoi region of P can be defined equivalently as the locus of points t whose maximum distance from any point of P is less than the maximum distance of t from any other cluster in S. The Hausdorff Voronoi region of P is subdivided into finer regions by the farthest-point Voronoi diagram of P. This structure generalizes both the ordinary Voronoi diagram of points and the farthest-point Voronoi diagram. It is equivalent to the ordinary Voronoi diagram of points if clusters degenerate to single points and to the farthest-point Voronoi diagram if S consists of a single cluster.

The Hausdorff Voronoi diagram has appeared in the literature under different names, defined in terms of the maximum distance and not in terms of the Hausdorff distance, and motivated by independent problems. In [3] it was termed the *Voronoi diagram of point clusters*, in [1] the *closest covered set diagram*, and in [7,9] the *min-max* Voronoi diagram. The equivalence to the Voronoi diagram under the Hausdorff metric was shown in [9]. In [3] combinatorial bounds regarding this diagram were derived by means of envelopes in three dimensions. It was shown that the size of this diagram is $O(n^2\alpha(n))$ for arbitrary clusters of points, and $O(n)$ for clusters of points with disjoint convex hulls, where n is the total number of points on the convex hulls of individual clusters in S. The latter

[1] The (directed) Hausdorff distance from set A to B is $h(A, B) = \{\max_{a \in A} \min_{b \in B} d(a, b)\}$. The Hausdorff distance between A and B is $d_h(A, B) = \max\{h(A, B), h(B, A)\}$.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 439–450, 2003.
© Springer-Verlag Berlin Heidelberg 2003

was also shown in [1] for disjoint convex shapes and arbitrary convex distance functions. An $\Omega(n^2)$ example of n intersecting segments was given in [3]. In [9] a tighter combinatorial bound on the size of the diagram was given and the linearity property was extended to the more general class of *non-crossing* clusters (see Def. 1). An $O(n^2\alpha(n))$-time algorithm for the construction of this diagram was given in [3] by applying a divide and conquer technique for envelopes of piece-wise linear functions in three dimensions. This time complexity was automatically improved to $O(n^2)$ by the tighter combinatorial bound given in [9]. In [1] the problem for disjoint convex sets was reduced to abstract Voronoi diagrams and the randomized incremental construction of [5] was proposed for the computation of the diagram. This approach resulted in an $O(kn \log n)$-time algorithm, where k is the time to construct the Hausdorff bisector of two convex polygons. In [9] a direct divide and conquer algorithm was given for the construction of the Hausdorff Voronoi diagram of time complexity $O((n + M + N + K) \log n)$, where M was the number of pairs of points on the convex hull of *crossing* clusters, N was $O(n \log n)$, and K was the total number of points of clusters entirely enclosed in the minimum enclosing circle of some $P \in S$. In [7] the simpler L_∞ version of the problem was investigated and a simple plane sweep algorithm of time complexity $O((n + K) \log n)$ was given for the non-crossing case.

In this paper we derive a tight combinatorial bound on the structural complexity of the Hausdorff Voronoi diagram. In particular we show that the size of the diagram is $O(n + m)$, where m is the number of *crucial* supporting segments among pairs of crossing clusters (see Def. 2). Crucial supporting segments are entirely enclosed in the minimum enclosing circle of one of the clusters. We also present a simple plane sweep algorithm for the construction of the Hausdorff Voronoi diagram of time complexity $O(M + (n + m + K) \log n)$, where $K = \Sigma_{P \in S} K(P)$, $K(P)$ is the number of clusters entirely enclosed in the *anchor circle* of P, $M = \Sigma_{P \in S} M(P)$ and $M(P)$ is the number of points $q \in Q$ enclosed in the *anchor circle* of P such that either Q is entirely enclosed in the *anchor circle* of P or Q is crossing with P. The *anchor circle* of P is a specially defined enclosing circle (see Def. 4), generally larger than the minimum enclosing circle of P. This algorithm improves the time complexity of previous results, generalizes the plane sweep construction for Voronoi diagrams, and remains simple.

Our motivation for studying the Hausdorff Voronoi diagram comes from an application in VLSI manufacturing as explained in [7,9], in particular *critical area* extraction for predicting the *yield* of a VLSI chip. The *critical area* is a measure reflecting the sensitivity of a VLSI design to manufacturing defects due to dust particles and other contaminants on materials and equipment. In [7,9] the critical area computation problem for *via-blocks* was shown to be reducible to the Hausdorff Voronoi diagram (termed the min-max Voronoi diagram). Via-blocks represent the 2nd most important defect mechanism (after shorts) for yield loss. For more details on the critical area computation problem and its connection to the Hausdorff Voronoi diagram see e.g. [6,7,8,9,11]. Plane sweep is our method of choice for this problem because of the very large data volume of VLSI designs. The advantage of plane sweep is that we never need to keep the entire Voronoi diagram in memory. Instead we only keep the *wavefront*, the portion of the

Voronoi diagram bounding the sweep-line. As soon as a Voronoi cell is computed, critical area computation can be performed independently within that cell, and the cell can be immediately discarded.

In the following, due to lack of space, we skip proofs that are easy to derive.

2 Preliminaries and Definitions

The *farthest distance* between two sets of points A, B is $d_f(A,B) = \max\{d(a,b), a \in A, b \in B\}$, where $d(a,b)$ denotes the ordinary distance between two points a, b. The (directed) Hausdorff distance from A to B, $h(A,B) = \{\max_{a\in A} \min_{b\in B} d(a,b)\}$. The (undirected) Hausdorff distance between A and B is $d_h(A,B) = \max\{h(A,B),$
$h(B,A)\}$. The Hausdorff bisector between A and B is $b_h(A,B) = \{y \mid d_h(y,A) = d_h(y,B)\}$ and the farthest bisector is $b_f(A,B) = \{y \mid d_f(y,A) = d_f(y,B)\}$. As it was shown in [9] the Hausdorff bisector and the farthest bisector are equivalent. In the following we simply use the generic term *inter-bisector* to denote both.

The *farthest* Voronoi region of point $p_i \in P$ is $freg(p_i) = \{x \mid d(x,p_i) > d(x,p_j), p_j \in P\}$ and the *farthest* Voronoi diagram of a set of points P is denoted as *f-Vor*(P). It is well known (see e.g. [10]) that *f-Vor*(P) consists of unbounded convex regions, one for each point on the convex hull of P. The bisectors of *f-Vor*(P) are portions of ordinary bisectors, denoted $b(p_i,p_j)$, $p_i, p_j \in P$, and they are called *intra-bisectors*. The convex hull of P is denoted as $CH(P)$. A segment $\overline{p_ip_j}$ connecting any two points on $CH(P)$ is called a *chord*.

The tree structure of *f-Vor(P)* is called the *intra-bisector tree* of P and it is denoted as $T(P)$. $T(P)$ is assumed to be rooted at an arbitrary point $y_0 \in T(P)$. Every point $y \in T(P)$ is weighted by $d_f(y,P)$, the radius of the smallest circle centered at y entirely enclosing P. The circle centered at y of radius $d_f(y,P)$ is called a *P-circle* and it is denoted as $\mathcal{K}_y$. $\mathcal{K}_y$ passes through $p_i, p_j \in P$ such that $y \in b(p_i,p_j)$. Point y partitions $T(P)$ in two parts: $T(y)$ and $T_c(y)$, where $T(y)$ consists of all descendents of y in the rooted $T(P)$ including y, and $T_c(y)$ is the complement of $T(y)$. If y is a vertex of $T(P)$ and an incident intra-bisector segment $\overline{yy_j}$ is explicitly specified to contain y then $T(y)$ consists of the subtree rooted at y containing segment $\overline{yy_j}$; $T_c(y)$ is still the complement of $T(y)$. For figures and more details see [9]. Chord $\overline{p_ip_j}$ ($y \in b(p_i,p_j)$) partitions $\mathcal{K}_y$ in two parts: $\mathcal{K}_y^r$, referred to as the *rear* portion of $\mathcal{K}_y$, enclosing the points of $CH(P)$ that induce $T(y)$, and $\mathcal{K}_y^f$, referred to as the *forward* portion, enclosing the points of $CH(P)$ inducing $T_c(y)$. The characterization of the portions of $\mathcal{K}_y$ as *rear* or *forward* depends on the root of $T(P)$ and can be reversed for a different choice of the root. Any point q in $\mathcal{K}_y^r$ is called *rear* and any point in $\mathcal{K}_y^f$ is called *forward*, with respect to intra-bisector point y, $b(p_i,p_j)$, and the root of $T(P)$. The following observation is a generalization of one given in [9] and is used throughout the paper.

Lemma 1. *For any point $y \in T(P)$, such that $y \in b(p_i,p_j)$, the following holds: For any $y_j \in T(y)$, $\mathcal{K}_y^f \subset \mathcal{K}_{y_j}^f$ and $\mathcal{K}_{y_j}^r \subset \mathcal{K}_y^r$. For any $y_k \in T_c(y)$, $\mathcal{K}_y^r \subset \mathcal{K}_{y_k}$. The observation is valid for any root of $T(P)$.*

Definition 1. *Two chords* $\overline{q_iq_j} \in Q$ *and* $\overline{p_ip_j} \in P$ *are called* crossing *iff they intersect and all* p_i, p_j, q_i, q_j *are points of the convex hull of* $P \cup Q$*; otherwise they are called* non-crossing*. Cluster* Q *is said to be* crossing *chord* $\overline{p_ip_j} \in P$ *iff there is a chord* $\overline{q_iq_j} \in Q$ *that is crossing* $\overline{p_ip_j}$*; otherwise* Q *is said to be* non-crossing *with* $\overline{p_ip_j}$*. Two clusters* P, Q *are called* non-crossing *iff their convex hulls admit at most two supporting segments.*

Let S be a set of point clusters. The Hausdorff Voronoi region of cluster $P \in S$ is $hreg(P) = \{x \mid d_f(x, P) < d_f(x, Q), \forall Q \in S\}$ and it may be disconnected. $hreg(P)$ is further partitioned into finer regions by *f-Vor(P)*. For any point $p \in P$, $hreg(p) = \{x \mid d(x, p) = d_f(x, P) < d_f(x, Q), \forall Q \in S\}$. The collection of all (non-empty) Hausdorff Voronoi regions defined by S, together with their bounding edges and vertices, is called the Hausdorff Voronoi diagram of S. The bounding edges of $hreg(P)$ consist of portions of *inter-bisectors* and the inner edges consist of portions of *intra-bisectors* among the points on the convex hull of P. The vertices are classified into three types: *inter-vertices* where at least three inter-bisectors meet, *intra-vertices* where at least three intra-bisectors meet, and *mixed-vertices* where at least one intra-bisector and two inter-bisectors meet.

By definition, a mixed Voronoi vertex v is the center of a P-circle $\mathcal{K}_v$ passing through $p_i, p_j \in P$ and $q_i \in Q$, entirely enclosing both clusters P and Q and not containing any other cluster in S. Clearly v is a vertex of the intra-bisector $b_h(P, Q)$ that is incident to $T(P)$. Since $b_h(P, Q)$ is a subgraph of *f-Vor*$(P \cup Q)$ [9], any vertex of the intra-bisector $b_h(P, Q)$ is a mixed Voronoi vertex of *H-Vor*$(\{P, Q\})$ and a candidate for a mixed Voronoi vertex of *H-Vor*(S). Vertex v is characterized as *crossing* (resp. *non-crossing*) iff Q is crossing (resp. *non-crossing*) with $\overline{p_ip_j}$. Furthermore, v is characterized as *rear* (resp. *forward*) iff $q_i \in \mathcal{K}_v^r$ (resp. $q_i \in \mathcal{K}_v^f$). The characterization of a mixed vertex as rear or forward depends on the choice of the root for the intra-bisector tree.

3 Structural Complexity

In this section we give a tight bound on the structural complexity of the Hausdorff Voronoi diagram.

Lemma 2. *Consider the mixed vertices induced on* $T(P)$ *by* $b_h(P, Q)$*. We have the following properties:*

- *For any rear non-crossing vertex* v, $T_c(v) \cap hreg(P) = \emptyset$*. Thus,* $T(P)$ *can contain at most one rear non-crossing mixed Voronoi vertex.*
- *For any forward non-crossing vertex* v, $T(v) \cap hreg(P) = \emptyset$*. Thus,* $T(P)$ *can contain at most* $|T(P)|$ *forward non-crossing mixed Voronoi vertices, at most one for each unbounded segment of* $T(P)$*.*
- *Any crossing forward vertex on* $T(P)$ *must be followed by a crossing rear vertex (considering only vertices of* $b_h(P, Q)$*).*
- *Any rear (resp. forward) vertex delimits the beginning (resp. ending) of a component of* $T(P) \cap hreg(P)$ *as we traverse* $T(P)$ *from the root to the leaves.*

Proof. For a rear non-crossing v, $Q \in \mathcal{K}_v^r \cup CH(P)$ and for a forward non-crossing v, $Q \in \mathcal{K}_v^f \cup CH(P)$. The first two statements are easy to derive by Lemma 1.

Consider a path from the root to a leaf of $T(P)$ that contains vertices of $b_f(P,Q)$. Let y_1 (resp. y_k) be the first (resp. last) vertex along this path that is contained in $hreg(P)$. y_1 may be the root of $T(P)$ (if $y_0 \in hreg(P)$) and y_k may extend to infinity. Because of the first two statements of this Lemma, any vertex between y_1 and y_k, except y_1, y_k, must be crossing. Let v be such a vertex between y_1 and y_k and let $p_i, p_j \in P$, $q_i \in Q$ be the points inducing v. If v is delimiting the ending of $hreg(P)$ and the begining of $hreg(Q)$, as we walk from y_1 to y_k, then q_i must be part of $\mathcal{K}_v^f$, that is v must be forward. On the contrary, if v is delimiting the begining of $hreg(P)$ after $hreg(Q)$, v must be in $\mathcal{K}_v^r$ i.e., v must be rear. Thus, all the vertices between y_1 and y_k must be crossing and must be alternating between forward and rear starting with a forward. Since y_k cannot be rear, any crossing forward mixed vertex on the path must be followed by a rear crossing mixed vertex. This also derives the last statement. □

Let v be a mixed vertex of $b_h(P,Q)$ induced by $(q_i, p_i, p_j), q_i \in Q, p_i, p_j \in P$. Clearly q_i, p_i, p_j are points of $CH(P \cup Q)$. We say that vertex v can be *attributed* to the first pair of supporting segments encountered as we traverse $CH(P \cup Q)$ from q_i to p_i and p_j. We have the following property.

Lemma 3. *Every rear vertex of $b_h(P,Q)$ on $T(P)$ can be attributed to a unique pair of supporting segments between $CH(P)$ and $CH(Q)$ that are entirely enclosed in the P-circle centered at the root of $T(P)$.*

Proof. Let v_i be a rear vertex of $b_h(P,Q)$ induced by $(q_i, p_i, p_j), q_i \in Q, p_i, p_j \in P$, that can be attributed to the pair of supporting segments (s_1, s_2). Since v_i is rear, $s_1, s_2 \in \mathcal{K}_{v_i}^r$. By Lemma 1, $\mathcal{K}_{v_i}^r \subset \mathcal{K}_{y_0}$, where y_0 is the root of $T(P)$, and thus, $s_1, s_2 \in \mathcal{K}_{y_0}$. Let v_j be another vertex of $b_h(P,Q)$ induced by a triplet of points $(q_j, p_i', p_j'), q_j \in Q, p_i', p_j' \in P$. The only way for $\mathcal{K}_{v_j}$ to enclose both P and Q is for all three q_j, p_i', p_j' be enclosed in the same slice of $\mathcal{K}_{v_i}$ as defined by triangle (q_i, p_i, p_j). But then either $q_j \in \mathcal{K}_{v_j}^f$ i.e., v_j is forward, or q_j must be part of a different component of $CH(P \cup Q)$ than q_i, i.e., v_j is attributed to a pair of supporting segments other than (s_1, s_2). □

Lemma 4. *If there is a P-circle through $p_i, p_j \in P$ that encloses Q then there can be no Q-circle through $q_i, q_j \in Q$ that encloses P such that $\overline{q_i q_j}$ and $\overline{p_i p_j}$ are crossing.*

Definition 2. *Let P, Q be a pair of crossing clusters and let (s_1, s_2) be a pair of supporting segments between $CH(P)$ and $CH(Q)$ enclosed in the minimum enclosing circle of P such that there is a vertex of $b_h(P,Q)$ that can be attributed to (s_1, s_2). Segments s_1, s_2 are called* crucial supporting segments *for P. The number of crucial supporting segments of P is denoted by $m(P)$.*

Theorem 1. *The structural complexity of H-Vor(S) is $O(n+m)$, where n is the total number of points on convex hulls of clusters in S, and $m = \Sigma_{P \in S} m(P)$ is the total number crucial supporting segments between pairs of crossing shapes. The bound is tight in the worst case.*

Proof. As it was shown in [9] the structural complexity of *H-Vor(S)* is proportional to the number of mixed Voronoi vertices of *H-Vor(S)*. By Lemma 2, the number of non-crossing mixed Voronoi vertices is $O(n)$. (This was also shown in [9]). Also by Lemma 2 (3rd statement) the number of crossing mixed Voronoi vertices is proportional to the number of rear crossing mixed Voronoi vertices. But by Lemma 3, any rear mixed Voronoi vertex can be attributed to a unique pair of supporting segments enclosed in $\mathcal{K}_{y_0}$. By considering the center of the minimum enclosing circle of P as the root of $T(P)$, the pair of supporting segments associated with a rear crossing mixed Voronoi vertex must be crucial. Therefore, the number of crossing mixed Voronoi vertices is $O(m)$.

To obtain the lower bound it is enough to construct a set S such that every rear crossing vertex of $b_h(P,Q)$, $P,Q \in S$, remains a Voronoi vertex in *H-Vor(S)*. Consider a vertical segment $P = \overline{p_1 p_2}$ and its minimum enclosing circle, $\mathcal{K}_P$. Let $\epsilon > 0$ be a small constant and let $l_i, 1 < i \leq k$, be a set of horizontal lines, each $\epsilon > 0$ above the other, where l_1 is the horizontal line through the midpoint of P. Let $Q_i, 1 \leq i \leq k$ be a set of horizontal line segments each located on line l_i (see Figure 1). Let $\mathcal{K}_i$ (resp. R_i) $1 \leq i \leq k$ be the P-circle passing through the leftmost (resp. rightmost) point of Q_i. The left endpoint of Q_i is chosen in the interior of $\mathcal{K}_{i-1}$, ϵ away from the boundary of $\mathcal{K}_{i-1}$. The right endpoint of Q_i is in the exterior of $\mathcal{K}_{i-1}$, $\epsilon/2$ away from the boundary. By construction, R_i and $\mathcal{K}_i$ enclose exactly P and Q_i in their interior and the same holds for any P-circle centered on l_i between the centers of R_i and $\mathcal{K}_i$. Thus, the center of $\mathcal{K}_i, 1 \leq i \leq k$, remains a vertex in *H-Vor(S)*. By Lemma 4 there can be no Q_i-circle enclosing P i.e., all vertices of $b_h(P,Q_i)$ must be incident to $T(P)$. Thus, the only crucial supporting segments are the $O(k)$ supporting segments between P and Q_i, each pair inducing a rear mixed Voronoi vertex in *H-Vor(S)*. The construction is shown using segments for clarity only. Each Q_i can be substituted by a cluster Q_i' of arbitrarily many points forming a thin convex shape around segment Q_i such that no endpoint of the original set of segments is enclosed in $CH(Q_i')$ as shown in Figure 2. Segment P can also be substituted by a cluster P' forming a thin convex shape to the right of segment P with no change in the arguments of the proof. □

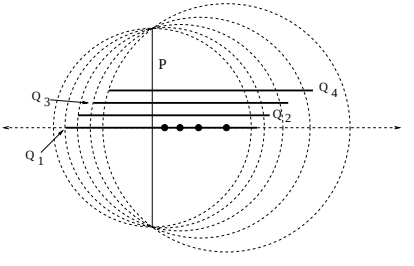


Fig. 1. The $\Omega(m)$ construction for the complexity of *H-Vor(S)*.

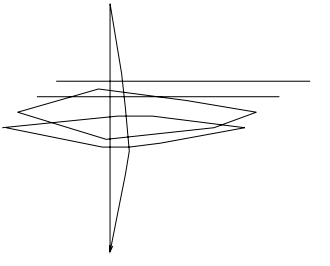


Fig. 2. Segments can be substituted by convex polygons.

4 A Plane Sweep Algorithm

In this section we give a plane sweep algorithm to construct the Hausdorff Voronoi diagram of S. The algorithm is based on the standard plane sweep paradigm of [2,4] but requires special *events* to handle mixed Voronoi vertices and disconnected Voronoi regions. The farthest Voronoi diagram of each individual cluster P is assumed to be available and it can be constructed by divide and conquer in $O(|P| \log |P|)$ time. The plane sweep basically stitches together the farthest Voronoi diagrams of the individual clusters of S into the Hausdorff Voronoi diagram of S.

The plane sweep process assumes a vertical sweep-line l_t sweeping the entire plane from left to right. The distance to the sweeping line is measured in the ordinary way and not in the Hausdorff metric i.e., $d(p, l_t) = \min\{d(p, y), y \in l_t\}$ for any point p. At any instant t of the sweeping process we compute $H\text{-}Vor(S_t \cup l_t)$ for $S_t = \{P \in S \mid \max_{p \in P} x(p) < t\}$ where $x(p)$ is the x-coordinate of point $p \in P$, and $d_h(p, l_t) = d(p, l_t)$. Following the terminology of [2] the boundary of the Voronoi region of l_t is called the *wavefront* at time t. The bisectors incident to the wavefront are called *spike bisectors* and consist of inter- and intra- spike bisectors. The wavefront consists of parabolic arcs, called *waves*, corresponding to ordinary bisectors between points $p \in S_t$ and the sweep line l_t. As the sweep line moves to the right, the wavefront and the endpoints of spike bisectors move continuously to the right. The combinatorial structure of the wavefront changes only at certain *events* organized in an *event queue*. We have two types of events, *site events* when new waves appear in the wavefront, and *spike events* when old waves disappear. Spike events correspond to the intersection of two neighboring spike bisectors and their treatment remains similar to the ordinary plane sweep paradigm. Site events are different and their handling is illustrated in the following.

Definition 3. *The* priority *of any point $v \in H\text{-}Vor(S)$ or $v \in f\text{-}Vor(P), P \in S$, is $priority(v) = x(v) + d(v, p_i)$, where p_i is the owner of region bounded by v in $H\text{-}Vor(S)$ or $f\text{-}Vor(P)$ respectively. In other words, $priority(v)$ equals the rightmost x-coordinate of the circle centered at v passing through p_i.*

The point of minimum priority for any cluster P is the intra-bisector point $Y_0(P)$ (for brevity Y_0) derived by shooting a horizontal ray backwards from the rightmost point p_r of P, until it hits the boundary of $freg(p_r)$ in $f\text{-}Vor(P)$. Throughout this section (unless explicitly noted otherwise) we assume that the root of $T(P)$ is $Y_0(P)$. As a result, the definition of a rear/forward mixed Voronoi vertex or rear/forward cluster always assumes that the corresponding intra-bisector tree is rooted at the point of minimum priority. The P-circle $\mathcal{K}_0$ centered at $Y_0(P)$ is referred to as the *minimum priority* circle of P. Clearly, $priority(Y_0) = x(p_r)$.

Lemma 5. *Let $y_i \in T(P)$. Then $priority(y_i) < priority(y_j)$ for any $y_j \in T(y_i)$ (assuming that the the root of $T(P)$ is $Y_0(P)$).*

Proof. The priority of any $y_i \in T(P)$ is given by the rightmost vertical line l tangent to $\mathcal{K}_{y_i}$. Since $priority(Y_0) \leq priority(y_i)$, and $\mathcal{K}^r_{y_i} \subset \mathcal{K}_0$ (Lemma 1), l must be tangent to $\mathcal{K}^f_{y_i}$. But by Lemma 1, $\mathcal{K}^f_{y_i} \subset \mathcal{K}^f_{y_j}$ for every $y_j \in T(y_i)$. Thus, $priority(y_i) < priority(y_j)$. □

The following Lemma can be easily derived from Lemma 2 and Lemma 5.

Lemma 6. *Let $hreg_i(P)$ be a connected component of $hreg(P)$, $P \in S$, that does not contain $Y_0(P)$. Then $hreg_i(P)$ must have exactly one rear mixed Voronoi vertex on $T(P)$ and this is the point of minimum priority of $hreg_i(P)$. If $Y_0(P) \in hreg_i(P)$ all mixed Voronoi vertices of $hreg_i(P) \cap T(P)$ are forward.*

By the definition of the wavefront, any Voronoi point $v \in$*H-Vor(S)* must enter the wavefront at time $t = priority(v)$. Thus, appropriate site events need to be generated so that at least one event exists (site or spike event) for every vertex of *H-Vor(S)*. We define two types of site events: *ordinary vertex events*, one for every vertex of $T(P)$, $P \in S$, and *mixed vertex events* (for brevity *mixed events*) whose purpose is to predict the rear mixed Voronoi vertices of *H-Vor(S)*. Ordinary vertex events are readily available from *f-Vor(P)*, $P \in S$. Mixed events get generated throughout the algorithm. An event is called *valid* if it falls on or ahead the wavefront at the time of its priority. An event falling behind the wavefront at the time of its priority is called *invalid*. Any event is processed at the time of its priority.

Let's first consider the vertex event corresponding to $Y_0(P)$ and let's assume that the event is valid. Let $\overline{p_i p_r}$ be the chord inducing Y_0 (i.e., $Y_0 \in b(p_i, p_r)$) where p_r is the rightmost point of P. Then at time t, the waves of p_i and p_r enter the wavefront for the first time, separated by the intra-bisector $b(p_i, p_r)$. In more detail, let q_j be the owner of the intersection point r where the horizontal ray from p_r hits the wavefront (see Figure 3). The wave of q_j is split at point r into two waves w_1, w_2 (say w_1 is above w_2), and the two rays of the spike bisector $b(q_j, p_r)$ emanating from r enter the wavefront, serving as new gliding tracks for waves w_1 and w_2. Furthermore, three new waves enter the wavefront: waves w_3, w_4 for p_r and wave w_5 for p_j gliding along the two rays of $b(p_j, p_r)$ emanating from Y_0. The ordering of the waves from top to bottom is w_1, w_3, w_5, w_4, w_2. Figures 3a and 3b depict the wavefront before and after the update, and Figure 3c shows the topological arrangement of the new waves. Spike events are generated as in the ordinary Voronoi diagram construction.

The update of the wavefront at any other valid ordinary vertex event $y_i \in T(P), y_i \neq Y_0$, is similar and is depicted in Figure 4. In detail, let $p_i, p_j, p_k \in P$ be the points inducing y_i. Since y_i is valid, at time $t = priority(y_i)$, point y_i must be a point of the wavefront incident to exactly one spike intra-bisector, say $b(p_i, p_j)$. Note that by Lemma 5 only one of the intra-bisectors incident to y_i can contain points of lower priority than y_i. At time t, a new wave for point p_k must enter the wavefront gliding between the spike intra-bisectors $b(p_k, p_i)$ and $b(p_k, p_j)$ as shown in Figure 4.

Let's now consider the handling of an invalid vertex event $y_i \in T(P)$ (y_i may be Y_0) and the generation of a mixed vertex event. At time $t = priority(y_i)$ point

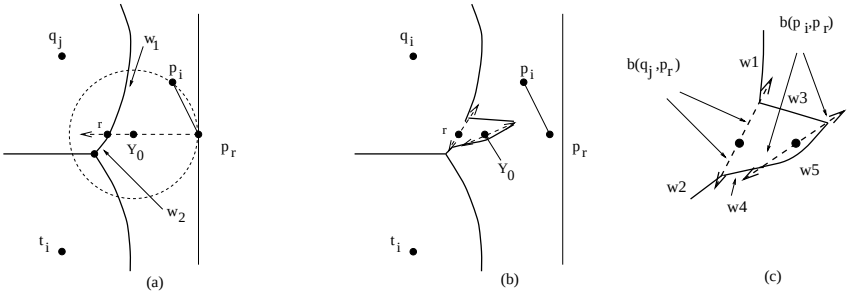


Fig. 3. The wavefront update at a valid minimum priority vertex event.

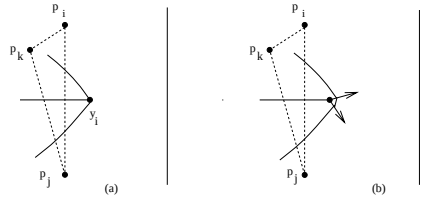


Fig. 4. The wavefront update at an ordinary valid vertex event.

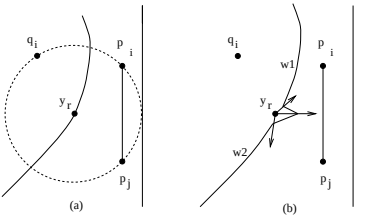


Fig. 5. The update of the wavefront at a valid mixed vertex event.

y_i is behind the wavefront. Let y_j be any immediate descendent of y_i in $T(P)$ that is not yet covered by the wavefront (if any). Then segment $\overline{y_iy_j}$ must intersect the wavefront at a point Y. The following process repeats for all immediate descendents of y_i that are not yet covered by the wavefront. Let $q_j \in Q_j$ be the owner of Y in *H-Vor*(S_t) and let $\overline{y_iy_j} \in b(p_i, p_j)$. Point q_j may be rear or forward with respect to $\overline{p_ip_j}$. Let's first consider the case where Q_j is non-crossing with $\overline{p_ip_j}$. If q_j is forward then by Lemma 2, $T(Y) \cap hreg(P) = \emptyset$. Thus, we eliminate all vertex events of $T(y_j)$ from the event queue. If q_j is rear, we eliminate from the event queue any vertex event associated with $T_c(Y)$. If q_j is rear and $d(y_j, p_j) < d_f(y_j, Q_j)$ then $b_f(Q_j, P)$ must intersect $\overline{y_iy_j}$ at a mixed vertex m_i. Since vertex m_i may or may not appear in *H-Vor*(S), a mixed-vertex event needs to get generated for m_i. Vertex m_i can be easily determined in $O(|CH(Q_j)|)$ time by considering intersections of $\overline{y_iy_j}$ with *f-Vor*(Q_j) (see also Lemma 9). Note that if $d(y_j, p_j) \geq d_f(y_j, Q_j)$, no mixed event gets generated as there can be no portion of $hreg(P)$ on $\overline{y_iy_j}$. Let's now assume that Q_j is crossing with $\overline{p_ip_j}$. Similarly to the non-crossing case, if $d(y_j, p_j) < d_f(y_j, Q_j)$ then a mixed vertex event m_i needs to get generated for $\overline{y_iy_j}$ at the point where $b_f(Q_j, P)$ intersects $\overline{y_iy_j}$. Although $b_f(Q_j, P)$ may induce several rear mixed vertices on $T(P)$, it can induce at most one on a single segment $\overline{y_iy_j}$. To determine m_i we walk on $\overline{y_iy_j}$ backwards starting at y_j, considering intersections with *f-Vor*(Q_j), until m_i is determined. The search starts at y_j and not at Y to maintain time complexity as it will be evident in Lemma 9.

