Lecture Notes in Computer Science 5431

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Sandip Das Ryuhei Uehara (Eds.)

WALCOM: Algorithms and Computation

Third International Workshop, WALCOM 2009
Kolkata, India, February 18-20, 2009
Proceedings

Springer

Volume Editors

Sandip Das
Indian Statistical Institute
Kolkata, 700 108, India
E-mail: sandipdas@isical.ac.in

Ryuhei Uehara
Japan Advanced Institute
of Science and Technology
Ishikawa 923-1292, Japan
E-mail: uehara@jaist.ac.jp

Library of Congress Control Number: 2009920096

CR Subject Classification (1998): F.2, C.2, G.2.2, I.3.5, G.1.6, J.3

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-642-00201-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-00201-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12615353 06/3180 5 4 3 2 1 0

Preface

Welcome to the third annual Workshop on Algorithms and Computation (WALCOM 2009). The workshop provided a forum for researchers working in algorithms and theory of computation from all over the world.

This volume contains the papers presented at WALCOM 2009 held during February 18-20, 2009 at the Indian Statistical Institute, Kolkata, India. The scientific program of WALCOM 2009 included 30 contributed papers selected through a very high quality refereeing process from 102 submissions with authors from 30 countries. In addition, there were four invited talks delivered by Otfried Cheong of KAIST, Korea, Janos Pach of Courant Institute, NY, USA, Sandeep Sen of Indian Institute of Technology, New Delhi, India and Chee Yap of Courant Institute, NY, USA, who are all eminent and well-known researchers.

As editors of these proceedings, we would like to thank all the authors who showed interest in WALCOM 2009. The reputation of a conference is enhanced by its Program Committee and the invited talks. We were able to get highly respected researchers to serve on our Program Committee. We are very much indebted to all members of the Program Committee who did excellent work in helping us to finalize the technical program. We also thank all external referees without whose help it would not have been possible to evaluate so many contributions in so little time. We thank the invited speakers for presenting their talks on current research areas of theoretical computer science.

Our sincerest thanks are due to Bhargab B. Bhattacharya for arranging a relevant pre-conference national workshop on Nano-Science and Bio-chips. It provided an excellent overview to the participants about the research on this emerging technology. We are immensely grateful to Sankar K. Pal, Director, Indian Statistical Institute, for his support in co-sponsoring the workshop and for providing infrastructural help.

We gratefully acknowledge the Organizing Committee led by Subhas C. Nandy for their excellent services that made the workshop a grand success. We thank the Steering Committee members for their continuous encouragement. We also thank Md. Saidur Rahman for valuable suggestions and support.

We would like to thank Springer for publishing these proceedings in their prestigious *Lecture Notes in Computer Science* series, which has in no small way contributed to elevating the status of the conference in academic circles.

We would like to thank our sponsors for their support. Finally, we would also like to thank the EasyChair system for providing a flexible, user-friendly conference management system.

February 2008

Sandip Das
Ryuhei Uehara

WALCOM Organization

Steering Committee

Kyung-Yong Chwa	KAIST, Korea
Costas S. Iliopoulos	KCL, UK
M. Kaykobad	BUET, Bangladesh
Petra Mutzel	University of Dortmund, Germany
Takao Nishizeki	Tohoku University, Japan
C. Pandu Rangan	IIT Madras, India

Organizing Committee

Arijit Bishnu	Indian Statistical Institute, India
Gautam Das	TorAnumana Pvt. Ltd., India
Sandip Das	Indian Statistical Institute, India
Partha P. Goswami	Kalyani University, India
Arindam Karmakar	Indian Statistical Institute, India
Priya Ranjan Sinha Mahapatra	Kalyani University, India
Krishnendu Mukhopadhyaya	Indian Statistical Institute, India
Subhas C. Nandy (Chair)	Indian Statistical Institute, India

Program Committee

Tetsuo Asano	JAIST, Japan
Binay Bhattacharya	Simon Fraser University, Canada
Bhargab Bikram Bhattacharya	Indian Statistical Institute, India
Arijit Bishnu	Indian Statistical Institute, India
Frédéric Chazal	INRIA Saclay - Île-de-France, France
Sandip Das (Co-chair)	Indian Statistical Institute, India
Tamal Dey	Ohio State University, USA
Adrian Dumitrescu	University of Wisconsin-Milwaukee, USA
Rudolf Fleischer	Fudan University, Shanghai, China
Subir Ghosh	TIFR, India
Ferran Hurtado	Universitat Politécnica de Catalunya, Spain
Giuseppe F. Italiano	Università degli Studi di Roma, Italy
Klara Kedem	Ben-Gurion University, Isreal
Rolf Klein	University of Bonn, Germany
Marc van Kreveld	Utrecht University, The Netherlands
Anil Maheshwari	Carleton University, Canada
Stefan Näher	Universität Trier, Germany

Shin-Ichi Nakano	Gunma University, Japan
János Pach	Courant Institute, NY, USA
Sudebkumar Prasant Pal	IIT Kharagpur, India
Jaikumar Radhakrishnan	TIFR, India
Md. Saidur Rahman	BUET, Bangladesh
Kavitha Telikepalli	IISc, Bangalore, India
Takeshi Tokuyama	Tohoku University, Japan
Ryuhei Uehara (Co-chair)	JAIST, Japan

External Reviewers

Mridul Aanjaneya
Muhammad Abdullah Adnan
Md. Jawaherul Alam
Luca Castelli Aleardi
Sang Won Bae
Rana Barua
Florian Berger
Partha Bhowmick
Gruia Calinescu
Ovidiu Daescu
Gautam K. Das
Amit Deshpande
Craig Dillabaugh
Chinmoy Dutta
Mohammad Farshi
Clara I. Grima
Niloy Ganguly
Arijit Ghosh
Alex Gilbers
Parikshit Gopalan
Partha P. Goswami
Martin Grohe
Arobinda Gupta
Takashi Horiyama
Toru Hasunuma
Herman Haverkort
Meng He
Debasish Jana
Minghui Jiang
Md. Rezaul Karim
Arindam Karmakar
Md. Abul Kashem
Matthew Katz
Masashi Kiyomi
Rajeev Kumar
Elmar Langetepe
Thijs van Lankveld
Giuseppe Liotta
Priya Ranjan Sinha Mahapatra
Henk Meijer
Pabitra Mitra
Takaaki Mizuki
Pat Morin
Krishnendu Mukhopadhyaya
M Mvpandurangarao
Hiroshi Nagamochi
Subhas Nandy
N. S. Narayanaswamy
Frank Nielsen
Hirotaka Ono
Yoshio Okamoto
Ekow Otoo
Md. Mostofa Ali Patwary
Rainer Penninger
Imran Pirwani
Sasanka Roy
Krishna S
J. Antoni Sellarès
Michael Segal
Sandeep Sen
Pranab Sen
Takeya Shigezumi
Akiyoshi Shioura
Bhabani P. Sinha
Sivaramakrishnan Sivasubramanian
Michiel Smid
Bettina Speckmann
Srikanth Srinivasan
Susmita Sur-Kolay
Suguru Tamaki

Keisuke Tanaka
Csaba Tóth
Dekel Tsur
Jan Vahrenhold
Kasturi Varadarajan
Isana Veksler-Lublinsky
Alexander Wolff
David R. Wood
Pawel Zylinski

Table of Contents

Invited Talks

Computational Geometry

Graph Algorithms

Complexity

Graph Drawing

Approximation Algorithms

Randomized Algorithms

A Separator Theorem for String Graphs and Its Applications

Jacob Fox[1,⋆] and János Pach[2,⋆⋆]

[1] Department of Mathematics, Princeton University, Princeton, NJ, USA
jacobfox@math.princeton.edu
[2] City College, New York, USA and EPFL, Lausanne, Switzerland
pach@cims.nyu.edu

Abstract. A *string graph* is the intersection graph of a collection of continuous arcs in the plane. It is shown that any string graph with m edges can be separated into two parts of roughly equal size by the removal of $O(m^{3/4}\sqrt{\log m})$ vertices. This result is then used to deduce that every string graph with n vertices and no complete bipartite subgraph $K_{t,t}$ has at most $c_t n$ edges, where c_t is a constant depending only on t. Another application is that, for any $c > 0$, there is an integer $g(c)$ such that every string graph with n vertices and girth at least $g(c)$ has at most $(1 + c)n$ edges.

1 Introduction

A large part of computational geometry deals with representation and manipulation of various geometric objects. Special attention is paid to pairs of objects that are in contact with each other: detecting intersections among line segments, for example, belongs to the oldest and best studied chapter of computational geometry, already addressed in the first monograph devoted to the subject [29]. Yet, even in the special case of segments, little is known about elementary structural properties of the arising intersection patterns. The recognition of such intersection patterns (intersection graphs) is known to be NP-hard [18], [19].

Given a collection $C = \{\gamma_1, \ldots, \gamma_n\}$ of arcwise connected sets in the plane, their *intersection graph* $G = G(C)$ is a graph on the vertex set C, where γ_i and γ_j $(i \neq j)$ are connected by an edge if and only if $\gamma_i \cap \gamma_j \neq \emptyset$. It is easy to show that every such intersection graph can be obtained as an intersection graph of a collection of (simple) continuous curves in the plane. Therefore, the intersection graphs of arcwise connected sets in the plane are often called *string graphs*.

Given a graph $G = (V, E)$ with vertex set V and edge set E, a *weight function* $w : V \to \mathbb{R}_{\geq 0}$ is a nonnegative function on the vertex set such that the sum of

⋆ Research supported by an NSF Graduate Research Fellowship and a Princeton Centennial Fellowship.
⋆⋆ Supported by NSF Grant CCF-05-14079, and by grants from NSA, PSC-CUNY, Hungarian Research Foundation OTKA, and BSF.

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 1–14, 2009.

the weights is at most 1. The *weight of a subset* $S \subseteq V$, denoted by $w(S)$, is defined as $\sum_{v \in S} w(v)$.

A *separator* in a graph $G = (V, E)$ with respect to a weight function w is a subset $S \subseteq V$ for which there is a partition $V = S \cup V_1 \cup V_2$ such that $w(V_1), w(V_2) \leq 2/3$ and there is no edge between V_1 and V_2. If the weight function is not specified, it is assumed that $w(v) = \frac{1}{|V|}$ for every vertex $v \in V$.

The Lipton-Tarjan separator theorem [22] states that for every planar graph G with n vertices and for every weight function w for G, there is a separator of size $O(n^{1/2})$. This has been generalized in various directions: to graphs embedded in a surface of bounded genus [14], graphs with a forbidden minor [1], intersection graphs of balls in $\mathbb{R}^d$ [24], intersection graphs of Jordan regions [9], and intersection graphs of convex sets in the plane [9]. Our main result is a separator theorem for string graphs.

Theorem 1. *For every string graph G with m edges and for every weight function w for G, there is a separator of size $O\left(m^{3/4}\sqrt{\log m}\right)$ with respect to w.*

We do not believe that the bound on the separator size in Theorem 1 is tight. In fact, as in [12], we make the following conjecture.

Conjecture 1. Every string graph with m edges has a separator of size $O(\sqrt{m})$.

This conjecture is known to be true in several special cases: (1) for intersection graphs of convex sets in the plane with bounded clique number [9], (2) for intersection graphs of curves, any pair of which have a bounded number of intersection points [9], and (3) for *outerstring graphs*, that is, intersection graphs of collections C of curves with the property that there is a suitable curve γ such that each member of C has one endpoint on γ, but is otherwise disjoint from it [10].

Separator theorems have many important applications (see, e.g., [21] and [23]). Despite the apparent weakness of the bound in Theorem 1, it is still strong enough to yield some interesting corollaries.

For any graph H, a graph G is called *H-free* if it does not have a (not necessarily induced) subgraph isomorphic to H. Given H and a positive integer n, the *extremal number* $\mathrm{ex}(H, n)$ is defined as the maximum number of edges over all H-free graphs on n vertices. The study of this parameter is a classical area of Turán type *extremal graph theory*; see [2]. The problem of investigating the same maximum restricted to intersection graphs of arcwise connected sets, convex bodies, segments, etc., was initiated in [27]. For partial results in this direction, see [27], [30], [9].

In the present paper, we use Theorem 1 to prove that for any bipartite graph H, there is a constant c_H such that every H-free intersection graph of n arcwise connected sets in the plane has at most $c_H n$ edges. Clearly, it is sufficient to prove this statement for *balanced complete* bipartite graphs $H = K_{t,t}$, as every bipartite graph with t vertices is a subgraph of $K_{t,t}$.

Theorem 2. *For any positive integer t, every $K_{t,t}$-free string graph with n vertices has at most $t^{c \log\log t} n$ edges, where c is an absolute constant.*

A graph G is called *d-degenerate* if every subgraph of G has a vertex of degree at most d. Every d-degenerate graph has chromatic number at most $d+1$. Theorem 2 implies that every $K_{t,t}$-free intersection graph of arcwise connected sets in the plane is $2t^{c \log \log t}$-degenerate. Thus, we obtain

Corollary 1. *For any positive integer t, the chromatic number of every $K_{t,t}$-free intersection graph of n arcwise connected sets in $\mathbb{R}^2$ is at most $2t^{c \log \log t} + 1$.*

In [9], it was shown that every $K_{t,t}$-free intersection graph of n curves, no pair of which has more than a fixed constant number of points in common, has at most $c_t n$ edges, where the dependence on t is exponential. In this case, our separator based approach gives a tight bound. In Section 5, we establish

Theorem 3. *Let k and t be positive integers. There exists a constant C_k depending only on k, such that the maximum number of edges of any $K_{t,t}$-free intersection graph of n curves in the plane, no pair of which have more than k points in common, is at most $C_k t n$. Apart from the value of the constant C_k, this bound cannot be improved.*

A collection of curves in the plane is called a collection of *pseudo-segments* if no two of them have more than one point in common. The *girth* of a graph is the length of its shortest cycle. Kostochka and Nešetřil [17] proved that for any $c > 0$, there is a positive integer $g(c)$ such that the intersection graph of any collection of pseudo-segments with girth at least $g(c)$ has at most $(1+c)n$ edges. Using our separator theorem, Theorem 1, this statement can be extended to all string graphs.

Theorem 4. *For any $c > 0$, there is a positive integer $g(c)$ such that every string graph on n vertices with girth at least $g(c)$ has at most $(1 + c)n$ edges.*

In particular, this theorem implies that there exists a positive integer g_0 such that every string graph with girth at least g_0 has chromatic number at most 3. It would be interesting to determine the smallest such integer g_0.

We mention another application of Theorem 1. The *bandwidth* of a graph G with n vertices is the minimum b such that there is a labeling of the vertices of G by $1, \ldots, n$ so that the labels of adjacent vertices differ by at most b. Chung [5] showed that every tree with n vertices and maximum degree Δ has bandwidth at most $O(n/\log_\Delta n)$. Böttcher, Pruessmann, Taraz, and Würfl [3] used the separator theorem for planar graphs to extend this result to show that every planar graph with n vertices and maximum degree Δ has bandwidth at most $O(n/\log_\Delta n)$. Replacing the separator theorem for planar graphs by Theorem 1 in the proof of this result, we obtain the following extension to all string graphs with a forbidden bipartite subgraph.

Corollary 2. *Every $K_{t,t}$-free string graphs with n vertices and maximum degree Δ has bandwidth at most $c_t n/\log_\Delta n$, where c_t only depends on t.*

The proof of Theorem 1 is given in Section 2. In Section 3, in a few lines we deduce from Theorem 1 a qualitative version of Theorem 2, which states that for

any fixed t, every $K_{t,t}$-free string graph with n vertices has $O(n)$ edges. The proof of Theorem 2 is given in Section 4. In Section 5, we establish Theorem 3 and a similar result for intersection graphs of convex sets in the plane. In Section 6, we prove Theorem 4 and two other results that can be obtained similarly. Throughout the paper, for the sake of clarity of the presentation, we systematically omit floor and ceiling signs, whenever they are not crucial.

2 Proof of Theorem 1

The *bisection width* $b_w(G)$ of a graph $G = (V, E)$ with respect to a weight function w is the least integer for which there is a partition $V = V_1 \cup V_2$ such that $w(V_1), w(V_2) \leq 2/3$ and the number of edges between V_1 and V_2 is $b_w(G)$. If w is the "homogeneous" weight function $w(v) = \frac{1}{|V|}$ for all $v \in V$, for simplicity we write $b(G)$ for $b_w(G)$.

A *topological graph* is a graph drawn in the plane with vertices as points and edges as curves connecting these points. The *pair-crossing number* $\mathrm{pcr}(G)$ of a graph G is the smallest number of pairs of edges that intersect in a drawing of G in the plane. For any graph G, let $\mathrm{ssqd}(G) = \sum_{v \in V(G)} (\deg(v))^2$. We will use the following result of Kolman and Matoušek [16].