The handling of a mixed vertex event is similar to the handling of an ordinary vertex event. In particular, let $y_r \in T(P)$ be a mixed vertex event induced by $p_i, p_j \in P, q_i \in Q$ and let y_j be the immediate descendent of y_r in $T(P)$. y_r is valid iff at time $t = priority(y_r)$, y_r is a point of the wavefront, in particular a point on the wave of q_i. If y_r is valid then the wave of q_i is split into two waves w_1, w_2, and new waves for p_i and p_j enter the wavefront between w_1 and w_2, separated by the spike intra-bisector $\overline{y_r y_j} \in b(p_i, p_j)$, as shown in Figure 5. If y_r is determined to be invalid then a new mixed-vertex event may be generated similarly to the case of an invalid vertex event.

The handling of spike events is identical to the ordinary Voronoi diagram construction with the exception of the need to generate mixed events for crossing clusters. A spike event between a spike intra-bisector $b(p_i, p_j)$ of P and a spike inter-bisector $b_h(P, Q)$ corresponds to a forward mixed Voronoi vertex v induced on $T(P)$ by Q. If Q is crossing with $\overline{p_i p_j}$ then v may be followed on $T(P)$ by a rear mixed Voronoi vertex. Thus, in this case, a mixed event induced by Q on $b(p_i, p_j)$ may need to be generated, exactly as for the case of an invalid ordinary vertex event. If Q is non crossing with $\overline{p_i p_j}$ then $T(v) \cap hreg(P) = \emptyset$ and thus, we can eliminate all the vertex events of $T(v)$ from the event queue.

The correctness of the algorithm follows from the correctness of the plane sweep paradigm of [2] as long as we can show that no rear mixed Voronoi vertex can be missed. In other words, we need to show that a mixed vertex event gets generated for every rear mixed Voronoi vertex of *H-Vor(S)*. Note that by Lemma 3, the minimum priority of any connected component of $hreg(P)$ occurs at a rear mixed Voronoi vertex of $T(P)$ (if not $Y_0(P)$), therefore a starting point for all connected Voronoi regions can be obtained if we have events for all rear mixed Voronoi vertices and the roots of intra-bisector trees. The following lemma shows that no mixed events can be missed and can be derived from the above discussion.

Lemma 7. *Let $y_i \in T(P)$ be an invalid site event or a valid spike event involving a crossing cluster Q. Let y_j be an immediate descendent of y_i in $T(P)$. If a mixed-vertex event y_r gets generated on segment $\overline{y_i y_j} \in T(P)$ during the handling of y_i, then $\overline{y_i y_r} \cap hreg(P) = \emptyset$. If no mixed-vertex event gets generated on $\overline{y_i y_j}$ then $\overline{y_i y_j} \cap hreg(P) = \emptyset$.*

The time complexity depends on the number of mixed vertex events that get generated and the time to produce them. In the following we concentrate in formally bounding this number.

Definition 4. *Let r be the horizontal ray extending backwards from the right-most point p_r of P. Let C_r be the circle centered on r passing through p_r that contains exactly one cluster in its interior (in the worst case that cluster is P). The cluster contained in C_r is called the* anchor *of P.*

Definition 5. *Let A be the anchor of P. If $A \neq P$, let Y_a be the nearest common ancestor of all rear vertices of $b_f(P, A)$ on $T(P)$. The P-circle centered at Y_a is called the* anchor circle *of P and it is denoted as $\mathcal{K}_a(P)$. If $A = P$, the anchor*

circle is the minimum enclosing circle of P. The priority of the anchor circle is priority(Y_a).

The anchor circle coincides with the minimum enclosing circle of P when $Y_0(P) \in hreg(P)$. In case of non-crossing clusters, the anchor circle of P is the minimum radius P-circle entirely enclosing both P and its anchor. In the worst case, the anchor circle of P coincides with the minimum priority circle of P, in which case $Y_a = Y_0$ and $T_c(Y_a) = \emptyset$. Since any component of $hreg(P) \cap T(P)$ must be bounded by a rear mixed vertex or Y_0, $T_c(Y_a) \cap hreg(P) = \emptyset$. For the sake of formally bounding the number of mixed vertex events, we can add the following step when handling the invalid site event corresponding to $Y_0(P)$, in order to ensure that no mixed vertex events get generated for $T_c(Y_a)$: Determine the anchor of P and produce a mixed vertex event for Y_a (if Y_a does not coincide with a vertex of $T(P)$); eliminate $T_c(Y_a)$ i.e., delete all vertex events of $T_c(Y_a)$ from the event queue. The correctness of the algorithm is not affected by the addition of this step.

Lemma 8. *The number of mixed vertex events generated throughout the algorithm is $O(K + m)$, where $K = \Sigma_{P \in S} K(P)$, $K(P)$ is the number of clusters entirely enclosed in the anchor circle of P, and m is as defined in Theorem 1.*

Proof. We have two types of mixed vertex events: *crossing* and *non-crossing*. Let m_i be a non-crossing mixed vertex event induced by Q on $T(P)$. By Lemma 2, m_i is unique as Q can induce at most one rear non-crossing mixed vertex on $T(P)$. Since $Q \in \mathcal{K}^r_{m_i} \cup CH(P)$ and $m_i \in T(Y_a)$, Q must be entirely enclosed in $\mathcal{K}_a(P)$ as $\mathcal{K}^r_{m_i} \subset \mathcal{K}_a(P)$. Thus, $O(K)$ bounds the total number of non-crossing mixed events. By construction the total number of crossing mixed vertex events is upper bounded by the the total number of crossing vertices on $b_h(P, Q)$ for any pair of crossing clusters (P, Q). But this number is $O(m)$ as it was shown in Theorem 1. □

Lemma 9. *The generation time for all mixed vertex events induced on $T(P)$ by a single cluster Q is $O(|CH(Q) \cap \mathcal{K}_a(P)|)$.*

Proof. The claim is easy to see for a non-crossing Q as $Q \in \mathcal{K}_a(P)$ and Q can induce at most one mixed event on $T(P)$ (see Lemma 8). If Q is crossing with P then Q may induce several crossing mixed vertex events on $T(P)$. However, we claim that any $q_r \in Q$ that gets visited during the generation of a single mixed vertex event m_i, cannot be visited again during the generation of another mixed vertex event on $T(P)$. Let $q_r \in Q$ be visited during the generation of $m_i \in \overline{y_i y_j} \in T(P)$, that is, $freg(q_r)$ is intersected by $\overline{m_i y_j}$. (Recall that the traversal of $\overline{y_i y_j}$ starts at y_j and y_i is an ancestor of y_j in $T(P)$). Then $q_r \in \mathcal{K}^r_{y_i}$ but $q_r \notin \mathcal{K}_y$ for any $y \in \overline{m_i y_j} \cap freg(q_r)$. Thus, $q_r \notin \mathcal{K}^r_v$ for any $v \in T(y_j)$ since $\mathcal{K}^r_v \subset \mathcal{K}^r_{y_j}$ (Lemma 1). Furthermore, $q_r \notin \mathcal{K}^r_u$ for any $u \in T_c(y_i)$ that is not an ancestor of y_i, since $\mathcal{K}^r_{y_i} \cap \mathcal{K}^r_u = \emptyset$. Thus, q_r cannot be considered again during the generation of another mixed vertex event on $T(P)$. Since $m_i \in T(Y_a)$, $q_r \in \mathcal{K}^r_{y_i} \subset \mathcal{K}_a(P)$. Thus, only points in $CH(Q) \cap \mathcal{K}_a(P)$ can be visited. □

Theorem 2. *H-Vor(S) can be computed in* $O(M + (n + m + K)\log n)$ *time by plane sweep, where* n, m *are as defined in Theorem 1,* K *is as defined in Lemma 8, and* $M = \Sigma_{P \in S} M(P)$*, where* $M(P)$ *is the total number of points* $q \in CH(Q)$ *that are enclosed in the anchor circle of* P *such that either* Q *is entirely contained in* $\mathcal{K}_a(P)$ *or* Q *is crossing with* P*.*

Proof. By Lemma 9, $O(M)$ time is attributed to the generation of mixed vertex events. The theorem follows from Lemma 8 and the fact that $O(\log n)$ time is spent for each event (without counting time for generation of mixed events). □

Whether the terms K, M can be eliminated from the time complexity remains an open problem. In our VLSI application shapes are rectilinear in nature, well spaced, and in their majority non-crossing. A small number of crossings may be present due to non-neighboring redundant vias. In this setting both K, M remain small. In the simpler L_∞ non-crossing case, experimental results on VLSI via-layers indicated that K was negligible compared to n [7].

References

1. M. Abellanas, G. Hernandez, R. Klein, V. Neumann-Lara, and J. Urrutia, "A Combinatorial Property of Convex Sets", *Discrete Computat. Geometry* 17, 1997, 307-318.
2. F. Dehne and R. Klein, "The Big Sweep": On the power of the Wavefront Approach to Voronoi Diagrams", *Algorithmica*, 17, 19-32, 1997.
3. H. Edelsbrunner, L.J. Guibas, and M. Sharir, "The upper envelope of piecewise linear functions: algorithms and applications", *Discrete Computat. Geometry* 4, 1989, 311-336.
4. S. J. Fortune, "A sweepline algorithm for Voronoi diagrams", *Algorithmica*, 2, 1987, 153-174.
5. R. Klein, K. Mehlhorn, S. Meiser, "Randomized Incremental Construction of Abstract Voronoi diagrams", *Computational geometry: Theory and Applications* 3,1993, 157-184
6. W. Maly, "Computer Aided Design for VLSI Circuit Manufacturability," *Proc. IEEE*, vol.78, no.2, 356-392, Feb. 90.
7. E. Papadopoulou, "Critical Area Computation for Missing Material Defects in VLSI Circuits", *IEEE Transactions on Computer-Aided Design*, vol. 20, no.5, May 2001, 583-597.
8. E. Papadopoulou and D.T. Lee, "Critical Area Computation via Voronoi Diagrams", *IEEE Trans. on Computer-Aided Design*, vol. 18, no.4, April 1999,463-474.
9. E. Papadopoulou and D.T. Lee, 'The Min-Max Voronoi diagram of polygonal objects and applications in VLSI manufacturing", *Proc. 13th Int. Symp. on Algorithms and Computation*, Nov 2002, *LNCS 2518*, 511-522.
10. Preparata, F. P. and M. I. Shamos, *Computational Geometry: an Introduction*, Springer-Verlag, New York, NY 1985.
11. H. Walker and S.W. Director, " VLASIC: A yield simulator for integrated circuits," *IEEE Trans. on Computer-Aided Design*, CAD-5,4, 541-556, Oct. 1986.

Output-Sensitive Algorithms for Computing Nearest-Neighbour Decision Boundaries*

David Bremner[1], Erik Demaine[2], Jeff Erickson[3], John Iacono[4], Stefan Langerman[5], Pat Morin[6], and Godfried Toussaint[7]

[1] Faculty of Computer Science, University of New Brunswick, bremner@unb.ca
[2] MIT Laboratory for Computer Science, edemaine@mit.edu
[3] Computer Science Department, University of Illinois, jeffe@cs.uiuc.edu
[4] Polytechnic University, jiacono@poly.edu
[5] Chargé de recherches du FNRS, Université Libre de Bruxelles, stefan.langerman@ulb.ac.be
[6] School of Computer Science, Carleton University, morin@cs.carleton.ca
[7] School of Computer Science, McGill University, godfried@cs.mcgill.ca

Abstract. Given a set R of red points and a set B of blue points, the *nearest-neighbour decision rule* classifies a new point q as red (respectively, blue) if the closest point to q in $R \cup B$ comes from R (respectively, B). This rule implicitly partitions space into a red set and a blue set that are separated by a red-blue *decision boundary*. In this paper we develop output-sensitive algorithms for computing this decision boundary for point sets on the line and in $\mathbb{R}^2$. Both algorithms run in time $O(n \log k)$, where k is the number of points that contribute to the decision boundary. This running time is the best possible when parameterizing with respect to n and k.

1 Introduction

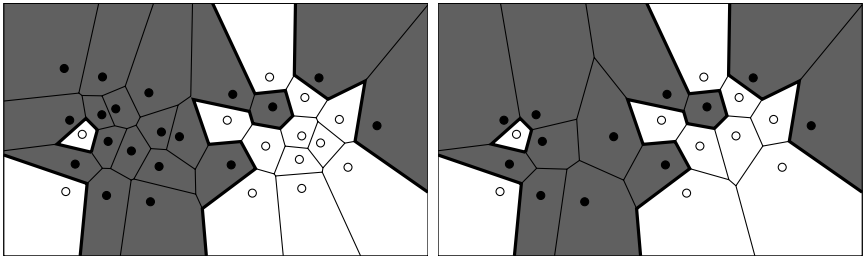
Let S be a set of n points in the plane that is partitioned into a set of *red points* denoted by R and a set of *blue points* denoted by B. The *nearest-neighbour decision rule* classifies a new point q as the color of the closest point to q in S. The nearest-neighbour decision rule is popular in pattern recognition as a means of learning by example. For this reason, the set S is often referred to as a *training set*.

Several properties make the nearest-neighbour decision rule quite attractive, including its intuitive simplicity and the theorem that the asymptotic error rate of the nearest-neighbour rule is bounded from above by twice the Bayes error rate [6,8,16]. (See [17] for an extensive survey of the nearest-neighbour decision rule and its relatives.) Furthermore, for point sets in small dimensions, there are efficient and practical algorithms for preprocessing a set S so that the nearest neighbour of a query point q can be found quickly.

* This research was partly funded by the Alexander von Humboldt Foundation and The Natural Sciences and Engineering Research Council of Canada.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 451–461, 2003.
© Springer-Verlag Berlin Heidelberg 2003

The nearest-neighbour decision rule implicitly partitions the plane into a red set and a blue set that meet at a red-blue *decision boundary*. One attractive aspect of the nearest-neighbour decision rule is that it is often possible to reduce the size of the training set S without changing the decision boundary. To see this, consider the *Voronoĭ diagram* of S, which partitions the plane into convex (possibly unbounded) polygonal *Voronoĭ cells*, where the Voronoĭ cell of point $p \in S$ is the set of all points that are closer to p than to any other point in S (see Figure 1.a). If the Voronoĭ cell of a red point r is completely surrounded by the Voronoi cells of other red points then the point r can be removed from S and this will not change the classification of any point in the plane (see Figure 1.b). We say that these points *do not contribute* to the decision boundary, and the remaining points *contribute* to the decision boundary.



(a) (b)

Fig. 1. The Voronoĭ diagram (a) before Voronoĭ condensing and (b) after Voronoĭ condensing. Note that the decision boundary (in bold) is unaffected by Voronoĭ condensing. Note: In this figure, and all other figures, red points are denoted by white circles and blue points are denoted by black disks.

The preceding discussion suggests that one approach to reducing the size of the training set S is to simply compute the Voronoĭ diagram of S and remove any points of S whose Voronoĭ cells are surrounded by Voronoĭ cells of the same color. Indeed, this method is referred to as *Voronoĭ condensing* [18]. There are several $O(n \log n)$ time algorithms for computing the Voronoĭ diagram a set of points in the plane, so Voronoĭ condensing can be implemented to run in $O(n \log n)$ time.[1] However, in this paper we show that we can do significantly better when the number of points that contribute to the decision boundary is small. Indeed, we show how to do Voronoĭ condensing in $O(n \log k)$ time, where k is the number of points that contribute to the decision boundary (i.e., the number of points of S that remain after Voronoĭ condensing). Algorithms, like these, in which the size of the input and the size of the output play a role in the running time are referred to as *output-sensitive* algorithms.

[1] Historically, the first efficient algorithm for specifically computing the nearest-neighbour decision boundary is due to Dasarathy and White [7] and runs in $O(n^4)$ time. The first $O(n \log n)$ time algorithm for computing the Voronoĭ diagram of a set of n points in the plane is due to Shamos [15].

Readers familiar with the literature on output-sensitive convex hull algorithms may recognize the expression $O(n \log k)$ as the running time of optimal algorithms for computing convex hulls of n point sets with k extreme points, in 2 or 3 dimensions [2,4,5,13,19]. This is no coincidence. Given a set of n points in $\mathbb{R}^2$, we can color them all red and add three blue points at infinity (see Figure 2). In this set, the only points that contribute to the nearest-neighbour decision boundary are the three blue points and the red points on the convex hull of the original set. Thus, identifying the points that contribute to the nearest-neighbour decision boundary is at least as difficult as computing the extreme points of a set.

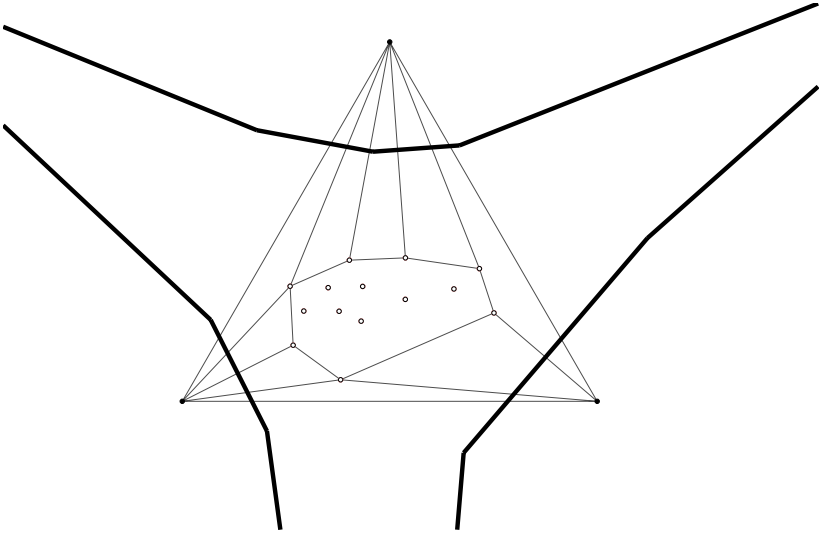


Fig. 2. The relationship between convex hulls and decision boundaries. Each vertex of the convex hull of R contributes to the decision boundary.

Observe that, once the size of the training set has been reduced by Voronoĭ codensing, the condensed set can be preprocessed in $O(k \log k)$ time to answer nearest neighbour queries in $O(\log k)$ time per query. This makes it possible to do nearest-neighbour classifications in $O(\log k)$ time. Alternatively, the algorithm we describe for computing the nearest neighbour decision boundary actually produces an explicit description of the boundary (of size $O(k)$) that can be preprocessed in $O(k)$ time by Kirkpatrick's point-location algorithm [12] to allow nearest neighbour classification in $O(\log k)$ time.

The remainder of this paper is organized as follows: In Section 2 we describe an algorithm for computing the nearest-neighbour decision boundary of points on a line that runs in $O(n \log k)$ time. In Section 3 we present an algorithm for points in the plane that also runs in $O(n \log k)$ time. Finally, in Section 4 we summarize and conclude with open problems.

2 A 1-Dimensional Algorithm

In the 1-dimensional version of the nearest-neighbour decision boundary problem, the input set S consists of n real numbers. Imagine sorting S, so that $S = \{s_1, \ldots, s_n\}$ where $s_i < s_{i+1}$ for all $1 \leq i < n$. The decision boundary consists of all pairs (s_i, s_{i+1}) where s_i is red and s_{i+1} is blue, or *vice-versa*. Thus, this problem is solveable in linear-time if the points of S are sorted. Since sorting the elements of S can be done using any number of $O(n \log n)$ time sorting algorithms, this immediately implies an $O(n \log n)$ time algorithm. Next, we give an algorithm that runs in $O(n \log k)$ time and is similar in spirit to Hoare's quicksort [11].

To find the decision boundary in $O(n \log k)$ time, we begin by computing the median element $m = s_{\lceil n/2 \rceil}$ in $O(n)$ time using any one of the existing linear-time median finding algorithms (see [3]). Using an additional $O(n)$ time, we split S into the sets $S_1 = \{s_1, \ldots, s_{\lceil n/2 \rceil - 1}\}$ and $S_2 = \{s_{\lceil n/2 \rceil + 1}, \ldots, s_n\}$ by comparing each element of S to the median element m. At the same time we also find $s_{\lceil n/2 \rceil - 1}$ and $s_{\lceil n/2 \rceil + 1}$ by finding the maximum and minimum elements of S_1 and S_2, respectively. We then check if $(s_{\lceil n/2 \rceil - 1}, m)$ and/or $(m, s_{\lceil n/2 \rceil + 1})$ are part of the decision boundary and report them if necessary.

At this point, a standard divide-and-conquer algorithm would recurse on both S_1 and S_2 to give an $O(n \log n)$ time algorithm. However, we can improve on this by observing that it is not necessary to recurse on a subproblem if it contains only elements of one color, since it will not contribute a pair to the decision boundary. Therefore, we recurse on each of S_1 and S_2 only if they contain at least one red element and one blue element.

The correctness of the above algorithm is clear. To analyze its running time we observe that the running time is bounded by the recurrence

$$T(n, k) \leq O(n) + T(n/2, l) + T(n/2, k - l) \quad ,$$

where l is the number of points that contribute to the decision boundary in S_1 and where $T(1, k) = O(1)$ and $T(n, 0) = O(n)$. An easy inductive argument that uses the concavity of the logarithm shows that this recurrence is maximized when $l = k/2$, in which case the recurrence solves to $O(n \log k)$ [5].

Theorem 1 *The nearest-neighbour decision boundary of a set of n real numbers can be computed in $O(n \log k)$ time, where k is the number of elements that contribute to the decision boundary.*

3 A 2-Dimensional Algorithm

In the 2-dimensional nearest-neighbour decision boundary problem the Voronoĭ cells of S are (possibly unbounded) convex polygons and the goal is to find all Voronoĭ edges that bound two cells whose defining points have different colors. Throughout this section we will assume that the points of S are in *general position* so that no four points of S lie on a common circle. This assumption is not very restrictive, since general position can be simulated using infinitesmal perturbations of the input points.

It will be more convenient to present our algorithm using the terminology of Delaunay triangulations. A *Delaunay triangle* in S is a triangle whose vertices (v_1, v_2, v_3) are in S and such that the circle with v_1, v_2 and v_3 on its boundary does not contain any point of S in its interior. A *Delaunay triangulation* of S is a partitioning of the convex hull of S into Delaunay triangles. Alternatively, a *Delaunay edge* is a line segment whose vertices (v_1, v_2) are in S and such that there exists a circle with v_1 and v_2 on its boundary that does not contain any point of S in its interior. When S is in general position, the Delaunay triangulation of S is unique and contains all triangles whose edges are Delaunay edges (see [14]). It is well known that the Delaunay triangulation and the Voronoi diagram are dual in the sense that two points of S are joined by an edge in the Delaunay triangulation if and only if their Voronoi cells share an edge.

We call a Delaunay triangle or Delaunay edge *bichromatic* if its set of defining vertices contains at least one red and at least one blue point of S. Thus, the problem of computing the nearest-neighbour decision boundary is equivalent to the problem of finding all bichromatic Delaunay edges.

3.1 The High Level Algorithm

In the next few sections, we will describe an algorithm that, given a value $\kappa \geq k$, finds the set of all bichromatic Delaunay triangles in S in $O((\kappa^2 + n) \log \kappa)$ time, which for $\kappa \leq \sqrt{n}$ simplifies to $O(n \log \kappa)$. To obtain an algorithm that runs in $O(n \log k)$ time, we repeatedly guess the value of κ, run the algorithm until we find the entire decision boundary or until it determines that $\kappa < k$ and, in the latter case, restart the algorithm with a larger value of κ. If we ever reach a point where the value of κ exceeds $\sqrt{n}$ then we stop the entire algorithm and run an $O(n \log n)$ time algorithm to compute the entire Delaunay triangulation of S.

The values of κ that we use are $\kappa = 2^{2^i}$ for $i = 0, 1, 2, \ldots, \lceil \log \log n \rceil$. Since the algorithm will terminate once $\kappa \geq k$ or $\kappa \geq \sqrt{n}$, the total cost of all runs of the algorithm is therefore

$$T(n,k) = \sum_{i=0}^{\lceil \log \log k \rceil} O(n \log 2^{2^i}) = \sum_{i=0}^{\lceil \log \log k \rceil} O(n 2^i) = O(n \log k) \ ,$$

as required.

3.2 Pivots

A key subroutine in our algorithm is the *pivot*[2] operation illustrated in Figure 3. A pivot in the set of points S takes as input a ray and reports the largest circle

[2] The term pivot comes from linear programming. The relationship between a (polar dual) linear programming pivot and the circular pivot described here is evident when we consider the parabolic lifting that transforms the problem of computing a 2-dimensional Delaunay triangulation to that of computing a 3-dimensional convex hull of a set of points on the paraboloid $z = x^2 + y^2$. In this case, the circle is the projection of the intersection of a plane with the paraboloid.

whose center is on the ray, has the origin of the ray on its boundary and has no point of S in its interior. We will make use of the following data structuring result, due to Chan [4]. For completeness, we also include a proof.

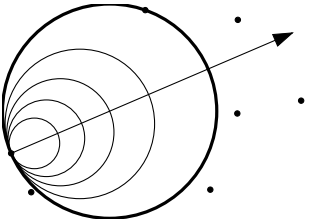


Fig. 3. A pivot operation.

Lemma 1 (Chan 1996) *Let S be a set of n points in $\mathbb{R}^2$. Then, for any integer $1 \leq m \leq n$, there exists a data structure of size $O(n)$ that can be constructed in $O(n \log m)$ time, and that can perform pivots in S in $O(\frac{n}{m} \log m)$ time per pivot.*

Proof. Dobkin and Kirkpatrick [9,10] show how to preprocess a set S of n points in $O(n \log n)$ time to answer pivot queries in $O(\log n)$ time per query. Chan's data structure simply partitions S into n/m groups each of size m and then uses the Dobkin-Kirkpatrick data structure on each group. The time to build all n/m data structures is $\frac{n}{m} \times O(m \log m) = O(n \log m)$. To perform a query, we simply query each of the n/m data structures in $O(\log m)$ time per data structure and report the smallest circle found, for a query time of $\frac{n}{m} \times O(\log m) = O(\frac{n}{m} \log m)$.

In the following, we will be using Lemma 1 with a value of $m = \kappa^2$, so that the time to construct the data structure is $O(n \log \kappa)$ and the query time is $O(\frac{n}{\kappa^2} \log \kappa)$. We will use two such data structures, one for performing pivots in the set R of red points and one for performing pivots in the set B of blue points.

3.3 Finding the First Edge

The first step in our algorithm is to find a single bichromatic edge of the Delaunay triangulation. Refer to Figure 4. To do this, we begin by choosing any red point r and any blue point b. We then perform a pivot in the set B along the ray with origin r that contains b. This gives us a circle C that has no blue points in its interior and has r as well as some blue point b' (possibly $b = b'$) on its boundary. Next, we perform a pivot in the set R along the ray originating at b' and passing through the center of C. This gives us a circle C_1 that has no point of S in its interior and has b' and some red point r' (possibly $r = r'$) on its boundary. Therefore, (r', b') is a bichromatic edge in the Delaunay triangulation of S.

The above argument shows how to find a bichromatic Delaunay edge using only 2 pivots, one in R and one in B. The second part of the argument also implies the following useful lemma.

Lemma 2 *If there is a circle with a red point r and a blue point b on its boundary, and no red (respectively, blue) points in its interior, then r (respectively, b) contributes to the decision boundary.*

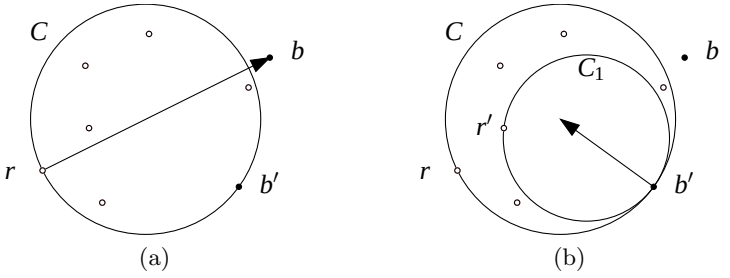


Fig. 4. The (a) first and (b) second pivot used to find a bichromatic edge (r', b').

3.4 Finding More Points

Let Q be the set of points that contribute to the decision boundary, i.e., the set of points that are the vertices of bichromatic triangles in the Delaunay triangulation of S. Suppose that we have already found a set $P \subseteq Q$ and we wish to either (1) find a new point $p \in Q \setminus P$ or (2) verify that $P = Q$.

To do this, we will make use of the *augmented Delaunay triangulation* of P (see Figure 5). This is the Delaunay triangulation of $P \cup \{v_1, v_2, v_3\}$, where v_1, v_2, and v_3 are three *black* points "at infinity" (see Figure 5). For any triangle t, we use the notation $C(t)$ to denote the circle whose boundary contains the three vertices of t (note that if t contains a black point then $C(t)$ is a halfplane). The following lemma allows us to tell when we have found the entire set of points Q that contribute to the decision boundary.

Lemma 3 *Let $\emptyset \neq P \subseteq Q$. The following statements are equivalent:*

1. *For every triangle t in the augmented Delaunay triangulation of P, if t has a blue (respectively, red) vertex then $C(t)$ does not have a red (respectively, blue) point of S in its interior.*
2. *$P = Q$.*

Proof. First we show that if Statement 1 of the lemma is not true, then Statement 2 is also not true, i.e., $P \neq Q$. Suppose there is some triangle t in the augmented Delaunay triangulation of P such that t has a blue vertex b and $C(t)$ contains a red point of S in its interior. Pivot in R along the ray originating at b and passing through the center of $C(t)$ (see Figure 6). This will give a circle C with b and some red point $r \notin P$ on its boundary and with no red points inits interior. Therefore, by Lemma 2, r contributes to the decision boundary and is therefore in Q, so $P \neq Q$. A symmetric argument applies when t has a red vertex r and $C(t)$ contains a blue vertex in its interior.

Next we show that if Statement 2 of the lemma is not true then Statement 1 is not true. Suppose that $P \neq Q$. Let r be a point in $Q \setminus P$ and, without loss of generality, assume r is a red point. Since r is in Q, there is a circle C with r and some other blue point b on its boundary and with no points of S in its interior. We will use r and b to show that the augmented Delaunay triangulation of P contains a triangle t such that either (1) b is a vertex of t and $C(t)$ contains r in its interior, or (2) $C(t)$ contains both r and b in its interior. In either case, Statement 1 of the lemma is not true because of triangle t.