Theorem 5. *(Kolman and Matoušek [16]) Every graph G on n vertices satisfies*

$$b(G) \leq c \log n \left(\sqrt{pcr(G)} + \sqrt{ssqd(G)} \right),$$

where c is an absolute constant.

By iterating this theorem, we obtain the following result.

Theorem 6. *Let G be a topological graph with n vertices and maximum degree d, and assume that every edge of G intersects at most D other edges. For any weight function w, we have*

$$b_w(G) = O\left(\left(\sqrt{dD} + d \right) \sqrt{n} \log n \right).$$

Proof. The maximum degree is d, so that the number of edges of G is at most $dn/2$. Since each edge of G intersects at most D other edges, the pair-crossing number of G is at most $\frac{dn}{2} \frac{D}{2} = dDn/4$.

Let A_0 denote the vertex set of G. By Theorem 5, there is a partition $A_0 = A_1 \cup B_1$ such that $|A_1|, |B_1| \leq \frac{2}{3}n$, and the number of edges with one vertex in A_1 and the other in B_1 is at most

$$c \log n \left(\sqrt{\mathrm{pcr}(G)} + \sqrt{\mathrm{ssqd}(G)} \right) \leq c \log n \left(\sqrt{dDn/4} + \sqrt{d^2 n} \right).$$

Without loss of generality, we may assume that $w(A_1) \geq w(B_1)$.

After i iterations, we have a vertex subset A_i with at most $\left(\frac{2}{3}\right)^i n$ vertices. By Theorem 5 applied to the subgraph $G[A_i]$ of G induced by A_i, there is a

partition $A_i = A_{i+1} \cup B_{i+1}$ such that $|A_{i+1}|, |B_{i+1}| \leq \frac{2}{3}|A_i| \leq \left(\frac{2}{3}\right)^{i+1} n$, and the number of edges with one vertex in A_{i+1} and the other in B_{i+1} is at most

$$\begin{aligned} & c \log n \left(\sqrt{\mathrm{pcr}(G[A_i])} + \sqrt{\mathrm{ssqd}(G[A_i])} \right) \\ \leq\ & c \log \left(\left(\tfrac{2}{3}\right)^i n \right) \left(\sqrt{dD \left(\tfrac{2}{3}\right)^i n/4} + \sqrt{d^2 \left(\tfrac{2}{3}\right)^i n} \right) \\ \leq\ & \left(\tfrac{2}{3}\right)^{i/2} c(\sqrt{dD} + d)\sqrt{n} \log n. \end{aligned}$$

Without loss of generality, we may assume that $w(A_{i+1}) \geq w(B_{i+1})$.

We stop the iterative process with i_0 if $w(A_{i_0}) \leq \frac{2}{3}$. Since $w(A_{i_0}) + w(B_{i_0}) = w(A_{i_0-1}) > 2/3$, we have $1/3 < w(A_{i_0}) \leq 2/3$. Let $X = A_{i_0}$ and $Y = A_0 \setminus A_{i_0} = B_1 \cup \ldots \cup B_{i_0}$. By construction, the number of edges of G with one vertex in X and the other vertex in Y is less than

$$\sum_{i=0}^{\infty} \left(\frac{2}{3}\right)^{i/2} c \left(\sqrt{dD} + d\right) \sqrt{n} \log n \leq 6c \left(\sqrt{dD} + d\right) \sqrt{n} \log n.$$

Thus, $A_0 = X \cup Y$ is a partition of the vertex set, demonstrating that the bisection width of G with respect to w is $O\left(\left(\sqrt{dD} + d\right) \sqrt{n} \log n\right)$. □

We next prove a separator theorem for string graphs of maximum degree Δ.

Theorem 7. *Let C be a collection of curves in the plane whose intersection graph G has m edges and maximum degree Δ, and let w be a weight function on G. Then G has a separator of size $O\left(\Delta\sqrt{m} \log m\right)$ with respect to w.*

Proof. By slightly perturbing the curves in C, if necessary, we can assume that no *three* curves in C share a point in common. We may also assume without loss of generality that every element of C intersects at least one other element.

For each pair of intersecting curves, pick a point of intersection, and let P be the set of these m points. Define the topological graph T on the vertex set P by connecting a pair of points of P with an edge if and only if they are consecutive points of P along a curve in C. The number of vertices of T is m. Since no *three* curves in C have a point in common, the maximum degree of the vertices of T is at most *four*. Each curve in C gives rise to a path in the topological graph T with at most Δ vertices and at most $\Delta - 1$ edges. Since each curve in C intersects at most Δ other curves, each edge of T crosses at most Δ curves, besides the one it is contained in. Each of these at most Δ curves contains at most $\Delta - 1$ edges of T, therefore, each edge of T intersects altogether at most $\Delta(\Delta - 1) < \Delta^2$ other edges.

For any $\gamma \in C$, let $d(\gamma)$ denote the number of points of P on γ, i.e., the number of curves in C that intersect γ. To each vertex v of T that is the intersection of two elements $\gamma_1, \gamma_2 \in C$, assign the weight

$$w'(v) = \frac{w(\gamma_1)}{d(\gamma_1)} + \frac{w(\gamma_2)}{d(\gamma_2)}.$$

Notice that $w'(P) = w(C) = 1$.

We now apply Theorem 6 to the topological graph T and to the weight function w'. Recall that T has m vertices, maximum degree at most *four*, and every edge intersects at most Δ^2 other edges. So there is a partition $P = P_1 \cup P_2$ with $w'(P_1), w'(P_2) \leq 2/3$ and the number of edges with one vertex in P_1 and the other in P_2 is

$$O\left((\Delta^2 m)^{1/2} \log m\right) = O\left(\Delta m^{1/2} \log m\right).$$

Let C_0 consist of those curves in C that contain an edge of the topological graph T with one vertex in P_1 and the other in P_2. There are $O\left(\Delta m^{1/2} \log m\right)$ such edges, therefore we have $|C_0| = O\left(\Delta m^{1/2} \log m\right)$.

For $i \in \{1, 2\}$, let C_i consist of those curves of C all of whose intersection points in P belong to P_i. Note that, by construction, $w(C_i) \leq w'(P_i) \leq 2/3$. We claim that $C = C_0 \cup C_1 \cup C_2$ is a partition of C, and hence C_0 is a separator for G with respect to w.

Each curve in C_0 contains an edge with one endpoint in P_1 and the other in P_2. Thus, C_0 is disjoint from C_1 and from C_2. To show that $C_1 \cap C_2 = \emptyset$, it is enough to notice that any curve γ of C which contains a point in P_1 and one in P_2 must belong to C_0, because it gives rise to a path in T, therefore it must contain an edge from P_1 to P_2. Thus, $C = C_0 \cup C_1 \cup C_2$ is a partition of C, and C_0 is a separator with respect to w, of the desired size. □

Proof of Theorem 1. Let $\Delta = m^{1/4}/\sqrt{\log m}$. Let C be a collection of curves in the plane whose intersection graph is the string graph G. Let C' denote the set of all curves in C, the degree of which in G is at least Δ. We have $|C'| \leq 2m/\Delta = 2m^{3/4}\sqrt{\log m}$. In the subgraph G' of G induced by the remaining vertices, the maximum degree is at most Δ. Applying Theorem 7 to this graph and to the weight function w restricted to $C \setminus C'$, we conclude that there is a separator C'' for G' of size $O(\Delta\sqrt{m}\log m)$. Hence, $C' \cup C''$ is a separator for G of size $O(m^{3/4}\sqrt{\log m})$, completing the proof. □

3 *H*-Free String Graphs Have Linearly Many Edges

In this section, we show how Theorem 2 can be deduced in a few lines from our separator theorem, Theorem 1, if we pay no attention to the dependence of the constant coefficient of n on t.

A weaker version of Theorem 2, established in [27], states that every $K_{t,t}$-free string graph on n vertices has at most $n \log^{c_t} n$ edges. Combining this theorem with Theorem 1, we obtain the following corollary.

Corollary 3. *For every $K_{t,t}$-free string graph G on n vertices and for every weight function w for G, there is a separator of size $n^{3/4} \log^{c_t} n$ with respect to w, where c_t is a constant depending only on t.*

A family of graphs is *hereditary* if it is closed under taking induced subgraphs. The following lemma of Lipton, Rose, and Tarjan [21] shows that if all members

of a hereditary family of graphs have small separators, then the number of edges of these graphs is at most linear in the number of vertices. Another proof with a slightly better bound can be found in [9].

Lemma 1. *(Lipton, Rose, Tarjan [21]) Let $\epsilon > 0$, and let F be a hereditary family of graphs such that every member of F with n vertices has a separator of size $O(n/(\log n)^{1+\epsilon})$. Then every graph in F on n vertices has at most $c_F n$ edges, where c_F is a suitable constant.*

Clearly, the family of $K_{t,t}$-free string graphs is hereditary. Therefore, Corollary 3 combined with Lemma 1 immediately implies that every $K_{t,t}$-free string graph on n vertices has at most $c_t n$ edges, where c_t only depends on t. □

4 Proof of Theorem 2

The aim of this section is to prove Theorem 2 in its full generality.

The first ingredient of the proof of Theorem 2 is a weaker upper bound on the number of edges of a $K_{t,t}$-free string graph on n vertices. Pach and Sharir proved that every $K_{t,t}$-free string graph on n vertices has at most $n \log^{c_t} n$ edges. Their proof gives that we may take $c_t = 2^{ct}$, for some absolute constant c. We first show how to modify their proof technique, in combination with other extremal results on string graphs, to show that the result also holds with $c_t = c \log t$.

Lemma 2. *Every string graph G with n vertices and at least $n \log^{c_1 \log t} n$ edges has $K_{t,t}$ as a subgraph.*

To prove this lemma, we need the following two auxiliary results.

Lemma 3. *There is an absolute constant c_2 such that every topological graph with n vertices and at least $n(\log n)^{c_2 \log s}$ edges has $s \geq 3$ pairwise crossing edges with distinct vertices.*

Lemma 4. *([11]) Every string graph with n vertices and ϵn^2 edges contains $K_{t,t}$ as a subgraph with $t = \epsilon^{c_3} n / \log n$ for some absolute constant c_3.*

The last statement was established in [11], while the previous one can be obtained by strengthening the argument in [10], where the same result was proved, except that the s pairwise crossing edges were allowed to share endpoints.

It follows from Lemma 4 that n vertex string graphs with positive constant edge density must contain balanced complete bipartite graphs of size $\Omega(n/\log n)$.

A *Jordan region* is a closed region of the plane, bounded by a simple closed Jordan curve. In other words, a Jordan region is homeomorphic to the closed unit disk.

Proof of Lemma 2. Let s be the smallest positive integer such that

$$(1/16)^{c_3}(2s)/\log 2s \geq t,$$

where c_3 is the absolute constant from Lemma 4. Let $c_t = c_2 \log s = O(\log t)$, where c_2 is the absolute constant from Lemma 3.

Suppose G is a string graph with n vertices and at least $n(\log n)^{c_t}$ edges. Let $\mathcal{C} = \{C_1, \ldots, C_n\}$ be a collection of n Jordan regions whose intersection graph is the string graph G and which have the property that any two intersecting Jordan regions in $\mathcal{C}$ intersect in their interiors (it is easy to see, by slightly fattening compact connected sets, that every string graph is the intersection graph of such a collection of Jordan regions). Fix distinct points p_i in the interior of C_i for $i = 1, \ldots, n$. For each intersecting pair $C_i, C_j \in \mathcal{C}$ with $i < j$, let p_{ij} be a point in $C_i \cup C_j$ such that all the points in $\{p_1, \ldots, p_n\} \cup \{p_{ij} : C_i \cup C_j \neq \emptyset\}$ are distinct, and let γ_{ij} be a simple (nonintersecting) curve contained in $C_i \cup C_j$ such that

1. γ_{ij} has endpoints p_i and p_j;
2. γ_{ij} is disjoint from all other p_ℓ;
3. γ_{ij} can be split into two subcurves γ_{ij}^0 and γ_{ij}^1 such that γ_{ij}^0 is contained in C_i and has endpoints p_i and p_{ij} and γ_{ij}^1 is contained in C_j and has endpoints p_{ij} and p_j.

The points $\{p_1, \ldots, p_n\}$ form the vertex set and curves $\{\gamma_{ij} : C_i \cap C_j \neq \emptyset\}$ form the edge set of a topological graph T with n vertices and at least $n(\log n)^{c_t}$ edges. Since $c_t = c_2 \log s$, by Lemma 3, there are at least s pairwise intersecting edges in T with distinct vertices. Each edge consists of two subcurves and these $2s$ subcurves have at least $\binom{s}{2} \geq \frac{1}{16}(2s)^2$ intersecting pairs. By Lemma 4, the intersection graph of these $2s$ subcurves contains $K_{h,h}$ with $h = (1/16)^{c_3}(2s)/\log 2s \geq t$. It follows from the construction that G contains $K_{t,t}$. □

The second ingredient of the proof of Theorem 2 is our separator theorem, Theorem 1, which, together with Lemma 2, implies that every $K_{t,t}$-free string graph with $m < n(\log n)^{c_t}$ edges has a separator of size

$$O(m^{3/4}\sqrt{\log m}) < O(n^{3/4} \log^{3c_t/4+1/2} n).$$

Thus, for $n_0 = 2^{O(c_t \log c_t)} \leq t^{c' \log \log t}$ (where c' is an absolute constant), every $K_{t,t}$-free string graph with $n \geq n_0$ vertices has a separator of size $n^{7/8}$. This fact, together with the following lemma from [9] (which is a more precise version of Lemma 1), immediately implies Theorem 2.

Lemma 5. *([9]) Let $\phi(n)$ be a monotone decreasing nonnegative function defined on the set of positive integers, and let n_0 and C be positive integers such that*

$$\phi(n_0) \leq \frac{1}{12} \quad \textit{and} \quad \prod_{i=0}^{\infty} \left(1 + \phi\left(\lceil (4/3)^i n_0 \rceil\right)\right) \leq C.$$

If F is an $n\phi(n)$-separable hereditary family of graphs, then every graph in F on $n \geq n_0$ vertices has fewer than $\frac{Cn_0}{2} n$ edges.

5 Stronger Versions of Theorem 2

First we establish Theorem 3, that is, we show how to improve considerably Theorem 2 for intersection graphs of collections of curves, in which every pair of curves intersect in at most a fixed constant number k of points.

In [9], we proved that the intersection graph of a collection of curves with x crossings has a separator of size $O(\sqrt{x})$. If each pair in a collection of curves intersect in at most k points, then the number m of edges of the intersection graph is at least x/k, and we obtain a separator of size $O(\sqrt{km})$. In [12], the following result was established, which is an analogue of Lemma 4 for families of curves in which every pair intersect in at most a constant k number of points.

Lemma 6. *Let G be the intersection graph of a collection C of n curves such that each pair of curves in C intersect in at most k points. If G has at least ϵn^2 edges, then it contains a complete bipartite subgraph $K_{t,t}$ with $t \geq c_k \epsilon^c n$, where c is an absolute constant and c_k is a constant that only depends on k.*

We now have the necessary tools to prove Theorem 3.

Proof of Theorem 3. Suppose that G has ϵn^2 edges. By Lemma 6, we have $t \geq c_k \epsilon^c n$, that is, $\epsilon \leq \left(c_k^{-1} \frac{t}{n}\right)^{1/c}$. Thus, according to the separator lemma mentioned above, G has a separator of size $O(\sqrt{km}) < O\left(\sqrt{k \left(c_k^{-1} \frac{t}{n}\right)^{1/c} \cdot n^2}\right) < c'_k (t/n)^{c_1} n$, where $c_1 = 1/(2c) > 0$ and c'_k only depends on k.

Letting $\phi(n) = c'_k (t/n)^{c_1}$ and $n_0 = (12c'_k)^{-1/c_1} t$, Lemma 5 implies that G has at most $C_k t n$ edges for some constant C_k only depending on k. □

In [13], it was shown that every intersection graph of n convex sets in the plane with $m = \epsilon n^2$ edges contains a complete bipartite subgraph $K_{t,t}$ with $t = \Omega(\epsilon^2 n)$. From this result, using a separator lemma from [10], which states that every K_t-free intersection graph of convex sets in the plane with m edges has a separator of size $O(\sqrt{tm})$, and Lemma 5, it is easy to deduce the following result.

Theorem 8. *Every $K_{t,t}$-free intersection graph of n convex sets in the plane has $O(t^3 n)$ edges.*

6 Proof of Theorem 4 and Related Results

Theorem 4 is a direct corollary of Theorem 1 and the following lemma, showing that all graphs of large girth, which belong to a hereditary family of graphs with small separators, are sparse.

Lemma 7. *Let $\epsilon > 0$, and let F be a hereditary family of graphs such that every member of F with n vertices has a separator of size $O(n/(\log n)^{1+\epsilon})$. Then for each $c > 0$ there is a positive integer $g = g_F(c)$ such that every graph in F on n vertices and girth at least g has at most $(1 + c)n$ edges.*

The aim of this section is to prove Lemma 7 and to discuss some of its consequences. The similarity between Lemmas 7 and 1 is no coincidence; their proofs are very similar.

Before turning to the proof, we briefly outline its main idea. Consider a hereditary family F of graphs, in which every graph has a small separator. We show that every graph G in F with n vertices has an induced subgraph with at most $\frac{3}{4}n$ vertices, whose average degree is not much smaller than the average degree of G. We repeatedly use this fact until we find an induced subgraph of G with fewer than g vertices, whose average degree is not much smaller than that of G. But if the girth of G is at least g, then this induced subgraph of G with fewer than g vertices is a forest and so has average degree less than 2. If g is chosen sufficiently large, we conclude that G has average degree at most $2 + 2c$ and hence at most $(1 + c)n$ edges.

Now we work out the details of the proof of Lemma 7. Given a nonnegative function f defined on the set of positive integers, we say that a family F of graphs is *f-separable*, if every graph in F with n vertices has a separator of size at most $f(n)$.

Lemma 8. *Let $\phi(n)$ be a monotone decreasing nonnegative function defined on the set of positive integers, and let g be a positive integer and $c > 0$ such that*

$$\phi(g) \le \frac{1}{12} \quad \textit{and} \quad \prod_{i=0}^{\infty} \left(1 + \phi\left(\lceil (4/3)^i g \rceil\right)\right) \le 1 + c.$$

If F is an $n\phi(n)$-separable hereditary family of graphs, then every graph in F with n vertices and girth at least g has fewer than $(1 + c)n$ edges.

Proof. Let $G_0 = (V, E)$ be a member of the family F with n vertices, girth at least g, and average degree d. If $n < g$, then G_0 is a forest and hence has at most $n - 1$ edges. We may assume that $n \ge g$. By definition, there is a partition $V = V_0 \cup V_1 \cup V_2$ with $|V_0| \le n\phi(n)$, $|V_1|, |V_2| \le \frac{2}{3}n$, such that no vertex in V_1 is adjacent to any vertex in V_2.

Let d' and d'' denote the average degree of the vertices in the subgraphs of G_0 induced by $V_0 \cup V_1$ and $V_0 \cup V_2$, respectively. Every edge of G_0 is contained in at least one of these two induced subgraphs. Hence,

$$d'(|V_0| + |V_1|) + d''(|V_0| + |V_2|) \ge 2|E| = d|V|,$$

so that

$$d' \frac{|V_0| + |V_1|}{|V| + |V_0|} + d'' \frac{|V_0| + |V_2|}{|V| + |V_0|} \ge d \frac{|V|}{|V| + |V_0|}.$$

Since $|V| = |V_0| + |V_1| + |V_2|$, we have $\frac{|V_0|+|V_1|}{|V|+|V_0|} + \frac{|V_0|+|V_2|}{|V|+|V_0|} = 1$ and the left hand side of the above inequality is a weighted mean of d' and d''. Consequently, d' or d'' is at least

$$d \frac{|V|}{|V| + |V_0|} \ge d \frac{1}{1 + \phi(n)}.$$

Suppose without loss of generality that d' is at least as large as this number, and let G_1 denote subgraph of G induced by $V_0 \cup V_1$. By assumption, we have that $\phi(n) \leq \frac{1}{12}$ and $|V_0| \leq n\phi(n)$. Therefore, G_1 has $|V_0| + |V_1| \leq \frac{1}{12}n + \frac{2}{3}n = \frac{3}{4}n$ vertices.

Proceeding like this, we find a sequence of induced subgraphs $G_0 \supset G_1 \supset G_2 \supset \ldots$ with the property that, if G_i has $n_i \geq g$ vertices and average degree d_i, then G_{i+1} has at most $\frac{3}{4}n_i$ vertices and average degree at least $\frac{1}{1+\phi(n_i)}d_i$. We stop with G_j if the number of vertices of G_j is less than g.

Since G_j is an induced subgraph of G, it also has girth at least g. The number of vertices of G_j is less than g, so G_j must be a forest and therefore has average degree less than 2. The above argument also shows that the average degree of G_j is at least $\frac{1}{1+c}d$, so $d < 2(1+c)$, and the number of edges of G is $dn/2 < (1+c)n$, completing the proof. □

Taking logarithms and approximating $\ln(1+x)$ by x, we obtain that $\prod_{i=0}^{\infty}(1+\phi(\lceil(4/3)^i\rceil)) \neq \infty$ if and only if $\sum_{i=0}^{\infty}\phi(\lceil(4/3)^i\rceil) \neq \infty$ if and only if $\sum_{i=0}^{\infty}\phi(2^i) \neq \infty$. Therefore, Lemma 8 has the following corollary.

Corollary 4. *Let F be an $n\phi(n)$-separable hereditary family of graphs, where $\phi(n)$ is a monotone decreasing nonnegative function such that $\sum_{i=0}^{\infty}\phi(2^i) \neq \infty$. Then, for each $c > 0$, there is $g_F(c)$ such that every graph in F on n vertices and girth at least g has at most $(1+c)n$ edges.*

Since $\sum_{i=1}^{\infty} 1/i^{1+\epsilon}$ converges for all $\epsilon > 0$, Lemma 7 is an immediate consequence of Corollary 4.

The condition that a connected graph has large girth means that the graph is locally "tree-like." In general, this local condition does not imply that the graph also has some global tree-like properties. For instance, in 1959, Erdős [7] proved that the existence of graphs with arbitrarily large girth and chromatic number. However, according to Lemma 7, if every member of a hereditary family F of graphs has a small separator, then the condition that a connected graph in F has large girth *does* imply that the graph is *globally tree-like*. Indeed, if $c < 1/2$, then every graph in F with girth at least $g_F(c)$ is 2-degenerate and hence has chromatic number at most *three*. Furthermore, any graph G in F with girth at least $\max(g_F(c/2), 2/c)$ can be turned into a forest by the removal of at most $c|G|$ edges.

We end this section by presenting two other corollaries of Lemma 7. The separator theorem for graphs with an excluded minor [1], together with Lemma 7, imply the following.

Corollary 5. *For any $c > 0$ and positive integer t, there exists a positive integer $g(c,t)$ such that every K_t-minor-free graph on n vertices with girth at least $g(c,t)$ has at most $(1+c)n$ edges.*

A well-known result of Thomassen [31] (see also Chapter 8.2 in [6]) states that for any positive integer t, there exists another integer $g(t)$ such that every graph with minimum degree at least *three* and girth at least $g(t)$ contains K_t as a minor. Obviously, Corollary 5 implies Thomassen's result. In fact, it can be shown by a simple argument that the two statements are equivalent. In the special case of planar graphs and, more generally, for graphs with bounded genus, the statement easily follows from Euler's polyhedral formula.

The separator theorem for intersection graphs of balls in $\mathbb{R}^d$ [24] together with Lemma 7 imply

Corollary 6. *For any $c > 0$ and positive integer d, there exists a positive integer $g(c, d)$ such that every intersection graph of balls in $\mathbb{R}^d$ with girth at least $g(c, d)$ has at most $(1 + c)n$ edges.*

7 Concluding Remarks

Theorem 2, with a much worse dependence of the coefficient of n on t, can also be deduced from the following result of Kuhn and Osthus [20]: For any graph H and any positive integer t, there is a constant $c(H, t)$ such that every graph with n vertices and at least $c(H, t)n$ edges, which contains no induced subdivision of H, contains $K_{t,t}$ as a subgraph. Let H_0 be the graph obtained from the complete graph K_5 by replacing each edge by a path of length two. Using the nonplanarity of K_5, it is easy to see that no subdivision of H_0 is a string graph. Since the family of string graphs is closed under taking induced subgraphs, no string graph contains an induced subdivision of H_0. Thus, the result of Kuhn and Osthus implies that any $K_{t,t}$-free string graph on n vertices has at most $c(H_0, t)n$ edges. However, this proof only shows that $c(H_0, t) < 2^{2^{2^{2^{ct \log t}}}}$, for some absolute constant c.

The dependence of the coefficient of n on t in Theorem 2 could be further improved if we could prove Conjecture 1. Indeed, Conjecture 1 combined with Lemmas 4 and 5 would imply the following.

Conjecture 2. Every $K_{t,t}$-free string graph with n vertices has $O((t \log t)n)$ edges.

Conjecture 2, if true, would be tight up to the constant factor. According to a construction in [8] and [28], there are string graphs with n vertices and $(1 - o(1))n^2/2$ edges, in which the size of the largest balanced bipartite subgraph is $O(n/\log n)$.

Another consequence of Conjecture 1 would be that, together with Lemma 4, it would imply that every K_t-free string graph with n vertices has chromatic number at most $(\log n)^{c \log t}$ for some absolute constant c. It is not even known if every triangle-free string graph with n vertices has chromatic number $n^{o(1)}$.

References

1. Alon, N., Seymour, P., Thomas, R.: A separator theorem for nonplanar graphs. J. Amer. Math. Soc. 3, 801–808 (1990)
2. Bollobás, B.: Modern Graph Theory. Springer, Heidelberg (1998)
3. Böttcher, J., Pruessmann, K.P., Taraz, A., Würfl, A.: Bandwidth, treewidth, separators, expansion, and universality, in. In: Proc. Topological. and Geometric Graph Theory (TGGT 2008); Electron. Notes. Discrete Math., vol. 31, pp. 91–96 (2008)
4. Capoyleas, V., Pach, J.: A Turán-type theorem on chords of a convex polygon. J. Combinatorial Theory, Ser. B 56, 9–15 (1992)
5. Chung, F.: Labelings of graphs. In: Selected Topics in Graph Theory, pp. 151–168. Academic Press, San Diego (1988)
6. Diestel, R.: Graph Theory, 2nd edn. Springer, Heidelberg (2000)
7. Erdős, P.: Graph theory and probability. Canad. J. Math. 11, 34–38 (1959)
8. Fox, J.: A bipartite analogue of Dilworth's theorem. Order 23, 197–209 (2006)
9. Fox, J., Pach, J.: Separator theorems and Turán-type results for planar intersection graphs. Advances in Mathematics 219, 1070–1080 (2008)
10. Fox, J., Pach, J.: Coloring K_k-free intersection graphs of geometric objects in the plane. In: Proc. 24th ACM Sympos. on Computational Geometry, pp. 346–354. ACM Press, New York (2008)
11. Fox, J., Pach, J.: String graphs and incomparability graphs (manuscript)
12. Fox, J., Pach, J., Tóth, C.D.: Intersection patterns of curves. J. London Math. Soc. (to appear)
13. Fox, J., Pach, J., Tóth, C.D.: Turán-type results for partial orders and intersection graphs of convex sets. Israel J. Math. (to appear)
14. Gilbert, J.R., Hutchinson, J.P., Tarjan, R.E.: A separator theorem for graphs of bounded genus. J. Algorithms 5, 391–407 (1984)
15. Koebe, P.: Kontaktprobleme der konformen Abbildung. Berichte über die Verhandlungen der Sachsischen Akademie der Wissenschaften, Leipzig. Mathematische-Physische Klasse 88, 141–164 (1936)
16. Kolman, P., Matoušek, J.: Crossing number, pair-crossing number, and expansion. J. Combin. Theory Ser. B 92, 99–113 (2004)
17. Kostochka, A.V., Nešetřil, J.: Coloring relatives of intervals on the plane. I. Chromatic number versus girth. European J. Combin. 19, 103–110 (1998)
18. Kratochvíl, J., Matoušek, J.: NP-hardness results for intersection graphs. Comment. Math. Univ. Carolin. 30, 761–773 (1989)
19. Kratochvíl, J., Matoušek, J.: Intersection graphs of segments. J. Combin. Theory Ser. B 62, 289–315 (1994)
20. Kühn, D., Osthus, D.: Induced subdivisions in $K_{s,s}$-free graphs of large average degree. Combinatorica 24, 287–304 (2004)
21. Lipton, R.J., Rose, D.J., Tarjan, R.E.: Generalized nested dissection. SIAM J. Numer. Anal. 16, 346–358 (1979)
22. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM J. Appl. Math. 36, 177–189 (1979)
23. Lipton, R.J., Tarjan, R.E.: Applications of a planar separator theorem. SIAM J. Comput. 9, 615–627 (1980)
24. Miller, G.L., Teng, S.-H., Thurston, W., Vavasis, S.A.: Separators for sphere-packings and nearest neighbor graphs. J. ACM 44, 1–29 (1997)
25. Pach, J., Agarwal, P.: Combinatorial Geometry. J. Wiley, New York (1995)

26. Pach, J., Pinchasi, R., Sharir, M., Tóth, G.: Topological graphs with no large grids. Graphs and Combinatorics 21, 355–364 (2005)
27. Pach, J., Sharir, M.: On planar intersection graphs with forbidden subgraphs. J. Graph Theory 59, 205-214 (2008)
28. Pach, J., Tóth, G.: Comment on Fox News. Geombinatorics 15, 150–154 (2006)
29. Preparata, F., Shamos, M.: Computational geometry. An introduction. Texts and Monographs in Computer Science. Springer, New York (1985)
30. Radoičić, R., Tóth, G.: The discharging method in combinatorial geometry and its application to Pach–Sharir conjecture on intersection graphs. In: Goodman, J.E., Pach, J., Pollack, J. (eds.) Proceedings of the Joint Summer Research Conference on Discrete and Computational Geometry. Contemporary Mathematics, AMS, vol. 453, pp. 319–342 (2008)
31. Thomassen, C.: Girth in graphs. J. Combin. Theory Ser. B 35, 129–141 (1983)

Foundations of Exact Rounding

Chee K. Yap and Jihun Yu

Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
251 Mercer Street
New York, NY 10012
{yap,jihun}@cs.nyu.edu

Abstract. Exact rounding of numbers and functions is a fundamental computational problem. This paper introduces the mathematical and computational foundations for exact rounding. We show that all the elementary functions in ISO standard (ISO/IEC 10967) for Language Independent Arithmetic can be exactly rounded, in any format, and to any precision. Moreover, a priori complexity bounds can be given for these rounding problems. Our conclusions are derived from results in transcendental number theory.

1 Introduction

Modern scientific and engineering computation is predominantly based on floating point computations. Central to such computations is the now ubiquitous floating-point hardware that performs the four arithmetic operations and square-root to double or quadruple precision, exactly rounded. The IEEE 754 standard [11] specifies the various rounding modes that must be supported: round up or round down, round toward or away from zero, and round to nearest (with suitable tie-breaking rule).

To supplement these built-in hardware operations, many scientific computation also need a mathematical library in software for the elementary operations such as exponentiation, logarithm, trigonometric functions, etc. There is an ISO standard ISO/IEC 10967 for Language Independent Arithmetic (LIA) which is compatible with the IEEE 754 standard [18,3]. The second part of this standard (known as LIA-2) specifies a list of elementary functions that should be implemented in such libraries. A very similar list of functions from Defour et al [3] is reproduced in Table 1.

An excellent general reference for the issues of exact rounding, including various algorithms for elementary functions is Muller [16]. Issues of hardware implementation and fixed precision formats are also addressed in this book. There is also a growing interest in arbitrary precision "big number" packages that need such algorithms. A much larger class of functions than elementary functions is the class of hypergeometric functions. In [5,6], we provide algorithms for computing hypergeometric functions to any desired precision.

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 15–31, 2009.

Logarithms:	$\log(x), \log_2(x), \log_{10}(x), \log_y(x), \log(1+x)$
Exponentials:	$\exp(x), \exp(x)-1, 2^x, 10^x, x^y$
Trigonometric:	$\sin(x), \cos(x), \tan(x), \cot(x), \sec(x), \csc(x)$
Inverse Trigonometric:	$\arcsin(x), \arccos(x), \arctan(x), \arctan(x/y)$, $\operatorname{arccot}(x), \operatorname{arcsec}(x), \operatorname{arccsc}(x)$
Hyperbolic:	$\sinh(x), \cosh(x), \tanh(x), \coth(x), \operatorname{sech}(x), \operatorname{csch}(x)$
Inverse Hyperbolic:	$\operatorname{arcsinh}(x), \operatorname{arccosh}(x), \operatorname{arctanh}(x), \operatorname{arctanh}(x/y)$, $\operatorname{arccoth}(x), \operatorname{arcsech}(x), \operatorname{arccsch}(x)$

Fig. 1. Elementary functions in the LIA-2

The ability to approximate a given function f to any desired precision is a prerequisite for the general problem[1] of exact rounding for the function f. But this ability alone is insufficient for the exact rounding of f. In general, this only be resolved by using nontrivial results from transcendental number theory. For instance, current knowledge does not allow us to exactly round the hypergeometric functions. According to [3], there is no IEEE standard analogous to LIA-2 because the theory of exact rounding for such functions is not understood.