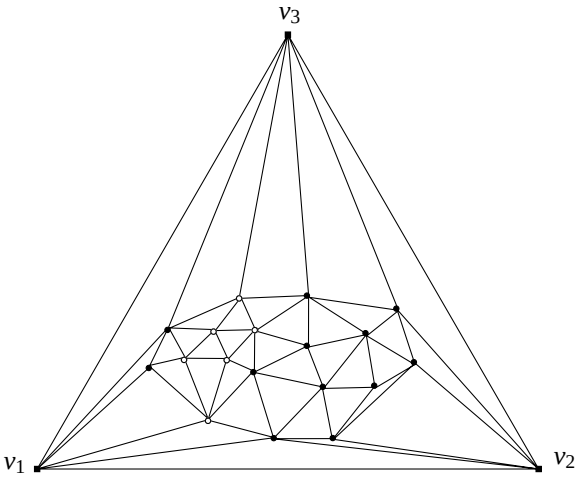


Fig. 5. The augmented Delaunay triangulation of S.

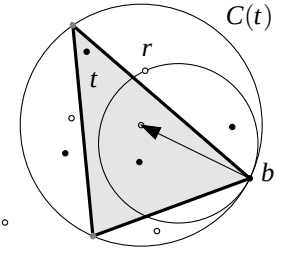


Fig. 6. If Statement 1 of Lemma 3 is not true then $P \neq Q$.

Refer to Figure 7 for what follows. Consider the largest circle C_1 that is concentric with C and that contains no point of P in its interior (this circle is at least as large as C). The circle C_1 will have at least one point p_1 of P on its boundary (it could be that $p_1 = b$, if $b \in P$). Next, perform a pivot in P along the ray originating at p_1 and containing the center of C_1. This will give a circle C_2 that contains C_1 and with two points p_1 and p_2 of $P \cup \{v_1, v_2, v_3\}$ on its boundary and with no points of $P \cup \{v_1, v_2, v_3\}$ in its interior. Therefore, (p_1, p_2) is an edge in the augmented Delaunay triangulation of P.

The edge (p_1, p_2) partitions the interior of C_2 into two pieces, one that contains r and one that does not. It is possible to move the center of C_2 along the perpendicular bisector of (p_1, p_2) maintaining p_1 and p_2 on the boundary of C_2. There are two directions in which the center of C_2 can be moved to accomplish this. In one direction, say $\overrightarrow{d}$, the part of the interior that contains r only increases, so move the center in this direction until a third point $p_3 \in P \cup \{v_1, v_2, v_3\}$ is on the boundary of C_2. The resulting circle has the points p_1, p_2, and p_3 on its boundary and no points of P in its interior, so p_1, p_2 and p_3 are the vertices of a triangle t in the augmented Delaunay triangulation of P. The circumcircle $C(t)$

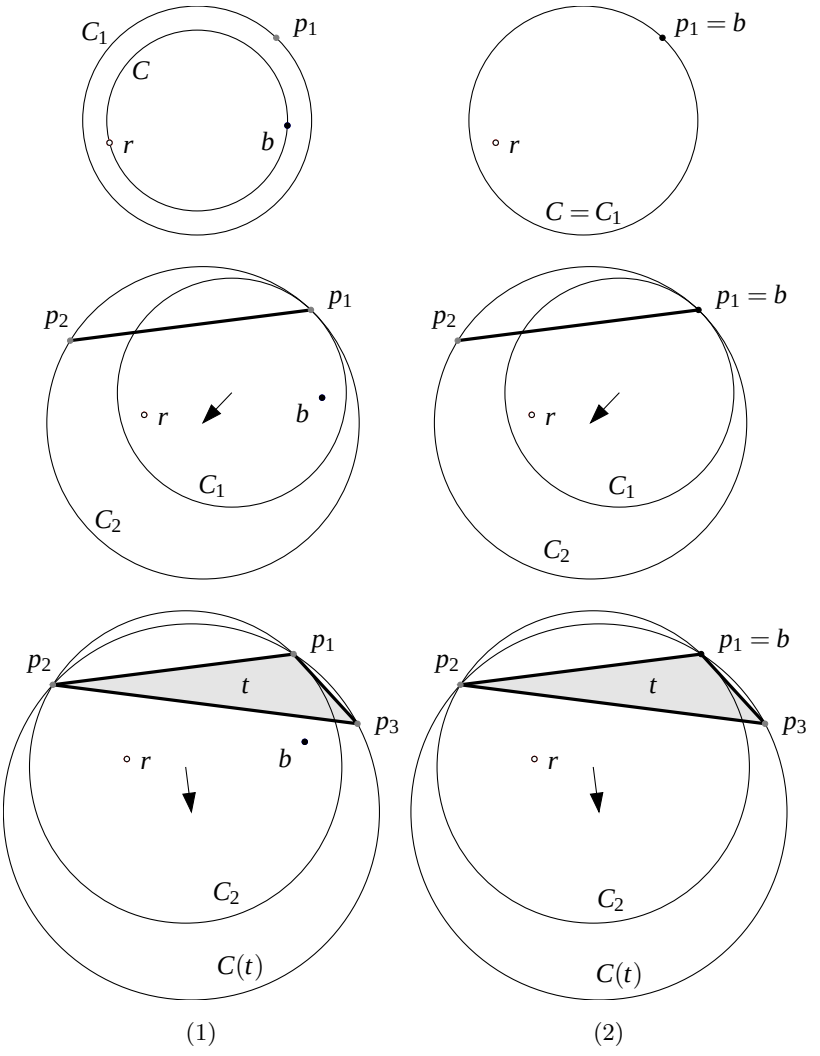


Fig. 7. If $P \neq Q$ then Statement 1 of Lemma 3 is not true. The left column (1) corresponds to the case where $b \notin P$ and the right column (2) corresponds to the case where $b \in P$.

contains r in its interior and contains b either in its interior or on its boundary. In either case, t contradicts Statement 1, as promised.

Note that the first paragraph in the proof of Lemma 3 gives a method of testing whether $P = Q$, and when this is not the case, of finding a point in $Q \setminus P$. For each triangle t in the Delaunay triangulation of P, if t contains a blue vertex b then perform a pivot in R along the ray originating at b and passing through $C(t)$. If the result of this pivot is $C(t)$, then do nothing. Otherwise, the pivot finds a circle C with no red points in its interior and that has one blue

point b and one red point $r \notin P$ on its boundary. By Lemma 2, the point r must be in Q. If t contains a red vertex, repeat the above procedure swapping the roles of red and blue. If both pivots (from the red point and the blue point) find the circle $C(t)$, then we have verified Statement 1 of Lemma 3 for the triangle t.

The above procedure performs at most two pivots for each triangle t in the augmented Delaunay triangulation of P. Therefore, this procedure performs $O(|P|) = O(\kappa)$ pivots. Since we repeat this procedure at most κ times before deciding that $\kappa < k$, we perform $O(\kappa^2)$ pivots, at a total cost of $O(\kappa^2 \times \frac{n}{\kappa^2} \log \kappa) = O(n \log \kappa)$. The only other work done by the algorithm is that of recomputing the augmented Delaunay triangulation of P each time we add a new vertex to P. Since each such computation takes $O(|P| \log |P|)$ time and $|P| \leq \kappa$, the total amount of work done in computing all these triangulations is $O(\kappa^2 \log \kappa)$.

In summary, we have an algorithm that given S and κ decides whether the condensed set Q of points in S that contribute to the decision boundary has size at most κ, and if so, computes Q. This algorithm runs in $O((\kappa^2 + n) \log \kappa)$ time. By trying increasingly large values of κ as described in Section 3.1 we obtain our main theorem.

Theorem 2 *The nearest-neighbour decision boundary of a set of n points in $\mathbb{R}^2$ can be computed in $O(n \log k)$ time, where k is the number of points that contribute to the decision boundary.*

Remark: Theorem 2 extends to the case where there are more than 2 color classes and our goal is to find all Voronoĭ edges bounding two cells of different color. The only modification required is that, for each color class, R, we use two pivoting data structures, one for R and one for $S \setminus R$. When performing pivots from a point in R, we use the data structure for pivots in $S \setminus R$. Otherwise, the details of the algorithm are identical.

Remark: In the pattern-recognition community pattern classification rules are often implemented as neural networks. In the terminology of neural networks, Theorem 2 states that it is possible, in $O(n \log k)$ time, to design a simple one-layer neural network that implements the nearest-neighbour decision rule and uses only k McCulloch-Pitts neurons (threshold logic units).

4 Conclusions

We have given $O(n \log k)$ time algorithms for computing nearest-neighbour decisions boundaries in 1 and 2 dimensions, where k is the number of points that contribute to the decision boundary. A standard application of Ben-Or's lower-bound technique [1] shows that even the 1-dimensional algorithm is optimal in the algebraic decision tree model of computation.

We have not studied algorithms for dimensions $d \geq 3$. In this case, it is not even clear what the term "output-sensitive" means. Should k be the number of points that contribute to the decision boundary, or should k be the complexity of the decision boundary? In the first case, $k \leq n$ for any dimension d, while in

the second case, k could be as large as $\Omega(n^{\lceil d/2 \rceil})$. To the best of our knowledge, both are open problems.

References

1. M. Ben-Or. Lower bounds for algebraic computation trees (preliminary report). In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 80–86, 1983.
2. B. K. Bhattacharya and S. Sen. On a simple, practical, optimal, output-sensitive randomized planar convex hull algorithm. *Journal of Algorithms*, 25(1):177–193, 1997.
3. M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computing and Systems Science*, 7:448–461, 1973.
4. T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16:361–368, 1996.
5. T. M. Chan, J. Snoeyink, and C. K. Yap. Primal dividing and dual pruning: Output-sensitive construction of four-dimensional polytopes and three-dimensional Voronoi diagrams. *Discrete & Computational Geometry*, 18:433–454, 1997.
6. T. M. Cover and P. E. Hart. Nearest neighbour pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.
7. B. Dasarathy and L. J. White. A characterization of nearest-neighbour rule decision surfaces and a new approach to generate them. *Pattern Recognition*, 10:41–46, 1978.
8. L. Devroye. On the inequality of Cover and Hart. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3:75–78, 1981.
9. D. P. Dobkin and D. G. Kirkpatrick. Fast detection of poyhedral intersection. *Theoretical Computer Science*, 27:241–253, 1983.
10. D. P. Dobkin and D. G. Kirkpatrick. A linear algorithm for determining the separation of convex polyhedra. *Journal of Algorithms*, 6:381–392, 1985.
11. C. A. R. Hoare. ACM Algorithm 64: Quicksort. *Communications of the ACM*, 4(7):321, 1961.
12. D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.
13. D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15(1):287–299, 1986.
14. F. P Preparata and M. I. Shamos. *Computational Geometry*. Springer-Verlag, 1985.
15. M. I. Shamos. Geometric complexity. In *Proceedings of the 7th ACM Symposium on the Theory of Computing (STOC 1975)*, pages 224–253, 1975.
16. C. Stone. Consistent nonparametric regression. *Annals of Statistics*, 8:1348–1360, 1977.
17. G. T. Toussaint. Proximity graphs for instance-based learning. Manuscript, 2003.
18. G. T. Toussaint, B. K. Bhattacharya, and R. S. Poulsen. The application of Voronoi diagrams to non-parametric decision rules. In *Proceedings of Computer Science and Statistics: 16th Symposium of the Interface*, 1984.
19. R. Wenger. Randomized quick hull. *Algorithmica*, 17:322–329, 1997.

Significant-Presence Range Queries in Categorical Data

Mark de Berg[1] and Herman J. Haverkort[2]

[1] Department of Computer Science, TU Eindhoven,
P.O.Box 513, 5600 MB Eindhoven, The Netherlands, m.t.d.berg@tue.nl
[2] Institute of Information and Computing Sciences, Utrecht University,
P.O.Box 80.089, 3508 TB Utrecht, The Netherlands, herman@cs.uu.nl

Abstract. In traditional colored range-searching problems, one wants to store a set of n objects with m distinct colors for the following queries: report all colors such that there is at least one object of that color intersecting the query range. Such an object, however, could be an 'outlier' in its color class. Therefore we consider a variant of this problem where one has to report only those colors such that at least a fraction τ of the objects of that color intersects the query range, for some parameter τ. Our main results are on an approximate version of this problem, where we are also allowed to report those colors for which a fraction $(1-\varepsilon)\tau$ intersects the query range, for some fixed $\varepsilon > 0$. We present efficient data structures for such queries with orthogonal query ranges in sets of colored points, and for point stabbing queries in sets of colored rectangles.

1 Introduction

Motivation. The range-searching problem is one of the most fundamental problems in computational geometry. In this problem we wish to construct a data structure on a set S of objects in $\mathbb{R}^d$, such that we can quickly decide for a query range which of the input objects it intersects. The range-searching problem comes in many flavors, depending on the type of objects in the input set S, on the type of allowed query ranges, and on the required output (whether one wants to report all intersected objects, to count the number of intersected objects, etc.). The range-searching problem is not only interesting because it is such a fundamental problem, but also because it arises in numerous applications in areas like databases, computer graphics, geographic information systems, and virtual reality. Hence, it is not surprising that there is an enormous literature on the subject—see for instance the surveys by Agarwal [1], Agarwal and Erickson [2], and Nievergelt and Widmayer [7].

In this paper, we are interested in range searching in the context of databases. Here one typically wants to be able to answer questions like: given a database of customers, report all customers whose ages are between 20 and 30, and whose income is between \$50,000 and \$75,000. In this example, the customers can be

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 462–473, 2003.
© Springer-Verlag Berlin Heidelberg 2003

represented as points in $\mathbb{R}^2$, and the query range is an axis-parallel rectangle.[1] This is called the (planar) *orthogonal range-searching problem*, and it has been studied extensively—see the surveys [1,2,7] mentioned earlier.

There are situations, however, where the data points are not all of the same type but fall into different categories. Suppose, for instance, that we have a database of stocks. Each stock falls into a certain category, namely the industry sector it belongs to—energy, banking, food, chemicals, etc. Then it can be interesting for an analyst to get answers to questions like: "In which sectors companies had a 10–20% increase in their stock values over the past year?" In this simple example, the input can be seen as points in 1D (namely for each stock its increase in value), and the query is a 1-dimensional range-searching query.

Now we are no longer interested in reporting all the points in the range, but in reporting only the categories that have points in the range. This means that we would like to have a data structure whose query time is not sensitive to the total number of points in the range, but to the total number of categories in the range. This can be achieved by building a suitable data structure for each category separately, but this is inefficient if the number of categories is large. This has led researchers to study so-called *colored range-searching problems*: store a given set of colored objects—the color of an object represents its category—such that one can efficiently report those colors that have at least one object intersecting a query range [3,6,8,9].

We believe, however, that this is not always the correct abstracted version of the range-searching problem in categorical data. Consider for instance the stock example sketched earlier. The standard colored range-searching data structures would report all sectors that have *at least one* company whose increase in stock value lies in the query range. But this does not necessarily say anything about how the sector is performing: a given sector could be doing very badly in general, but contain a single 'outlier' whose performance has been good. It is much more natural to ask for all sectors for which *most* stocks, or at least a significant portion of them, had their values increase in a certain way. Therefore we propose a different version of the colored range-searching problem: given a fixed threshold parameter τ, with $0 < \tau < 1$, we wish to report all colors such that at least a fraction τ of the objects of that color intersect the query range. We call this a *significant-presence query*, as opposed to the standard *presence query* that has been studied before.

Problem statement and results. We study significant-presence queries in categorical data in two settings: orthogonal range searching where the data is a set of colored points in $\mathbb{R}^d$ and the query is a box, and stabbing queries where the data is a set of colored boxes in $\mathbb{R}^d$ and the query is a point. We now discuss our results on these two problems in more detail.

Let $S = S_1 \cup \cdots \cup S_m$ be a set of n points in $\mathbb{R}^d$, where m is the number of different colors and S_i is the subset of points of color class i. Let τ be a fixed

[1] From now on, whenever we use terms like "rectangle" or "box" we implicitly assume these are axis-parallel.

parameter with $0 < \tau < 1$. We are interested in answering significant-presence queries on S: given a query box Q, report all colors i such that $|Q \cap S_i| \geq \tau \cdot |S_i|$. For $d = 1$, we present a data structure that uses $O(n)$ storage, and that can answer significant-presence queries in $O(\log n + k)$ time, where k is the number of reported colors. Unfortunately, the generalization of our approach to higher dimensions leads to a data structure using already cubic storage in the planar case. To show this fact, we obtain the following result which is of independent interest. Let P be a set of n points in $\mathbb{R}^d$, and t a parameter with $1 \leq t \leq n/(2d)$. Then the maximum number of combinatorially distinct boxes containing exactly t points from P is $\Theta(n^d t^{d-1})$ in the worst case.

As a data structure with cubic storage is prohibitive in practice, we study an approximate version of the problem. More precisely, we study *ε-approximate significant-presence queries*: here we are required to report all colors i with $|Q \cap S_i| \geq \tau \cdot |S_i|$, but we are also allowed to report colors with $|Q \cap S_i| \geq (1-\varepsilon)\tau \cdot |S_i|$, where ε is a fixed positive constant. For such queries we develop a data structure that uses $O(M^{1+\delta})$ storage, for any $\delta > 0$, and that can answer such queries in $O(\log n + k)$ time, where $M = m/(\tau^{2d-2}\varepsilon^{2d-1})$ and k is the number of reported colors. We obtain similar results for the case where τ is not fixed, but part of the query—see Theorem 2. Note that the amount of storage does not depend on n, the total number of points, but only on m, the number of colors. This should be compared to the results for the previously considered case of presence queries on colored points sets. Here the best known results are: $O(n)$ storage with $O(\log n + k)$ query time for $d = 1$ [9], $O(n \log^2 n)$ storage with $O(\log n + k)$ query time for $d = 2$ [9], $O(n \log^4 n)$ storage with $O(\log^2 n + k)$ query time for $d = 3$ [8], and $O(n^{1+\delta})$ storage with $O(\log n + k)$ query time for $d \geq 4$ [3]. These bounds all depend on n, the total number of points; this is of course to be expected, since these results are all on the exact problem, whereas we allow ourselves approximate answers.

In the point-stabbing problem we are given a parameter τ and a set $B = B_1 \cup \cdots \cup B_m$ of n colored boxes in $\mathbb{R}^d$, and we wish, for a query point q, to report all colors i such that the number of boxes in B_i containing q is at least $\tau \cdot |B_i|$. We study the ε-approximate version of this problem, where we are also allowed to report colors such that the number of boxes containing q is at least $(1-\varepsilon)\tau \cdot |B_i|$. Our data structure for this case uses $O(M^{1+\delta})$ storage, for any $\delta > 0$, and it has $O(\log n + k)$ query time, where $M = m/(\tau\varepsilon)^d$. The best results for standard colored stabbing queries, where one has to report all colors with at least one box containing the query point, are as follows. For $d = 2$, there is a structure using $O(n \log n)$ storage with $O(\log^2 n + k)$ query time [8], and for $d > 2$ there is a structure using $O(n^{1+\delta})$ storage with $O(\log n + k)$ query time [3].

2 Orthogonal Range Queries

Our global approach is to first reduce significant-presence queries to standard presence queries. We do this by introducing so-called *test sets*.

2.1 Test Sets for Orthogonal Range Queries

Let P be a set of n points in $\mathbb{R}^d$, and let τ be a fixed parameter with $0 < \tau < 1$. A set T of boxes—that is, axis-parallel hyperrectangles—is called a *τ-test set* for P if:

1. any box from T contains at least τn points from P, and
2. any query box Q that contains at least τn points from P fully contains at least one box from T.

We call the boxes in T *test boxes*. We can answer a significant-presence query on P by answering a presence query on T: a query box Q contains at least τn points from P if and only if it contains at least one test box. This does not yet reduce the problem to a standard presence-query problem, because T contains boxes instead of points. However, like Agarwal *et al.* [3], we can map the set T of boxes in $\mathbb{R}^d$ to a set of points in $\mathbb{R}^{2d}$, and the query box Q to a box in $\mathbb{R}^{2d}$, in such a way that a box $b \in T$ is fully contained in Q if and only if its corresponding point in $\mathbb{R}^{2d}$ is contained in the transformed query box.[2] This means we can apply the results from the standard presence queries on colored point sets.

It remains to find small test sets. As it turns out, this is not possible in general: below we show that there are point sets that do not admit test sets of near-linear size. Hence, after studying the case of exact test sets, we will turn our attention to approximate test sets.

Exact test sets. Let t be a parameter with $1 \leq t \leq n$. Define a *t-box* to be a minimal box containing at least t points from P, that is, a box b containing at least t points such that there is no strictly smaller box $b' \subset b$ that contains t or more points. It is easy to see that any (τn)-box must be a test box, and that the collection of all (τn)-boxes forms a τ-test set. Hence, the smallest possible test set consists exactly of these (τn)-boxes.

In the 1-dimensional case a box is a segment, and a minimal segment is uniquely defined by the point from P that is its left endpoint. This means that any set of n points on the real line has a test set that has size $(1 - \tau)n + 1$. Unfortunately, the size of test sets increases rapidly with the dimension, as the next lemma shows.

Lemma 1. *For any set P of n points in $\mathbb{R}^d$, there is a τ-test set that has size $O(\tau^{d-1}n^{2d-1})$. Moreover, for some sets P, any τ-test set has size $\Omega(\tau^{d-1}n^{2d-1})$.*

Proof. By the observation made before, bounding the size of a test set boils down to bounding the number of (τn)-boxes. In this proof, when we use the term direction we mean one of the $2d$ directions $+x_1, -x_1, ..., +x_d, -x_d$. Let b be a (τn)-box, and let $D(b)$ be a set of points in b such that there is at least one point of $D(b)$ on each facet of b. If there are more such sets, let $D(b)$ be a set with minimum cardinality.

[2] In fact, the transformed query box is unbounded to one side along each coordinate-axis, so it is a d-dimensional 'octant'.

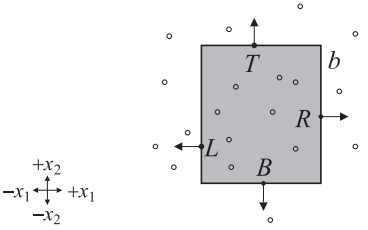


Fig. 1. Peeling a (τn)-box b in two dimensions ($\tau n = 12$). The black dots are the four points of $D(b)$. Initially, each point is extreme in only one direction, as indicated by the arrows. We can choose any of them, let us take T.

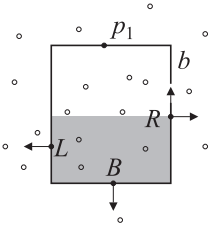


Fig. 2. For p_2, we cannot take R, since it is extreme in two directions among the remaining points of $D(b)$. We have to take one of the others, for example L.

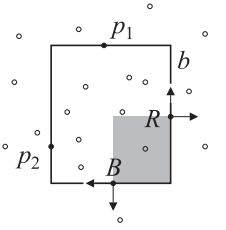


Fig. 3. Now, all remaining points of $D(b)$ are extreme in 2 directions: we stop peeling here. R and B together form the basis $D^*(b)$ of b. We conclude that b has a peeling sequence of type $+x_2, -x_1$.

The central concept in the proof is that of a peeling sequence, which is defined as follows: a *peeling sequence* for $D(b)$ is a sequence $p_1, p_2, ...$ of points from $D(b)$ with the following property: any p_i in the sequence is extreme in exactly one direction among the points in $D(b) - \{p_1, ..., p_{i-1}\}$. Ties are broken arbitrarily, i.e. if multiple points are extreme in the same direction, we appoint one of them to be the extreme point in that direction. The *type* of a peeling sequence is the sequence $\boldsymbol{d}_1, \boldsymbol{d}_2, ...$ of directions such that $\boldsymbol{d}_i$ is the unique direction in which p_i is extreme among $D(b) - \{p_1, ..., p_{i-1}\}$. Note that there are $(2d)!/(2d-\ell)! = O(1)$ different sequence types of a given length ℓ, so we have $O(1)$ different sequence types of length between 0 and d.

It is easy to see that there must be a peeling sequence $\sigma(b)$ of length $q = \max(0, |D(b)| - d)$: consider an incremental construction of the sequence, peeling off points from $D(b)$ one at a time, as illustrated in Figs. 1–3. There are $2d$ directions, so as long as there are more than d points left there must be a point that is extreme in only one direction, which we can peel off.

Call $D^*(b) := D(b) - \sigma(b)$ the *basis* of b. We charge the box b to its basis $D^*(b)$, and we claim that each basis is charged $O((\tau n)^{d-1})$ times. Since there are $O(n^d)$ possible bases, this proves the theorem. To prove the claim, consider a basis D^*, and choose a sequence type. Any (τn)-box b whose basis $D(b)$ is equal to D^* and whose peeling sequence has the given type can be constructed incrementally as follows—see Figs. 4 and 5 for an illustration. Start with $D = D^*$. Now consider the last direction $\boldsymbol{d}_q$ of the sequence type. Since the last point p_q of the peeling sequence is extreme only in direction $\boldsymbol{d}_q$, it must be contained in the semi-infinite box which is bounded in all other directions by planes through points in D. Hence, only the first τn points in this semi-infinite box are candidates for p_q, otherwise the box would already contain too many points. A similar argument shows there are only τn choices for $p_{q-1}, ..., p_2$. The first point p_1 from the sequence is then fixed, as b must contain exactly τn points.

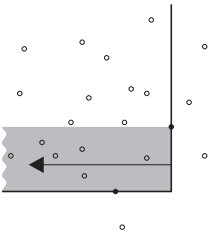


Fig. 4. Constructing a (τn)-box with sequence type $+x_2, -x_1$ in two dimensions. First choose a basis of two points for the remaining directions (the black dots). Then follow the sequence type in reverse order. The extreme point for direction $-x_1$ must be one of the first τn points found when traversing the shaded area in the direction of the arrow.

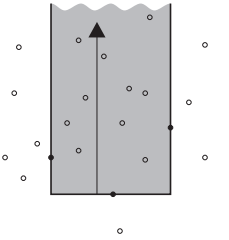


Fig. 5. The extreme point for the first direction of the sequence, $+x_2$, must be the (τn)'th point in the shaded area.

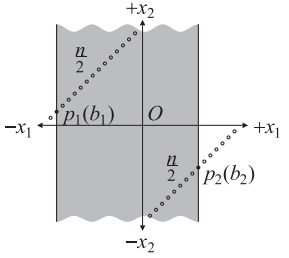


Fig. 6. A lower bound on the number of (τn)-boxes in two dimensions. The four directions are grouped in two pairs $(-x_1, +x_2)$ and $(+x_1, -x_2)$. We place a staircase of $n/2$ points in the positive quadrant for each pair (in two dimensions, these quadrants are coplanar; in higher dimensions this is not necessarily the case). Choosing one defining point on each staircase fixes two sides of a box. We have $\Theta(n^2)$ ways to do so.

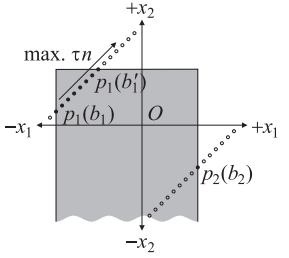


Fig. 7. Choosing one additional point on one staircase fixes another side of the box. This additional point must be one of the first $\Theta(\tau n)$ points found when walking up the staircase from the first defining point on that staircase. On the remaining staircase, we will have no choice but to choose the point such that the box will contain exactly τn points.

To prove the lower bound, consider the following configuration (shown in Fig. 6 for the planar case). We pair the $2d$ directions $+x_1, -x_1, ..., +x_d, -x_d$ into d pairs $(\boldsymbol{d}_{11}, \boldsymbol{d}_{12})$, $(\boldsymbol{d}_{21}, \boldsymbol{d}_{22}), \ldots$, $(\boldsymbol{d}_{d1}, \boldsymbol{d}_{d2})$ so that no pair contains opposite directions, that is $\boldsymbol{d}_{i1} \neq -\boldsymbol{d}_{i2}$ for $1 \leq i \leq d$. Let h_i be the 2-plane spanned by the directions $\boldsymbol{d}_{i1}$ and $\boldsymbol{d}_{i2}$ and containing the origin. On each 2-plane h_i, we place n/d points $p_i(1), ..., p_i(n/d)$ such that all of them are in the positive quadrant with respect to the origin and both directions $\boldsymbol{d}_{i1}$ and $\boldsymbol{d}_{i2}$. We place these points along a staircase. More precisely, we require that for $1 < j \leq n/d$, the point $p_i(j)$ is closer to the origin than $p_i(j-1)$ with respect to direction $\boldsymbol{d}_{i1}$, and further from the origin with respect to direction $\boldsymbol{d}_{i2}$. Any box containing at least one point from each of these sets can now be specified by choosing two points $p_i(b_i)$ and $p_i(b_i')$ in each 2-plane h_i; we define the box b to be the minimum

bounding box of the points chosen. By choosing $b'_i \leq b_i + (\tau n - 1)/(d-1) - 1$ for $1 \leq i < d$, and $b'_d = b_d - 1 + \sum_{i=1}^{d-1}(b'_i - b_i + 1)$, we get a box containing exactly τn points. Having $\Theta(n)$ choices for each b_i $(1 \leq i \leq d)$ and $\Theta(\tau n)$ choices for each b'_i $(1 \leq i \leq d-1)$, we can construct $\Theta(\tau^{d-1} n^{2d-1})$ different (τn)-boxes. □

Note that already in the plane, the bound is cubic in n.

Remark 1. A different way to state the result above is as follows. Let P be a set of n points in $\mathbb{R}^d$, and let t be a parameter with $1 \leq t \leq n/(2d)$. Then the maximum number of combinatorially distinct boxes containing exactly t points from P is $\Theta(n^d t^{d-1})$. In other words, we have proved a tight bound on the number of t-sets for ranges that are boxes instead of hyperplanes. Since t-sets have been studied extensively—see e.g. [5] and [10]—we suspected that the case of box-ranges would have been considered as well, but we have only found a result on this for $t = 2$: Alon *et al.* [4] proved that the maximum number of 2-boxes is $(1 - \frac{1}{2^{2^{d-1}-1}})n^2/2 + o(n^2)$.

Approximate test sets. The worst-case bound from Lemma 1 is quite disappointing. Therefore we now turn our attention to approximate test sets. A set T of boxes is called an *ε-approximate τ-test set* for a set P of n points if

1. any box from T contains at least $(1-\varepsilon)\tau n$ points from P;
2. any query box Q that contains at least τn points from P fully contains at least one box from T.

This means we can answer ε-approximate significant-presence queries on P by answering a presence query on T.