Until the advent of modern computers, various elementary functions are precomputed and stored in tables for hand calculation. The **Table Maker's Problem** amounts to producing exactly rounded function values at a finite set G of inputs, with the function values also exactly rounded to G. We call G a "grid" for rounding. E.g., G can be the set of IEEE 754 double precision numbers in binary format. The associated **Table Maker's Dilemma** arises because it is not apparent that there are terminating algorithms to produce the table entries correctly [10,16,13, p. 166]. Indeed, this dilemma is another form of the Zero Problem [19,23].

Why is exact rounding important? An exactly rounded math library is the foundation for trustworthy numerical computations, for code portability, for use in computed-assisted proofs, etc. See [3,16] for other considerations. Another application arises in Computational Geometry, where the problem of numerical non-robustness is especially acute. A highly successful approach for achieving robust algorithms is **Exact Geometric Computation** (EGC) [23]. To speed up EGC computations, we need floating point filters [2,15]. A basis for such filters is exactly rounded arithmetic at machine-level. In our Core Library (version 2), we introduced transcendental functions in expressions [15]. The corresponding filters need exactly rounded elementary functions in some fixed precision library, such as specified by LIA-2.

Let us briefly review some basic results on exact rounding of elementary functions. In Lefèvre et al [13], the exact rounding problem was solved for elementary functions for the IEEE double precision format in radix 2. Their general approach

[1] The problem is also known as "correct rounding". We prefer the terminology of "exact rounding" as there are situations when we would settle for less stringent notions of exactness, but consider it to be "correct" nevertheless. For instance, [3] describes 3 levels of rounding.

was to search for the worst case input numbers in the IEEE format. By exhaustive search, it is determined that the smallest gap between a function value and a breakpoint is greater than 2^{-119}. (A "break point" is either an possible input number, or the midpoint between consecutive input numbers.) It follows that if we evaluate the elementary functions to 119 bits of accuracy, we can round exactly.

The exhaustive search approach is important for hardware implementation of exactly rounded functions. But it's extendability is fragile: a new search must be mounted for input numbers in other precision, or in other formats. In short, each choice of a rounding grid G needs its own search. For example, Lefèvre et al [14] describe the search for the exponential function on 64-bit IEEE numbers in decimal format. They show that exact rounding can be achieved by approximating the exponential function to 47 digits. With 64-bit inputs, the search space is too large for naive search. Among the sophisticated search methods they employed is one based on lattice reduction. But exhaustive search for 128-bit formats is way beyond current techniques and computing power [16, p. 168].

Another important application for exact rounding is in multiprecision floating point libraries. Unlike hardware implementation applications, we do not need to determine the worst case precision for a fixed grid. We only need an algorithm that is guaranteed to produce the exactly rounded answer for any potential input number. One of the most widely used multiprecision libraries is `gmp` (Gnu Multiprecision library). The bigfloat subsystem in `gmp` does not guarantee exact rounding. The `mpfr` project [8] aims to remedy this shortcoming; but the underlying basis for such algorithms does not seem to have been published. Ziv and others [25,9] have proposed to use arbitrary precision computation to achieve exact rounding. The problem with a naive application of arbitrary precision computation is termination. Only heuristic arguments have been used to argue for a high probability of termination.

The purpose of the present article is to clearly formulate the fundamental principles behind exact rounding. We show that there are terminating algorithms for computing exactly rounded values for the standard elementary functions, in any format and to any desired precision. In this sense, we solve the Table Maker's Dilemma for such functions. Worst case complexity estimates for some of these algorithms are also given. Our conclusions are based on basic results from transcendental number theory (TNT). Note that Muller [16, p. 167] had already observed that Lindemann's result in TNT can be used to justify exact rounding; he also gave quantitative bounds from Baker type theory in TNT. A general reference for TNT is [7].

¶*1. Computational Model.* We consider only $\mathbb{R}$ (the real numbers). Usually, extensions of our results to $\mathbb{C}$ is routine. In computing with reals, we use the computational model of [22,24] in which numerical inputs and outputs for algorithms are members of some set $\mathbb{F} \subseteq \mathbb{R}$ such that $\mathbb{F}$ contains the integers $\mathbb{Z}$, and $\mathbb{F}$ is dense in $\mathbb{R}$ and is closed under the ring operations $(+, -, \times)$ as well as division by 2 (so $x \in \mathbb{F}$ implies $x/2 \in \mathbb{F}$), and allow exact comparisons. We

may call $\mathbb{F}$ the **computational ring** of our model. There are several common choices for this ring. Consider the following tower of rings

$$\mathbb{Z}[1/2] \subseteq \mathbb{Z}[1/10] \subseteq \mathbb{Q} \subseteq \mathbb{A} \cap \mathbb{R}$$

where $\mathbb{Z}[1/2] = \{m2^n : m, n \in \mathbb{Z}\}$ is the set of bigfloats or dyadic numbers, $\mathbb{Z}[1/10] = \{m10^n : m, n \in \mathbb{Z}\}$ is the set of decimal numbers, $\mathbb{Q}$ is the set of rational integers, and $\mathbb{A}$ is the set of algebraic numbers. We can choose $\mathbb{F}$ to be any one of these four rings. For these "standard" choices, our computational model can be implemented by standard Turing machines [24].

Unlike the computational model of computable analysis [12], ours allows exact comparison between members of $\mathbb{F}$; this is crucial for the rounding problem. Not only can we recognize if a given $x \in \mathbb{F}$ is 0, but we also determine if x is in $\mathbb{Z}$ (resp., $\mathbb{Q}$). E.g., this is needed for rounding the function y^x where the case $x \in \mathbb{Q}$ is an exceptional case that is separately treated.

The most important computational ring is $\mathbb{F} = \mathbb{Z}[1/2]$; it is also the minimal computational ring by our axioms. In the following, $\mathbb{F} = \mathbb{Z}[1/2]$ is our default, unless otherwise noted. In order to formulate the rounding problem using $\mathbb{F}$ as the rounding target, we need to introduce a "scale" or grading on $\mathbb{F}$. A tower of proper inclusions $\mathbb{F}_0 \subseteq \mathbb{F}_1 \subseteq \mathbb{F}_2 \subseteq \cdots$ such that $\cup_{n \geq 0} \mathbb{F}_n = \mathbb{F}$ is called a **grading** of $\mathbb{F}$,

For our minimal ring $\mathbb{F} = \mathbb{Z}[1/2]$, we use the grading in which $\mathbb{F}_n = \mathbb{Z}/2^n := \{m2^{-n} : m \in \mathbb{Z}\}$ for each $n \in \mathbb{N}$. In general, for any set $G \subseteq \mathbb{R}$, let[2] $G/N := \{a/N : a \in G\}$ where $N \neq 0$. We can formulate similar grading for other choices of $\mathbb{F}$, with the proviso that $\mathbb{Z}/2^n \subseteq \mathbb{F}_n$. E.g., for $\mathbb{Q}$, we can choose $\mathbb{Q}_n = \{m/b : 1 \leq b \leq n+1, m \in \mathbb{Z}\}$.

Bigfloats are a convenient abstraction of the finite-precision binary formats of the IEEE numbers [11]. In particular, the IEEE numbers in double precision is a finite subset of $\mathbb{F}_{1075}$. The choice $\mathbb{F} = \mathbb{A} \cap \mathbb{R}$ is mostly of theoretical interest. It is conceivable that $\mathbb{F} = \mathbb{Z}[\sqrt{2}]$ has attractive computational properties.

Let $f : \mathbb{R} \to \mathbb{R}$ be any real function. An **approximation** for f is any function $\widetilde{f} : \mathbb{F} \times \mathbb{N} \to \mathbb{F}$ with the property that $\widetilde{f}(x, n) \in \mathbb{F}_n$ and $\widetilde{f}(x, n) = f(x) \pm 2^{-n}$ (i.e., $\widetilde{f}(x, n)$ is correct to n bits). In general, a numerical expression of the form "$x \pm \varepsilon$" (involving the symbol '$\pm$') denotes some value of the form $x + \theta\varepsilon$ where θ is an implicit variable satisfying $|\theta| \leq 1$.

Suppose we fix some rounding rule r and some grading $\{\mathbb{F}_n : n \in \mathbb{N}\}$ of $\mathbb{F}$. Then an **exactly rounded approximation** for f is

$$\widehat{f} : \mathbb{F} \times \mathbb{N} \to \mathbb{F} \tag{1}$$

such that (1) $\widehat{f}$ is an approximation for f, and (2) the value $\widehat{f}(x, n) \in \mathbb{F}_n$ is rounded following the rule r. In the next section, we will clarify the notion of rounding rules.

Our main task is to provide an algorithm for computing $\widehat{f}$. For this, we make two natural assumptions: First, we assume the availability of an algorithm to compute an approximation $\widetilde{f}$ for f. Such algorithms are well-known

[2] $\mathbb{Z}/N$ should not be confused with integers mod N, a concept not used in this paper.

(e.g., [1,22,6]). Second, we assume that given $x \in \mathbb{F}$ and $n \in \mathbb{N}$, we know how to round x in $\mathbb{F}_n$. Both these assumptions are easily satisfied in our computational model, assuming the standard choices of f and $\mathbb{F}$. We will next see what else is needed to complete our task.

2 The Rounding Framework

We describe an abstract framework for rounding a real number y. In the context of our main task of computing $\widehat{f}$, the number y is the value of the function f at some point $x \in \mathbb{F}$.

Rounding of numbers takes place with respect to some discrete set G of points: a countable set $G \subseteq \mathbb{R}$ that is closed is called a (rounding) **grid**. Each $g \in G$ is called a **grid point**. Intuitively, the grid points serve as potential values for the approximation of real numbers. Note that G may be a finite set.

Examples: Typically, $G = \mathbb{F}_n$ in arbitrary precision arithmetic computation. For hardware-oriented algorithms, G can be the set of IEEE machine doubles. In evaluation of the arctangent function, it is desirable to output an element $y \in G$ that is in $[-\pi/2, \pi/2]$, to ensure monotonicity properties (see [16,3]). One way to do this is to insert all integer multiples of $\pi/2$ into G; it is possible to compute effective with this extended set [24].

Relative to grid G, we introduce the **floor** and **ceiling** functions on $y \in \mathbb{R}$:

$$\lfloor y \rfloor_G := \max\{x \in G : y \geq x\}, \qquad \lceil y \rceil_G := \min\{y \in G : y \leq x\}.$$

If y is less than the smallest grid point, $\lfloor y \rfloor_G = -\infty$ and if y is greater than the largest grid point, $\lceil y \rceil_G = +\infty$. Thus, $-\infty$ and $+\infty$ are implicitly included in all grids. A **rounding rule** for G is a function $r : \mathbb{R} \to G \cup \{-\infty, +\infty\}$ such that for all $y \in \mathbb{R}$, we have $r(y) \in \{\lfloor y \rfloor_G, \lceil y \rceil_G\}$. Here are 5 standard rounding rules:

r_1: Round Up	$r_1(y) = \lceil y \rceil_G$
r_2: Round Down	$r_2(y) = \lfloor y \rfloor_G$
r_3: Round toward Zero	$r_3(y) = \lfloor y \rfloor_G$ iff $y \geq 0$
r_4: Round away from Zero	$r_4(y) = \lceil y \rceil_G$ iff $y \geq 0$
r^*: Round To Nearest	$\lvert r^*(y) - y \rvert = \min\{\lvert y - \lfloor y \rfloor_G \rvert, \lvert \lceil y \rceil_G - y \rvert\}$

Note rounding to nearest $r^*(y)$ is sometimes denoted $\lfloor y \rceil$. But this rule r^* is incomplete as stated because when $y = (\lfloor y \rfloor_G + \lceil y \rceil_G)/2$, the round-to-nearest rule does not give a unique determination of $r^*(y)$. In this case, $r^*(y)$ can be decided by using one of the other four rules $(r_1, \ldots, r_4)$ as tie-breaker rule. We denote the corresponding rule by r_i^* $(i = 1, \ldots, 4)$.

There are two more rounding rules, based on an additional property of G: suppose there is **parity function** $par : G \to \{odd, even\}$. By definition, the parity function assigns different parity to any two adjacent y's in G. Clearly, there are only two possible parity functions for G. The standard parity function is the one where 0 has even parity. In case $G = \mathbb{F}_n$, this choice assigns to $g \in \mathbb{F}_n$

an even parity iff the mantissa of y is even (i.e., $y = m2^{-n}$ and m is even). We now have two more rounding rules: **Round to Even** (r_e) or **Round to Odd** (r_o). For instance, $r_e(y) = y$ if $y \in G$; otherwise $r_e(y)$ is the even parity grid point from $\{\lfloor y \rfloor_G, \lceil y \rceil_G\}$. These rules work because $\lfloor y \rfloor_G$ and $\lceil y \rceil_G$ have different parity when they are distinct. As before, we can also use these two rules as tie-breaker for round to nearest, and thus obtain the rounding rules denoted by r_e^* and r_o^*. We remark that rule r_e^* is empirically found to be a good rule for reducing error in a computation. So it is often the default rounding mode.

Naively, the problem of exact rounding is this: given a rounding rule $r : \mathbb{R} \to G$, construct an algorithm to compute r. This is naive because ordinary algorithms (in the sense of Turing) cannot possibly accept arbitrary real numbers, which are uncountably many, and which generally has no finite representation. Instead, suppose each real number y is given by some "black box" which, given $n \in \mathbb{N}$, will return an n-bit approximation of y. Formally, a **black box number** (or Cauchy function [12]) is a function $\widetilde{y} : \mathbb{N} \to \mathbb{F}$ such that there exists a $y \in \mathbb{R}$ with the property that $\widetilde{y}(n) = y \pm 2^{-n}$ for all $n \in \mathbb{N}$. Call $\widetilde{y}$ a black box for y. In computable analysis, y is called a **computable real number** if it has a black box $\widetilde{y}$ that is recursive. Turing machines can be extended to use such black boxes as inputs and outputs; these are called **oracle Turing machines** (see [12,24]).

The **black box rounding problem** relative to some rounding function r on grid G is this: given a black box $\widetilde{y}$ for y, to compute $r(y) \in G$ by using $\widetilde{y}$ as an oracle. The black box $\widetilde{y}$ arise naturally when y is the value of an approximable function f at a point $a \in \mathbb{F}$. In this case, $\widetilde{y}(n)$ is just $\widetilde{f}(a, n)$ where $\widetilde{f}$ is any fixed approximation of f.

We call G an **effective grid** if (1) for all $y \in \mathbb{F}$, we can compute $\lceil y \rceil_G$ and $\lfloor y \rfloor_G$, and (2) for all $g \in G$, we can determine the next larger g^+ and next smaller g^- grid point, $g^- < g < g^+$. Typically $G \subseteq \mathbb{F}_n$ for some n, and the effectiveness of G is easy to see. But the introduction of non-algebraic grid points like π requires special considerations which we must address.

Recall the definition of the exactly rounded approximation $\widehat{f}$ in (1). We can now interpret the problem of computing $\widehat{f}(x, n)$ as rounding $f(x)$ to the grid $\mathbb{F}_n$.

3 Two Preconditions of Exact Rounding

Given an effective grid G, and $y \in \mathbb{R}$, our goal is to round y in G according to some rounding rule. Define $\|y\|_G := \inf_{x \in G} |x - y|$ to be the distance from y to the nearest grid point. Thus $y \in G$ iff $\|y\|_G = 0$. Clearly, one of these two equalities must hold:

$$\lfloor y \rfloor_G = y - \|y\|_G \quad \text{or} \quad \lceil y \rceil_G = y + \|y\|_G.$$

For any $\varepsilon \geq 0$, we say that y is **ε-discrete** in G provided that $y \notin G$ implies $\|y\|_G > \varepsilon$. We shall denote this condition by

$$\Delta_\varepsilon(y, G). \tag{2}$$

We now state two (alternative) preconditions on (y, G) for this:

- Precondition A: $y \notin G$.
- Precondition B: $\Delta_\varepsilon(y, G)$ holds for a known $\varepsilon > 0$.

Our first result shows that either of these two preconditions suffices to achieve exact rounding; it further shows that some precondition is unavoidable.

Theorem 1. *Let G be a fixed effective grid.*
(i) There are oracle Turing machines M_A and M_B such that for all black boxes $\overline{y}$ for y, the machine M_X ($X = A$ or B) on $\overline{y}$ will output $\lceil y \rceil_G$ if Precondition X holds.
(ii) For all oracle Turing machines M, there exists black boxes $\overline{y}$ such that M on $\overline{y}$ does not compute $\lceil y \rceil_G$.

Proof. The oracle Turing machines M_A and M_B will be given below. So we only prove (ii) here. Let M be an oracle Turing machine that takes an input oracle $\widetilde{y}$ (any black box for $y \in \mathbb{R}$) and outputs $\lceil y \rceil_G$ after finitely many steps. Let $g < g'$ be two consecutive grid points. Let $\overline{g}$ be a black box for g with $\overline{g}(n) = g - 2^{-n-1}$ for all n. Then M on input $\overline{g}$ must output g. Let n_0 be the largest value of n such that the computation of M on $\overline{g}$ queries the oracle $\overline{g}(n)$. Now let $y = g + 2^{-n_0-1}$, and choose any black box $\widetilde{y}$ for y that agrees with $\overline{g}$ for the first n_0 values. Clearly M on input $\widetilde{y}$ will still output g. This is wrong since $\lceil y \rceil_G = g'$. **Q.E.D.**

¶*2. METHOD A.* If $y \notin G$, we have the following method for computing $\lceil y \rceil_G$: Compute $\widetilde{y}(n) = y \pm 2^{-n}$ for $n = 0, 1, 2, \ldots$ until the interval $[\widetilde{y}(n) \pm 2^{-n}] = [\widetilde{y}(n) - 2^{-n}, \widetilde{y}(n) + 2^{-n}]$ contains no grid points,

$$[\widetilde{y}(n) \pm 2^{-n}] \cap G = \emptyset. \tag{3}$$

We then output $\lceil \widetilde{y}(n) \rceil_G$.

Correctness: By our assumption about the effective grid G, we can check (3) since this amounts to checking that $|\lceil \widetilde{y}(n) \rceil_G - \widetilde{y}(n)| > 2^{-n}$ and $|\lfloor \widetilde{y}(n) \rfloor_G - \widetilde{y}(n)| > 2^{-n}$. Furthermore, (3) ensures that $\lceil y \rceil_G = \lceil \widetilde{y}(n) \rceil_G$. Finally, observe that (3) will eventually hold since $y \notin G$.

¶*3. METHOD B.* In contrast to the precondition A, the ε-discreteness property $\Delta_\varepsilon(y, G)$ does not exclude the possibility that $y \in G$. The constant $\varepsilon > 0$ (or a positive lower bound) must be effectively known. We use the following lemma.

Lemma 1 (Discreteness Lemma). *Suppose $\Delta_\varepsilon(y, G)$ and $\widetilde{y} = y \pm \varepsilon/2$. Then: $\lfloor y \rfloor_G < y < \lceil y \rceil_G$ iff*

$$\lfloor y \rfloor_G + \varepsilon/2 < \widetilde{y} < \lceil y \rceil_G - \varepsilon/2.$$

Proof. It is enough (by symmetry) to show that $y < \lceil y \rceil_G$ iff $\widetilde{y} < \lceil y \rceil_G - \varepsilon/2$. If $y < \lceil y \rceil_G$, then by ε-discreteness, $y + \varepsilon < \lceil y \rceil_G$. Thus $\widetilde{y} \leq y + \varepsilon/2 < \lceil y \rceil_G - \varepsilon/2$. Conversely, if $\widetilde{y} < \lceil y \rceil_G - \varepsilon/2$, then $y \leq \widetilde{y} + \varepsilon/2 < \lceil y \rceil_G$. **Q.E.D.**

METHOD B goes as follows: Fix $N = \lceil \lg(2/\varepsilon) \rceil$. Compute $\widetilde{y}(n) = y \pm 2^{-n}$ for $n = 0, 1, 2, \ldots$ until the first time that one of the following cases hold:

Case (i): If (3) holds (as in METHOD A), we output $\lceil \widetilde{y}(n) \rceil_G$ (as before).
Case (ii): $[\widetilde{y}(n) - 2^{-n}, \widetilde{y}(n) + 2^{-n}] \cap G$ contains exactly one point g and $n \geq N$. We output g in this case.

Note that termination is assured since the interval $[\widetilde{y}(n) \pm 2^{-n}] \cap G$ will eventually have at most one point. The output in case (i) is clearly correct. In case (ii), by ε-discreteness, we know that $y = g$ and so $\lceil y \rceil_G = g$. It should be observed how ε-discreteness is used in an essential way.

¶*4. Other Rounding Modes.* The preceding METHODs A and B were for rounding up (or r_1). They can clearly be modified for rounding down (r_2). It can also be used for round to odd (r_o) and round to even (r_e): assuming that we can decide if a grid point is odd of even, we first compute $\lceil y \rceil_G$.

To extend them to r_3 and r_4, we need to know the sign of y. Actually, all we need to know if whether $y = 0$. If $y \neq 0$, we can then determine the sign of y from $\widetilde{y}$. Unfortunately, deciding if $y = 0$ from $\widetilde{y}$ is a well-known undecidable problem in computable analysis. However, in our applications below, y is the value of the elementary functions evaluated at dyadic points; we could explicitly detect when $y = 0$, and METHOD B will not be invoked if $y = 0$.

Finally, consider the round to nearest function r_i^* (various i). Here, we must extend the grid G to G' where $G \subseteq G'$ and the extra grid points of G' are the "breakpoints" that lie midway between two consecutive grid points of G. Method A extends directly to the grid G'. For Method B, we need some discreteness property, $\Delta_\varepsilon(y, G')$ for a suitable ε. This concludes our discussion of the rounding algorithms.

REMARK: we can extend the above rounding methods to complex grids, $G \subseteq \mathbb{C}$ with grading with $G_n = \{x + \mathbf{i}y : x, y \in \mathbb{F}_n\}$.

4 Rounding the Elementary Functions

We now consider exact rounding of the elementary functions found in Table 1. The functions there can be put into two groups: the "pure" functions are given by

$$\left.\begin{array}{ll} \textit{Exponentiation}: & \exp(x), \exp(x) - 1 \\ \textit{Trigonometric func.}: & \sin(x), \cos(x), \tan(x), \cot(x), \sec(x), \csc(x) \\ \textit{Hyperbolic func.}: & \sinh(x), \cosh(x), \tanh(x), \coth(x), \mathrm{sech}(x), \mathrm{csch}(x) \end{array}\right\} \quad (4)$$

together with all their inverses

$$\log(x), \log(1 + x); \arcsin(x), \arccos(x), \ldots; \mathrm{arcsinh}(x), \ldots.$$

The inverses (except for log) are multivalued functions: we will assume principal values for these functions. However, they are easily generalized to allow the specification of any desired branch, by introducing an extra integer argument. The second group of functions from Table 1 are

$$\left.\begin{array}{ll} \textit{Logarithm}: & \log_2(x), \log_{10}(x), \log_y(x) \\ \textit{Exponential}: & 2^x, 10^x, x^y \\ \textit{Inverse Trigonometric}: & \arctan(x/y) \\ \textit{Inverse Hyperbolic}: & \mathrm{arctanh}(x/y) \end{array}\right\} \quad (5)$$

This group is treated separately. We appeal to the following classic results from transcendental number theory:

Proposition 1
(a) [Lindemann] *For any complex number* $\alpha \neq 0$*, either* α *or* e^α *is transcendental.*
(b) [Gelfond-Schneider] *For any complex numbers* α, β *where* $\alpha(1-\alpha) \neq 0$ *and* β *is irrational, one of the numbers in the set* $\{\alpha, \beta, \alpha^\beta\}$ *is transcendental.*

In order to apply this proposition, we need some basic facts about algebraic functions. A partial function $f : \mathbb{C} \to \mathbb{C}$ is **algebraic** if there is an integer polynomial $F(X,Y)$ such that $F(x, f(x)) = 0$ for all x in $\mathrm{dom}(f) := \{x : f(x) = \downarrow\}$, the domain of f. To ensure closure of algebraic functions under composition, it turns out that we also require that if $F(X,Y) = F_1(X)F_2(X,Y)$ then $F_1(X)$ must be a constant. Here $f(x) = \downarrow$ means f is defined at x. We say $F(X,Y)$ is a **defining polynomial** for f. The constant function $f(x) = c$ where $c \in \mathbb{A}$ is the simplest example of an algebraic function; another is $f(x) = \sqrt{x}$. If a function is not algebraic, we say it is **transcendental**. By an inverse function of f, we mean any partial function $g : \mathbb{C} \to \mathbb{C}$ such that $f(g(x)) = x$ holds for all $x \in \mathrm{dom}(g)$. Write $f^{-1}(x)$ to denote any choice of an inverse functions of f. (The notation f^{-1} will not be used for $1/f$ in this paper.) Also, let $f \circ g$ denote function composition, $(f \circ g)(x) = f(g(x))$.

Our application of Prop. 1 is reduced to an application of the following:

Lemma 2. *Let* f *be an algebraic function. If* $x \in \mathrm{dom}(f)$ *is algebraic then* $f(x)$ *is algebraic.*

Proof. Let $F(X,Y)$ be a defining polynomial for f. For algebraic x, the relation $F(x, f(x)) = 0$ implies that $f(x)$ is the zero of a polynomial with algebraic coefficients. Thus $f(x)$ is algebraic. **Q.E.D.**

Combining this lemma with the results of Lindemann and Gelfond-Schneider Prop. 1, we conclude:

Corollary 1. *Let* $y \in \mathbb{A}$ *with* $y(1-y) \neq 0$*. The functions* $f(x) = e^x$ *and* $g(x) = y^x$ *are transcendental functions.*

In this corollary, we could have stated that $g(x,y) = y^x$ is a transcendental (i.e., non-algebraic) function. The definition of algebraic functions on more than one variable is a straightforward extension of the univariate case; but we avoided this for simplicity.

Here is the analogue of Lemma 2 for transcendental functions: *if* f *is transcendental, and* $x \in \mathrm{dom}(f)$ *is algebraic then* $f(x)$ *is transcendental.* Unfortunately, this is falsified even by $f(x) = e^x$: just take $x = 0$. We say x is an **exceptional argument** for a function f if both $f(x)$ and x are algebraic. Let $E_f \subseteq \mathbb{A}$ denote the set of exceptional arguments for a function f. Lindemann says that 0 is the only exceptional argument for the function e^x, $E_f = \{0\}$. Fortunately, all the elementary functions has exactly one exceptional argument and they are

easy to detect. Gelfond-Schneider says that the set of exceptional arguments for $g(x) = y^x$ is contained in $\mathbb{Q}$. We easily verify that $E_g = \mathbb{Q}$.

Lemma 3. *The exceptional arguments of f^{-1} are contained in the set $f(E_f) = \{f(x) : x \in E_f\}$. In particular, f^{-1} cannot have more exceptional arguments than f.*

Proof. Let y be exceptional for f^{-1}. We must show that $y = f(x)$ for some $x \in E_f$. Then $\{f^{-1}(y), y\} \subseteq \mathbb{A}$. Writing $x = f^{-1}(y)$, we also have $\{x, f(x)\} \subseteq \mathbb{A}$. So $x \in E_f$ and $y = f(x)$ as desired. **Q.E.D.**

The following allows us to conclude that certain functions are algebraic:

Theorem 2 (Closure of Algebraic Functions). *If f, g are algebraic functions, so are*

$$f + g,\ f - g,\ fg,\ 1/f,\ \sqrt{f},\ f^{-1},\ f \circ g.$$

Proof. Assume $F(x, f(x)) = 0$ and $G(x, g(x)) = 0$. Our basic tool is resultant theory [21, chap. 6]. View $F(X, Y) \in \mathbb{Z}[X, Y]$ as a polynomial in $D[X]$ where $D = \mathbb{Z}[X]$. Then we can view $f(x)$ as an element of $\overline{D}$ (algebraic closure of D). The constructions for $f + g, f - g, fg, 1/f$ are obtained from the corresponding resultants for adding, subtracting, multiplying and taking reciprocals of algebraic numbers (Lemma 6.16 in [21, p.158]). To see that $h = \sqrt{f}$ is algebraic, let $H(X, Y) = F(X, Y^2)$. Then $H(x, h(x)) = F(x, h(x)^2) = F(x, f(x)) = 0$. To see that the inverse $h = f^{-1}$ is algebraic, let $H(X, Y) = F(Y, X)$. Then for all y in the domain of $h = f^{-1}$, $H(y, h(y)) = F(f^{-1}(y), y) = F(f^{-1}(y), f(f^{-1}(y))) = F(x, f(x)) = 0$. For function composition $h = f \circ g$, let $H(X, Y) = \mathrm{res}_Z(F(Z, Y), G(X, Z))$. Then $H(x, f(g(x)) = 0$. **Q.E.D.**

An application of the preceding theorem shows that certain functions are transcendental:

Corollary 2. *If f is transcendental and g is algebraic, then the following are transcendental:*

$$f + g,\ f - g,\ fg,\ f/g,\ 1/f,\ \sqrt{f},\ f^{-1},\ f \circ g,\ g \circ f.$$

The next theorem assumes the general setting where the set $\mathbb{F}$ is the set of real algebraic numbers.

Theorem 3. *Assume $\mathbb{F} = \mathbb{A} \cap \mathbb{R}$. Let f or the inverse of f be an elementary function from the list (4).*
(a) f is transcendental with one exceptional argument.
(b) The exact rounding problem for f in an effective grid $G \subseteq \mathbb{F}$ is solvable.

Proof. Suppose we have shown part (a), i.e., f is transcendental, and determined its single exceptional argument x_0. Then part (b) follows in a generic way: the algorithm for exact rounding of f amounts to detecting if an input $x \in \mathbb{F}$ is

exceptional. If so, explicitly output the exact rounding of $f(x_0)$ in $G \subseteq \mathbb{F}$ (invariably, this is trivial). Otherwise, x is non-exceptional, and we use METHOD A to round $f(x)$ in the grid G.

Exponential function: If $f(x) = e^x$, we already saw that $f(x)$ is transcendental provided $x \neq 0$. If $x = 0$, we output $e^x = 1$. Otherwise, we use METHOD A to round $f(x) = e^x$ to G. Clearly, the method extends to the variant where $f(x) = e^x - 1$.

Trigonometric functions: Suppose $y = \sin(x)$ (the case $y = \cos(x)$ is very similar). If $x = 0$, we may output $\sin(0) = 0$. Otherwise, consider $f(y) = f(\sin x) := \sqrt{1 - \sin^2 x} + \mathbf{i} \sin x = \sqrt{1 - y^2} + \mathbf{i}y$. By Thm. 2, $f(y)$ is an algebraic function of y. But $f(y) = f(\sin x) = e^{\mathbf{i}x}$ and $\mathbf{i}x$ is non-zero algebraic; so Lindemann says $f(y) = e^{\mathbf{i}x}$ is transcendental. From this fact, and Lemma 2, we conclude that y is transcendental.

Suppose $y = \tan(x) = \frac{s}{\sqrt{1-s}}$ where $s = \sin(x)$. Thus, $\tan(x)$ is an algebraic function of $\sin(x)$. By Thm. 2, we conclude $\sin(x)$ is an algebraic function of $\tan(x)$. If $x = 0$ then we may output $\tan(0) = 0$. Otherwise, we already know that $\sin(x)$ in transcendental. Then $\tan(x)$ is transcendental.

Since $\cot(x) = 1/\tan(x)$, $\sec(x) = 1/\cos(x)$, $\csc(x) = 1/\sin(x)$, it follows from Corollary 2 that $\cot(x), \sec(x), \csc(x)$ are transcendental functions. Moreover, their exceptional argument are directly obtained from the exceptional argument of their reciprocals (Lemma 3). We can round exactly at these exceptional points.

The hyperbolic functions are seen to be transcendental by Corollary 2, since they are derived from transcendental trigonometric functions using the following algebraic relations:

$$\sinh(x) = \mathbf{i} \sin(\mathbf{i}x), \qquad \cosh(x) = \cos(\mathbf{i}x), \qquad \tanh(x) = -\mathbf{i} \tan(\mathbf{i}x).$$

Inverse functions: finally, we address the inverses of all the preceding functions. It is clear from the preceding proofs that these functions are transcendental and each has exactly one exceptional argument. By Corollary 2, the inverses are all transcendental. Moreover, by Lemma 3, the inverses has at most one exceptional argument. We can directly round the functions at the single exceptional argument. **Q.E.D.**

We now turn to the functions (5) of the second group.