Lemma 2. *For any set P of n points in $\mathbb{R}^d$ $(d > 1)$ and any ε with $0 < \varepsilon < 1/2$, there is an ε-approximate τ-test set of size $O(1/(\varepsilon^{2d-1}\tau^{2d-2}))$. Moreover, there are sets P for which any ε-approximate τ-test set has size $\Omega(1/(\varepsilon^{2d-1}\tau^d))$.*

Proof. To prove the upper bound, we proceed as follows. We will construct test sets recursively, starting with the full set P as input. If the size of the current set P is less than τn_0, where n_0 is the original number of points, there is nothing to do. Otherwise, we choose a hyperplane h orthogonal to the x_1-axis, such that at most half of the points in P lies on either side of h. Then we construct three test sets, one for queries on one side of h, one for queries on the other side, and one for queries intersecting h. The first two test sets are constructed by applying the procedure recursively. The latter set is constructed as follows.

Let n be the number of points in the current set P. We construct a collection $H_2(P)$ of $n(2d-1)/(\varepsilon\tau n_0)$ hyperplanes orthogonal to the x_2-axis, such that there are $\varepsilon\tau n_0/(2d-1)$ points of P between any pair of consecutive hyperplanes.[3] We do the same for the other axes, except the x_1-axis, obtaining sets $H_3(P), \ldots, H_d(P)$.

[3] If there are more points with the same x_2-coordinate, we choose the hyperplanes such that we have at most $\varepsilon\tau n_0/(2d-1)$ points strictly in between consecutive hyperplanes, and at least $\varepsilon\tau n_0/(2d-1)$ points in between or on consecutive hyperplanes.

From these collections of hyperplanes we construct our test set as follows. Take any possible subset H^* of $2d-2$ hyperplanes from $H_2(P) \cup \cdots \cup H_d(P)$ such that $H_2(P)$ up to $H_d(P)$ each contribute exactly two hyperplanes to H^*. Let $P(H^*)$ be the set of points in P that lie on or between the hyperplanes contributed by $H_i(P)$, for all $2 \leq i \leq d$. Construct a collection $H_1(H^*)$ of hyperplanes orthogonal to the x_1-axis, such that there are $\varepsilon\tau n_0/(2d-1)$ points of $P(H^*)$ between each pair of consecutive hyperplanes. For each such hyperplane $h' \in H_1(H^*)$, construct a test box b with the following properties:

1. b is bounded by h', the hyperplanes from H^*, and one additional hyperplane parallel to h' and through a point of $P(H^*)$;
2. b is a $((1-\varepsilon)\tau n_0)$-box.

Of all the test boxes thus constructed, we discard those that do not intersect h. Hence we will only keep boxes for which h' is relatively close to h: there cannot be more than $(1-\varepsilon)\tau n_0$ points from $P(H^*)$ between h and h'.

This implies that the total number of test boxes we create in this step is bounded by $(1-\varepsilon)\tau n_0 \,/\, (\varepsilon\tau n_0/(2d-1)) \leq (2d-1)/\varepsilon$ for a fixed set H^*. Hence, we create at most $(n(2d-1)/(\varepsilon\tau n_0))^{2d-2} \cdot (2d-1)/\varepsilon$ boxes in total. The number $T(n)$ of boxes created in the entire recursive procedure therefore satisfies:

$$T(n) = 0 \qquad \text{if } n < \tau n_0$$
$$T(n) \leq 2T(n/2) + \left(\tfrac{2d-1}{\varepsilon\tau n_0}\right)^{2d-2} \cdot \tfrac{2d-1}{\varepsilon} \cdot n^{2d-2} \quad \text{otherwise.}$$

This leads to $|T| = T(n_0) = O(1/(\varepsilon^{2d-1}\tau^{2d-2}))$.

We now argue that T is an ε-approximate τ-test set for P. By construction, every box in T contains at least $(1-\varepsilon)\tau n_0$ points, so it remains to show that every box Q that contains at least τn_0 points from P fully contains at least one box b from T. Let h be the first hyperplane used in the recursive construction. If at least τn_0 points in Q lie to the same side of h, we can assume that there is a test box contained in Q by induction. If this is not the case, we will show that a test box b inside Q was created for queries intersecting h. To see that such a box must exist, observe that for any i with $2 \leq i \leq d$, there must be a hyperplane $h_i \in H_i(P)$ that intersects Q and has at most $\varepsilon\tau n_0/(2d-1)$ points from $Q \cap P$ below it. Similarly, there is a hyperplane $h'_i \in H_i(P)$ intersecting Q with at most $\varepsilon\tau n_0/(2d-1)$ points from $Q \cap P$ above it. Note that $h_i \neq h'_i$. Let H^* be the set $\{h_2, h'_2, h_3, h'_3, \ldots, h_d, h'_d\}$. Since each of these hyperplanes 'splits off' at most $\varepsilon\tau n_0/(2d-1)$ points from Q, they define, together with the facets of Q orthogonal to the x_1-axis, a box contained in Q and containing at least $(1-\varepsilon+\varepsilon/(2d-1))\tau n_0$ points. From this, one can argue that our construction, when processing this particular H^*, must have produced a test box $b \subset Q$. The proof is illustrated in Fig. 8.

To prove the lower bound, recall the construction used in Lemma 1 for the lower bound for the exact case. There we used d staircases of n/d points each. We then picked two points from each staircase, with at most $(\tau n-1)/(d-1)$ points between (and including) the first and second point, except for the last staircase,

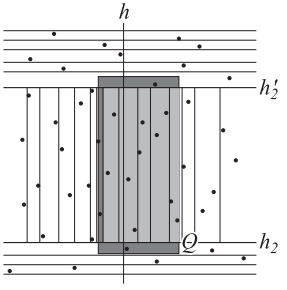


Fig. 8. An example query range Q (shaded area) that intersects h, showing also h_2, h'_2 and the grid $H_1(\{h_2, h'_2\})$. The three dark areas of Q each contain at most $\varepsilon\tau n_0/3$ points. Hence, if Q contains at least τn_0 points, the bright area of Q contains at least $(1-\varepsilon)\tau n_0$ points, and a test box like the one shown above, bounded by h_2, h'_2 and a grid line from $H_1(\{h_2, h'_2\})$, must lie inside Q.

where we picked only one point. Each such combination of points defined a different (τn)-box, thus given $\Omega(\tau^{d-1}n^{2d-1})$ different (τn)-boxes. Now, for the approximate case, we consider a subset of $(n/d)/(\varepsilon\tau n+2)$ so-called *anchor points* along each staircase, such that two consecutive anchor points have $\varepsilon\tau n+1$ points in between. We now pick two anchor points from each staircase, except the last staircase, where we pick one. We make sure that in between two chosen anchor points from the same staircase, there are at most $(\tau n-1)/(d-1)$ points. We then pick a final point on the last staircase to obtain a (τn)-box. Each of these boxes must be captured by a different test box, because the intersection of two such boxes contains less than $(1-\varepsilon)\tau n$ points. The lower bound follows. □

Putting it all together. To summarize, the construction of our data structure for ε-approximate significant-presence queries on $S = S_1 \cup \cdots \cup S_m$ is as follows. We construct an ε-approximate τ-test set T_i for each color class S_i. This gives us a collection of $M = O(m/(\varepsilon^{2d-1}\tau^{2d-2}))$ boxes in $\mathbb{R}^d$. We map these boxes to a set $\hat{S}$ of colored points in $\mathbb{R}^{2d}$, and construct a data structure for the standard colored range-searching problem (that is, presence queries) on P, using the techniques of Agarwal *et al.* [3]. Their structure was designed for searching on a grid, but using the standard trick of normalization—replace every coordinate by its rank, and transform the query box to a box in this new search space in $O(\log n)$ time before running the query algorithm—we can employ their results in our setting.

The same technique works for exact queries, if we use exact test sets. This gives a good result for $d = 1$, if we use the results from Gupta *et al.* [8] on quadrant range searching.

Theorem 1. *Let $S = S_1 \cup \cdots \cup S_m$ be a colored point set in $\mathbb{R}^d$, and τ a fixed constant with $0 < \tau < 1$. For $d = 1$, there is a data structure that uses $O(n)$ storage such that exact significant-presence queries can be answered in $O(\log n + k)$ time, where k is the number of reported colors. For $d > 1$, there is, for any ε with $0 < \varepsilon < 1/2$ and any $\delta > 0$, a data structure for S that uses $O(M^{1+\delta})$ storage such that ε-approximate significant-presence queries on S can be answered in $O(\log n + k)$ time, where $M = O(m/(\varepsilon^{2d-1}\tau^{2d-2}))$.*

Remark 2. Observe that, since we only have constantly many points per color, we could also use standard range-searching techniques. But this would increase the factor k in the reporting time to $O(k/(\varepsilon^{2d-1}\tau^{2d-2}))$, which is undesirable.

The case of variable τ. Now consider the case where the parameter τ is not given in advance, but is part of the query. We assume that we have a lower bound τ_0 on the value of τ in any query. Then we can still answer queries efficiently, at only a small increase in storage. To do so, we build a collection of $O(T)$ substructures, where $T = \log(1/\tau_0)/\log(1+\varepsilon/2)$. More precisely, for integers i with $0 \leq i \leq T$, we define $\tau_i := (1+\varepsilon/2)^i\tau_0$, and for each such i we build a data structure for $(\varepsilon/2)$-approximate τ_i-significant-presence queries on S. To answer a query with a query box Q and query parameter τ, we first find the largest τ_i smaller than or equal to τ, and we query with Q in the corresponding data structure. This leads to the following result.

Theorem 2. *Let $S = S_1 \cup \cdots \cup S_m$ be a colored point set in $\mathbb{R}^d$, and τ_0 a fixed constant with $0 < \tau_0 < 1$. For $d > 1$, any $0 < \varepsilon < 1/2$ and any $\delta > 0$, there is a data structure for S that uses $O(M^{1+\delta}/\varepsilon)$ storage such that ε-approximate significant-presence queries on S can be answered in $O(\log n + k)$ time, where $M = O(m/(\varepsilon^{2d-1}\tau_0^{2d-2}))$ and k is the number of reported colors.*

Proof. By Theorem 1, the size of substructure i is $O(M^{1+\delta}(\tau_0/\tau_i)^D) = O(M^{1+\delta}/(1+\varepsilon/2)^{Di})$, where $M = O(m/(\varepsilon^{2d-1}\tau_0^{2d-2}))$ and $D = (2d-2)(1+\delta)$. The total size of all substructures is therefore $O(M^{1+\delta}\sum_{i=0}^{T}(1+\varepsilon/2)^{-Di}) = O(M^{1+\delta}/\varepsilon)$.

It remains to show that queries are answered correctly. Note that $\tau_i \leq \tau \leq (1+\varepsilon/2)\tau_i$. Now, any color j with $|Q \cap S_j| \geq \tau_i|S_j|$ will be reported by our algorithm, so certainly any color with $|Q \cap S_j| \geq \tau|S_j|$ will be reported. Second, for any reported color j we have:
$|Q \cap S_j| \geq (1-\varepsilon/2)\cdot\tau_i|S_j| \geq (1-\varepsilon/2)\cdot\tau/(1+\varepsilon/2)\cdot|S_j| \geq (1-\varepsilon)\tau\cdot|S_j|$.
This proves the correctness of the algorithm. □

3 Stabbing Queries

Let $B = B_1 \cup \cdots \cup B_m$ be a set of n colored boxes in $\mathbb{R}^d$, where B_i denotes the subset of boxes of color i. Let τ be a constant with $0 < \tau < 1$. For a point q, we use $B_i(q)$ to denote the subset of boxes from B_i that contain q. We want to preprocess B for the following type of stabbing queries: given a query point q, report all colors i such that $|B_i(q)| \geq \tau\cdot|B_i|$. As was the case for range queries, we are not able to obtain near-linear storage for exact queries for $d > 1$, so we focus on the ε-approximate variant, where we are also allowed to report a color if $|B_i(q)| \geq (1-\varepsilon)\tau\cdot|B_i|$.

Our approach is similar to our approach for range searching. Thus we define an *ε-approximate τ-test set* for a set B_i to be a set T_i of test boxes such that

1. for any point q with $|B_i(q)| \geq \tau\cdot|B_i|$, there is a test box b with $q \in b$, and
2. for any test box b and any point $q \in b$, we have $|B_i(q)| \geq (1-\varepsilon)\tau\cdot|B_i|$.

This means we can answer a query by reporting all colors i for which there is a test box $b \in T_i$ that contains q.

Lemma 3. *For any set B_i of boxes in $\mathbb{R}^d$, there is an ε-approximate τ-test set T_i consisting of $O(1/(\varepsilon\tau)^d)$ disjoint boxes. Moreover, for $\varepsilon < 1/(2d)$, there are sets of boxes in $\mathbb{R}^d$ for which any ε-approximate τ-test set has size $\Omega(((1-\tau)/(\varepsilon\tau))^d)$.*

Proof. For each of the d main axes, sort the facets of the input boxes orthogonal to that axis, and take a hyperplane through every $(\varepsilon\tau n_i/d)$-th facet, where $n_i := |B_i|$. This gives d collections of $d/(\varepsilon\tau)$ parallel planes, which together define a grid with $O(1/(\varepsilon\tau)^d)$ cells. We let T_i consist of all cells that are fully contained in at least $(1-\varepsilon)\tau \cdot |B_i|$ boxes from B_i. Clearly T_i has the required number of boxes, and has property (2). (Note: using the fact that, coming from infinity, we must cross at least $d(1-\varepsilon)/\varepsilon \geq (1/\varepsilon) - 1$ hyperplanes before we can come to a cell from T_i, we can in fact obtain a slightly stronger bound on the size of T_i for the case where τ is large.)

It remains to show that T_i has property (1). Let q be a point for which $|B_i(q)| \geq \tau \cdot |B_i|$, and let C be the cell containing q. Since any cell is crossed by at most $\varepsilon\tau n_i$ facets, we must have $C \in T_i$.

The lower bound is proved as follows. For each of the main axes, take a collection of $(1-\tau)/(2d\varepsilon\tau)$ hyperplanes orthogonal to that axis. Slightly 'inflate' each hyperplane to obtain a very thin box. This way each intersection point of d hyperplanes becomes a tiny hypercube. Next, each of these thin boxes is replaced by $2\varepsilon\tau n_i$ identical copies of itself. Note that each tiny hypercube is now covered by $2d\varepsilon\tau n_i$ boxes, and that there are $((1-\tau)/(2d\varepsilon\tau))^d$ such hypercubes. Add a collection of $(1-2d\varepsilon)\tau n_i$ big boxes, each containing all the tiny hypercubes. The tiny hypercubes are now covered by exactly τn_i boxes, and the remaining space is covered by at most $(1-2\varepsilon)\tau n_i$ boxes. (Since we have used slightly less than n_i boxes in total, we need to add some more boxes, at some arbitrary location disjoint from all other boxes.) Any test set must contain each of the hypercubes, and the result follows. □

To solve our problem, we construct a test set T_i for each color class B_i according to the lemma above. This gives us a collection of $M = O(m/(\varepsilon\tau)^d)$ colored boxes. Applying the results of Agarwal *et al.* [3] again, we get the following result.

Theorem 3. *Let $B = B_1 \cup \cdots \cup B_m$ be a colored set of boxes in $\mathbb{R}^d$, and τ a fixed constant with $0 < \tau < 1$. For $d = 1$, there is a data structure that uses $O(n)$ storage such that exact significant-presence queries can be answered in $O(\log n + k)$ time, where k is the number of reported colors. For $d > 1$, there is, for any ε with $0 < \varepsilon < 1/2$ and any $\delta > 0$, a data structure for B that uses $O(M^{1+\delta})$ storage such that ε-approximate significant-presence queries on B can be answered in $O(\log n + k)$ time, where $M = O(m/(\varepsilon\tau)^d)$.*

Remark 3. Note that, since the test boxes from any given color are disjoint, we can simply report the color of each box containing the query point q. Thus we do not have to use the structure of Agarwal *et al.*, but we can apply results from standard non-colored stabbing queries [2]. This way we can slightly reduce storage to $O(M \log^{d-2+\delta} M)$ at the cost of a slightly increased query time of $O(\log^{d-1} M + k)$. Also note that we can treat the case of variable τ in exactly the same way as for range queries.

4 Concluding Remarks

Standard colored range searching problems ask to report all colors that have at least one object of that color intersecting the query range. We considered the variant where a color should only be reported if some constant pre-specified fraction of the objects intersects the range. We developed efficient data structures for an approximate version of this problem for orthogonal range searching queries and for stabbing queries. One obvious open problem is whether there exists a data structure for the exact problem with near-linear space. We have shown that this is impossible using our test-set approach, but perhaps a completely different approach is possible. Another open problem is to close the gap between our upper and lower bounds for the size of approximate test sets for orthogonal range searching.

Acknowledgements. We thank Joachim Gudmundsson and Jan Vahrenhold for inspiring discussions about the subject of this paper. Herman Haverkort's work is supported by the Netherlands' Organization for Scientific Research (NWO).

References

1. P.K. Agarwal. Range Searching. In: J. Goodman and J. O'Rourke (Eds.), *CRC Handbook of Computational Geometry*, CRC Press, pages 575–598, 1997.
2. P.K. Agarwal and J. Erickson. Geometric range searching and its relatives. In: B. Chazelle, J. Goodman, and R. Pollack (Eds.), *Advances in Discrete and Computational Geometry*, Vol. 223 of *Contemporary Mathematics*, pages 1–56, American Mathematical Society, 1998.
3. P.K. Agarwal, S. Govindarajan, and. S. Muthukrishnan. Range searching in categorical data: colored range searching on a grid. In *Proc. 10th Annu. European Sympos. Algorithms* (ESA 2002), pages 17–28, 2002.
4. N. Alon, Z. Füredi, and M. Katchalski. Separating pairs of points by standard boxes. *European J. Combinatorics* 6:205–210 (1985).
5. T.K. Dey. Improved bounds for planar k-sets and related problems. *Discrete and Computational Geometry* 19(30):373–382 (1998).
6. M. van Kreveld. New Results on Data Structures in Computational Geometry. PhD thesis, Utrecht University, 1992.
7. J. Nievergelt and P. Widmayer. Spatial data structures: concepts and design choices. In: J.-R. Sack and J. Urrutia (Eds.) *Handbook of Computational Geometry*, pages 725–764, Elsevier Science Publishers, 2000.
8. J. Gupta, R. Janardan, and M. Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization. In *Proc. 3rd Workshop on Algorithms and Data Structures*, LNCS 709, pages 361–373, 1993.
9. R. Janardan and M. Lopez. Generalized intersection searching problems. *Internat. J. Comput. Geom. Appl.* 3:39–70 (1993).
10. M. Sharir, S. Smorodinsky, and G. Tardos. An Improved Bound for k-Sets in Three Dimensions. *Discrete and Computational Geometry* 26(2):195–204 (2001).

Either/Or: Using VERTEX COVER Structure in Designing FPT-Algorithms — The Case of k-INTERNAL SPANNING TREE

Elena Prieto[1] and Christian Sloper[2]

[1] School of Electrical Engineering and Computer Science,
The University of Newcastle
NSW, Australia
elena@cs.newcastle.edu.au
[2] Department of Informatics,
University of Bergen
Norway
sloper@ii.uib.no

Abstract. To determine if a graph has a spanning tree with few leaves is NP-hard. In this paper we study the *parametric dual* of this problem, k-INTERNAL SPANNING TREE (Does G have a spanning tree with at least k internal vertices?). We give an algorithm running in time $O(2^{4k \log k} \cdot k^{7/2} + k^2 \cdot n^2)$. We also give a 2-approximation algorithm for the problem.

However, the main contribution of this paper is that we show the following remarkable structural bindings between k-INTERNAL SPANNING TREE and k-VERTEX COVER:

- NO for k-VERTEX COVER implies YES for k-INTERNAL SPANNING TREE.
- YES for k-VERTEX COVER implies NO for $(2k + 1)$-INTERNAL SPANNING TREE.

We give a polynomial-time algorithm that produces *either* a vertex cover of size k *or* a spanning tree with at least k internal vertices. We show how to use this inherent *vertex cover structure* to design algorithms for FPT problems, here illustrated mainly by k-INTERNAL SPANNING TREE. We also briefly discuss the application of this vertex cover methodology to the parametric dual of the DOMINATING SET problem. This design technique seems to apply to many other FPT problems.

Keywords: Spanning trees, fixed-parameter tractability, vertex cover, kernelization.

1 Introduction

The investigations on which we report here are carried out in the framework of parameterized complexity, so we will begin by making a few general remarks about this context of our research. The subject is concretely motivated by an abundance of natural examples of two different kinds of complexity behavior. These include the well-known problems MIN CUT LINEAR ARRANGEMENT, BANDWIDTH, VERTEX COVER, and INDEPENDENT SET (for definitions the reader may refer to [GJ79]).

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 474–483, 2003.
© Springer-Verlag Berlin Heidelberg 2003

Definition 1. *(Fixed Parameter Tractability) A parameterized problem $L \subseteq \Sigma^* \times \Sigma^*$ is* fixed-parameter tractable *if there is an algorithm that correctly decides, in time $f(k)\, n^{\alpha}$, for input $(x, y) \in \Sigma^* \times \Sigma^*$ whether or not $(x, y) \in L$, where n is the size of the input x, $|x| = n$, k is the parameter, α is a constant (independent of k) and f is an arbitrary function.*

The analog of NP is the parameterized complexity class $W[1]$ [DF95b]. The fact that the k-NDTM problem is complete for $W[1]$ was proved by Cai, Chen, Downey and Fellows in [CCDF97]. Since BANDWIDTH and INDEPENDENT SET are hard for $W[1]$, we thus have strong natural evidence that they are not fixed-parameter tractable, as VERTEX COVER and MIN CUT LINEAR ARRANGEMENT are.

Further background on parameterized complexity can be found in [DF98].

The problems we address in this paper concern spanning trees, namely k-(MIN)LEAF SPANNING TREE (Does G have a spanning tree with at most k leaves?) and its parametric dual, k-INTERNAL SPANNING TREE (Does G have a spanning tree with at most $n - k$ leaves?). In the classical complexity theory these two problems are indistinguishable from each other. In the following section we show that the problems are intrinsically different when analyzed from a parameterized point of view. We prove that while k-(MIN)LEAF SPANNING TREE is $W[P]$-hard, k-INTERNAL SPANNING TREE is in FPT . In Section 3 we describe how to use the bounded *vertex cover structure* to design an FPT algorithm for k-INTERNAL SPANNING TREE. We give an analysis of the running time of the algorithm generated by the method in Section 4, we show how the methodology created in Section 3 can be applied to other FPT problems and we conclude with some remarks about future research. Also, as a consequence of the preprocessing of the graph necessary to create our fixed-parameter algorithm, we easily obtain a polynomial time 2-approximation algorithm for k-INTERNAL SPANNING TREE.

2 Parametric Duality

Koth and Raman were the first to notice a remarkable empirical pattern in the parameterized complexity of familiar NP-complete problems, having to do with *parametric duality* [KR00]. Koth and Raman observed that if a problem is fixed parameter tractable then its parametric dual usually is not tractable. The idea is perhaps best presented by the dual pair of problems INDEPENDENT SET and VERTEX COVER. The duality between the two problems consists in the fact that a graph G has a vertex cover of size k if and only if it has an independent set of size $n - k$. Thus, if we consider the two naturally parameterized problems, for input G and parameter k, where in the one case we are "parameterizing upward" and in the other case we are "parameterizing downward". As we have mentioned VERTEX COVER is fixed-parameter tractable, whereas INDEPENDENT SET is complete for the class $W[1]$ and therefore very unlikely to be in FPT unless both classes are proven to be equal, which is as unlikely as proving P = NP. This is also the case with DOMINATING SET and its parametric dual, NONBLOCKER, the former being W[2]-complete and the latter being in FPT.

We prove that Koth and Raman's observation holds for k-(MIN)LEAF SPANNING TREE and its parametric dual k-INTERNAL SPANNING TREE. We start with the following easy lemma:

Lemma 1. *The* k-(MIN)LEAF SPANNING TREE *problem is hard for* $W[P]$.

Proof: It is trivially true that the HAMILTONIAN PATH problem is a special case of k-(MIN)LEAF SPANNING TREE when we make the parameter $k = 2$. Since HAMILTONIAN PATH is *NP*-complete [GJ79] we can see immediately that k-(MIN)LEAF SPANNING TREE is not FPT unless FPT $= W[P]$. □

Using Robertson and Seymour's Graph Minor Theorem it is also quite straightforward to prove the following membership in FPT.

Lemma 2. *The* k-INTERNAL SPANNING TREE *problem is in* FPT.

Proof: Let $\mathcal{F}_k$ denote the family of graphs that do not have spanning trees with at least k internal vertices. It is easy to observe that for each k this family is a lower ideal in the minor order. Less formally, let (G, k) be a NO-instance of k-INTERNAL SPANNING TREE, that is a graph G for which there is not a spanning tree with k internal vertices. The local operations which configure the minor order (i.e. edge contractions, edge deletions and vertex deletions) will always transform this NO-instance into another NO-instance. By the Graph Minor Theorem of Robertson and Seymour and its companion result that order testing in the minor order is FPT [RS99] we can claim that k-INTERNAL SPANNING TREE is also FPT. (An exposition of well-quasiordering as a method of FPT algorithm design can be found in [DF98].) □

Unfortunately, this FPT proof technique suffers from being nonuniform and nonconstructive, and gives an $O(f(k)n^3)$ algorithm with a very fast-growing parameter function compared to the one we obtain in Section 3.

We remark that it can be shown that all fixed graphs with a vertex cover of size k are well-quasi ordered by ordinary subgraphs and have linear time order tests [FFLR03]. The proof of this is substantially shorter than the Graph Minor Project and could be used to simplify Lemma 2.

3 Either/Or: We Win

Currently, the main practical methods of FPT algorithm design are based on *kernelization* and *bounded search trees*. The idea of kernelization is relatively simple and can be quickly illustrated for the VERTEX COVER problem. If the instance is (G, k) and G has a pendant vertex $v \in V(G)$ of degree 1 connected to the vertex $u \in V(G)$, then it would be silly to include v in any solution (it would be better, and equally necessary, to include u), so (G, k) can be reduced to $(G', k-1)$, where G' is obtained from G by deleting u and v. Some more complicated and much less obvious reduction rules for the VERTEX COVER problem can be found in the current state-of-the-art FPT algorithms (see [BFR98,DFS99,NR99b,Ste00,CKJ01]). The basic schema of this method of FPT-algorithm design is that reduction rules are applied until an *irreducible* instance (G', k') is obtained. At this point in the FPT algorithm, a *Kernelization Lemma* is invoked to decide all those instances where the reduced instance G' is larger than $g(k')$ for some function g.

To find the function g in the case of k-INTERNAL SPANNING TREE we are going to make use of the intrinsic relationship between this problem and the thoroughly studied

VERTEX COVER. We give a polynomial time algorithm that outputs *either* a Spanning Tree with many internal vertices *or* a small Vertex Cover. A proof of this result and its applications to the design of FPT-algorithms will be shown later on in the paper.

Lemma 3. *Any graph G has a spanning tree T such that all the leaves of T are independent vertices in G or G has a spanning tree T' with only two leaves.*

Proof: Given a spanning tree T of a graph G we say that two leaves $u, v \in T$ are in conflict if $uv \in E(G)$. We now show that given a spanning tree with i conflicts it is possible to obtain a spanning tree with $< i$ conflicts using one of the rules below:

1. If x and y are in conflict and z the parent of x, has degree 3 or higher, then a new spanning tree T' could be constructed using the edge xy as an edge instead of xz.
2. If x and y are in conflict and both their parents are of degree of 2, then let x' be the first vertex on a path from x that has degree different from 2, and let y' be the first vertex on a path from y that has degree different from 2. If $y' = x$ (and thus $x' = y$) we know that the spanning tree is a Hamiltonian path and has only two leaves. Otherwise we create a new spanning tree disconnecting the path from x to x'(leaving x') and connecting x to y, repairing the conflict between x and y. Since x' is now of degree at least 2 we do not create any new conflicts.

The validity of the rules are easy to verify and it is obvious that they can be executed in polynomial time. Lemma 3 then follows by recursively applying the rules until no conflicts exists. □

For a spanning tree T we define two sets, $A(T)$ of internal vertices of T, and $B(T)$ of leaves of T. If it is obvious from the context which spanning tree is in question we will for simplicity write A and B.

Several corollaries follow easily from this Lemma. One of them gives an approximation for k-INTERNAL SPANNING TREE, the others relate the problem to the well-studied VERTEX COVER.

Corollary 1. *k-INTERNAL SPANNING TREE has a 2-approximation algorithm.*

Proof: Note that because B is an independent set (due to Lemma 3) it is impossible to include more than $|A|$ elements of B as internals in the optimal spanning tree. The maximum number of internal vertices is at most $2|A|$, and hence the spanning tree generated by the algorithm in Lemma 3 has $|A|$ internal vertices and is a 2-approximation for k-INTERNAL SPANNING TREE. □

Corollary 2. *If a graph $G = (V, E)$ is a* NO*-instance for k-*VERTEX COVER *then G is a* YES*-instance for k-*INTERNAL SPANNING TREE.

Proof: If a graph does not have a vertex cover of size k then we know that it does not have an independent set of size $\geq n - k$ (see discussion in section 2). This implies that $|B| < n - k$ and $|A| \geq k$, a YES-instance of k-INTERNAL SPANNING TREE. □

Corollary 3. *If a graph $G = (V, E)$ is a* YES*-instance for k-*VERTEX COVER *then G is a* NO*-instance for $(2k + 1)$-*INTERNAL SPANNING TREE.

Proof: Again, since the set $B(T)$ in the transformed spanning tree is independent, we know that a if G is a YES-instance for k-VERTEX COVER then there are $n-k$ vertices in B. For each vertex in the independent set B that we include as a internal in the spanning tree we must include at least one other vertex in A. Thus, at most $2k$ vertices can be internal in the spanning tree and therefore G is a NO-instance for $(2k+1)$-INTERNAL SPANNING TREE. □

We now know that if a graph does not have a k-vertex cover then it is a YES-instance for k-INTERNAL SPANNING TREE. We can use the structure provided by VERTEX COVER to bound the size of the kernel.

Lemma 4. *(Kernelization Lemma) Either G is a* YES*-instance for* k-INTERNAL SPANNING TREE *or it has less than* $g(k) = 2k^3 + k^2 + 2k$ *vertices.*

Proof: By Corollary 2 we know that *either* a graph is a YES-instance for k-INTERNAL SPANNING TREE *or* it has a k-vertex cover. Thus, we assume that there is a k-vertex cover in G. We will use this vertex cover structure to prove the lemma.

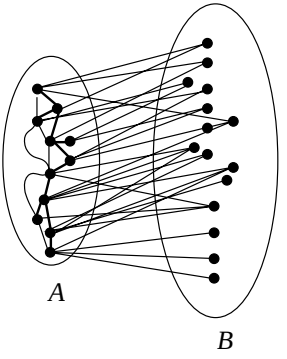


Fig. 1. Example of inherent vertex cover structure
(Set A)

Note that the set A produced in Lemma 3 is a vertex cover and that the set B, its complement, is an independent set. (We assume that Lemma 3 did not produce a Hamilton path as it would be a optimal spanning tree and we could determine the result immediately.)