Theorem 4. *Let $\mathbb{F} = \mathbb{A} \cap \mathbb{R}$ and $y \in \mathbb{A}$ and $y(y-1) \neq 0$. Consider the functions $f(x) = y^x$ and its inverse $g(x) = \log_y x$.*
(a) $f(x)$ and $g(x)$ are transcendental with exceptional values $E_f = \mathbb{Q}$ and $E_g = \{y^x : x \in \mathbb{Q}\}$.
(b) The exact rounding of $f(x)$ and $g(x)$ to grid $G = \mathbb{F}_n$ is solvable.

Proof. (a) We already know that $f(x) = y^x$ is transcendental with $E_f = \mathbb{Q}$. The transcendence of $g(x) = \log_y x$ with $E_g = \{y^x : x \in \mathbb{Q}\}$ follows from Lemma 3.
(b) To perform exact rounding for $f(x) = y^x$, we first check if $f(x) \in G$, and if so, we directly output $f(x)$. If not, we can use METHOD A.

To check if $f(x) \in G$, we proceed as follows: we check if $x \in \mathbb{Q}$. If not, $f(x) \notin G$. Otherwise let $x = a/b$ be a rational in lowest terms. If $f(x) = y^x \in G$, let $f(x) = m2^{-n}$ for some positive $m \in \mathbb{Z}$. We can compute some approximation $\widetilde{w} = 2^n y^x \pm 1/4$. Then $m = \lfloor \widetilde{w} \rfloor$ or $m = 1 + \lfloor \widetilde{w} \rfloor$. So we have to check if one of two cases hold: $2^n y^x = \lfloor \widetilde{w} \rfloor$ or $2^n y^x = 1 + \lfloor \widetilde{w} \rfloor$. Let us focus of testing the first case (the other case is similar). The height H of $E = 2^n y^x - \lfloor \widetilde{w} \rfloor$ is easy to determine [15]. It follows that if $E \neq 0$ then $|E| > 1/(1+H)$. We compute an approximation $\widetilde{E} = E \pm (4(1+H))^{-1}$. If $|\widetilde{E}| < (2(1+H))^{-1}$, we know that $E = 0$, and we may output $f(x) = m2^{-n} \in G$. **Q.E.D.**

Finally, we address the two argument functions of $\arctan(x/y)$ and $\operatorname{arctanh}(x/y)$ in (5). Let $f(x) = \arctan(x/y)$ where $y \in \mathbb{A}$ is nonzero. By Corollary 2, $f(x)$ is a transcendental function. Moreover, its only exceptional argument is $x = 0$, and $f(0) = 0$. Thus METHOD A applies for rounding $f(x) \neq 0$. Similarly, we can treat $\operatorname{arctanh}(x/y)$.

All our algorithms above are based on METHOD A. It is also possible to base our algorithms on METHOD B but these require ε-discreteness properties. As we shall see in the next section, current estimates for ε (from TNT) is extremely pessimistic. So it is best to use METHOD A whenever possible.

5 Complexity of Exact Rounding

No complexity bounds can be deduced using only Precondition A. To deduce bounds, we need to invoke the ε-discreteness conditions of Precondition B (*even* when our algorithms are based on METHOD A). If $\Delta_\varepsilon(f(x), G)$ holds, the complexity of rounding $f(x)$ in G is basically the time to compute $f(x) \pm \varepsilon/2$. When f is an elementary function, Brent [1] tells us that the running time is $O(M(\log(1/\varepsilon))\log(1/\varepsilon))$ where $M(n)$ is the time to multiply two n-bit numbers. One caveat is that Brent's result is "local", i.e., it is only applicable when x lies in a bounded range [20]. More global results are obtained in [6].

The ε-discreteness results are obtained from effective forms of the Lindemann or Gelfond-Schneider theorems, such as are provided by Baker's theory of linear form in logarithms [7]. In particular, we use some explicit bounds from Nesterenko and Waldschmidt [17], somewhat simplified.

Proposition 2 (Nesterenko-Waldschmidt, 1995). *Let $\alpha, \beta \in \mathbb{A}$ with $D = [\mathbb{Q}(\alpha, \beta) : \mathbb{Q}]$ and the heights of α and β are at most H. Define*

$$B_1(D, H) := 10550 \cdot D^2 A \cdot (H + \log(A) + \log(D) + 1)(D\log(D) + 1)$$

where $A = \max\{H, D^{-1}(1+H)e\}$.

- *(i) If $\beta \neq 0$ then $|e^\beta - \alpha| \geq \exp(-B_1(D, H))$.*
- *(ii) If $\alpha \neq 0$ and $\log \alpha$ is any non-zero logarithm of α, then $|\beta - \log \alpha| \geq \exp(-B_1(D, H))$.*

Write $B_1(H) := B_1(1, H)$. If $D = 1$, then $A = (1 + H)e$ and we have

$$B_1(H) = 10550 \cdot (1 + H)e \cdot (H + \log(1 + H) + 2) \tag{6}$$

Since $H \geq 1$, we see that $B_1(H) > 10550$.

¶*5. Transfer Functions: Inhomogeneous Case.* To make our bounds as useful as possible, we state them as "transfer functions" that convert bounds such as $B_1(D, H)$ from TNT into ε-discreteness bounds. This is illustrated by the next lemma.

Lemma 4. *Let $G = \mathbb{Z}/N$ $(n \geq 0)$. If $x \in \mathbb{Q}$ $(x > 0)$ has height at most h_0, and $\log(x)$ is real, then*

$$\| \log(x) \|_G \geq \exp(-B_1(H))$$

where $H = \max\{h_0, N \log(e(1 + h_0))\}$.

Proof. Let $\Lambda_1 = g - \log x$ where $g \in G$ and $|\Lambda_1| = \| \log(x) \|_G$. So $\Lambda_1 \neq 0$. If $|\Lambda_1| \geq 1$, then our lemma is immediate. Otherwise, $|g| \leq 1 + |\log x| \leq 1 + \log(1 + h_0) = \log(e(1 + h_0))$. If $g = a/N \in \mathbb{F}_n$ then $h(g) \leq \max\{|a|, N\} \leq \max\{N \log(e(1 + h_0)), N\}$. So $h(g) \leq N \log(e(1 + h_0))$. Hence by Prop. 2(ii),

$$|g + \log x| \geq \exp(-B_1(H))$$

where $H = \max\{h(x), h(g)\} \leq \max\{h_0, N \log(e(1 + h_0))\}$. **Q.E.D.**

We can clearly obtain a similar transfer function for rounding the function $f(x) = e^x$. Suppose G is the IEEE double precision binary numbers and $x \in G$. Then $h_0 \leq 2^{1075}$ and $N = 2^{1075}$. Hence $H = 2^{1075}$. Thus it is clear that the bound $B_1(H)$ is nowhere near practical. On the other hand, the worst-case searches from Lefèvre et al [13,14] gives very practical bounds for this case. This suggests that our ability to derive sharp transcendence bounds are rather limited.

¶*6. Transfer Function: Homogeneous Case.* An expression of the form

$$\Lambda = \beta_0 + \sum_{i=1}^{m} \beta_i \log \alpha_i$$

where $\alpha_i, \beta_i \in \mathbb{A}$ is called a "linear form in logarithms". The form is homogeneous when $\beta_0 = 0$; if, in addition, the β_i's are integers, then Λ is said to be in "algebraic form". This is the case (with $m = 2$) needed for bounding $\| \log_2(x) \|_G$.

Lemma 5. *Let $\Lambda_0 = \beta_1 \log \alpha_1 + \beta_2 \log \alpha_2$ where $\alpha_1, \alpha_2 \in \mathbb{A}$ and $\beta_1, \beta_2 \in \mathbb{Z}$. If $\alpha := {\alpha_1}^{\beta_1} {\alpha_2}^{\beta_2}$, then*

$$|\Lambda_0| \geq \exp\left(-B_1\left(D, H\right)\right). \tag{7}$$

where $D = [\mathbb{Q}(\alpha) : \mathbb{Q}]$ and $h(\alpha) = H$.

Proof. Note that $\alpha \in \mathbb{A}$ so that its degree D and height H is well-defined. Thus $|\Lambda_0| = |\beta_1 \log \alpha_1 + \beta_2 \log \alpha_2| = |\log \alpha|$. Then Prop. 2(ii), with $\beta = 0$, shows that $|\beta - \log \alpha| = |\log \alpha| \geq \exp(-B_1(D, H))$. **Q.E.D.**

We now apply this bound:

Lemma 6. *Let $G = \mathbb{Z}/N$ $(N \geq 1)$. If $x \in \mathbb{Q}$ $(x > 0)$ has height at most h_0, then*

$$\|\log_2(x)\|_G \geq \exp(-B_1(H))/N$$

where $H = 2^{2N \log(1+h_0)+2} h_0^N$.

Proof. Let $\Lambda_1 = g - \log_2 x$ where $|\Lambda_1| = \|\log_2 x\|_G$ and $g \in G$. So $Ng \in \mathbb{Z}$, and $(N \log 2)\Lambda_1 = \Lambda_0$ where

$$\Lambda_0 = Ng \log 2 - N \log x.$$

If $|\Lambda_0| \geq 1$, the lemma is true. Otherwise, it is easy to see that $|g| \leq 2/N + 2|\log(x)| \leq 2/N + 2\log(1 + h_0)$. We then use Lemma 5 to lower bound $|\Lambda_0|$. To do this, we need to bound the degree D and height H of $\alpha = e^{\Lambda_0}$. Clearly, $D = 1$. Moreover, $H = h(2^{Ng} x^{-N}) = h(2^{Ng}) h(x^{-N}) \leq 2^{Ng} h_0^N \leq 2^{2N \log(1+h_0)+2} h_0^N$. **Q.E.D.**

¶7. *Non-algebraic Grid Points.* In evaluating the tangent function, it useful to know the relation of an input argument $x \in \mathbb{F}$ to multiples of $\pi/2$. The complexity of this comparison, can be deduced from the following result from [17, Theorem 2].

Proposition 3 (Nesterenko-Waldschmidt). *Let $L \geq 3$. Let ξ be a real algebraic number with $d = \deg \xi$ and $L(\xi) \leq L$. Then*

$$-\log|\pi - \xi| \leq B_\pi(d, L)$$

where $B_\pi(d, L) = 1.2 \cdot 10^6 d \cdot (\log L + d \log d)(1 + \log d)$.

We provide a corresponding transfer function:

Lemma 7. *If $G = \mathbb{F}_n = \mathbb{Z}/N$ then*

$$\|\pi\|_G \geq \exp(-B_\pi(5N))$$

Proof. The $\Lambda = g - \pi$ such that $g \in G$ and $|\Lambda| = \|\pi\|_G$. If $|\Lambda| \geq 1$, the result is immediate. Else, $|g| < 1 + \pi$. If $g = a/N$ then $L(g) \leq |a| + N \leq N(1 + \pi) + N < 5N$. **Q.E.D.**

¶8. *How to Use ε-discreteness for π.* Suppose we are given $x \in \mathbb{Z}/N$. We want to determine the sign of $\tan(x)$. Suppose $\frac{m\pi}{2} < x < \frac{(m+1)\pi}{2}$. Then $\text{sign}(\tan(x)) = 1 - 2 \cdot \text{parity}(m)$ where $\text{parity}(m) = 0$ if m is even, and $\text{parity}(m) = 1$ if m is odd. Thus, our goal is to determine m.

We first compute $\widetilde{m} = \frac{2x}{\pi} \pm 2^{-3}$. Let $m' = \lfloor m \rceil$ (rounding, with ties arbitrarily taken). If $|m' - \widetilde{m}| > 1/4$, we know that $m = \lfloor \widetilde{m} \rfloor$, so we can output $1 - 2 \cdot \text{parity}(\lfloor \widetilde{m} \rfloor)$.

Otherwise, notice that $m'\pi/2 > x$ iff $\pi > 2x/m'$. But $2x/m' \in \mathbb{Z}/(m'N)$. According to Lemma 7, $-\log|\pi - (2x/m')| \leq B_\pi(5m'N)$. Hence we compute $\widetilde{\pi} = \pi \pm 2^{-1-B_\pi(5m'N)}$. We then know that $\pi > 2x/m'$ iff $\widetilde{\pi} > 2x/m'$. The latter is easy to determine. If $\widetilde{\pi} > 2x/m'$, we output $1 - 2 \cdot \text{parity}(m')$. Otherwise, we output $2 \cdot \text{parity}(m') - 1$.

The above transfer functions illustrate the three types ε-discreteness bounds that are of interest. In a full paper, we will describe other transfer functions for the other elementary functions.

6 Conclusions

As this paper shows, the proper foundations for exact rounding are (1) the computational framework of arbitrary-precision approximation, and (2) the mathematical properties of transcendence.

We have shown that in the most important case of elementary functions, the exact rounding problem is solvable (thereby avoiding the Table Maker's Dilemma). However, pending much improved bounds from transcendental number theory, the worst-case complexity analysis of these algorithms is hopelessly pessimistic. Evidence (in the case of single and double precision IEEE number formats) suggests that the actual bounds are much better than we are able to prove. In any case, our algorithms based on METHOD A are naturally adaptive and run in time (roughly) proportional to the actual bounds.

Acknowledgments

This work is supported by NSF Grant CCF-0728977.

References

1. Brent, R.P.: Fast multiple-precision evaluation of elementary functions. J. of the ACM 23, 242–251 (1976)
2. Brönnimann, H., Burnikel, C., Pion, S.: Interval arithmetic yields efficient dynamic filters for computational geometry. Discrete Applied Mathematics 109(1-2), 25–47 (2001)
3. Defour, D., Hanrot, G., Lefèvre, V., Muller, J.-M., Revol, N., Zimmermann, P.: Proposal for a standardization of mathematical function implementation in floating-point arithmetic. Numerical Algorithms 37(1-4), 367–375 (2004)
4. Du, Z.: Guaranteed Precision for Transcendental and Algebraic Computation made Easy. Ph.D. thesis, New York University, Department of Computer Science, Courant Institute (May 2006), `http://cs.nyu.edu/exact/doc/`

5. Du, Z., Eleftheriou, M., Moreira, J., Yap, C.: Hypergeometric functions in exact geometric computation. In: Brattka, V., Schoeder, M., Weihrauch, K. (eds.) Proc. 5th Workshop on Computability and Complexity in Analysis, Malaga, Spain, July 12–13. Electronic Notes in Theoretical Computer Science, vol. 66(1), pp. 55–66 (2002), http://www.elsevier.nl/locate/entcs/volume66.html
6. Du, Z., Yap, C.: Uniform complexity of approximating hypergeometric functions with absolute error. In: Pae, S., Park, H. (eds.) Proc. 7th Asian Symp. on Computer Math. (ASCM 2005), pp. 246–249 (2006)
7. Fel'dman, N., Nesterenko, Y.V.: Number Theory IV: Transcendental Numbers. Encyclopaedia of Mathematical Sciences, vol. 44. Springer, Berlin (1998); translated from Russian by N. Koblitz
8. Fousse, L., Hanrot, G., Lefévre, V., Pélissier, P., Zimmermann, P.: Mpfr: A multiple-precision binary floating-point library with correct rounding. ACM Trans. Math. Softw. 33(2), 13 (2007)
9. Gal, S., Bachelis, B.: An accurate elementary mathematical library for the IEEE floating point standard. ACM Trans. on Math. Software 17(1), 26–45 (1991)
10. Goldberg, D.: What every computer scientist should know about floating-point arithmetic. ACM Computing Surveys 23(1), 5–48 (1991)
11. IEEE. ANSI/IEEE Standard 754-1985 for binary floating-point arithmetic, The Institute of Electrical and Electronic Engineers Inc., New York (1985)
12. Ko, K.-I.: Complexity Theory of Real Functions. Progress in Theoretical Computer Science. Birkhäuser, Boston (1991)
13. Lefévre, V., Muller, J.-M., Tisserand, A.: Towards correctly rounded transcendentals. IEEE Trans. Computers 47(11), 1235–1243 (1998)
14. Lefèvre, V., Stehlé, D., Zimmermann, P.: Worst cases for the exponential function in the IEEE 754r decimal64 format. In: Hertling, P., Hoffmann, C.M., Luther, W., Revol, N. (eds.) Real Number Algorithms. LNCS, vol. 5045, pp. 114–126. Springer, Heidelberg (2008)
15. Li, C., Pion, S., Yap, C.: Recent progress in Exact Geometric Computation. J. of Logic and Algebraic Programming 64(1), 85–111 (2004); special issue on "Practical Development of Exact Real Number Computation"
16. Muller, J.-M.: Elementary Functions: Algorithms and Implementation. Birkhäuser, Boston (1997)
17. Nesterenko, Y., Waldschmidt, M.: On the approximation of the values of exponential function and logarithm by algebraic numbers. Mat. Zapiski 2, 23–42 (1996); Diophantine Approximations, Proc. of papers dedicated to the memory of Prof. N.I. Feldman, Moscow, http://arxiv.org/abs/math.NT/0002047
18. I. S. O. ISO/IEC 10967-2-2001 for Language Independne Arithmetic: Elementary Numerical Functions. International Standards Organisation, Geneva, Switzerland (2001)
19. Richardson, D.: How to recognize zero. J. of Symbolic Computation 24, 627–645 (1997)
20. Sharma, V., Du, Z., Yap, C.: Robust approximate zeros. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 874–887. Springer, Heidelberg (2005)
21. Yap, C.K.: Fundamental Problems of Algorithmic Algebra. Oxford University Press, Oxford (2000)

22. Yap, C.K.: On guaranteed accuracy computation. In: Chen, F., Wang, D. (eds.) Geometric Computation, ch. 12, pp. 322–373. World Scientific Publishing Co., Singapore (2004)
23. Yap, C.K.: Robust geometric computation. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, 2nd edn., ch. 41, pp. 927–952. Chapman & Hall/CRC, Boca Raton (2004)
24. Yap, C.K.: Theory of real computation according to EGC. In: Hertling, P., Hoffmann, C.M., Luther, W., Revol, N. (eds.) Real Number Algorithms. LNCS, vol. 5045, pp. 193–237. Springer, Heidelberg (2008)
25. Ziv, A.: Fast evaluation of elementary mathematical functions with correctly rounded last bit. ACM Trans. on Math. Software 17(3), 410–423 (1991)

Approximating Shortest Paths in Graphs

Sandeep Sen

Department of Comp. Sc. & Engg., Indian Institute of Technology Delhi,
New Delhi-110016, India
{ssen}@cse.iitd.ernet.in

Abstract. Computing all-pairs distances in a graph is a fundamental problem of computer science but there has been a status quo with respect to the general problem of weighted directed graphs. In contrast, there has been a growing interest in the area of algorithms for approximate shortest paths leading to many interesting variations of the original problem.

In this article, we trace some of the fundamental developments like spanners and distance oracles, their underlying constructions, as well as their applications to the approximate all-pairs shortest paths.

1 Introduction

The all-pairs shortest paths (APSP) problem is one of the most fundamental algorithmic graph problems of computer science. Efficient algorithms for the APSP problem are very important in several applications. In many applications the aim is not to compute *all* distances, but to have a mechanism (data structure) through which we can extract distance/shortest-path for any pair of vertices efficiently. Therefore, the following is a useful alternate formulation of the APSP problem.

Preprocess a given graph efficiently to build a data structure that can answer a shortest-path query or a distance query for any pair of vertices.

There are various algorithms for the APSP problem depending upon whether the graph is directed or undirected, edges are weighted or unweighted, weights are non-negative or negative. In its most generic version, that is, for directed graph with real edge weights, the best known algorithm [25] for this problem requires $O(mn + n^2 \log\log n)$ time. However, for graphs with $m = \Theta(n^2)$, this algorithm has a running time of $\Theta(n^3)$ which matches that of the old and classical algorithm of Floyd and Warshal [18]. The existing lower bound on the time complexity of APSP is the trivial $\Omega(n^2)$ lower bound. Researchers have striven for years to improve the time complexity of APSP but with little success - the best known upper bound on the worst case time complexity of APSP problem is $O(n^3/\log^2 n)$ due to Chan [14], which is marginally subcubic. There exist subcubic algorithms for the APSP problem if the edge weights are integers in a bounded range. All these algorithms employ the fast (subcubic) algorithm for matrix multiplication.

The underlying intuition for taking this approach is the fact that computing all-pairs distances in a graph is related to computing $(min, +)$ product (called *distance* product) of matrices. Let ω be the exponent of matrix multiplication,

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 32–43, 2009.

i.e., the smallest constant for which matrix multiplication can be performed using $O(n^\omega)$ algebraic operations - additions, *subtractions*, and multiplications. The fastest known algorithm for matrix multiplication due to Coppersmith and Winograd [17] implies $\omega < 2.376$. For undirected unweighted graphs, Seidel gave a very simple and elegant algorithm to solve APSP in $\tilde{O}(n^\omega)$ [1] time. In fact he showed that APSP in undirected unweighted graphs is harder than Boolean matrix multiplication by at most a polylogarithmic factor. For APSP in undirected graphs with edge weights from $\{0, 1, ..., M\}$, Shoshana and Vick [28] designed an $\tilde{O}(Mn^\omega)$ algorithm. Note the dependence of the time complexity on the absolute value of max edge weight - this is not acceptable in most of the situations. So, even in the case of integer edge weights from arbitrary range, the goal of achieving sub-cubic running time seems too far at present.

What do we actually mean by an algorithm which computes *approximate* shortest-paths/distances ? The distance reported by the algorithm between any given pair of vertices is not the exact distance, instead it may have some error. This error can be additive (surplus) or multiplicative (stretch). Let $\delta(u, v)$ denote the actual distance between vertices u and v in a given graph $G = (V, E)$. An algorithm is said to compute all-pairs t-approximate (*stretch*) distances for G if for any pair of vertices $u, v \in V$, the distance reported by it is at least $\delta(u, v)$ and at most $t\delta(u, v)$. In a similar manner, an algorithm is said to compute distance with *surplus* a if the distance reported is at least $\delta(u, v)$ and at most $\delta(u, v) + a$.

In the last ten years, many novel algorithms [2,19,16,6] have been designed for all-pairs approximate shortest paths (APASP) problem in undirected graphs that achieve sub-cubic running time and/or sub-quadratic space. To achieve this goal, the following ideas lie at the core of each of these algorithms.

1. Compute shortest paths from a large set of vertices in a sparse sub graph
2. Compute shortest paths from a small set of vertices in a dense sub-graph.

To implement the first idea, most of the algorithms employ a suitable and sparse sub graph which preserves approximate distance pairwise. Such a sub graph is called spanner. In the subsequent paragraphs, we will also trace the developments in the area of graph spanners and related structures.

2 Use of Dominating Sets

Aingworth et al. [2] showed a simple and elegant $\tilde{O}(n^{5/2})$ algorithm for finding all distances in unweighted undirected graphs with an additive error of at most 2. Their paper [2] has significantly inspired the research for designing simple and combinatorial algorithms (without matrix multiplication) for all-pairs approximate shortest paths.

The main idea is based on the following observation (for undirected, unweighted graphs) about a *dominating* set. A *dominating* set of a graph $G = (V, E)$ is a subset $W \subset V$ such that for every $v \in V$, there exists a vertex $w \in N(v)$

[1] We shall use $\tilde{O}(f(n))$ to denote $O(f(n)\,\mathrm{polylog}\,n)$, where $f(n)$ is poly(n).

that belongs to W. Here $N(v)$ is the set of neighboring vertices of v (including v). It follows that if we can compute the all-pair distances between the vertices of a dominating set, then we can approximate the distances between the original vertices within surplus 2 (for unweighted graphs).

The savings in the running time will depend on the size of the dominating set and for certain graphs, like a chain, the size of the smallest dominating set can be half the number of vertices. Although the problem of computing a minimal size dominating set is NP-hard, there is a simple algorithm to approximate a dominating set of a graph with minimum degree d. Using a simple probabilistic argument it can be shown that there exists a dominating set of size $O(n/d \log n)$. A graph can be very dense and still have very small minimum degree. Therefore, Aingworth et al. [2] used the idea of splitting the set of vertices V into two categories - *light* and *heavy*. The set of vertices with degree less than $\sqrt{n}$ is called the set of light vertices, and is denoted by $L(V)$, and the complement of $L(V)$ is called the set of heavy vertices, and is denoted by $H(V)$. Let $G_L(V)$ be the subgraph induced by $L(V)$. It is easy to observe that $G_L(V)$ will have $O(n^{3/2})$ edges and is *sparse*, whereas $H(V)$ will have a $O(\sqrt{n} \log n)$ size dominating set which is *small*. The following algorithm of Aingworth et al. [2] exploits these two facts together to compute a matrix $d[\ ,\ ]$ which will store approximate distances.

1. Execute Dijkstra's algorithm from each vertex $v \in V$ in the subgraph $G_L(V)$. Total time for this step is $O(n^{5/2})$).
2. Execute Dijkstra's algorithm from each $v \in W$ in the given graph G, where W is the dominating set for $H(V)$. Total time for this step is $O(m\sqrt{n} \log n))$.
3. For each u, v
$$d[u, v] = \min_{x \in W} (d[u, v] + d[x, u] + d[x, v])$$
Total time for this step $= O(n^{5/2} \log n)$.

The claim for the path length is based on a relatively simple and elegant argument. Consider any two vertices $u, v \in V$. If the shortest path between u and v passes through only light vertices, then $d[u, v] = \delta(u, v)$. Otherwise, let y be a heavy vertex on this path, and let $x \in W$ be a neighbor of y. Since we have exact distance information from x to all vertices in the graph. So $d[u, v] \leq \delta(x, u) + \delta(x, v)$. Since the graph is unweighted, $\delta(x, u) \leq \delta(u, y) + 1$, and $\delta(x, v) \leq \delta(y, v) + 1$. Hence

$$d[u, v] \leq \delta(u, y) + \delta(y, v) + 2 = \delta(u, v) + 2$$

Hence the algorithm approximates all-pair distances with additive error of 2 only. Dor et al. [19] improved and extended the algorithms of Aingworth et al. [2] - their algorithm requires $O(kn^{2-\frac{1}{k}} m^{\frac{1}{k}} \operatorname{polylog} n)$ time for finding distances with surplus $2(k-1)$ for all pair of vertices in unweighted undirected graphs.

Cohen and Zwick designed an $\tilde{O}(n^{3/2} m^{1/2})$ time algorithm for all-pairs 2-approximate distances for weighted undirected graphs. They also designed an $\tilde{O}(n^{7/3})$ time algorithm for stretch 7/3. The space complexity of these algorithms is $\Theta(n^2)$ which is indeed optimal as seen from the above mentioned lower bound

for stretch < 3. In the same paper, Cohen and Zwick also designed an $O(n^2 \log n)$ time algorithm for stretch 3, and the space required is $\Theta(n^2)$. Their algorithms were nontrivial extensions of the algorithms by Dor et al. [19].

3 Graph Spanners

A spanner is a *sparse* subgraph of a given undirected graph that preserves approximate distance between each pair of vertices. More precisely, a t-spanner of a graph $G = (V, E)$ is a subgraph $(V, E_S), E_S \subseteq E$ such that, for any pair of vertices, their distance in the subgraph is at most t times their distance in the original graph. The parameter t is called the *stretch factor* associated with the t-spanner. The concept of spanners was defined formally by Peleg and Schäffer [26] though the associated notion was used implicitly by Awerbuch [4] in the context of network synchronizers.

The reader may realise that constructing a spanner doesn't directly yield the pairwise distances; however, if we were to run any APSP algorithm we can obtain the pairwise distances within stretch t. Evidently, the APSP algorithms will be much faster if the number of edges in the t-spanner is significantly lower.

Computing a minimum size t-spanner for a given graph is a well motivated combinatorial problem with many applications. Clearly, the smallest size of a t-spanner depends on the input graph. For example, it is easy to observe that a complete graph on n vertices has a 2-spanner (star) of size $n - 1$ whereas no bipartite graph has a 2-spanner except the graph itself. In general, computing t-spanner of smallest size for a graph is NP-hard. In fact, for $t > 2$, it is NP-hard [21] even to approximate the smallest size of t-spanner of a graph with ratio $O(2^{(1-\mu)\ln n})$ for any $\mu > 0$. Having realized this fact, researchers have pursued another direction which is quite interesting and useful. Let S_G^t be the size of the sparsest t-spanner of a graph G, and let $\mathcal{S}_n^t$ be the maximum value of S_G^t over all possible graphs on n vertices. Does there exist a polynomial time algorithm which computes, for any weighted graph on n vertices and parameter t, its t-spanner with size $O(\mathcal{S}_n^t)$? Such an algorithm would be the best one can hope for given the hardness of the original t-spanner problem. Naturally the question arises as to how large can $\mathcal{S}_n^t$ be ? A 43 years old girth lower bound conjecture by Erdös [23] implies that there are graphs on n vertices whose $2k$ as well as $(2k-1)$-spanner will require $\Omega(n^{1+1/k})$ edges. This conjecture has been proved for $k = 1, 2, 3$ and 5. Note that a $(2k-1)$-spanner is also a $2k$-spanner and the lower bound on the size is the same for both $2k$-spanner and $(2k-1)$-spanner. So the objective is to design an algorithm that, for any weighted graph on n vertices computes a $(2k-1)$-spanner of $O(n^{1+1/k})$ size. Needless to say, one would like to design the fastest algorithm for this problem, and the most ambitious aim would be to achieve the linear time complexity.

Althöfer et al. [3] were the first to design a polynomial time algorithm for computing a $(2k-1)$-spanner of $O(n^{1+1/k})$ size for any weighted graph. They are able to meet the worst case size bound, however, their algorithm requires $O(\min(kn^{2+1/k}, mn^{1+1/k}))$ time. Cohen [15], and later Thorup and Zwick [30]

designed faster algorithms for computing spanners but the time complexities of these algorithms are far from being linear.

An algorithm of [10] computes a spanner without any sort of local/global distance computation, and runs in expected linear time. To achieve this goal, the algorithm employs a novel clustering which is based on a very local approach. The main result can be stated formally as follows :

> Given a weighted graph $G = (V, E)$, and integer $k > 1$, a spanner of $(2k-1)$-stretch and $O(kn^{1+1/k})$ size can be computed in expected $O(km)$ time.

The notion of a spanner has been generalized in the past by many researchers. *Additive spanners* : A t-spanner as defined above approximates pairwise distances with multiplicative error, and can be called a multiplicative spanner. In an analogous manner, one can define spanners that approximate pairwise distances with additive error. Such a spanner is called an additive spanner and the corresponding error is called *surplus*. Aingworth et al. [2] presented the first additive spanner of size $O(n^{3/2} \log n)$ with surplus 2. Baswana et al. [9] presented a construction of $O(n^{4/3})$ size additive spanner with surplus 6. It is a major open problem if there exists any sparser additive spanner.
(α, β)-*spanner* : Elkin and Peleg [22] introduced the notion of (α, β)-spanner for unweighted graphs, which can be viewed as a hybrid of multiplicative and additive spanners. An (α, β)-spanner is a subgraph such that the distance between any pair of vertices $u, v \in V$ in this subgraph is bounded by $\alpha\delta(u, v) + \beta$, where $\delta(u, v)$ is the distance between u and v in the original graph. Elkin and Peleg showed that an $(1 + \epsilon, \beta)$-spanner of size $O(\beta n^{1+\delta})$, for arbitrarily small $\epsilon, \delta > 0$, can be computed at the expense of sufficiently large surplus β. Recently Thorup and Zwick [31] introduced a spanner where the additive error is sublinear in terms of the *distance* being approximated.

Other interesting variants of spanner include *distance preserver* proposed by Bollobás et al. [5] and *Light-weight* spanner proposed by Awerbuch et al. [1]. A subgraph is said to be a d-preserver if it preserves exact distances for each pair of vertices which are separated by distance at least d. A light-weight spanner tries to minimize the number of edges as well as the total edge weight. A *lightness* parameter is defined for a subgraph as the ratio of total weight of all its edges and the weight of the minimum spanning tree of the graph. Awerbuch et al. [1] showed that for any weighted graph and integer $k > 1$, there exists a polynomially constructible $O(k)$-spanner with $O(k\rho n^{1+1/k})$ edges and $O(k\rho n^{1/k})$ lightness, where $\rho = \log(Diameter)$.

4 Approximate Distance Oracles

The notion of *Distance Oracles* was formalised in the seminal paper of Thorup and Zwick [30] which is a milestone in the area of all-pairs approximate shortest paths. They showed that for any integer $k \geq 2$, an undirected weighted graph can be preprocessed in $O(kmn^{1/k})$ time to build a data structure of size $O(kn^{1+1/k})$.

This data structure is capable of answering any distance query with a *stretch* $2k-1$ in just $O(k)$ time. The fact that the data structure doesn't store the all-pairs approximate distances explicitly, and is yet capable of answering any distance query in *essentially* constant time is indeed surprising. Due to this feature, this data structure is named *approximate distance oracle*. In addition, the space requirement of the oracle is essentially optimal, assuming a 1963 girth lower bound conjecture of Erdös [23]. To highlight the beauty and simplicity of the data structure, we provide a sketch of 3-approximate distance oracle.