We define a B-bridge over a pair of vertices $w_1, w_2 \in A$ as a vertex $u \in B$ such that both uw_1 and uw_2 are in E.

We bound the number of vertices in B by showing that there is a limited amount of B-bridges. To do so we need to prove the following two claims which can be seen as kernelization rules for k-INTERNAL SPANNING TREE.

Claim 1 *If there exists a vertex $u \in B$ such that for all vertex pairs v_1, v_2 where u is a B-bridge over v_1 and v_2 there exists $2k+1$ other B-bridges over v_1, v_2 then u can be removed from G (see Figure 2).*

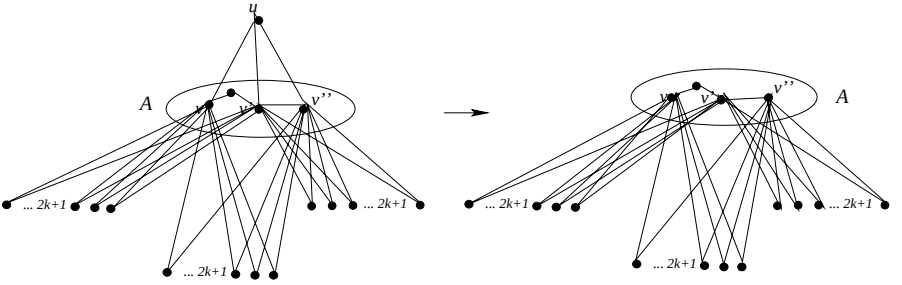


Fig. 2. Kernelization Rule 1

Proof of Claim 1. We prove this claim by showing that G has a k-internal spanning tree if and only if $G' = G \setminus u$ has a k-internal spanning tree.

If G' has a k-internal spanning tree then G obviously has a k-internal spanning tree as the introduction of u could not decrease the number of internals.

If G' has a k-internal spanning tree T then one of three cases apply:

1. u has degree 1 in the spanning tree (u is a leaf) and its neighbor, vertex z, has one or more other leaves. In this case $T \setminus u$ would be a spanning tree with k internal vertices for G'.
2. u has degree 1 in the spanning tree (u is a leaf) and its neighbor, vertex z, has no other leaves. We know that z has at least $2k+1$ other neighbors. No more than k of the $2k+1$ are internal vertices and at most k are needed as leaves elsewhere. We are left with at least one vertex of the $2k+1$ and it can be used as a leaf on z. Case 1 now applies.
3. u has degree $i \geq 2$ in the spanning tree (u is internal). In this case it is possible to change the spanning tree to a spanning tree where u has degree <i. Consider any two vertices $x, y \in N(u)$ in T. We know that x and y have at least $2k+1$ other B-bridges. The same argument as above applies, hence at least one vertex z of these B-bridges is an unessential leaf. Remove xu from the spanning tree and add xz, zy to obtain a spanning tree with more internals and where u is of degree $(i-1)$ in the spanning tree. Recursively apply this rule to obtain a spanning tree where u is of degree 1, where case 1 or 2 applies.

We see that we can always obtain a spanning tree where u is an unessential leaf, and can be removed without lowering the number of internal vertices of k. □

This rule gets rid of many vertices of degree greater than or equal to two in the set B since they are B-bridges. We still could have countless vertices of degree 1 in the graph and therefore be unable to bound the size of B. We need the following reduction rule to eliminate those.

Claim 2 *If a graph G is a* YES*-instance for* k-INTERNAL SPANNING TREE *and there exist two vertices u, v such that the degree of u and v is 1 and u and v have the same neighbor, then $G' = G \setminus v$ is a* YES*-instance for* k-INTERNAL SPANNING TREE

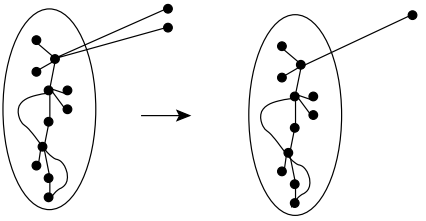


Fig. 3. Kernelization Rule 2

Proof of Claim 2. We prove this by showing that G has a k-internal spanning tree if and only if G' has a k-internal spanning tree.

If G' has a k-internal spanning tree then obviously G has a k-internal spanning tree since adding a leaf cannot decrease the number of internal vertices.

If G has a k-internal spanning tree we know that z is one of the internal vertices (because of x and y) and that x and y are leaves. In G' the vertex z is still an internal vertex (because of y) and y is a leaf. Thus, the number of internals in a spanning tree in G' is not affected by the missing leaf x. □

We recursively apply Claims 1 and 2 to obtain a reduced instance where the claims no longer can be applied.

There are no more than k^2 pairs $p_1, \ldots, p_{k^2}$ of elements in A. Claim 1 implies that at least one of these pairs p_i has no more than $2k + 1$ unmarked B-bridges. Mark all these B-bridges and the pair p_i. Again by Claim 1 we have that at least one unmarked pair p_j with less than $2k + 1$ B-bridges. Mark these B-bridges and the pair p_j. Repeat this operation until all pairs are marked. This can go on for at most k^2 steps, thus no more than a total of $(2k + 1) \cdot k^2 = 2k^3 + k^2$ vertices of degree greater than or equal to 2 can exists in set B.

Now, due to Claim 2 we know that each vertex in A can have at most one "pendant" vertex of degree 1 in B, which gives us total of k vertices of degree 1 in any reduced instance of G. This, together with the $2k^3 + k^2$ vertices of degree greater that 1 in B and another k vertices in set A give us a maximum size of $2k^3 + k^2 + 2k$.

If a reduced instance has more than $2k^3 + k^2 + 2k$ we have reached a contradiction and the assumption that there was a vertex cover of size k must be wrong. Hence, by Corollary 2 we can conclude that G has a k-internal spanning tree.

This concludes the proof of Lemma 4. □

4 Analysis of the Running Time

Our algorithm works in several stages. It first runs a regular spanning tree algorithm and then modifies it to make the leaves independent. Then, if the spanning tree doesn't contain enough internals, we run our reduction rules to reduce the instance in size to $O(k^3)$. We would like to note that a limited number of experiments suggest that this algorithm is a very good heuristic. Then, we finally employ a brute-force spanning tree algorithm to find an optimal solution for the reduced instance.

Since we do not require a minimum spanning tree, we can use a simple breadth-first search to obtain a spanning tree. This can be done in time $O(|V| + |E|)$ [CLR90]. The conflicts can be detected in time $O(|E|)$ and repaired in time $O(|V|)$. Note that we could obtain the Vertex Cover structure by running one of the celebrated VERTEX COVER-algorithms instead, but for this particular problem our heuristic is sufficient.

To reduce the number of B-bridges we can apply the following algorithm. For each vertex $u \in B$ we count the number of vertices that are B-bridges for each pair of neighbors of u. If all pairs have more than $2k + 1$ B-bridges we can remove u (Claim 1). As there is less than k^2 such pairs this can be done in time $O(k^2 \cdot |V|)$ for each vertex in B.

We can easily remove superfluous leaves (Claim 1) in time $O(|V|)$. Thus reducing the instance to a cubic kernel requires in total $O(k^2 \cdot |V|^2)$ time.

To determine if there is a k-internal spanning tree in the reduced kernel we test every possible k-set of the kernel, there are less than k^{3k} such k-sets. Note that this can be rewritten as $2^{3k \log k}$. We now have to verify if these k vertices can be used as the internal vertices of a spanning tree. To do this we try every possible construction of a tree T with these k vertices, by Cayley's formula there are no more than k^{k-2} such trees. This, again, can be rewritten as $(2^{k \log k})/k^2$. Then we test whether or not each leaf in T can be assigned at least one vertex in the remaining kernel as its leaf. This is equivalent to testing if the leaves and the remaining kernel have a perfect bipartite matching, which can be done in time $O(\sqrt{|V|} \cdot |E|)$. In this particular bipartite subset there are not more than k^4 edges giving us a total of $k^{11/2}$ for the matching. Thus for each k-set we can verify if it is a valid solution in $2^{k \log k} \cdot k^{7/2}$ time.

The total running time of the algorithm is $O(2^{4k \log k} \cdot k^{7/2} + k^2 \cdot n^2)$.

5 Vertex Cover Structure: Further Applications

In this section we give another example of how to use the methodology described in Section 3 to produce algorithms for other FPT problems.

Consider the parametric dual of k-DOMINATING SET, k-NONBLOCKER (Does $G = (V, E)$ have a subset of size k, V', such that every element of V' has at least one neighbor in $V \setminus V'$?). Using the bounded vertex cover structure we can produce an algorithm for k-NONBLOCKER running in time $O(4^k + n^\alpha)$ as follows: We first compute a *maximal* independent set I in G. The complement of I, $\overline{I}$, is *either* a vertex cover of size $\leq k$ *or* a nonblocking set of size $\geq k + 1$. This simple algorithm, an analog of our Lemma 3, was first suggested by Faisal Abu-Khzam [A03]. This vertex cover structure allows us now to easily compute a path decomposition of the graph of width k. Let $j_1, j_2, j_3, \ldots$ be an arbitrary sequence of I. The path decomposition is a sequence of bags $B_i = \overline{I} \cup j_i$. Now, using the algorithm introduced by Telle and Proskurowski [PT93] and further improved by Alber and Niedermeier [AN02] we can compute a minimum dominating set in time $O(4^k + n^\alpha)$.

This result matches the running time of McCartin's algorithm [McC03] who used a completely different route to get the same running time.

We would also like to mention that $(n - k)$-COLORING has recently been proven to be FPT using vertex cover structure [CFJ03].

6 Conclusions

In this paper we have given a hardness result for k-(MIN)LEAF SPANNING TREE and a fixed parameter algorithm for its *parametric dual*, k-INTERNAL SPANNING TREE. The algorithm runs in time $O(2^{4k \log k} \cdot k^{7/2} + k^2 \cdot n^2)$ which is the best currently known for this problem.

We also give a 2-approximation algorithm for the problem which can easily be further improved and the same idea used to find more approximation algorithms for other related problems.

We have shown the remarkable structural bindings between k-INTERNAL SPANNING TREE and k-VERTEX COVER in Corollaries 2 and 3. We believe that similar structural bindings exist between VERTEX COVER and other fixed-parameter tractable problems and we are confident that this inherent vertex cover structure can be used to design potent algorithms for these problems, especially when combined with constructive polynomial time algorithms that produce either a vertex cover or a solution for the problem in question. Even if such polynomial time *either/or*-algorithms do not exist, we may still use the quite practical FPT VERTEX COVER-algorithm to find the vertex cover structure.

The current state of the art algorithm for VERTEX COVER runs in time $O(1.286^k + n)$ [CKJ01] and has been proven useful in implementations by groups at the University of Carleton and University of Tennessee in Knoxville for exact solutions for values of n and k up to $2,500$ [L03]. We believe that exploiting vertex cover structure may be one of the most powerful tools to designing algorithms for other fixed parameter tractable problems for which structural bindings with VERTEX COVER exist. For example, we suspect that the parameterized versions of MAX LEAF SPANNING TREE, MINIMUM INDEPENDENT DOMINATING SET and MINIMUM PERFECT CODE are very likely to fall into this class of problems.

Acknowledgements. We would like to thank Mike Fellows, Andrzej Proskurowski and Frances Rosamond for very helpful conversations and encouragement.

References

[A03] F. Abu-Khzam. Private communication.

[AN02] J. Alber and R. Niedermeier. Improved tree decomposition based algorithms for domination-like problems. *Proceedings of the 5th Latin American Theoretical INformatics (LATIN 2002)*, number 2286 in Lecture Notes in Computer Science, pages 613–627, Springer (2002).

[BFR98] R. Balasubramanian, M. R. Fellows, and V. Raman. An Improved Fixed Parameter Algorithm for Vertex Cover. *Information Processing Letters* 65:3 (1998), 163–168.

[CFJ03] B. Chor, M. Fellows, D. Juedes. Private communication concerning manuscript in preparation.

[CCDF97] Liming Cai, J. Chen, R. Downey and M. Fellows. The parameterized complexity of short computation and factorization. *Archive for Mathematical Logic* 36 (1997), 321–338.

[CKJ01] J. Chen, I. Kanj, and W. Jia. Vertex cover: Further Observations and Further Improvements. *Journal of Algorithms* Volume 41, 280–301 (2001).

[CLR90] T.H.Cormen, C.E.Leierson, R.L.Rivest, *Introduction to Algorithms*, MIT Press.

[DF95a] R. Downey and M. Fellows. Parameterized Computational Feasibility. *P. Clote, J. Remmel (eds.): Feasible Mathematics II* Boston: Birkhauser (1995), 219–244.

[DF95b] R. Downey and M. Fellows. Fixed-parameter tractability and completeness II: completeness for $W[1]$. *Theoretical Computer Science A* 141 (1995), 109-131.

[DF98] R. Downey and M. Fellows. *Parameterized Complexity* Springer-Verlag (1998).

[DFS99] R. Downey, M. Fellows and U. Stege. Parameterized complexity: a framework for systematically confronting computational intractability. *Contemporary Trends in Discrete Mathematics* (R. Graham, J. Kratochvil, J. Nesetril and F. Roberts, eds.), *AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 49 (1999), 49–99.

[FFLR03] A. Faisal, M. Fellows, M. Langston, F. Rosamond. Private communication concerning manuscript in preparation.

[FMRS01] M. Fellows, C. McCartin. F. Rosamond and U. Stege. Spanning Trees with Few and Many Leaves. *To appear*

[GMM94] G. Galbiati, F. Maffioli, and A. Morzenti. A Short Note on the Approximability of the Maximum Leaves Spanning Tree Problem. *Information Processing Letters* 52 (1994), 45–49.

[GMM97] G. Galbiati, A. Morzenti and F. Maffioli. On the Approximability of some Maximum Spanning Tree Problems. *Theoretical Computer Science* 181 (1997), 107–118.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.*W.H. Freeman, San Francisco, 1979.

[KR00] Subhash Khot and Venkatesh Raman. *Parameterized Complexity of Finding Hereditary Properties*. Proceedings of COCOON. Theoretical Computer Science (COCOON 2000 special issue)

[L03] M. Langston. Private communication.

[LR98] H.-I. Lu and R. Ravi. Approximating Maximum Leaf Spanning Trees in Almost Linear Time. *Journal of Algorithms* 29 (1998), 132–141.

[McC03] Catherine McCartin. Ph.D. dissertation in Computer Science, Victoria University, Wellington, New Zealand, 2003.

[NR99b] R. Niedermeier and P. Rossmanith. Upper Bounds for Vertex Cover Further Improved. In C. Meinel and S. Tison, editors, *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science*, number 1563 in Lecture Notes in Computer Science, Springer-Verlag (1999), 561–570.

[PT93] J.A.Telle and A.Proskurowski. Practical algorithms on partial k-trees with an application to domination-like problems. *Proceedings WADS'93 - Third Workshop on Algorithms and Data Structures.* Springer Verlag, Lecture Notes in Computer Science vol.709 (1993) 610–621.

[RS99] N. Robertson, PD. Seymor. Graph Minors. XX Wagner's conjecture. *To appear*.

[Ste00] Ulrike Stege. Ph.D. dissertation in Computer Science, ETH, Zurich, Switzerland, 2000.

Parameterized Complexity of Directed Feedback Set Problems in Tournaments

Venkatesh Raman[1] and Saket Saurabh[2]

[1] The Institute of Mathematical Sciences, Chennai 600 113.
vraman@imsc.res.in
[2] Chennai Mathematical Institute, 92, G. N. Chetty Road, Chennai-600 017.
saurabh@cmi.ac.in

Abstract. Given a directed graph on n vertices and an integer parameter k, the feedback vertex (arc) set problem asks whether the given graph has a set of k vertices (arcs) whose removal results in an acyclic directed graph. The parameterized complexity of these problems, in the framework introduced by Downey and Fellows, is a long standing open problem in the area. We address these problems in the well studied class of directed graphs called tournaments.
While the feedback vertex set problem is easily seen to be fixed parameter tractable in tournaments, we show that the feedback arc set problem is also fixed parameter tractable. Then we address the parametric dual problems (where the k is replaced by 'all but k' in the questions) and show that they are fixed parameter tractable in oriented directed graphs (where there is at most one directed arc between a pair of vertices). More specifically, the dual problem we show fixed parameter tractable are: Given an oriented directed graph, is there a subset of k vertices (arcs) that forms an acyclic directed subgraph of the graph?

1 Introduction

We explore efficient fixed parameter algorithms for the directed feedback set problems and their parametric dual problems in tournaments under the framework introduced by Downey and Fellows[3].

In the framework of parameterized complexity, a problem with input size n and parameter k is fixed parameter tractable (FPT) if there exists an algorithm to solve the problem in $O(f(k)n^{O(1)})$ time where f is any function of k. Such an algorithm is quite useful in practice for small ranges of k (against a naive $n^{k+O(1)}$ algorithm). Some of the well known fixed parameter tractable problems include parameterized versions of Vertex Cover, MaxSat and Max Cut (see [3]). On the contrary, for the parameterized versions of problems like Clique and Dominating Set the best known algorithms have only $n^{k+O(1)}$ running time and these problems are also known to be hard for some parameterized complexity classes (see [3] for details). The central aim of the study of parameterized complexity is to identify problems exhibiting this contrasting behaviour.

Given a directed graph on n vertices and an integer parameter k, the feedback vertex (arc) set problem asks whether the given graph has a set of k vertices

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 484–492, 2003.
© Springer-Verlag Berlin Heidelberg 2003

(arcs) whose removal results in an acyclic directed graph. While these problems in undirected graphs are known to be FPT [11] (in fact the edge version in undirected graphs can be trivially solved), the parameterized complexity of these problems in directed graphs is a long standing open problem in the area. In fact, there are problems, on sequences and trees in computational biology, that are related to the directed feedback vertex set problem [5]. In this paper, we address these feedback set problems (and their duals, explained later) for the well studied special class of directed graphs – tournaments.

A tournament $T = (V, E)$ is a directed graph in which there is exactly one directed arc between every pair of vertices. Since a tournament has a directed cycle if and only if it has a directed triangle, one can first find a directed triangle in the tournament, and then branch on each of its three vertices to get an easy recursive $O(3^k n^3)$ algorithm to find a feedback vertex set of size at most k (or determine its absence). Alternatively we can write the feedback vertex set problem in tournaments as a 3-hitting set problem (feedback vertex set is a hitting set of all directed triangles in the tournament) and can apply the algorithm of [9] to get an $O(2.27^k + n^3)$ algorithm.

There are parameterized reductions between feedback vertex set problem and feedback arc set problem in directed graphs (actually the NP-complete reductions for these problems are parameterized reductions [4]), but they don't preserve the tournament structure. Also, it is not sufficient to hit all triangles by arcs to get a feedback arc set in a tournament. Furthermore, after we remove an arc from a tournament, we no longer have a tournament. Hence it is not straightforward to apply the ideas of the fixed parameter tractable algorithms for feedback vertex set to the arc set problem. In Section 2, we show that the feedback arc set problem is also fixed parameter tractable by giving an $O((c\sqrt{k/e})^k n^\omega \lg n)$ algorithm (where ω is the exponent of the best matrix multiplication algorithm, e is the base of the natural logarithm and c is some positive constant).

In Section 3, we consider the parametric duals of these feedback set problems. More specifically the dual problems are: Given a directed graph G, (a) is there a set of at least k vertices of G that induces a directed acyclic graph, and (b) is there a directed acyclic subgraph of G with at least k arcs? In undirected graphs, the former ((a)) question is $W[1]$-complete [7] while the latter question is easily solvable in polynomial time (since in any connected graph on n vertices and m edges, it is necessary and sufficient to remove $m - (n + 1)$ edges to make it acyclic).

In directed graphs where cycles of length 2 are allowed, we show that the parametric dual of the feedback vertex set problem is $W[1]$-hard, while it is fixed parameter tractable for oriented directed graphs (where cycles of length 2 are not allowed). We show that the dual of the feedback arc set problem is fixed parameter tractable in general directed graphs. We also consider variations of these problems where the parameter is above the default lower bound. In Section 4, we conclude with some remarks and open problems. By an oriented directed graph, we mean a directed graph where there is at most one directed arc between

every pair of vertices. By an inneighbour of a vertex x in a directed graph G, we mean a vertex y such that there is a directed arc from y to x in G. By $\lg n$ we mean the logarithm to the base 2 of n.

2 Feedback Arc Set Problem in Tournaments

Feedback vertex set problem is known to be NP-Complete for tournaments [12] but its arc counterpart is still open for unweighted tournaments. It is conjectured to be NP-Complete[2]. In this section we give a $O(\sqrt{k}^k n^\omega \lg n)$ algorithm for the feedback arc set problem in tournaments. Our algorithm relies on the following main lemma.

Lemma 1. *Let $G = (V, E)$ be a directed graph such that $|V| = n$ and $|E| \geq \binom{n}{2} - k$, for some non-negative integer k. Then either G is acyclic or has a directed cycle of length at most $c\sqrt{k}$ for some positive constant c.*

Proof. Assume G is not acyclic and choose c such that $c^2k - 3c\sqrt{k} \geq 2k$ ($c = 3$ suffices for $k \geq 2$). Note that the shortest directed cycle C of G is chordless; i.e. for all non-adjacent pairs of vertices u, v in C, there is no arc (u, v) or (v, u) since otherwise that arc between u and v will give raise to a shorter directed cycle. Suppose that the length l of the shortest directed cycle C in G is strictly greater than $c\sqrt{k}$. Then G misses $\binom{l}{2} - l = l(l-3)/2 > (c^2k - 3c\sqrt{k})/2 \geq k$ arcs. This is a contradiction since G has at least $\binom{n}{2} - k$ arcs. □

Now we are ready to show the following theorem.

Theorem 1. *Given a tournament $T = (V, E)$, we can determine whether it has a feedback arc set of size at most k in $O((c\sqrt{k/e})^k n^\omega \lg n)$ time, where n^ω is the running time for the best matrix multiplication algorithm, and c is a positive constant. I.e. the feedback arc set problem is fixed parameter tractable in tournaments.*

Proof. We will give an algorithm *TFES* which constructs a search tree for which each node has at most $c\sqrt{k}$ children. Each node in the tree is labeled with a set of vertices S that represents a partially constructed feedback arc set.
Algorithm TFES$(T = (V, E)$**,** $k)$ (* $k \geq 0$ *)
(returns a feedback arc set of size k in G if there is one and returns NO otherwise).

- *Step 1*: Find a shortest cycle C in T, if exists.
- *Step 2*: If G is acyclic, then answer YES (T has a FES of size at most k), return $\emptyset$ and EXIT.
- *Step 3*: If $k = 0$, then answer NO and EXIT.
- *Step 4:* If for some arc $e = (u, v) \in C$, TFES$(T', k-1)$ is true, where $T' = (V, E - e)$ then answer YES and return $\{e\} \cup TFES(T', k-1)$, else answer NO.

If the algorithm exits at Step 2, then G can be made acyclic by deleting no arc and hence its answer is correct (since $k \geq 0$). If it exits at Step 3, then G has a cycle and so it can't be made acyclic by deleting $k = 0$ arcs, and so its answer is correct. Finally the correctness of Step 3 follows from the fact that any feedback arc set must have one of these arcs of the cycle C, and the step is recursively checking for each arc e in the cycle whether $T - e$ has a feedback arc set of size at most $k - 1$.

To show that the algorithm takes the claimed bounds, observe that since k decreases at every recursive Step 3 (after an edge deletion), the recursion depth is at most k. Also the resulting directed graph after the i-th step of the recursion has at most i arcs deleted from a tournament. Hence Lemma 1 applies and so there is a cycle of length at most $c\sqrt{i}$ in the resulting graph after the i-th step. So the number of nodes in the search tree is $O(c^k\sqrt{k!})$ and the claimed bound follows from Stirling's approximation. Here, we assume that the shortest cycle in a directed graph can be found in $O(n^\omega \lg n)$ time [6]. □

From the above algorithm and Lemma 1, we observe the following for dense directed graphs.

Corollary 1. *Let $G = (V, E)$ be a directed graph with n vertices and $m \geq \binom{n}{2} - l$ arcs. Let l and k be fixed integer parameters. Then it is fixed parameter tractable to determine whether there are k arcs (vertices) whose removal makes the resulting graph acyclic.*

Proof. To start with, G will have a directed cycle of length at most $c\sqrt{l}$, by Lemma 1 . At the recursive steps of the algorithm TFES, the resulting graph will have at least $\binom{n}{2} - l - k$ arcs and so the resulting directed graph will a cycle of length at most $c\sqrt{l+k}$. Since l and k are parameters, the result follows. □

3 Parametric Duals

The parametric dual of a parameterized problem with parameter k is the same problem with k replaced by 'all but k' ([7], [1]). For example, the parametric dual of the k-vertex cover is the $(n - k)$- vertex cover or equivalently k-independent set problem. As in this case, typically parameterized dual problems have complementary parameterized complexity ([7], [1]).

In this section, we show that the parametric dual problems of the directed feedback set problems are themselves some natural optimization problems and their parameterized versions are fixed parameter tractable in oriented directed graphs. This might be an evidence that parameterized versions of directed feedback vertex set ($DFVS$) and directed feedback arc set ($DFAS$) problems are possibly W hard, in oriented directed graphs (using the strong but informal, notion of parametric duality; see [1] or [7]).

3.1 Parametric Dual of Directed Feedback Vertex Set

Here, the question is, given a directed graph on n vertices, are there at most $n-k$ vertices whose removal makes the graph acyclic. Or equivalently, is there a set of at least k vertices that induces an acyclic directed graph? We call this the $maxv-acyclic$ subgraph problem. We show that it is fixed parameter tractable in oriented directed graphs and $W[1]$ hard in general directed graphs where cycles of length 2 are allowed.

Since every tournament has a vertex with outdegree at least $(n-1)/2$, the following lemma [2] is immediate.

Lemma 2. *Every tournament T on n vertices contains an acyclic (transitive) subtournament on $\lfloor \lg n \rfloor$ vertices and can be found in the $O(n^2)$ time.*

Since any oriented directed graph can be completed to a tournament by adding the missing arcs (with arbitrary directions), we observe the following corollary for oriented directed graphs.

Corollary 2. *Every oriented directed graph G on n vertices and m arcs has at least $\lfloor \lg n \rfloor$ vertices that induce an acyclic subgraph and such a subgraph can be found in $O(n^2)$ time.*

We give another algorithm to find the acyclic subgraph defined in the above corollary taking $O(m \lg n)$ time.

Algorithm TS ($G = (V, E)$
(returns an acyclic subgraph $H = (S, E^{'})$

$S \leftarrow \emptyset$
Repeat
  Let u be a vertex in G with the smallest in degree in G.
  $S \leftarrow S \bigcup \{u\}$;
  $G \leftarrow G - (\{v \mid v$ is an inneighbour of u in $G\} \bigcup \{u\})$;
Until G is empty

Lemma 3. *Let G be an oriented directed graph with n vertices and m arcs. Then the algorithm TS finds a subset S of vertices such that $|S| \geq \lfloor \lg n \rfloor$ and the induced subgraph of G on S is acyclic. The running time of the algorithm is $O(m|S|)$.*

Proof. Let the oriented directed graph G be given as an adjacency list to TS where associated with every vertex x is a list of vertices y such that y is an inneighbour of x. It is easy to implement each step of the 'repeat' loop in $O(m)$ time (We can have a bitvector for the list of vertices to be deleted and scan through the adjacency list and remove those vertices.).

It is also clear that the induced subgraph on S is acyclic. For, if we order the vertices by the order in which they are included in S, then the arcs go only from smaller vertices to bigger vertices.

To show that $|S| \geq \lfloor \lg n \rfloor$, it suffices to show that the 'repeat' loop will execute for at least $\lfloor \lg n \rfloor$ steps. This follows because in any oriented directed graph there is a vertex with indegree at most $(n-1)/2$. Thus, after one step of the loop at most $(n+1)/2$ vertices are deleted. □

The following corollary follows from Corollary 2 and Lemma 3.

Corollary 3. *Let G be an oriented directed graph with n vertices and m arcs. Then there exists a subset of $\lfloor \lg n \rfloor$ vertices which induces an acyclic subgraph and it can be found in $O(\min\{m \lg n, n^2\})$ time.*

Now we can design a fixed parameter tractable algorithm for the parameterized $maxv - acyclic$ subgraph problem as follows. If $k \leq \lfloor \lg n \rfloor$, then answer YES, otherwise $n \leq 2^k$ and then we check all k sized subsets of the vertex set to see whether the subset induces an acyclic subgraph. If any one of them does, then we answer YES, otherwise answer NO. Since $\binom{n}{k} \leq \binom{2^k}{k} \leq (e2^k/k)^k$ in the latter case, we have the following theorem. (Here e is the base of natural logarithm.)

Theorem 2. *Given an oriented directed graph G and an integer k, we can determine whether or not G has at least k vertices that induce an acyclic subgraph in time $O((e2^k/k)^k k^2 + min(\{m \lg n\}, \{n^2\})$; i.e $maxv-acyclic$ subgraph problem is fixed parameter tractable.*

The fixed parameter tractable algorithm for the $maxv - acyclic$ subgraph problem follows from the easy observation that there is a "guarantee" (lower bound) of $\lfloor \lg n \rfloor$ for the solution size. In such situations, it is natural to parameterize above the guarantee [8] and so a natural question to ask is whether a given directed graph has a set of at least $\lfloor \lg n \rfloor + k$ vertices that induces an acyclic subgraph. The parameterized complexity of this question is open.

Now we show that $maxv - acyclic$ subgraph problem is $W[1]$ hard in general directed graphs. We reduce the $W[1]$-hard parameterized Independent Set problem in undirected graphs to $maxv - acyclic$ subgraph problem in directed graphs. For the proof of $W[1]$ hardness of the Independent set problem and to find more about reductions and $W[1]$ classes see [3].

Theorem 3. *It is $W[1]$-hard to determine whether a given directed graph has k vertices that induce an acyclic subgraph; i.e. $maxv - acyclic$ subgraph problem is $W[1]$-hard in directed graphs.*

Proof. We will reduce the k-independent set problem in undirected graphs to the given problem. Given an undirected graph $G = (V, E)$, an instance of the independent set problem we construct $D = (V, E')$, an instance of $maxv-acyclic$ subgraph problem in directed graph by adding both arcs $u \to v$ and $v \to u$ for every (u, v) in E. If G has an independent set of size k, then those corresponding vertices of D form an acyclic subgraph. Conversely, if D has an acyclic subgraph on k vertices, then those k vertices must form an independent set in D as if there is an arc between a pair of vertices in D, then there actually is a directed cycle (of length 2) between them. Thus those k vertices form an independent set in G. □

3.2 Parametric Dual of Directed Feedback Arc Set

Here the problem is, given a directed graph on n vertices and m arcs, are there at most $m - k$ arcs whose removal makes the graph acyclic. Or equivalently, is

there a set of at least k arcs that induces an acyclic directed graph? We call this the $maxe - acyclic$ subgraph problem and show that it is fixed parameter tractable. It follows from the following easy lemma.

Lemma 4. *Given a directed graph G on n vertices and m arcs, there always exists a set of at least $\lceil m/2 \rceil$ arcs that form an acyclic directed graph. Such a set of arcs can be found in $O(m)$ time.*

Proof. Order the vertices of the directed graph G arbitrarily. If m is the number of arcs in the graph, then at least $\lceil m/2 \rceil$ of these arcs go in one direction (all from a smaller vertex to a higher vertex or vice versa). Choose these arcs and they form an acyclic directed graph. □

Theorem 4. *Given a directed graph G on n vertices and m arcs and an integer parameter k, we can determine whether or not G has at least k arcs that form an acyclic subgraph in time $O(4^k k + m)$, i.e $maxe - acyclic$ subgraph problem is FPT in directed graphs.*

Proof. If $k \leq m/2$, then say YES, otherwise $m \leq 2k$, and then check all subsets of k of the arc set of the graph. If any of these k subsets of arcs induces an acyclic subgraph, then say YES, and answer NO otherwise.

Since $\binom{m}{k} \leq 2^m \leq 2^{2k}$, and we can check in $O(k)$ time if k arcs forms an acyclic graph, we have the desired running time. □

Just as in the parametric dual of the feedback vertex set problem, a natural parameterized question here is whether the given directed graph has a set of at least $\lceil m/2 \rceil + k$ arcs that form an acyclic subgraph. This question is open for directed graphs. In fact, $\lceil m/2 \rceil$ is a tight lower bound for the solution size in directed graphs. This bound is realized, for example, in a directed graph obtained by taking an undirected path on n vertices (and n edges) and replacing every edge by a pair of directed arcs one in each direction.

However, this bound of $\lceil m/2 \rceil$ is not tight for oriented directed graphs. We prove that there exists an acyclic subgraph on $m/2 + 1/2\lceil (n-c)/2 \rceil$ arcs in any oriented directed graph and then use this to give fixed parameter algorithm for the question 'whether the given oriented directed graph has a set of at least $m/2 + k$ arcs that forms an acyclic subgraph'.

To prove our result, we mimic the proof of the the following lemma proved in [10].

Lemma 5. *[10] If G is a simple undirected graph with m edges, n vertices and c components, then the maximum number of edges in a bipartite subgraph of G is at least $m/2 + 1/2\lceil (n-c)/2 \rceil$. Such a bipartite graph can be found in $O(n^3)$ time.*

Lemma 6. *Any oriented directed graph $G = (V, E)$ with m arcs and n vertices, with the underlying undirected graph having c components, has an acyclic subgraph with at least $m/2 + 1/2\lceil (n-c)/2 \rceil$ arcs and such a subgraph can be found in $O(n^3)$ time.*

Proof. Without loss of generality assume that the underlying undirected graph is connected, otherwise we will apply this lemma on each component to get the result. The proof is exactly along the lines of the proof of Lemma 5. Just for completion, we give the main steps.

The proof is by induction on the number of vertices. The lemma is clearly true for oriented directed graphs on 1 or 2 vertices. At the induction step, there are three cases. In case 1, we assume that the underlying undirected graph has a cut vertex x. In this case, we apply induction on each of the 2-connected components of $G - x$ including x in each component. In case 2, we assume that the underlying undirected graph has no cut vertex and the oriented directed graph G has a vertex u whose indegree and outdegree are not the same. In this case we apply induction on $G - u$ (whose underlying undirected graph will be clearly connected), and also include all arcs coming into x or all arcs going out of x whichever set is larger, to get an acyclic subgraph in the resulting directed graph. In case 3, the underlying undirected graph is Eulerian, and has no cut vertex. Hence there exists a pair of adjacent vertices u and v such that $G-\{u, v\}$ is connected (for a proof see [10]). Here, we apply induction on $G - \{u, v\}$ and include all arcs coming into u and all arcs coming into v. It is easy to verify that the resulting set of arcs forms an acyclic subgraph, and is of the required cardinality and can be found in the claimed bound. See [10] for details. □

Theorem 5. *Let G be an oriented directed graph on n vertex and m arcs. Let c be the number of components in the underlying undirected graph. Then given an integer k, we can determine whether or not G has at least $m/2 + k$ arcs which forms an acyclic subgraph in time $O(c2^{O(k^2)}k^2 + m + n^3)$.*

Proof. First, find all the c components of the underlying undirected graph corresponding to G. If $k \leq 1/2\lceil(n-c)/2\rceil$, then say YES, else $k > 1/2\lceil(n-c)/2\rceil \geq (n-c)/4 - 1$ or $n \leq 4k + 4 + c$. Thus n_i, the number of vertices in the i-th component is at most $n - (c-1) \leq 4k + 5$. Hence the number of arcs in each of the components is $O(k^2)$. By trying all subsets of the arcs in the component, we can find m_i, the maximum number of arcs of the i-th component that form an acyclic subgraph, for any i, in $O(2^{O(k^2)}k^2)$ time. If $\sum_{i=1}^{c} m_i \geq \lceil m/2\rceil + k$ then answer YES and otherwise answer NO. □

4 Conclusions

In this paper, we have given fixed parameter tractable (FPT) algorithms for the feedback vertex set and the feedback arcs set problem in the special class of directed graphs, namely tournaments. These problems are still not known to be FPT even in oriented directed graphs. We have given FPT algorithms for the parametric duals of directed feedback vertex and arc set problems in oriented directed graphs.

Getting a $O(c^k)$ algorithm (for some constant c) for the directed feedback arc set problem in tournaments is an interesting open problem. In line with

parameterizing above the guaranteed values, the parameterized complexity of the following questions are also interesting.

- Given an oriented directed graph on n vertices, does it have a subset of at least $\lfloor \lg n \rfloor + k$ vertices that induces an acyclic subgraph?
- Given an oriented directed graph on n vertices and m arcs, does it have a subset of at least $m/2 + 1/2(\lceil n - 1/2 \rceil) + k$ arcs that induces an acyclic subgraph ?
- Given a directed graph on n vertices and m arcs, does it have a subset of at least $m/2 + k$ arcs that induces an acyclic subgraph ?

Acknowledgement. We thank P. Paulraja for fruitful discussions and comments on an earlier version of the paper.

References

1. V. Arvind, M. R. Fellows, M. Mahajan, V. Raman, S. S. Rao, F. A. Rosamond and C. R. Subramanian, "Parametric Duality and Fixed Parameter Tractability", manuscript 2001.
2. J. Bang-Jensen and G. Gutin, 'Digraphs Theory, Algorithms and Applications', Springer Verlag 2001.
3. R. Downey and M. R. Fellows, 'Parameterized Complexity', Springer Verlag 1998.
4. G. Even, J. (Seffi) Naor, B. Schieber, M. Sudan, 'Approximating Minimum Feedback Sets and Multicuts in Directed Graphs', *Algorithmica* **20** (1998) 151–174.
5. M. Fellows, M. Hallett, C. Korostensky, U. Stege, 'Analogs and Duals of the MAST Problem for Sequences and Trees', *in Proceedings of 6th Annual European Symposium on Algorithms (ESA '98)*, Venice, Lecture Notes in Computer Science **1461** (1998) 103–114.
6. A. Itai and M. Rodeh, 'Finding a Minimum Circuit in a Graph', *Siam Journal of Computing* **7** (4) (1978) 413–423.
7. S. Khot and V. Raman, 'Parameterized Complexity of Finding Subgraphs with Hereditary Properties', *Theoretical Computer Science* **289** (2002) 997–1008.
8. M. Mahajan and V. Raman, 'Parameterizing above Guaranteed Values: MaxSat and MaxCut', *Journal of Algorithms* **31** (1999) 335–354.
9. R. Niedermeier and P. Rossmanith, 'An efficient Fixed Parameter Algorithm for 3-Hitting Set', *Journal of Discrete Algorithms* **2** (1) (2001).
10. S. Poljak and D. Turzik, 'A Polynomial Algorithm for Constructing a Large Bipartite Subgraph, with an Application to a Satisfiability Problem',*Canad. J. Math* **34**, No.3 (1982) 519–524.
11. V. Raman, S. Saurabh and C. R. Subramanian, 'Faster Fixed Parameter Tractable Algorithms for Undirected Feedback Vertex Set', *in the Proceedings of the 13th International Symposium on Algorithms and Computation (ISAAC 2002)*, Lecture Notes in Computer Science **2518** (2002) 241–248.
12. E. Speckenmeyer, 'On Feedback Problems in Digraphs', *in Proceedings of the 15th International Workshop WG'89*, Lecture Notes in Computer Science **411** (1989) 218–231.

Compact Visibility Representation and Straight-Line Grid Embedding of Plane Graphs*

Huaming Zhang and Xin He

Department of Computer Science and Engineering,
SUNY at Buffalo, Buffalo, NY, 14260, USA
{huazhang,xinhe}@cse.buffalo.edu

Abstract. We study the properties of Schnyder's realizers and canonical ordering trees of plane graphs. Based on these newly discovered properties, we obtain compact drawings of two styles for any plane graph G with n vertices. First we show that G has a *visibility representation* with height at most $\lceil \frac{15n}{16} \rceil$. This improves the previous best bound of $n-1$. The drawing can be obtained in linear time. Second, we show that every plane graph G has a straight-line grid embedding on an $(n-\Delta_0-1)\times(n-\Delta_0-1)$ grid, where Δ_0 is the number of cyclic faces of G with respect to its minimum realizer. This improves the previous best bound of $(n-1)\times(n-1)$. This embedding can also be found in $O(n)$ time.

1 Introduction

The concepts of *canonical ordering* and *canonical ordering tree* of plane triangulations and tri-connected plane graphs played crucial roles in designing several graph drawing algorithms [5,6,7,9,11,12,13]. Recently, Chang et. al. generalized these concepts to arbitrary connected plane graphs [4], which leads to improvements in several graph-drawing algorithms [3,4].

A *visibility representation* (VR for short) of a 2-connected plane graph G is a drawing of G, where the vertices of G are represented by non-overlapping horizontal segments (called *vertex segments*), and each edge of G is represented by a vertical line segment touching the vertex segments of its end vertices. The problem of computing a compact VR is important not only in algorithmic graph theory, but also in practical applications such as VLSI layout. A VR of a plane graph G can be obtained from an *st-numbering* of G and the corresponding *st*-numbering of its dual G^* [18,21]. Using this approach, the height of the VR is bounded by $(n-1)$ and the width of the VR is bounded by $(2n-5)$ [18,21].

Some work have been done to reduce the width of the VR by carefully choosing a special *st*-numbering of G. Kant proved that every plane graph has a VR with width at most $\lfloor \frac{3n-6}{2} \rfloor$ [13]. Very recently, Lin et. al. reduced the width to $\lfloor \frac{22n-42}{15} \rfloor$ by choosing the best *st*-numbering from three *st*-numberings derived from a Schnyder's realizer of G [16]. However, the height of the VR of a general plane graph remains to be the trivial bound of $n-1$. In this paper we prove

* Research supported in part by NSF Grant CCR-9912418.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 493–504, 2003.
© Springer-Verlag Berlin Heidelberg 2003

that every plane graph G has a VR with height at most $\lceil \frac{15n}{16} \rceil$, which can be obtained in linear time.

Finding a *straight-line grid embedding* of a plane graph G is another extensively studied problem. It is known for long time that such an embedding exists. However, the drawings produced by earlier algorithms require grids whose size is exponential in n (see the references cited in [6]). A breakthrough was achieved in [6,7]: it was shown that such an embedding can be done on a $(2n-4)\times(n-2)$ grid. The grid size was reduced to $(n-1)\times(n-1)$ in [20]. For a 4-connected plane triangulation G with at least four exterior vertices, the size of the grid can be reduced to $n/2 \times n/2$ [8,17]. It is known that there exists a plane graph whose straight line embedding requires a grid of size at least $\lceil 2n/3 \rceil \times \lceil 2n/3 \rceil$. It has been conjectured that every plane graph G has such an embedding on a $\lceil 2n/3 \rceil \times \lceil 2n/3 \rceil$ grid. However, the best known bound remains to be $(n-1)\times(n-1)$ given in [20]. In this paper, we show that every plane graph has a straight-line grid embedding on a $(n-\Delta_0-1)\times(n-\Delta_0-1)$ grid, where Δ_0 is the number of clockwise cyclic faces of G with respect to its *minimum realizer* $\mathcal{R}_0$. While Δ_0 can be 0 in the worst case, the number of clockwise cyclic faces with respect to $\mathcal{R}_0$ is the largest among all realizers of G. Our work is the first result that reduces the grid size below $n-1$ by a non-trivial parameter.

The present paper is organized as follows. Section 2 introduces definitions and preliminary results. Section 3 presents the construction of a VR with height at most $\lceil \frac{15n}{16} \rceil$. In Section 4, we present the algorithm for compact straight-line grid embedding.

2 Preliminaries

In this section, we give definitions and preliminary results. $G = (V, E)$ denotes a graph with $n = |V|$ vertices and $m = |E|$ edges. A *planar graph* G is a graph which can be drawn on the plane without edge crossings. Such a drawing (if one exists) is called an *embedding* of G on the plane. A *plane graph* is a planar graph with a fixed embedding. The embedding of a plane graph divides the plane into a number of regions, called *faces*. The unbounded region is the *exterior face*. Other regions are *interior faces*. The vertices and the edges on the exterior face are called *exterior vertices* and *exterior edges*. Other vertices and edges are called *interior vertices* and *interior edges*. If all facial cycles of G are triangles, G is called a *plane triangulation*. We abbreviate the words "counterclockwise" and "clockwise" as ccw and cw respectively.

An *orientation* of a graph G is a digraph obtained from G by assigning a direction to each edge of G. We will use G to denote both the resulting digraph and the underlying undirected graph unless otherwise specified. (Its meaning will be clear from the context).

Let G be a plane graph with two specified exterior vertices s and t. An orientation of G is called an *st-orientation*, if it is acyclic with s as the only source and t as the only sink,

Let G be a plane graph and s, t two distinct exterior vertices of G. An *st-numbering* of G is a one-to-one mapping $\xi : V \to \{1, 2, \cdots, n\}$, such that $\xi(s) = 1$, $\xi(t) = n$, and each vertex $v \neq s, t$ has two neighbors u, w with $\xi(u) < \xi(v) < \xi(w)$, where u (w, resp.) is called a *smaller neighbor (bigger neighbor, resp.)* of v. Given an *st*-numbering ξ of G, we can orient G by directing each edge in E from its lower numbered end vertex to its higher numbered end vertex. The resulting orientation will be called the *orientation derived from* ξ. Obviously, this orientation is an *st*-orientation of G. On the other hand, if $G = (V, E)$ has an *st*-orientation $\mathcal{O}$, we can define a 1-1 mapping $\xi : V \to \{1, \cdots, n\}$ by using topological sort. It is easy to see that ξ is an *st*-numbering and the orientation derived from ξ is $\mathcal{O}$.

Lempel et. al. [14] showed that for every 2-connected plane graph G and any two exterior vertices s and t, there exists an *st*-numbering ξ of G. Thus, G has an *st*-orientation derived from ξ, with s as the only source and t as the only sink.

The following VR drawing algorithm was given in [18,21]:

Lemma 1. *Let G be a 2-connected plane graph with n vertices. Let ξ be an st-numbering of G. A VR of G can be obtained from ξ in linear time. The height of the VR is the length of the longest directed path in the st-orientation of G derived from ξ.*

Let G be a plane triangulation with $n \geq 3$ vertices and $m = 3n - 6$ edges. Let $v_1, v_2, \cdots, v_n$ be an ordering of the vertices of G where v_1, v_2, v_n are the three exterior vertices of G in ccw order. For a fixed ordering, let G_k denote the subgraph of G induced by $v_1, v_2, \cdots, v_k$ and H_k the exterior face of G_k. Let $G - G_k$ be the subgraph of G obtained by removing $v_1, v_2, \cdots, v_k$.

Definition 1. [7] *An ordering $v_1, \cdots, v_n$ of a plane triangulation G is* canonical *if the following hold for every $k = 3, \cdots, n$;*

1. *G_k is biconnected, and its exterior face H_k is a cycle containing the edge (v_1, v_2).*
2. *The vertex v_k is on the exterior face of G_k, and its neighbors in G_{k-1} form a subinterval of the path $H_{k-1} - (v_1, v_2)$ with at least two vertices. Furthermore, if $k < n$, v_k has at least one neighbor in $G - G_k$. (Note that the case $k = 3$ is degenerated, and $H_2 - (v_1, v_2)$ is regarded as the edge (v_1, v_2) itself.)*

Let G be a plane triangulation with canonical ordering $v_1, v_2, \cdots, v_n$. At step k, when v_k is added to construct G_k, let $c_l, c_{l+1}, \cdots, c_r$ be the lower ordered neighbors of v_k from left to right on the exterior face of G_{k-1}. We call (v_k, c_l) the *left edge* of v_k, (v_k, c_r) the *right edge* of v_k, and the edges (c_p, v_k) with $l < p < r$ the *internal edges* of v_k. The collection T of the left edges of the vertices v_j for $3 \leq j \leq n$ plus the edge (v_1, v_2) is a spanning tree of G and is called a *canonical ordering tree* of G [7,10].

Let T be a rooted spanning tree of a plane graph G. Two distinct vertices of G are *unrelated* with respect to T if neither of them is an ancestor of the other in T. An edge of G is *unrelated* with respect to T if its end vertices are unrelated. While traveling T in ccw (cw, resp.) preorder (postorder, resp.), if each vertex of

G is assigned a number from $\{1, 2, \cdots, n\}$ according to the order being visited, the resulting ordering is called the *ccw (cw, resp.) preordering (postordering, resp.)* of the vertices of G with respect to T.

In [4], the concept of canonical ordering tree was generalized to any connected plane graph, which they call *orderly spanning tree*, as follows:

Definition 2. *Let G be a plane graph and T be a spanning tree of G. Let $v_1, v_2, \cdots, v_n$ be the ccw preordering of the vertices of G with respect to T.*

(1) A vertex v_i of G is orderly *with respect to T if the neighbors of v_i in G form the following four blocks in ccw order around v_i:*

$B_1(v_i)$: the parent of v_i,

$B_2(v_i)$: the unrelated neighbors v_j of v_i with $j < i$,

$B_3(v_i)$: the children of v_i, and

$B_4(v_i)$: the unrelated neighbors v_j of v_i with $j > i$.

where each block could be empty.

(2) T is called an orderly spanning tree *of G if v_1 is an exterior vertex, and each v_i $(1 \leq i \leq n)$ is orderly with respect to T.*

The following is another related concept called Schnyder's realizer [19,20]:

Definition 3. *Let G be a plane triangulation with three exterior vertices v_1, v_2, v_n in ccw order. A* realizer $\mathcal{R}$ *of G is a partition of the interior edges on G into three sets T_1, T_2, T_n of directed edges such that the following hold.*

- *For each $i \in \{1, 2, n\}$, the interior edges incident to v_i are in T_i and directed toward v_i.*
- *For each interior vertex v of G, the neighbors of v form six blocks U_1, D_n, U_2, D_1, U_n, and D_2 in ccw order around v, where U_j and D_j $(j = 1, 2, n)$ are the parent and the children of v in T_j.*

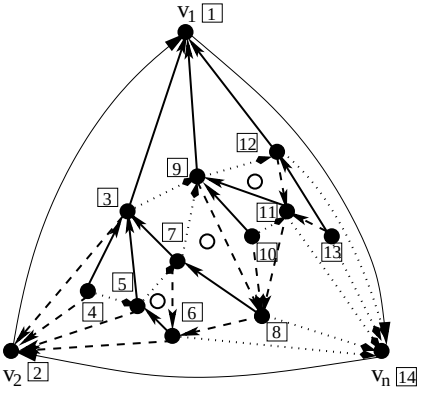


Fig. 1. A plane triangulation G and the minimum realizer of G.

It was shown in [19,20] that every plane triangulation G has a realizer $\mathcal{R}$, and each T_i ($i \in \{1, 2, n\}$) is a tree rooted at the vertex v_i containing all interior vertices of G. Fig. 1 shows a realizer of a plane triangulation G. Three trees T_1, T_2, T_n are drawn as thick solid line, dashed line, and dotted line, respectively. (Ignore the small boxes containing integers and the directions of the exterior edges for now. Their meaning will be explained later.)

We summarize known results in the following lemma [3,15,16,19]:

Lemma 2. *Let G be a plane triangulation with n vertices.*

1. *If T is an orderly spanning tree of G, then the ccw preordering of the vertices of G with respect to T is a canonical ordering of G. Hence T is also a canonical ordering tree of G. A canonical ordering tree T' of G is also an orderly spanning tree of G.*
2. *Let $\{T_1, T_2, T_n\}$ be a realizer of G, where T_i is rooted at v_i for each $i \in \{1, 2, n\}$. Then the union of T_i and the two exterior edges of G incident to v_i is an orderly spanning tree of G.*
3. *A realizer of G can be obtained from an orderly spanning tree T in $O(n)$ time, where one of the tree in the realizer is obtained from T by removing the two edges on the exterior face.*

The following lemma shows the connection between canonical ordering trees and *st*-numberings, the proof is omitted.

Lemma 3. *Let G be a plane triangulation and T be a canonical ordering tree (or equivalently, an orderly spanning tree) of G rooted at v_1. Let $v_1, v_2, \cdots, v_n$ be the ccw preordering of the vertices of G with respect to T. Then this ordering is an st-numbering of G.*

For example, consider the tree T_1 (rooted at v_1) shown in Fig. 1. The union of T_1 and the two exterior edges (v_2, v_1) and (v_n, v_1) is a canonical ordering tree of G, by Lemma 2 (1) and (2). Denote it by ${T_1}'$. The ccw preordering of the vertices of G with respect to ${T_1}'$ are shown in integers inside the small boxes. It is an *st*-numbering of G by Lemma 3.

Let G be a plane triangulation and v_1, v_2, v_n be the exterior vertices of G in ccw order. Let $\mathcal{R} = \{T_1, T_2, T_n\}$ be a realizer of G. We direct the exterior edges as $v_1 \rightarrow v_n$, $v_n \rightarrow v_2$, $v_2 \rightarrow v_1$. The resulting orientation is called the *orientation induced by* $\mathcal{R}$ and denoted by $G(\mathcal{R})$. Note that $G(\mathcal{R})$ is **never** an *st*-orientation because it is always cyclic.

Let $\Delta(\mathcal{R})$ denote the number of interior cyclic faces of $G(\mathcal{R})$. Let ξ_1, ξ_2, ξ_n be the number of internal (namely non-leaf) vertices in the trees T_1, T_2, T_n, respectively. The following results were proved in [1,2,16]:

Lemma 4. *Let G be a plane triangulation with n vertices. Let v_1, v_2, v_n be the exterior vertices of G in the ccw order.*

1. *Let $\mathcal{R} = \{T_1, T_2, T_n\}$ be any realizer of G, where T_i, $i \in \{1, 2, n\}$ is rooted at v_i respectively. Then $\xi_1 + \xi_2 + \xi_n - \Delta(\mathcal{R}) = n - 1$.*

2. *There is an unique realizer $\mathcal{R}_0$ of G such that all interior cyclic faces of $G(\mathcal{R}_0)$ are directed in cw direction. $\mathcal{R}_0$ can be obtained in $O(n)$ time.*

The realizer $\mathcal{R}_0$ stated in the statement (2) of Lemma 4 is called the *minimum realizer* of G [1,2]. Among all realizers of G, $G(\mathcal{R}_0)$ has the largest number of interior cyclic faces in cw direction. $\mathcal{R}_0$ plays a central role in our algorithms for compact VR representation and straight-line grid embedding.

For example, the realizer shown in Fig. 1 is actually the minimum realizer $\mathcal{R}_0$ of G. The orientation of G drawn in the figure is the orientation of $G(\mathcal{R}_0)$. It has three cw interior cyclic faces, marked by empty circles. We will call such faces the cyclic faces of $G(\mathcal{R}_0)$. (Note that although the exterior face is a cyclic face in $G(\mathcal{R}_0)$, we do not count it in $\Delta(\mathcal{R}_0)$ since it is not an interior face).

In the induced orientation $G(\mathcal{R}_0)$:

- $\mathcal{F}_{3cw}$ denotes the set of interior faces of $G(\mathcal{R}_0)$ with three cw edges and let $\Delta_0 = |\mathcal{F}_{3cw}|$.
- $\mathcal{F}_{2cw}$ denotes the set of interior faces of $G(\mathcal{R}_0)$ with two cw edges and one ccw edge and let $\alpha_0 = |\mathcal{F}_{2cw}|$.
- $\mathcal{F}_{1cw}$ denotes the set of interior faces of $G(\mathcal{R}_0)$ with one cw edge and two ccw edges and let $\beta_0 = |\mathcal{F}_{1cw}|$.

Note that $\mathcal{F}_{3cw}, \mathcal{F}_{2cw}, \mathcal{F}_{1cw}$ form a partition of the set of the interior faces of $G(\mathcal{R}_0)$. The following theorem can be proved by using Lemma 4.

Theorem 1. *1.* $\Delta_0 + \alpha_0 = n - \Delta_0 - 1$.
2. $\Delta_0 \leq \lfloor \frac{n-1}{2} \rfloor$.
3. G has a canonical ordering tree with at least $\lceil \frac{n+1}{2} \rceil$ *leaves.*

3 Compact Visibility Representation

In this section we present our theorem on the height of the compact VR.

Let G be a plane triangulation and v_1, v_2, v_n the exterior vertices in ccw order. Let T be a canonical ordering tree of G rooted at v_1. T is an orderly spanning tree of G by Lemma 2 (1). Let ρ be the ccw preordering of the vertices of G with respect to T. In this ordering, $\rho(v_1) = 1, \rho(v_2) = 2$ and $\rho(v_n) = n$. For any vertex v other than v_1, v_2, v_n, v has a nonempty set $B_2(v)$ of smaller neighbors and a nonempty set $B_4(v)$ of bigger neighbors.

We will construct two vertex numberings of G according to T simultaneously. The first vertex numbering ξ_T of G is defined as follows:

Step 1: Traveling from the leftmost unassigned leaf of T by **ccw postordering** with respect to T. (The first visited vertex is v_2). Stop assigning numbers when we reach either the next leaf of T, or the root v_1. When we reach v_1, the numbering process is complete. When we reach a leaf, do not assign a number to it at this moment. Continue to step 2 if there are leaves remaining to be traveled.

Step 2: Traveling from the rightmost unassigned leaf of T by **cw postordering** with respect to T. (Initially, it is v_n.) Stop assigning numbers when we reach

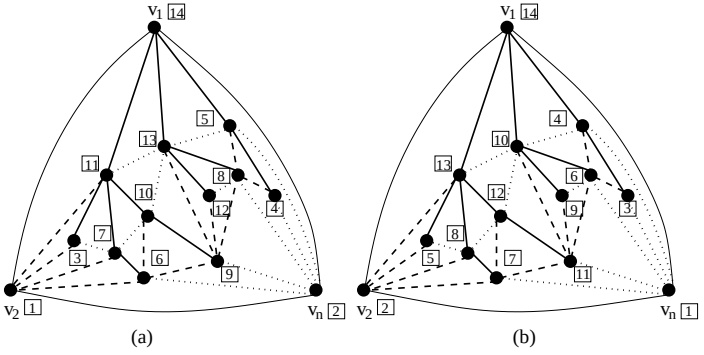


Fig. 2. Two *st*-numberings of the graph G in Fig. 1.

either the next leaf of T, or the root v_1. When we reach v_1, we are done. When we reach a leaf, do not assign a number to it at this moment. Loop back to step 1 if there are leaves remaining to be traveled.

For example, in Fig. 2 (a), ξ_T visits v_2 first in step 1, then it moves to step 2 to visit v_n, then it loops back to step 1 to visit vertex ordered 3, followed by step 2 to visit vertices ordered 4 and 5, and so on, until it reaches v_1.

The second vertex numbering ξ'_T is defined similarly, except that the order of step 1 and step 2 are swapped. Note: $\xi'_T(v_n) = 1$, $\xi'_T(v_2) = 2$ and $\xi'_T(v_1) = n$.

Fig. 2 (a) and (b) show two such numberings for the graph G in Fig. 1. They are constructed from the canonical ordering tree ${T_1}'$, which is obtained from the minimum realizer in Fig. 1 by applying Lemma 2 (2). The integers inside the small boxes are the numbers assigned to the vertices.

We omit the proof of the following technical lemma:

Lemma 5. *Let G be a plane triangulation, T a canonical ordering tree of G. Then $\xi_T, {\xi'}_T$ are two st-numberings of G. The st-orientation of G derived from ξ_T (ξ'_T, resp.) will be denoted by G_T (G'_T, resp.).*

Let v', v'' (u', u'', resp.) be the $(2i-1)$-th and $2i$-th leaf traveled by ξ_T in step 1 (step 2, resp.). Then, u', u'' (v', v'', resp.) are the $(2i-1)$-th and $2i$-th leaf traveled by ξ'_T in its own step 1 (step 2, resp.). These four leaves will be called the *i-th block of leaves of* T. For example, in Fig. 2 (b), the vertices $v' = 2, v'' = 5, u' = 1, u'' = 3$ are the first block of leaves of T'_1. The order of the four leaves in the i-th block traveled by ξ_T (ξ'_T, resp.) is v', u', v'', u'' (u', v', u'', v'', resp.). Though it is possible that some non-leaf vertices of T are traveled by ξ_T (ξ'_T, resp.) between the four leaves.

Lemma 6. *Let G be a plane triangulation, T a canonical ordering tree of G. Let P be a directed path in G_T, P' a directed path in G'_T. Then for any block of leaves of T, one of P and P' cannot pass through all the four vertices in the block.*

Proof. If P does not pass through all the four vertices in the block, we are done. Let's assume that P passes through all the four vertices. Let $v'_1, \cdots, v'_{t_1}$ (can

be degenerated to empty set) be all the vertices traveled by ξ_T between v' and u'. Then $v'_1, \cdots, v'_{t_1}$ are all the vertices traveled between v' and u'' by ξ'_T. Let $u'_1, \cdots, u'_{t_2}$ (can be empty set) be all the vertices traveled by ξ_T between u' and v''. Then $u', u'_1, \cdots, u'_{t_2}$ are on the unique path of T from u' to the root v_1, denote this unique path by Q_1. Let $v''_1, \cdots, v''_{t_3}$ (can be empty set) be all the vertices traveled by ξ_T between v'' and u''. Then $v'', v''_1, \cdots, v''_{t_3}$ are on the unique path of T from v'' to the root v_1, denote this unique path by P_1.

P passes through both u' and v'', and $u'_1, \cdots, u'_{t_2}$ are all the vertices traveled by ξ_T between u' and v'', so there is a vertex w such that the edge (w, v'') is in P, and $\xi_T(v'') > \xi_T(w) \geq \xi_T(u')$. Then either w is u' or w is u'_j for some $j \in \{1, \cdots, t_2\}$. Denote the unique path from w to v_1 in T by Q_2, the unique path from the parent of u'_{t_2} to v_1 in T by Q_3. Both Q_2 and Q_3 are parts of Q_1. See Fig. 3 for an illustration, some edges on P are drawn in solid lines.

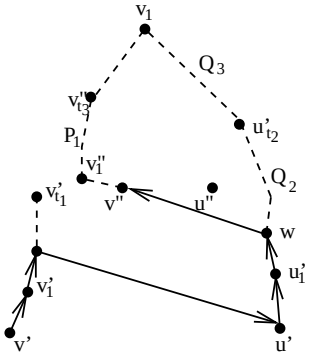


Fig. 3. The proof of Lemma 6.

The edge (w, v''), the path P_1 and the path Q_2 enclose a closed region, denote this region by R. Now, consider the position of u'' with respect to R. Because Q_1 is the path connects u' to the root v_1 in T and u'' is a leaf of T other than u', u'' cannot be on Q_1. Therefore, u'' cannot be on Q_2. Similarly, u'' cannot be on P_1. So either u'' is inside R, or u'' is outside of R.

Suppose u'' is outside of R. There are two possibilities of the unique path in T from u'' to the root v_1: (1) It intersects with P_1 from left. Then ξ_T travels u'' before v'', which is impossible. (2) It intersects with Q_2 from right. (Note, in this case, the intersection vertex has to be on the path Q_3.) Then ξ_T travels u'' before u', which is also impossible. Therefore, u'' has to be inside R.

Similarly, it can be shown that all $v', v'_j, 1 \leq j \leq t_1$ are outside of R. Therefore none of v' or $v'_j, 1 \leq j \leq t_1$ can be a neighbor of u''.

Consider the path P'. If v' is not on P', we are done. Assume v' is on P', $v'_1, \cdots, v'_{t_1}$ are all the vertices traveled between v' and u'' by ξ'_T, so if P' passes through u'' after it passes through v', it has to reach u'' from one of $v', v'_1, \cdots, v'_{t_1}$. However, none of $v', v'_j, 1 \leq j \leq t_1$ is a neighbor of u'', so there is no way for

P' to pass through u''. Thus P' cannot pass through all the four vertices in this block of leaves. □

Theorem 2. *Every plane graph G with n vertices has a VR with height at most $\lceil \frac{15n}{16} \rceil$, which can be obtained in linear time.*

Proof. Without loss of generality, we assume G is a plane triangulation. By Theorem 1 (3), from the minimal realizer $\mathcal{R}_0$ of G, we can obtain a canonical ordering tree T of G with at least $l \geq \lceil \frac{n+1}{2} \rceil$ leaves. Thus T has at least $\lfloor \frac{n}{8} \rfloor$ disjoint blocks of leaves. Therefore, T induces two st-orientations G_T and G'_T by Lemma 5. This can be done in $O(n)$ time.

Let P be a longest directed path in G_T, P' a longest directed path in G'_T. By Lemma 6, for any block of leaves of T, one of P or P' has to bypass at least one leaf in the block. T has at least $\lfloor \frac{n}{8} \rfloor$ disjoint blocks of leaves, thus P and P' have to bypass at least $\lfloor \frac{n}{8} \rfloor$ leaves all together. One of them has to bypass at least $\lfloor \frac{n}{16} \rfloor$ leaves. Therefore, its length is at most $\lceil \frac{15n}{16} \rceil$. So the length of the longest directed path in one of G_T, G'_T is at most $\lceil \frac{15n}{16} \rceil$.

Choosing the st-numbering which gives shorter longest directed path, we then apply the VR algorithm in Lemma 1 to the chosen st-numbering. It results in a VR of G with height at most $\lceil \frac{15n}{16} \rceil$. The running time is $O(n)$. □

For example, in Fig. 2 (b), the longest directed path in the orientation derived from the given st-numbering passes through the vertices numbered $1, 3, 4, 6, 9, 10, 11, 12, 13, 14$. However, in the st-numbering shown in Fig. 1, the longest directed path passes through all vertices.

4 Compact Straight-Line Grid Embedding

In this section we present our theorem on the size of a compact straight-line grid embedding. Without loss of generality, we assume G is a plane triangulation.

Define an order relation $<_{lex}$ on two pairs of numbers (x_1, x_2) and (y_1, y_2) as follows: $(x_1, x_2) <_{lex} (y_1, y_2)$ iff either $x_1 < y_1$, or $x_1 = y_1$ and $x_2 < y_2$. The following definition and lemma were given in [19,20].

Definition 4. *A* weak barycentric representation *of a graph $G = (V, E)$ is an injective mapping $\pi : v \in V \rightarrow (\pi_1(v), \pi_2(v), \pi_n(v)) \in R^3$ that satisfies the following conditions:*

1. *$\pi_1(v) + \pi_2(v) + \pi_n(v) = 1$ for all vertices v.*
2. *For each edge $(x, y) \in E$ and each vertex $z \notin \{x, y\}$, there is a $k \in \{1, 2, n\}$ such that $(\pi_k(x), \pi_{k-1}(x)) <_{lex} (\pi_k(z), \pi_{k-1}(z))$ and $(\pi_k(y), \pi_{k-1}(y)) <_{lex} (\pi_k(z), \pi_{k-1}(z))$ (when $k = 1$ or n, $k - 1$ denotes n or 2, respectively).*

Lemma 7. *Let $\pi : v \in V \rightarrow (\pi_1(v), \pi_2(v), \pi_n(v))$ be a weak barycentric representation of G. Then given any noncolinear points $\mathbf{a}, \mathbf{b}, \mathbf{c}$ on the plane, the mapping $f : v \in V \rightarrow \pi_1(v)\mathbf{a} + \pi_2(v)\mathbf{b} + \pi_n(v)\mathbf{c}$ defines a straight-line embedding of G in the plane spanned by $\mathbf{a}, \mathbf{b}, \mathbf{c}$.*

Let $G = (V, E)$ be a plane graph with three exterior vertices v_1, v_2, v_n in ccw order. Let $\mathcal{R}_0 = \{T_1, T_2, T_n\}$ be the minimum realizer of G. Let $\mathcal{F} = \mathcal{F}_{3cw} \cup \mathcal{F}_{2cw}$ be the set of interior faces of $G(\mathcal{R}_0)$ that have either three or two cw edges on their boundaries. Thus $|\mathcal{F}| = \Delta_0 + \alpha_0$.

For each vertex $v \in G$, let $P_1(v)$ ($P_2(v)$ and $P_3(v)$, respectively) be the path in T_1 (T_2 and T_n, respectively) from v to the root v_1 (v_2 and v_n, respectively). $P_1(v), P_2(v), P_n(v)$ divides G into three regions $R_1(v), R_2(v)$ and $R_n(v)$ where $R_i(v)$ denotes the closed region bounded by the paths $R_j(v), R_k(v)$ and the exterior edge (v_j, v_k) $(i \neq j, k)$.

For each interior vertex v of G, let $\pi_1(v)$ ($\pi_2(v)$ and $\pi_n(v)$, respectively) be the number of faces of $G(\mathcal{R}_0)$ in $R_1(v)$ ($R_2(v)$ and $R_n(v)$, respectively) that belong to $\mathcal{F}$. Note that since the two interior faces of $G(\mathcal{R}_0)$ incident to the exterior edges (v_1, v_2) and (v_1, v_n) are not in $R_1(v)$ and these two faces belong to $\mathcal{F}$, we have $\pi_1(v) \leq \Delta_0 + \alpha_0 - 2$. Similarly, we have $\pi_2(v), \pi_n(v) \leq \Delta_0 + \alpha_0 - 2$.

For the three exterior vertices v_1, v_2 and v_n, define $\pi_i(v_i) = \Delta_0 + \alpha_0$ and $\pi_j(v_i) = 0$ for $j \neq i$. Note: For all vertices v, $\pi_1(v) + \pi_2(v) + \pi_n(v) = \Delta_0 + \alpha_0$.

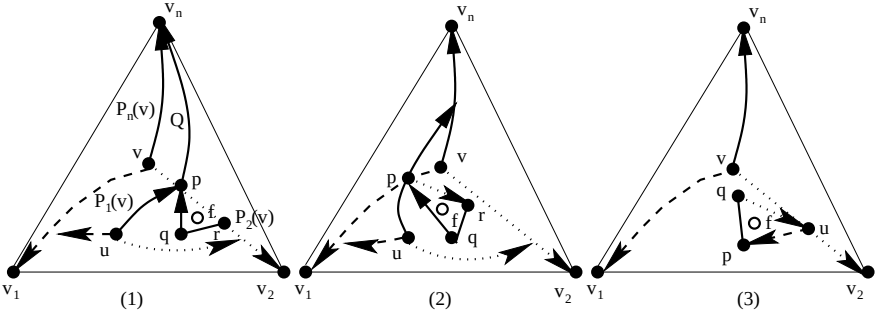


Fig. 4. The proof of Lemma 8.

Next, we present two lemmas, the proof of the second lemma is omitted.

Lemma 8. *If $u \in R_i(v) - P_{i+1}(v)$, then $\pi_i(u) < \pi_i(v)$ (when $i = 2$ or n, $i + 1$ denotes n or 1 respectively).*

Proof. We only prove the case $i = n$. The other cases are symmetric. We need to show that $u \in R_n(v) - P_1(v)$ implies $\pi_n(u) < \pi_n(v)$.

Case 1: u is in the interior of the region $R_n(v)$. Let Q be the path in T_n from u to the root v_n of T_n. Q must intersect either $P_1(v)$ or $P_2(v)$ at a vertex p.

Case 1 (a): Suppose p is on $P_2(v)$. Let r be the parent of p in T_2. Let (q, p) be the edge incident to p that is next to the edge (r, p) in cw order around p (see Fig. 4 (1).) Let f be the face of G bounded by the edges $(p, r), (r, q), (q, p)$. Note that f is in $R_n(v)$ but not in $R_n(u)$. By the property of the realizer, the edge (q, p) must be in T_n and directed toward p. The edge (p, r) is in T_2 and directed toward r. So f has at least two cw edges on its boundary, and hence is in $\mathcal{F}$. Thus, $\pi_n(u) \leq \pi_n(v) - 1 < \pi_n(v)$.

Case 1 (b): Suppose p is on $P_1(v)$ (but $p \neq v$). By the property of the realizer, there must be an edge (p, r) in T_2 directed from p to r and this edge must be in the region $R_n(v)$. (See Fig. 4 (2).) Let (q, p) be the edge incident to p that is next to the edge (r, p) in cw order around p. Let f be the face of G bounded by the edges $(p, r), (r, q), (q, p)$. Note that f is in $R_n(v)$ but not in $R_n(u)$. By the property of the realizer, the edge (q, p) must be in T_n and directed toward p. The edge (p, r) is in T_2 and directed toward r. So f has at least two cw edges on its boundary, and hence is in $\mathcal{F}$. Thus, $\pi_n(u) \leq \pi_n(v) - 1 < \pi_n(v)$.

Case 2: u is on the path $P_2(v)$. By the property of the realizer, there must be an edge (u, p) in T_1 directed toward p and this edge must be in the region $R_n(v)$. Let the edge (u, q) be the edge incident to u that is next to the edge (u, p) in cw order around u. Let f be the face bounded by the edges $(u, p), (p, q), (q, u)$. (See Fig. 4 (3).) Note that f is in $R_n(v)$ but not in $R_n(u)$. By the property of the realizer, the edge (q, u) must be in T_2 and directed toward u. The edge (u, p) is in T_1 and directed toward p. So f has at least two cw edges on its boundary and hence is in $\mathcal{F}$. Thus, $\pi_n(u) \leq \pi_n(v) - 1 < \pi_n(v)$. □

Lemma 9. *Let u and v be two distinct vertices of G. If v is an interior vertex of G and $u \in R_i(v)$, then $(\pi_i(u), \pi_{i-1}(u)) <_{lex} (\pi_i(v), \pi_{i-1}(v))$ (when $i = 1$ or n, $i - 1$ denotes n or 2 respectively).*

Theorem 3. *The mapping $\pi : v \in V \rightarrow \frac{1}{\Delta_0 + \alpha_0}(\pi_1(v), \pi_2(v), \pi_n(v))$ is a weak barycentric representation.*

Proof. The first condition of the barycentric representation is clearly satisfied, which implies the injectivity of π. We need to verify the second condition in the definition 4. Consider an edge (x, y) of G and a vertex $z \neq x, y$. If z is an exterior vertex v_i, then $\pi_i(z) = \Delta_0 + \alpha_0 > \pi_i(x), \pi_i(y)$, which implies $(\pi_i(x), \pi_{i-1}(x)) <_{lex} (\pi_i(z), \pi_{i-1}(z))$ and $(\pi_i(y), \pi_{i-1}(y)) <_{lex} (\pi_i(z), \pi_{i-1}(z))$.

Otherwise, z is an interior vertex and $x, y \in R_i(z)$ for some $i \in \{1, 2, n\}$. By Lemma 9, we have $(\pi_i(x), \pi_{i-1}(x)) <_{lex} (\pi_i(z), \pi_{i-1}(z))$ and $(\pi_i(y), \pi_{i-1}(y)) <_{lex} (\pi_i(z), \pi_{i-1}(z))$. □

Theorem 4. *Every plane triangulation with n vertices has a straight line grid embedding on a grid of size $(n - \Delta_0 - 1) \times (n - \Delta_0 - 1)$ where Δ_0 is the number of cyclic faces of G with respect to its minimum realizer $\mathcal{R}_0$. The embedding can be constructed in linear time.*

Proof. Pick three points $\mathbf{a} = (\Delta_0 + \alpha_0, 0)$, $\mathbf{b} = (0, \Delta_0 + \alpha_0)$ and $\mathbf{c} = (0, 0)$. By Theorem 3, Lemma 7 and Lemma 9, the mapping $v \rightarrow (\pi_1(v), \pi_2(v))$ is a straight-line grid embedding on a grid of size $(\Delta_0 + \alpha_0) \times (\Delta_0 + \alpha_0)$. By Lemma 1 (1), this equals to the size stated in the theorem.

To construct the embedding, we need to construct the minimal realizer $\mathcal{R}_0$ of G, which can be done in linear time [2]. The calculation of the coordinates of the embedding can be done in $O(n)$ time by using the techniques given in [20]. □

References

1. N. Bonichon, B. L. Saëc and M. De la Libération, Wagner's theorem on realizers. *Proc. of The 29th International Colloquium on Automata, Languages and Programming*, LNCS 2380, pp. 1043–1053, 2002.
2. E. Brehm, 3-Orientations and Schnyder 3-tree-decompositions. *Diploma Thesis*, FB mathematik und Informatik, Freie Universität Berlin, 2000.
3. H.-L. Chen, C.-C. Liao, H.-I. Lu and H.-C. Yen, Some applications of orderly spanning trees in graph drawing. *Proc. Graph Drawing'02*, LNCS 2528, pp. 332–343, 2002.
4. Y.-T. Chiang, C.-C. Lin and H.-I. Lu, Orderly spanning trees with applications to graph encoding and graph drawing. *Proc. of the 12th Annual ACM-SIAM SODA*, pp. 506–515, 2001.
5. U. Fößmeier, G. Kant and M. Kaufmann, 2-Visibility drawings of planar graphs. *Proc. 4th International Symposium on Graph Drawing*, LNCS 1190, pp. 155–168, 1996.
6. H. de Fraysseix, J. Pach and R. Pollack, Small sets supporting Straight-Line embeddings of planar graphs. *Proc. 20th Annual Symposium on Theory of Computing*, pp. 426–433, 1988.
7. H. de Fraysseix, J. Pach and R. Pollack, How to draw a planar graph on a grid. *Combinatorica* 10, pp. 41–51, 1990.
8. X. He, Grid Embedding of 4-connected Plane Graphs. *Discrete Comput. Geom.* 17, pp. 339–358, 1997.
9. X. He, On floor-plan of plane graphs. *SIAM Journal on Computing* 28(6), pp. 2150–2167, 1999.
10. X. He, M.-Y. Kao and H.-I. Lu, Linear-time succinct encodings of planar graphs via canonical orderings. *SIAM Journal Discrete Math* 12(3), pp. 317–325, 1999.
11. G. Kant, Drawing planar graphs using the lmc-ordering. *Proc. 33rd Symposium on Foundations of Computer Science*, Pittsburgh, pp.101–110, 1992.
12. G. Kant, Algorithms for Drawing Planar Graphs, Ph.D. Dissertation, Department of Computer Science, University of Utrecht, 1993.
13. G. Kant, A more compact visibility representation. *International Journal of Computational Geometry & Applications* 7(3), pp. 197–210, 1997
14. A. Lempel, S. Even and I. Cederbaum, An algorithm for planarity testing of graphs. *Theory of Graphs (Proc. of an International Symposium, Rome, July 1966)*, pp. 215–232, 1967.
15. C.-C. Liao, H.-I. Lu and H.-C. Yen, Floor-planning via orderly spanning trees. *Proc. 9th International Symposium on Graph Drawing* (GD 2001), LNCS 2265, pp. 367–377, 2002.
16. C.-C. Lin, H.-I. Lu and I-F. Sun, Improved Compact Visibility Representation of Planar Graph via Schnyder's Realizer. *Proc. 20th Annual Symposium on Theoretical Aspects of Computer Science*, LNCS 2607, pp. 14–25, 2003.
17. K. Miura, S. Nakano and T. Nishizeki, Grid Drawings of 4-Connected Plane Graphs. *Discrete Comput. Geom.* 26, pp. 73–87, 2001.
18. P. Rosenstiehl and R. E. Tarjan, Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete Comput. Geom.* 1, pp. 343–353, 1986.
19. W. Schnyder, Planar graphs and poset dimension. *Order* 5, pp. 323–343, 1989.
20. W. Schnyder, Embedding planar graphs on the grid. *Proc. of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 138–148, 1990.
21. R. Tamassia and I.G.Tollis, An unified approach to visibility representations of planar graphs. *Discrete & Computational Geometry* 1(4), pp. 321–341, 1986.

New Directions and New Challenges in Algorithm Design and Complexity, Parameterized

Michael R. Fellows

School of Electrical Engineering and Computer Science
University of Newcastle, University Drive, Callaghan NSW 2308, Australia
mfellows@cs.newcastle.edu.au

Abstract. The goals of this survey are to:
(1) Motivate the basic notions of parameterized complexity and give some examples to introduce the toolkits of FPT and W-hardness as concretely as possible for those who are new to these ideas.
(2) Describe some new research directions, new techniques and challenging open problems in this area.

1 Introduction

Let's immediately consider a compelling concrete motivation for the subject of parameterized complexity. Almost every computational problem that has been considered to date has turned out to be NP-hard (or worse). Vast efforts have subsequently been expended on the program of understanding the extent to which NP-hard problems might be approximated in polynomial time.

Thanks to the wonderful PCP Theorem [ALMSS92], it has been shown that:
• Large numbers, perhaps the majority, of the "classic" NP-hard optimization problems of computing cannot be approximated to within any constant factor, in polynomial time, unless $P = NP$.
• Of the remaining NP-hard problems, many do not admit *polynomial-time approximation schemes* (PTAS's) unless $P = NP$.
• But, "*good news*" (in a generally bleak landscape): dozens of hard problems do admit PTAS's. This means that for any fixed ϵ, there is an algorithm to compute a solution within a factor of $(1 + \epsilon)$ of optimal in polynomial time. (For a comprehensive account of approximation complexity see [ACGKMP99].) Upon examination, however, most of these PTAS results appear to be completely useless.

- The PTAS for the Euclidean TSP due to Arora [Ar96] has a running time of around $O(n^{300/\epsilon})$. Thus for a 20% error, we have a "polynomial-time" algorithm that runs in time $O(n^{1500})$.
- The PTAS for the Multiple Knapsack problem due to Chekuri and Khanna [CK00] has a running time of $O(n^{12(\log(1/\epsilon)/\epsilon^8)})$. Thus for a 20% error we have a "polynomial-time" algorithm that runs in time $O(n^{9375000})$.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 505–519, 2003.
© Springer-Verlag Berlin Heidelberg 2003

- The PTAS for the MAXIMUM SUBFOREST PROBLEM due to Shamir and Tsur [ST98] has a running time of $O(n^{2^{2^{1/\epsilon}-1}})$. For a 20% error we thus have a "polynomial" running time of $O(n^{958267391})$.
- The PTAS for the MAXIMUM INDENDENT SET problem on geometric graphs due to Erlebach, Jansen and Seidel [EJS01] has a running time of $O(n^{(4/\pi)(1/\epsilon^2+2)^2(1/\epsilon^2+1)^2})$. Thus for a 20% error, $O(n^{532804})$.
- The PTAS for the GENERAL MULTIPROCESSOR JOB SCHEDULING PROBLEM due to Chen and Miranda [CM99] runs in time $O(n^{(3mm!)^{(m/\epsilon)+1}})$ for m machines. Thus for 4 machines with a 20% error we have an algorithm that runs in time $O(n^{10000000000000000000000000000000000000000000000000000000000000000000000000000})$.

Most PTAS's are like that. The qualitative problem with the polynomial running times is that the parameter $k = 1/\epsilon$ is in the exponent. In some cases, an initial PTAS classification has subsequently been improved to one that does not have this problem. Arora described in 1997 an improved, nearly linear-time PTAS for the EUCLIDEAN TSP [Ar97]. A contemporary challenge is *to investigate which problems that have PTAS's admit PTAS's with any hope of practical significance, as exemplified by Arora's work on the Euclidean TSP.* The essential issue is how two measurements: n, the overall input size, and $k = 1/\epsilon$, the goodness-of-approximation parameter, qualitatively interact. The one-dimensional complexity framework of P *versus* NP is unable to speak to this. The two-dimensional parameterized complexity perspective provides the appropriate tools. The basic definitions are as follows.

Definition 1. *A parameterized language L is a subset $L \subseteq \Sigma^* \times \Sigma^*$. If L is a parameterized language and $(x, k) \in L$ then we will refer to x as the* main part, *and refer to k as the* parameter.

Definition 2. *A parameterized language L is fixed-parameter tractable (FPT) if it can be determined in time $f(k)q(n)$ whether $(x, k) \in L$, where $|x| = n$, $q(n)$ is a polynomial in n, and f is a function (unrestricted).*

The analogous additive "$f(k) + q(n)$" definition of FPT is equivalent.

Definition 3. *A parameterized language L belongs to the class XP if it can be determined in time $f(k)n^{g(k)}$ whether $(x, k) \in L$, where $|x| = n$, where f and g are unrestricted functions.*

It is known that FPT is (unconditionally) a proper subset of XP [DF99].

Examples of Parameterized Problems That Are in XP

(1) All of the problems that have PTAS's, where the parameter is $k = 1/\epsilon$.
(2) Many graph problems that take as input a graph G and a positive integer (parameter) k, such as determining whether a graph has a k-element dominating set, which is solvable in time $O(n^{k+1})$ by brute force checking of all k-subsets.
(3) The GRAPH ISOMORPHISM PROBLEM, parameterized by maximum vertex degree [Luks82].

(4) The μ-calculus model-checking problem, parameterized by formula size [EJ91].
(5) The problem of determining whether there is a relational structure homomorphism $h : A \to B$, parameterized by the hypertreewidth of A [GLS99].

Examples of Parameterized Problems That Are FPT

(1) The problem of approximating a solution to the Euclidean TSP problem, where the parameter is $k = 1/\epsilon$, the goodness of the approximation [Ar97].
(2) Many (maybe half) of the NP-hard graph problems that take as input a graph G and a positive integer parameter k, such as the VERTEX COVER problem, for which the current best FPT algorithm runs in time $O((1.29)^k + kn)$ [NR99,CKJ01]. Another nice example is the MAX LEAF SPANNING TREE problem. The trajectory of FPT algorithms for this problem includes:
• A quadratic FPT algorithm based on graph minors [FL92] having a completely ridiculous parameter function.
• The first linear time FPT algorithm for the problem due to Bodlaender [Bod93] had a parameter function of $f(k) = (17k^4)!$ or thereabouts.
• A linear time FPT algorithm by Downey-Fellows with $f(k) = (2k)^{4k}$ [DF95a].
• An improved linear time FPT algorithm due to Fellows, McCartin, Rosamond and Stege [FMRS00] with parameter function $f(k) = (14.23)^k$.
• The current best FPT algorithm for the problem, due to Bonsma, Brüggemann and Woeginger, running in time $O(n^3 + 9.4815^k k^3)$ [BBW03].
(3) The INTEGER LINEAR PROGRAMMING problem, where the parameter is the number of variable [Len83].
(4) The problem of determining whether a graph H is a minor of a graph G, where the parameter is H [RS95].
(5) The problem of determining whether there is a relational structure homomorphism $h : A \to B$, for structures of bounded arity, parameterized by the treewidth of the core of A [Gr03].

A parameter can be many things, and a single classical problem may be parameterized in a number of relevant ways. A parameter may be compound. The CLOSEST SUBSTRING problem is interestingly parameterized by the aggregate parameter consisting of: (1) the number of sequences, (2) the size of the alphabet, and (3) the Hamming distance from the input substrings to the motif [FGN03]. (Whether this parameterization of the problem is FPT is an open problem.)

In the next two sections we will survey, with some easy examples, the two toolkits that a working algorithmist might be interested in mastering. In the next section we survey the **bad news toolkit** of $W[1]$-hardness, the parameterized analog of the classical notion of NP-hardness that provides us with the means to show that some parameterized problems are unlikely to be in FPT. We will also discuss the new idea of $M[1]$-hardness, and its "lower bounds" applications. In §3 we will overview the **good news toolkit** of techniques for designing FPT algorithms. We will describe some striking new methods using vertex cover structure generally, and discuss the practical significance of FPT. In §4 we will summarize some of the main challenges for this area of research.

2 The Bad News Toolkit of $W[1]$ and $M[1]$ Hardness

The GRAPH k-CUT problem has a well-known $O(n^{k^2})$ algorithm due to Goldschmidt and Hochbaum [GH88,GH94] (also [KS96]).

GRAPH k-CUT
Input: A graph $G = (V, E)$, an edge-weighting function $w : E \to \mathbb{N}$, and positive integers k, m.
Parameter: k
Question: Is there a set $C \subseteq E$ with $\sum_{e \in C} w(e) \leq m$, such that the graph $G' = G - C$ obtained from G by removing the edges of the *cut* C has at least k connected components?

An $O(n^{k^2})$ algorithm is impractical for $k > 2$, and the same question arises as with the PTAS's. Before getting into the definitions that set up $W[1]$-hardness, let's look at a simple concrete example of the kind of problem reduction needed for the theory. (This example is taken from [DEFPR03].) Note that this reduction looks very similar to an NP-completeness reduction, and that it is *not difficult.*

Theorem 1. *If* GRAPH k-CUT *is FPT then so is the problem of determining whether a graph has a k-clique.*

Proof. We reduce the k-CLIQUE problem to GRAPH k-CUT. Let $G = (V, E)$ be the graph on n vertices for which we wish to determine if G has a k-clique. We construct a graph G' as follows:
(1) Start with $n + 2$ disjoint cliques of size n^4. We will refer to these as the *top clique* C_0, the *bottom clique* C_1 and the n *vertex cliques* C_v, $v \in V$.
(2) For each $v \in V$, connect C_v to C_1 by a matching of size $n^2 + n - deg(v)$.
(3) For each $v \in V$, connect C_v to C_0 by a matching of size $\binom{k}{2}$.
(4) For each edge $uv \in E$, make a single edge from C_u to C_v.
Each edge weight is 1; take $m = k(n^2 + n) + (k - 1)\binom{k}{2}$; take $k' = k + 1$.

We claim that G has a k-clique if and only if (G', k', m) is a yes-instance for the GRAPH k-CUT problem.

If G has a k-clique on the k vertices $V' \subseteq V$, then we can cut out k vertex cliques as follows (leaving the rest in a further component, noting $k' = k + 1$):
(1) To cut the k vertex cliques C_u, $u \in V'$, from C_1 requires almost all of the m cuts we are allowed. After we have accomplished this much, we have only $(k - 1)\binom{k}{2} + \sum_{u \in V'} deg(u)$ further cuts that we can make.
(2) The second term in the sum of our remaining budget of cuts from (1) allows us to cut the single edges joining the cliques corresponding to V' to the cliques corresponding to $V - V'$, and to each other (that is, the single edges between C_u and C_v where both $u, v \in V'$), requiring a number of cuts equal to all of this term, except for a savings of 1 for each adjacency of vertices in V'. Since there are $\binom{k}{2}$ such adjacencies, we have a remaining budget of $k\binom{k}{2}$ after this stage of cutting.
(3) The remaining budget of cuts allows us to cut all of the cliques corresponding to V' free from C_0.

The other direction is nearly obvious from the discussion above, and is easily checked.

To finish the argument, suppose that the GRAPH k-CUT problem has an FPT algorithm, that is, one that runs in time $f(k)n^c$ (where c is a constant and f is some function, unrestricted, as per the definition of FPT). The combinatorial reduction described above transforms an instance (G, k) of the k-CLIQUE problem to an instance (G', k') of the GRAPH k-CUT problem, where $k' = k+1$. But then we can solve k-CLIQUE in time $f(k+1)n^{c'}$ where $c' = 5c$.

A key point above is that the parameter k in the source problem is mapped to parameter k' of the target so that k' is purely a function of k.

Definition 4. *A parameterized language L is many:1 parametrically reducible to a parameterized language L' if there is an FPT algorithm that transforms (x, k) into (x', k') so that:*
(1) $(x, k) \in L$ if and only if $(x', k') \in L'$, and
(2) $k' = g(k)$ (for some function g).

A notion of problem reduction allows us to explore equivalence classes of parameterized problems. The main classes are organized in the hierarchy:

$$FPT \subseteq M[1] \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[P] \subseteq AW[P] \subseteq XP$$

The maining working tool (so far) for showing likely intractability is $W[1]$-hardness, the parameterized analogue of NP-hardness. (The class $M[1]$ is something new, and we will say more about it below.) The k-CLIQUE problem is $W[1]$-complete [DF99], and is a frequent source for $W[1]$-hardness proofs. $W[1]$ is strongly analogous to NP because the k-STEP HALTING PROBLEM FOR NONDETERMINISTIC TURING MACHINES (with unlimited nondeterminism) is complete for $W[1]$ ([DF95b] + [CCDF97]).

2.1 How Do We Analyze PTAS's?

The following definition captures the essential issue.

Definition 5. *An optimization problem Π has an* efficient P-time approximation scheme *(EPTAS) if it can be approximated to a goodness of $(1+\epsilon)$ of optimal in time $f(k)n^c$ where c is a constant and $k = 1/\epsilon$.*

The following important connection was first proved by Bazgan, and later independently by Cesati and Trevisan [Baz95,CT97].

Theorem 2. *Suppose that Π_{opt} is an optimization problem, and that Π_{param} is the corresponding parameterized problem, where the parameter is the value of an optimal solution. Then Π_{param} is fixed-parameter tractable if Π_{opt} has an EPTAS.*

Here is an example of how Bazgan's Theorem can be used to analyze PTAS complexity. Khanna and Motwani introduced three planar logic problems in an effort to give a general explanation of PTAS-approximability. Their suggestion is that "hidden planar structure" in the logic of an optimization problem is what allows PTASs to be developed [KM96]. One of their three general planar logic optimization problems is defined as follows.

PLANAR TMIN
Input: A collection of Boolean formulas in sum-of-products form, with all literals positive, where the associated bipartite graph is planar (this graph has a vertex for each formula and a vertex for each variable, and an edge between two such vertices if the variable occurs in the formula).
Output: A truth assignment of minimum weight (i.e., a minimum number of variables set to *true*) that satisfies all the formulas.

By Bazgan's Theorem, it is enough to show that the problem of deciding whether there is a truth assignment of weight k, taking this as the parameter, is $W[1]$-hard.

Theorem 3. *Planar TMIN is hard for* $W[1]$ *and therefore does not have an EPTAS unless* $FPT = W[1]$.

The other two planar logic problems of Khanna and Motwani are also $W[1]$-hard [CFJR01]. The reductions are all from the parameterized CLIQUE problem and are comparable in difficulty to the example we have seen concerning the GRAPH k-CUT problem.

2.2 A Surprising New Development

A natural (and useful!) class of parameterized problems that may be intermediate between FPT and $W[1]$ has recently been identified. The situation is:

$$FPT \subseteq M[1] \subseteq W[1]$$

The combinatorics of $M[1]$ are rich and easy to work with. It may actually be a better primary reference point for intractability than $W[1]$. There are two natural "routes" to $M[1]$.
• First of all, there are FPT algorithms for many parameterized problems (e.g., VERTEX COVER) that run in time $2^{O(k)}n^c$. VERTEX COVER has a simple $2^k n$ FPT algorithm that we can consider here as an example. We now "renormalize" and define the problem:

$k \log n$ VERTEX COVER
Input: A graph G on n vertices and a positive integer k.
Parameter: k
Question: Does G have a vertex cover of size at most $k \log n$?

The simple FPT algorithm for the original VERTEX COVER problem, parameterized by the number of vertices in the vertex cover, allows us to solve this new problem in time $O(n^{k+1})$. It now makes sense to ask if the $k \log n$ VERTEX

Cover problem is in FPT — or is it $W[1]$-hard? It turns out that $k \log n$ Vertex Cover is $M[1]$-complete. Call this the *renormalization route* to $M[1]$.
• There is another route to $M[1]$ via *miniaturization*. We certainly know an algorithm to solve n-variable 3SAT in time $O(2^n)$. So consider the following parameterized problem.

Mini-3SAT
Input: Positive integers k and n in unary, and a 3SAT expression E having at most $k \log n$ variables.
Parameter: k
Question: Is E satisfiable?
Using our exponential time algorithm for 3SAT, Mini-3SAT is in XP and we can wonder where it belongs — is it is FPT or $W[1]$-hard? This problem also turns out to be complete for $M[1]$.

Dozens of renormalized FPT problems and miniaturized arbitrary problems are now known to be $M[1]$-complete [DFPR03]. However, what is known is quite problem-specific. For example, one might expect Mini-Max Leaf to be $M[1]$-complete, but all that is known presently is that it is $M[1]$-hard. It is not known to be $W[1]$-hard, nor is it known to belong to $W[1]$.

The following theorem would be interpreted by most people as indicating that probably $FPT \neq M[1]$. (The theorem is essentially due to Cai and Juedes [CJ01].)

Theorem 4. *$FPT = M[1]$ if and only if n-variable 3SAT can be solved in time $2^{o(n)}$.*

Importantly, it is pretty easy to use $M[1]$ for various purposes. It supports convenient although unusual combinatorics. For example, one of the problems that is $M[1]$-complete is the miniature of the Independent Set problem defined as follows.

Mini-Independent Set
Input: Positive integers k and n in unary, a positive integer $r \leq n$, and a graph G having at most $k \log n$ vertices.
Parameter: k
Question: Does G have an independent set of size at least r?

The following example of some of the combinatorial possibilities available in making reductions from an $M[1]$-hard problem is due to [DEFPR03].

Theorem 5. *There is an FPT reduction from* Mini-Independent Set *to ordinary parameterized* Independent Set *(parameterized by the number of vertices in the independent set).*

Proof. Let $G = (V, E)$ be the miniature, for which we wish to determine whether G has an independent set of size r. Here, of course, $|V| \leq k \log n$ and we may regard the vertices of G as organized in k blocks $V_1, ..., V_k$ of size $\log n$. We now employ a simple but useful *counting trick* that can be used when reducing

miniatures to "normal" parameterized problems. Our reduction is a Turing reduction, with one branch for each possible way of writing r as a sum of k terms, $r = r_1 + \cdots + r_k$, where each r_i is bounded by $\log n$. The reader can verify that $(\log n)^k$ is an FPT function, and thus that there are an allowed number of branches. A branch represents a commitment to choose r_i vertices from block V_i (for each i) to be in the independent set.

We now produce (for a given branch of the Turing reduction) a graph G' that has an independent set of size k if and only if the miniature G has an independent set of size r, distributed as indicated by the commitment made on that branch. The graph G' consists of k cliques, together with some edges between these cliques. The ith clique consists of vertices in 1:1 correspondence with the subsets of V_i of size r_i. An edge connects a vertex x in the ith clique and a vertex y in the jth clique if and only if there is a vertex u in the subset $S_x \subseteq V_i$ represented by x, and a vertex v in the subset $S_y \subseteq V_j$ represented by y, such that $uv \in E$.

Verification that the reduction works correctly is straightforward.

The hypothesis that $FPT \neq M[1]$ allows us to say some very interesting things. Chor, Fellows and Juedes have recently proved the following using arguments similar to the proof of Theorem 5.

Theorem 6. *If $FPT \neq M[1]$ then determining whether a graph has a k-element dominating set (k-element independent set) cannot be solved in time $O(n^{o(k)})$.*

Cai and Juedes [CJ01] proved the following path-breaking results, which established an **FPT optimality program** of broad applicability.

Theorem 7. *If $FPT \neq M[1]$ then there cannot be an FPT algorithm for the general* Vertex Cover *problem with a parameter function of the form $f(k) = 2^{o(k)}$, and there cannot be an FPT algorithm for the* Planar Vertex Cover *problem with a parameter function of the form $f(k) = 2^{o(\sqrt{k})}$.*

3 The Good News Toolkit of FPT Methods

The best available general reference regarding FPT methods is Rolf Niedermeier's recent Habilitationschrift [Nie02], that we can only hope will be expanded into a comprehensive monograph. In a few pages, we can only point to some of the range and mathematical depth of this area, and sample a few key ideas and recent developments.

Some of the methods in the FPT toolkit, such as **well-quasi-ordering and graph minors** depend on very deep results that provide what are best regarded as *FPT classification tools*. These are powerful and general, but because of that generality, unlikely to yield directly useful FPT algorithms. The FPT classification just gives a starting point — recall the trajectory of improved FPT algorithms for Max Leaf discussed in §1.

Such classification tools can be quite easy to use. For example, to *use* the graph minors well-quasiordering (and accompanying GRAPH MINOR CONTAINMENT FPT result) of Robertson and Seymour [RS95] (based on nearly a thousand journal pages of intricate argument) to conclude that GRAPH GENUS is FPT — one only needs to check that, for every fixed k, the family of graphs that have genus at most k, is closed under the local operations that define the minor order: (1) deleting an edge, (2) deleting an isolated vertex, and (3) contracting an edge. The results of Robertson and Seymour on graph minors were observed to initially classify VERTEX COVER, MAX LEAF and many other problems as FPT in this way [FL92].

A less expensive alternative for these classifications and some others (via well-quasiordering methodlogy) is established by the following theorem [AFLR03].

Theorem 8. *For every fixed k, the family of graphs having a vertex cover of size at most k is well-quasiordered by ordinary subgraphs. Furthermore, restricting to this family of graphs, the* SUBGRAPH *problem (Is H a subgraph of G?), parameterized by H, is linear time FPT.*

The proof of this theorem requires less than two pages. It is more general and easier to use (than graph minors) — on those parameterized problems that are subject to *bounded vertex cover structure*. For example, VERTEX COVER could be classified as FPT by the following algorithm:
(1) Compute a maximal matching by a greedy procedure. If the matching has size greater than k then the answer is NO.
(2) If the size of the matching is at most k, then we have a $2k$ vertex cover, and can use Theorem 8, noting that the NO instances (for fixed k) are closed in the subgraph order.

The same argument works to classify as FPT the problem MAX INTERNAL of determining whether a graph has a spanning tree with at least k internal vertices — although in case (1) the answer is YES rather than NO (because an easy induction shows that if a connected graph has a k-matching, then it is a YES instance for k-MAX INTERNAL). A better FPT algorithm for this problem is described elsewhere in these proceedings [PS03].

Duality. It is an important fact that it makes sense to parameterize "upward" as well as "downward" for NP-hard problems. Note that a graph G has a k-element independent set, if and only if G has a vertex cover of size $n - k$, if and only if the complement of G has a k-clique. The naturally parameterized INDEPENDENT SET (and CLIQUE) problems are thus *parametrically dual* in a sense whose importance was first recognized by Raman and Mahajan [MR99, KR00]. They noted the remarkable (but not universal) phenomenon that for a great many NP-hard problems, one of a dual pair will be $W[1]$-hard, and the other will be FPT.

Many of the exciting new applications in computational biology of the hugely successful new parallel implementations of the current best FPT VERTEX COVER algorithms of Dehne [DRST02], and Langston, et al. [ALSS03], *are really applications of* CLIQUE*, in various data clustering contexts, that are being solved*

"through the back door" by means of the FPT vertex cover algorithm. Initial results for their joint implementations suggest practicality for reduced graphs (not subject to further kernelization) for k up to something like 2,000. This means that for $n \leq 2,000$, the CLIQUE problem can be solved exactly by deleting a minimum vertex cover from the dual. The "method" of **parametric duality** is this: if a problem is $W[1]$-hard, consider the dual!

Taking our own advice, we note that DOMINATING SET is $W[2]$-complete [DF99], and that GRAPH COLORING (parameterized by the number of colors) is hard for $W[P]$ [DFS99]. The parametric duals of both of these problems are easily classified as FPT by using Theorem 8. (The graph minor results do not apply to either of these problems.)

Another powerful and deep FPT classification technique is the **method of color-coding** introduced by Alon, Yuster and Zwick [AYZ95], and exposited also in [DF99,Nie02]. This is mathematically interesting, although not yet of direct practical importance.

The FPT method of **bounded variable integer linear programming** [Len83] has been used to classify a number of problems as FPT for which no other FPT algorithm is known. The method is exposited in [Nie02] with some important examples. The practicality of this FPT methodology also has yet to be determined.

3.1 The Universal Parameter of Treewidth

The FPT algorithmics of **bounded treewidth** seems to be of universal applicability (cf. [GLS99,Gr03]), although Bodlaender's algorithm [Bod96] for producing a tree-decomposition of width at most k (if one exists) that runs in time $O(2^{35k^3}n)$ is not practical for very large values of k.

However, Bodlaender's algorithm for bounded treewidth is not always necessary. Sometimes, one can polynomial-time either determine the answer, or get a bounded tree or path-width decomposition "for free". A nice example of this is described in [PS03].

3.2 Crown Rules: A Surprising New Kind of Kernelization Method

A surprisingly general new approach to kernelization rules has recently developed. Consider the VERTEX COVER problem, and the reduction rule for this problem for vertices of degree 1. If (G, k) is an instance to the VERTEX COVER problem where G has a vertex v of degree 1 with neighbor $u \neq v$, then we can reduce to the instance (G', k') where $G' = G - u - v$ and $k' = k - 1$. This simple *local* reduction rule admits a global structural generalization.

Definition 6. *A* crown decomposition *of G is a partition of the vertex set of G into three sets, H, C and J such that the following conditions hold:*
(1) C is an independent set,
(2) H is matched into C, and
(3) H is a cutset, i.e., there are no edges between C and J.

If (G, k) is an instance of the VERTEX COVER problem, and G admits a crown decomposition, then we can reduce to (G', k') where $G' = G - H - C$ and $k' = k - |H|$, as a generalization of the degree 1 reduction rule. Presently, it is an open problem whether determining if a graph has a crown decomposition is in P, or is NP-complete. However, if we are given: (1) a graph $G = (V, E)$ without isolated vertices, and (2) a vertex cover V' for G that satifies the inequality $2|V'| < |V|$ (in other words, the vertex cover is less than half the size of G), then we can compute a crown decomposition in polynomial time as follows. Let $V'' = V - V'$, and compute a maximum matching between V' and V''. For $x \in V'$, let $m(x) \in V''$ denote the vertex of V'' (if any) to which x is matched. For $U \subseteq V'$, define $m(U) = \{v \in V'' : \exists x \in U \text{ with } m(x) = v\}$. For $W \subseteq V''$ let $N(W) = \{x \in V' : \exists y \in W, xy \in E\}$. Let $A = V'' - m(V')$. The size condition on te vertex cover V' insures that A is nonempty. Note that since V' is a vertex cover, V'' is an independent set in G.

If we now take $C = m(N(A)) \cup A$, $H = N(A)$, and $J = V - C - H$, then it is easy to verify (using the fact that m is a *maximum* matching) that we have a nontrivial crown decomposition of G — and thus any instance (G, k) of the VERTEX COVER problem can be reduced by our *crown reduction rule*. It seems that many parameterized problems that are "subject to vertex cover structure" have reduction rules that are flavors of this approach. This includes many of the problems mentioned in this section.

Data reduction and kernelization rules are one of the primary outcomes of research on parameterized complexity. After all, whether k is small or not — and no matter if you are going to do simulated annealing eventually — it *always* makes sense to simplify and reduce (pre-process) your input! This humble activity, because reduction rules frequently cascade, can sometimes lead to unexpected success against hard problems on real input distributions. A fine example of this has been described by Weihe [Wei98].

4 Major Challenges

These seem to be some of the main challenges that lie ahead in this area of research.

Challenge 1: Explore the parameterized complexity of the fundamental problems of computer science subdiscipline X. Possible application areas abound across computer science and many are still completely unexplored.

Challenge 2: Clarify the complexity of PTAS's. This is a big project, with many dozens of problems with PTAS's unresolved. Only a handful of $W[1]$-hardness results are so far known [CT97,CFJR01].

Challenge 3: Show that $M[1] = W[1]$. This is arguably the most important unsolved problem in the foundations of parameterized complexity. Showing this equality would join the parameterized and classical worlds at the root in a very elegant way. Intuitively, this challenge involves reducing CLIQUE (where the parameter k is the size of the "small" clique) to MINI-CLIQUE, where the graph

has been miniaturized, but the clique we are looking for may be "as big" as the graph. The coding that is required seems reminiscent of the holographic issues that arise in the proof of the PCP Theorem.

Challenge 4: Use the unusual but supple combinatorics of $M[1]$-hardness to expand the realm of parameterized intractability results. It may be that $M[1]$ gives us the means to show that GRAPH ISOMORPHISM for the parameter maximum degree is unlikely to be in FPT (and, in particular, therefore unlikely to be in P). Will it turn out that $M[1]$-hardness is *generally* easier (or more powerful) to work with than $W[1]$-hardness?

Challenge 5: Investigate the known FPT algorithms according to the optimality program of Cai and Juedes. For example, Bodlaender's FPT algorithm for TREEWIDTH has an exponential function of the form $f(k) = 2^{O(k^3)}$. We should therefore seek either a lower bound result that says this cannot be improved to $f(k) = 2^{o(k^3)}$ unless $FPT = M[1]$, or an improved FPT algorithm accompanied by a matching lower bound.

Challenge 6: Explore whether a lower bound program concerning P-time kernelization can be developed along similar lines into a routine methodology.

Challenge 7: Understand the broader opportunities for *crown rules* and other global structure problem reduction rules of this type.

Challenge 8: Explore the parameterized complexity of local search. For a concrete example of this, the k-change neighborhood of a tour for the TRAVELING SALESMAN PROBLEM can be explored for an improved solution in time $O(n^{2k+1})$ by brute force. Is this problem is FPT, or is it hard for $W[1]$. A moment's reflection shows that we can ask similar questions for almost every local search problem. Nothing much is currently known about this universally relevant parameterization of a stalwart method of practical algorithmics.

Challenge 9: Explore the possibility of cryptosystems whose security rests on the unlikeliness that some problem is in FPT. The cryptosystem would be parameterized, and for every fixed parameter value, crackable in polynomial time. Such possibilities have attracted very little investigation in the reigning paradigm of one-dimensional complexity.

Challenge 10: Develop the structure theory required to settle some of the major unresolved concrete complexity problems. If Robertson and Seymour had set out to prove that the GRAPH MINOR CONTAINMENT problem (determining whether H is a minor of G, parameterized by H) is FPT, they would *still* have had to develop much of the deep and beautiful structure theory of minors. One wonders whether some of the notable open problems of parameterized complexity require similar structure theory to be resolved. A notable example is the problem of determining whether it is possible to delete k vertices from a directed graph so that the result is acyclic. (An FPT algorithm for this would have important applications in computational biology [Hal03].)

References

[ACGKMP99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela and M. Protasi. *Complexity and Approximation.* Springer-Verlag, 1999.

[AFN02] J. Alber, M. Fellows and R. Niedermeier. "Efficient Data Reduction for Dominating Set: A Linear Problem Kernel for the Planar Case." *Proceedings of Scandinavian Workshop on Algorithms and Theory* (SWAT 2002), Springer-Verlag, *Lecture Notes in Computer Science* 2368 (2002), 150–159.

[AFLR03] F. Abu-Khzam, M. Fellows, M. Langston and F. Rosamond. "Bounded Vertex Cover Structure and the Design of FPT Algorithms." Manuscript, 2003.

[ALSS03] F. Abu-Khzam, M. A. Langston, P. Shanbhag and C. T. Symons. "High Performance Tools for Fixed-Parameter Tractable Implementations." WADS'03 Workshop Presentation, 2003.

[ALMSS92] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. Prof verification and intractability of optimization problems. *Proceedings of the IEEE Symposium on the Foundations of Computer Science* (1992).

[Ar96] S. Arora. "Polynomial Time Approximation Schemes for Euclidean TSP and Other Geometric Problems." In: *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science* (1996), 2–12.

[Ar97] S. Arora. "Nearly Linear Time Approximation Schemes for Euclidean TSP and Other Geometric Problems." *Proc. 38th Annual IEEE Symposium on the Foundations of Computing* (FOCS'97), IEEE Press (1997), 554-563.

[AYZ95] N. Alon, R. Yuster and U. Zwick. "Color-coding." *Journal of the ACM* 42 (1995), 844–856.

[Baz95] C. Bazgan, "Schémas d'approximation et complexité paramétrée." Rapport de stage de DEA d'Informatique à Orsay, 1995.

[BBW03] P.S. Bonsma, T. Brüggemann and G.J. Woeginger. A faster FPT algorithm for finding spanning trees with many leaves. Manuscript, 2003.

[Bod93] H. L. Bodlaender. "On Linear Time Minor Tests and Depth-First Search." *Journal of Algorithms* 14 (1993), 1–23.

[Bod96] H. L. Bodlaender. "A Linear Time Algorithm for Finding Tree-Decompositions of Small Treewidth." *SIAM Journal on Computing* 25 (1996), 1305–1317.

[CCDF97] L. Cai, J. Chen, R. Downey and M. Fellows. "On the Parameterized Complexity of Short Computation and Factorization." *Archive for Mathematical Logic* 36 (1997), 321–337.

[CFJR01] Liming Cai, M. Fellows, D. Juedes and F. Rosamond, "Efficient Polynomial-Time Approximation Schemes for Problems on Planar Structures: Upper and Lower Bounds." Manuscript, 2001.

[CJ01] L. Cai and D. Juedes. "On the Existence of Subexponential Parameterized Algorithms," manuscript, 2001. Revised version of the paper, "Subexponential Parameterized Algorithms Collapse the W-Hierarchy," in: *Proceedings 28th ICALP*, Springer-Verlag LNCS 2076 (2001), 273–284. (The conference version contains some major flaws.)

[CK00] C. Chekuri and S. Khanna. "A PTAS for the Multiple Knapsack Problem." *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms* (SODA 2000), pp. 213-222.

[CKJ01] J. Chen, I. A. Kanj and W. Jia. "Vertex Cover: Further Observations and Further Improvements." *Journal of Algorithms* 41 (2001), 280–301.

[CM99] J. Chen and A. Miranda. "A Polynomial-Time Approximation Scheme for General Multiprocessor Scheduling." *Proc. ACM Symposium on Theory of Computing* (STOC '99), ACM Press (1999), 418–427.

[CT97] M. Cesati and L. Trevisan. "On the Efficiency of Polynomial Time Approximation Schemes." *Information Processing Letters* 64 (1997), 165–171.

[DEFPR03] R. Downey, V. Estivill-Castro, M. Fellows, E. Prieto-Rodriguez and F. Rosamond. "Cutting Up is Hard to Do: the Parameterized Complexity of k-Cut and Related Problems." *Electronic Notes in Theoretical Computer Science* 78 (2003), 205–218.

[DF95a] R. G. Downey and M. R. Fellows. "Parameterized Computational Feasibility." In: *P. Clote and J. Remmel (eds.), Feasible Mathematics II.* Birkhauser, Boston (1995), 219–244.

[DF95b] R. G. Downey and M. R. Fellows. "Fixed Parameter Tractability and Completeness II: Completeness for $W[1]$." *Theoretical Computer Science* A 141 (1995), 109–131.

[DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer-Verlag, 1999.

[DFPR03] R. Downey, M. Fellows, E. Prieto-Rodriguez and F. Rosamond. "Fixed-parameter Tractability and Completeness V: Parametric Miniatures." Manuscript, 2003.

[DFS99] R. Downey, M. Fellows and U. Stege. "Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability." In: *Contemporary Trends in Discrete Mathematics*, (R. Graham, J. Kratochvíl, J. Nestríl and F. Roberts, eds.), Proceedings of the DIMACS-DIMATIA Workshop, Prague, 1997, *AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 49 (1999), 49–99.

[Dow03] R. Downey. "Parameterized Complexity for the Skeptic." *Proceedings of the IEEE Computational Complexity Conference* (CCC'03), IEEE Press (2003), to appear.

[DRST02] F. Dehne, A. Rau-Chaplin, U. Stege and P. Taillon. "Solving Large FPT Problems on Coarse Grained Parallel Machines." Manuscript, 2002.

[EJ91] E.A. Emerson and C.S. Jutla. "Tree Automata, μ-Calculus and Determinacy." *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS'91)*, IEEE Press (1991), 368–377.

[EJS01] T. Erlebach, K. Jansen and E. Seidel. "Polynomial Time Approximation Schemes for Geometric Graphs." *Proc. ACM Symposium on Discrete Algorithms (SODA'01)*, ACM Press (2001), 671–679.

[FGN03] M. Fellows, J. Gramm and R. Niedermeier. "On the Parameterized Intractability of Motif Search Problems." Manuscript, 2003.

[FL92] M.R. Fellows and M.A. Langston. "On Well-Partial-Order Theory and its Applications to Combinatorial Problems of VLSI Design." *SIAM Journal on Discrete Mathematics* 5, 117–126.

[FMRS00] M. Fellows, C. McCartin, F. Rosamond and U. Stege. "Coordinatized Kernels and Catalytic Reductions: An Improved FPT Algorithm for Max Leaf Spanning Tree and Other Problems." *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'2000)*, Springer, Lecture Notes in Computer Science 1974 (2000), 240-251.

[GH88] O. Goldschmidt and D.S. Hochbaum. "A Polynomial Algorithm for the k-Cut Problem." *Proc. 29th Annual Symp. on the Foundations of Computer Science (FOCS)* (1988), 444-451.

[GH94] O. Goldschmidt and D.S. Hochbaum. "A Polynomial Algorithm for the k-Cut Problem for Fixed k." *Mathematics of Operations Research* 19 (1994), 24–37.

[GLS99] G. Gottlob, N. Leone and F. Scarcello. "Hypertree Decompositions and Tratable Queries." In *Proceedings of the 18th ACM Symposium on Database Systems* (1999), 21–32.

[Gr03] M. Grohe. "The Complexity of Homomorphism and Constraint Satisfaction Problems Seen from the Other Side." Manuscript, 2003.

[GSS01] G. Gottlob, F. Scarcello and M. Sideri. "Fixed Parameter Complexity in AI and Nonmonotonic Reasoning." To appear in *The Artificial Intelligence Journal*. Conference version in: *Proc. of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning* (LPNMR'99), vol. 1730 of *Lecture Notes in Artificial Intelligence* (1999), 1–18.

[Hal03] M. Hallett. Presentation at the Annual Conference of the New Zealand Mathematics Research Institute, New Plymouth, January 2003.

[KM96] S. Khanna and R. Motwani. "Towards a Syntactic Characterization of PTAS." In: *Proc. STOC 1996*, ACM Press (1996), 329–337.

[KR00] S. Khot and V. Raman. "Parameterized Complexity of Finding Subgraphs with Hereditary properties." *Proceedings of the Sixth Annual International Computing and Combinatorics Conference (COCOON 2000)* July 2000, Sydney, Australia, Lecture Notes in Computer Science, Springer-Verlag 1858 (2000), 137-147.

[KS96] D.R. Karger and C. Stein. "A New Approach to the Minimum Cut Problem." *Journal of the ACM* 43 (1996), 601–640.

[Len83] H. W. Lenstra. "Integer Programming with a Fixed Number of Variables." *Mathematics of Operations Research* 8 (1983), 538–548.

[Luks82] E. Luks. "Isomorphism of Graphs of Bounded Valence Can Be Tested in Polynomial Time." *Journal of Computing and Systems Sciences 25* (1982), 42–65.

[MR99] M. Mahajan and V. Raman. "Parameterizing Above Guaranteed Values: MaxSat and MaxCut." *J. Algorithms* 31 (1999), 335-354.

[Nie02] R. Niedermeier. "Invitation to Fixed-Parameter Algorithms." Habilitationschrift, University of Tübingen, 2002.

[NR99] R. Niedermeier and P. Rossmanith. "Upper Bounds for Vertex Cover Further Improved." *Proceedings 16th STACS*, Springer-Verlag LNCS 1563 (1999), 561–570.

[PS03] E. Prieto and C. Sloper. "Either/Or: Using Vertex Cover Structure in Designing FPT Algorithms — the Case of k-Internal Spanning Tree." In this volume.

[RS95] N. Robertson and P. D. Seymour. "Graph Minors XIII. The Disjoint Paths Problem." *Journal of Combinatorial Theory, Series B* 63 (1995), 65–110.

[ST98] R. Shamir and D. Tzur. "The Maximum Subforest Problem: Approximation and Exact Algorithms." *Proc. ACM Symposium on Discrete Algorithms* (SODA'98), ACM Press (1998), 394–399.

[Wei98] K. Weihe, "Covering Trains by Stations, or the Power of Data Reduction," *Proc. ALEX'98* (1998), 1–8.

Author Index

Aichholzer, Oswin 12, 377
Albers, Susanne 162
Amir, Amihood 340, 353
Anand, R. Sai 186
Andersson, Mattias 412
Atallah, Mikhail 231
Aurenhammer, Franz 12

Barequet, Gill 281
Berg, Mark de 462
Brass, P. 243
Brassard, Gilles 1
Bremner, David 451
Broadbent, Anne 1
Büttner, Markus 162
Butman, Ayelet 353

Cenek, E. 243

Demaine, Erik 451
Dey, Tamal K. 25
Duncan, Christian A. 219, 243

Efrat, A. 243
Elmasry, Amr 103
Eppstein, David 307
Erickson, Jeff 451
Erlebach, Thomas 186
Erten, C. 243
Esfahbod, Behdad 59
Evans, Patricia A. 47

Fekete, Sándor P. 150
Fellows, Michael R. 505
Franceschini, Gianni 114
Frederiksen, Jens S. 174
Frikken, Keith 231

Gąsieniec, Leszek 329
Ghodsi, Mohammad 59
Giesen, Joachim 25
Goodrich, Michael T. 281
Goswami, Samrat 25
Grossi, Roberto 114
Gudmundsson, Joachim 412

Harvey, Nicholas J.A. 294
Haverkort, Herman J. 462
He, Xin 493

Iacono, John 451
Ismailescu, D. 243

JaJa, Joseph 91

Kapoor, Sanjiv 365
Karpinski, Marek 401
Kobourov, S.G. 243
Kothari, Anshul 67
Krasser, Hannes 12

Ladner, Richard E. 294
Landau, Gad M. 340
Langerman, Stefan 451
Larsen, Kim S. 174
Levcopoulos, Christos 412
Lewenstein, Moshe 340, 353
Li, Xiang-Yang 365
Lingas, Andrzej 329
Lister, Raymond 319
Lovász, László 294
Lubiw, A. 243

Măndoiu, Ion I. 401
Matichin, Rachel 271
Meijer, Henk 150
Mitchell, J.S.B. 243
Moret, Bernard M.E. 37
Morin, Pat 451
Müller-Hannemann, Matthias 207

Olshevsky, Alexander 401

Papadopoulou, Evanthia 439
Peleg, David 271
Peyer, Sven 207
Phan, Vinhthuy 424
Porat, Ely 353
Prieto, Elena 474

Raman, Venkatesh 484
Riley, Chris 281
Rote, Günter 377

Saurabh, Saket 484
Sengupta, Sudipta 79
Sharifi, Ali 59
Sharir, Micha 127
Shi, Qingmin 91
Skiena, Steven 424
Sloper, Christian 474
Smith, Andrew D. 47
Smorodinsky, Shakhar 127
Sokol, Dina 340
Solomon, Andrew 319
Speckmann, Bettina 377
Spielman, Daniel A. 256
Streinu, Ileana 377
Sumazin, Pavel 424
Suri, Subhash 67
Sutcliffe, Paul 319

Tamir, Tami 294
Tang, Jijun 37
Tapp, Alain 1
Teng, Shang-Hua 256
Tóth, Csaba D. 389
Toussaint, Godfried 451

Wagner, Dorothea 198

Zelikovsky, Alexander 401
Zhang, Huaming 493
Zhou, Yunhong 67
Ziegler, Martin 140