The objective is to achieve sub-quadratic space and sub-cubic preprecessing time simultaneously. The 3-approximate distance oracle of Thorup and Zwick [30] meet this objective as follows. It selects a *small* set S of vertices. From each vertex of the set S, distance to all the vertices in the graph is computed and stored. From each $v \in V \backslash S$, distance to a only a small *neighborhood* of vertices is computed. More specifically, this neighborhood around v consists of all those vertices in the graph whose distance from v is less than the distance between v and the nearest vertex from S. This neighborhood may be viewed as a *ball* around v, and by employing simple random sampling in selecting S, it can be shown that expected size of *ball* turns out to be sublinear. The data structure stores the exact distances from each vertex of S to every vertex in the graph, stores one hash table for *ball*(v) for each $v \in V \backslash S$. In addition, it computes the nearest vertex from S for each v. The overall space requirement turns out to be $O(n^{3/2})$ when S is formed by selecting each vertex with probability $1/\sqrt{n}$. To answer any distance query between u and v, one can proceed as follows. If either u or v belong to S, exact distance can be retrieved. Otherwise, if u lies with in *ball*(v) (or vice versa), exact distance can still be reported in $O(1)$ time using the hash table storing *ball*(v). The only nontrivial case left is when u does not lie in *ball* of v. In this case, the oracle reports the distance between v and its nearest vertex, say x, from S plus the distance between x and u. By using triangle inequality, it follows easily that this sum is at most 3 times the exact distance between u and v.

The idea of 3-approximate distance oracle can be very nicely extended to $(2k-1)$-approximate distance oracle as shown by Thorup and Zwick [30]. To construct a $(2k-1)$-approximate distance oracle a hierarchy of subsets of vertices : $V \supseteq S_1 \supseteq \cdots \supseteq S_{k-1} \supseteq S_k = \emptyset$ is built using random sampling. In order to achieve a subquadratic space data structure, for each vertex u and level i, the preprocessing algorithm computes distance to a small subset, $ball^i(v)$ of vertices from S_{i-1}. In particular, $ball^i(v)$ consists of consists of all those vertices in S_{i-1} that are closer to u than its nearest vertex from S_i. Computing these balls efficiently turns out to require building truncated shortest path trees from vertices of S_{i-1}. Thorup and Zwick employed a modified Dijkstra's algorithm in a novel way for this task.

Baswana and Sen [12,11] showed that for unweighted graphs, a $(2k-1)$-approximate distance oracle of size $O(kn^{1/k})$ that returns $2k-1$ stretch distances in $O(k)$ time can be computed in expected $O(\min(n^2,\ kmn^{1/k}))$ time. Roditty,

Thorup, and Zwick [27] showed that the results in [30] and [12] can be achieved deterministically also.

Recently Baswana and Kavitha [8] improved the preprocessing time of approximate distance oracle to $O(\min(n^2,\ kmn^{1/k}))$ for weighted graphs as well.

5 A Generic Scheme for APASP for Stretch ≤ 3

The reader may notice that there exist two algorithms for 3-APASP: 3-approximate distance oracle of Thorup and Zwick [30] and 3-APASP algorithm of Cohen and Zwick [16]. It can be observed that the Cohen-Zwick algorithm [16] is preferred when time has to be optimized and the Thorup-Zwick algorithm [30] is preferred when space has to be optimized. Ideally, one would like an algorithm with the worst case preprocessing time of the former *and* the space requirement of the latter. This was posed as an open problem by Thorup and Zwick [30]. It is worth mentioning at this point that on studying the two algorithms [16,30] (for stretch 3) the reader would realize that these algorithms and their underlying ideas appear to be quite different.

Baswana and Kavitha [8] present a simple and generic scheme for APASP. which answers the open question of Thorup and Zwick in affirmative. We present an overview of this scheme as follows.

The generic scheme $\mathcal{A}(\mathcal{H})$ has an input parameter $\mathcal{H}$ which is a hierarchy of $k+1$ subsets $S_0 \supset S_1 \supset \cdots S_k$ of vertices of the given graph. Construction of this hierarchy employs random sampling. For each set S_i, a subset of edges $E_{S_i} \subseteq E$ is defined in a very natural manner which ensures that larger the set S_i, sparser would be the set E_{S_i}, and vice versa. The scheme executes Dijkstra's algorithm from each vertex in S_i in the graph $(V, E_{S_{i+1}})$. This hierarchical scheme is indeed simple and efficient, and it proves to be very generic and powerful technique for the APASP problem leading to improved algorithms for various values of stretch in the interval [2,3]. The 3-approximate distance oracle of Thorup-Zwick and $\Theta(n^2 \log n)$ algorithm of Cohen-Zwick for 3-APASP appear to be emerging from the new generic scheme $\mathcal{A}$ as can be seen from the following short summary of the scheme.

1. For a specific hierarchy $\mathcal{H}$ with $V = S_0 \supseteq S_1 \supseteq, S_k = \emptyset$ and $k = \log n$, the scheme $\mathcal{A}(\mathcal{H})$ turns out to be similar to the Cohen-Zwick algorithm for stretch 3. However, the new scheme with a more careful analysis leads to stretch which is *almost* 2. In particular, the distance reported between any pair of vertices $u, v \in V$ is at most $2\delta(u,v) + w_{\max}$, where $w_{\max}$ is the weight of the heaviest edge on the shortest path between u and v.
2. In a very natural manner, this scheme also leads to an $O(n^{3/2})$ space data structure capable of answering any 3-approximate distance query efficiently. The hierarchy $\mathcal{H}$ used here has the base set S_0 of size $O(\sqrt{n})$ which is formed by random sampling, and the set S_k is $\emptyset$. This data structure is parametrized such that its preprocessing time is $O(m\sqrt{n} + \min(m\sqrt{n}, kn^{2+\frac{1}{2k}}))$ and query time is $O(k)$. For the special case when there are only two levels in the hierarchy $\mathcal{H}$, this data structure degenerates to the 3-approximate distance oracle of

Thorup and Zwick [30]. Further, a bottleneck in the construction of this data structure is removed and this leads to an improved $O(\min(m\sqrt{n}, kn^{2+\frac{1}{2k}}))$ preprocessing time for the data structure. For $k = \log n$, this data structure achieves $O(\min(m\sqrt{n}, n^2 \log n))$ preprocessing time and $O(\log n)$ query time and this *almost* answers the open question posed by Thorup and Zwick [30].

3. The scheme $\mathcal{A}(\mathcal{H})$ also forms the foundation of the faster algorithms for different values of stretch in the interval $[2, 3]$. In particular, it provides faster algorithms for stretch $2, \frac{7}{3}, \frac{5}{2}$ and $(2+\epsilon)$. For each stretch, a hierarchy $\mathcal{H}$ is constructed suitably such that $\mathcal{A}(\mathcal{H})$ directly or after slight augmentation leads to a faster algorithm for APASP for that particular value of stretch.

6 A Subquadratic Algorithm for Approximate Distance Oracle

Having achieved optimal size-stretch trade offs, and essentially constant query time, it is only the preprocessing time of these oracles which may be improved. In fact, all the existing algorithms for approximate shortest paths achieve $\Omega(n^2)$ running time. Therefore, a natural question is whether it is possible to achieve $O(m + n^{2-\epsilon})$ - a subquadratic upper bound - for approximate shortest paths, and in particular for approximate distance oracles. At this point, it should be noted that if the aim is to just achieve sub-quadratic preprocessing time for these oracles, the task is not difficult at all if one employs spanners. By first computing a sufficiently sparse spanner (using linear time algorithms by [10,9]), and then building approximate distance oracle of this spanner by Thorup and Zwick algorithm [30] would achieve sub-quadratic time. However, in doing so, the size-stretch trade-off would be way off from being optimal. So the challenge is to achieve sub-quadratic preprocessing time for approximate distance oracles without violating the size-stretch trade off. This challenge becomes even more significant due to the fact that the quadratic upper bound of the existing preprocessing algorithms [30][12] for these oracles is indeed tight - there exist a family of graphs on which these algorithms would execute in $\Theta(n^2)$ time. This suggests that, in order to achieve sub-quadratic upper bound on these oracles, either a significant change in the data structure or a completely new approach would be needed.

In [7], the authors design approximate distance oracles for undirected, unweighted graphs[2] which, at the expense of constant additive error, are constructible in sub-quadratic time and preserve size stretch trade-off optimally. More precisely, they show the following. For any $k > 1$, there is a data-structure which occupies $O(kn^{1+1/k})$ space and for any pair of vertices $u, v \in V$, takes $O(k)$ time to return $\hat{\delta}(u, v)$ satisfying

$$\delta(u, v) \leq \hat{\delta}(u, v) \leq (2k-1)\delta(u, v) + c_k \qquad \text{where } \; c_k \leq \max(10, 2k-2)$$

[2] A somewhat weaker result holds for weighted graphs.

Table 1. Comparing the new algorithms with the existing algorithms for approximate distance oracles

Stretch	Space	Preprocessing Time	Reference
$(2k-1,0)$	$O(kn^{1+1/k})$	$O(\min(n^2, kmn^{1/k}))$	[30], [8]
$(3,10)$	$O(n^{3/2})$	$O(\min(m+n^{\frac{23}{12}}, m\sqrt{n}))$	[7]
$(5,10)$	$O(n^{4/3})$	$O(\min(m+n^{\frac{11}{6}}, m\sqrt[3]{n}))$	[7]
$(2k-1,2k-2)$, $k>3$	$O(kn^{1+1/k})$	$O(\min(m+n^{\frac{3}{2}+\frac{4k-3}{k(4k-6)}}, kmn^{\frac{1}{k}}))$	[7]

The preprocessing time for these oracle is $O(m+n^{2-\alpha_k})$, where $\alpha(k)$ takes value in the interval $[\frac{1}{12}, \frac{1}{2}]$ - takes value $\frac{1}{12}$ for $k=3$ and approaches $\frac{1}{2}$ steadily as k increases. The value of the exact preprocessing time of these oracles and the additive error c_k has been shown in the following Table 1. It should be noted that the effect of the extra $O(k)$ additive error would be quite insignificant compared to the $(2k-1)$ stretch except for pairs of vertices separated by small distance. However, this small additive error has allowed to break the quadratic barrier of preprocessing of approximate distance oracles. Nevertheless, it would be very important to explore the limits to which the preprocessing time can be further improved.

6.1 An Overview of the Algorithm

The starting point is the 3-approximate distance oracle of Thorup and Zwick [30]. In order to achieve sub-quadratic space, the underlying intuition of their data structure is a novel combination of computing local distance information from all the vertices and global distance information from a few selected vertices only as follows. A subset $S \subseteq V$ is formed by selecting each vertex independently with probability q. From each vertex of $S \subset V$, distance to every other vertex in the graph is computed and stored in the data structure. For every other vertex $v \in V \backslash S$, distance to all those vertices is computed which lie closer to v than the vertex of S nearest to v. Let us call this set as $ball(v, V, S)$. The collection of these balls and the shortest path trees built on vertices of S together provide an answer to a distance query with stretch 3 at most. The space requirement is $O(n/q + n^2 q)$. To achieve $O(n^{3/2})$ space requirement, this forces q to take value $1/\sqrt{n}$.

There are thus two computation tasks for 3-approximate oracles of Thorup and Zwick [30]. The first task is building shortest path trees from each of the sampled vertices in S that increases with $|S|$. The second task is construction of balls, and Thorup and Zwick gave a novel algorithm to construct Balls. Recently Baswana and Sen [12] showed that this algorithm for computing $Ball(v, V, S)$ for all $v \in V \backslash S$ has expected running time $O(\min(m/q, n/q^2))$. This upper bound on these algorithms is tight. There exists a family of graphs on n vertices and $\Theta(n/q)$ edges, for which the algorithm of Thorup and Zwick for building balls will indeed run in $\Theta(n/q^2)$ time. Thus the above theorem implies that if we

want to reduce the time required for building balls, we should increase p beyond $1/\sqrt{n}$. However increasing q beyond $\frac{1}{\sqrt{n}}$ would violate the optimality of the size of the oracle. Secondly, this would increase the time complexity of the first task as well. However, we pursue this approach and overcome these hurdles using a nontrivial combination of the following ideas/tools (for $(2k-1, c_k)$-approximate distance oracle).

1. *Partitioning graph into sparse and dense subgraphs* : Using a random sample of a set of vertices, we define a sparse subgraph with $m = o(n^{2-1/k})$, and execute Thorup and Zwick algorithm on this sub graph. This algorithm will execute in $o(n^2)$ time and will take care of approximate shortest paths between those pairs of vertices whose shortest path is fully preserved in the sparse graph. For shortest paths which are not preserved in the subgraph, the following tools (2,3) will be used. This idea has been used in the past in the context of algorithms [2,19] for computing approximate distances with purely additive error only.
2. *Storing distances between pairs* $S \times S$: Instead of storing distance for all pairs from $S \times V$ (which imposed the restriction of $S = \theta(\sqrt{n})$ for getting optimal size for 3-oracle), we store $S \times S$ for suitably large size S. This takes care of the space optimality. In doing so, the stretch needs to be preserved at $(2k-1)$.
3. $(t, t-1)$*-spanner of [9]* : An (α, β)-spanner is a subgraph such that the distance between any two vertices u, v in the sub graph is at most $\alpha\delta(u, v)+\beta$. We shall use $(t, t-1)$-spanner introduced by Baswana et al. [9] which occupy $O(n^{1+1/t})$ space and are constructible in $O(tm)$ time. These spanner are used for the first time in their full generality for improving running time of approximate distance oracles.

7 Concluding Remarks and Open Problems

In this presentation we have focused on the static problem; there have been similar developments in the context of the dynamic problem (under deletion and addition of edges). Not surprisingly, the task of maintaining approximate shortest paths has yielded faster and simpler algorithms than the exact shortest paths.

We conclude with the following two major open problems in the area of all-pairs approximate shortest paths, and the problem of graph spanners.

- Does there exist an algorithm which can compute approximate distance oracles in $O(m \operatorname{polylog} n)$ time ?
- Does there exist any purely additive spanner with size $o(n^{4/3})$?

Acknowledgement. The author is grateful to Surender Baswana who carefully went through the manuscript and edited some portions to enhance readability.

References

1. Awerbuch, B., Baratz, A., Peleg, D.: Efficient broadcast and light weight spanners. Tech. Report CS92-22, Weizmann Institute of Science (1992)
2. Aingworth, D., Chekuri, C., Indyk, P., Motwani, R.: Fast estimation of diameter and shortest paths(without matrix multiplication). SIAM Journal on Computing 28, 1167–1181 (1999)
3. Althöfer, I., Das, G., Dobkin, D.P., Joseph, D., Soares, J.: On sparse spanners of weighted graphs. Discrete and Computational Geometry 9, 81–100 (1993)
4. Awerbuch, B.: Complexity of network synchronization. Journal of Association of Computing Machinery 32(4), 804–823 (1985)
5. Bollobás, B., Coppersmith, D., Elkin, M.: Sparse distance preserves and additive spanners. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 414–423 (2003)
6. Baswana, S., Goyal, V., Sen, S.: All-pairs nearly 2-approximate shortest paths in $O(n^2 \operatorname{polylog} n)$ time. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 666–679. Springer, Heidelberg (2005)
7. Baswana, S., Gaur, A., Sen, S., Upadhyaya, J.: Distance Oracle for unweighted graphs: breaking the quadratic barrier with constant additive error. In: Proceedings of the ICALP (1) pp. 609–621 (2008)
8. Baswana, S., Kavitha, T.: Faster Algorithms for Approximate Distance Oracles and All-Pairs Small Stretch Paths. In: Proc. 47th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 591–602. IEEE, Los Alamitos (2006)
9. Baswana, S., Telikepalli, K., Mehlhorn, K., Pettie, S.: New construction of (α,β)-spanners and purely additive spanners. In: Proceedings of 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 672–681 (2005)
10. Baswana, S., Sen, S.: A simple linear time algorithm for computing a $(2k-1)$-spanner of $O(kn^{1+1/k})$ size in weighted graphs. In: Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP), pp. 384–396 (2003)
11. Baswana, S., Sen, S.: Approximate distance oracles for unweighted graphs in $\tilde{O}(n^2)$ time. In: Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 271–280 (2004)
12. Baswana, S., Sen, S.: Approximate distance oracles for unweighted graphs in expected $O(n^2)$ time. ACM Transactions on Algorithms 2, 557–577 (2006)
13. Baswana, S., Sen, S.: A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. Random Structures and Algorithms 30, 532–563 (2007)
14. Chan, T.M.: More algorithms for all-pairs shortest paths in weighted graphs. In: Proceedings of 39th Annual ACM Symposium on Theory of Computing, pp. 590–598 (2007)
15. Cohen, E.: Fast algorithms for constructing t-spanners and paths with stretch t. SIAM Journal on Computing 28, 210–236 (1998)
16. Cohen, E., Zwick, U.: All-pairs small stretch paths. Journal of Algorithms 38, 335–353 (2001)
17. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation 9, 251–280 (1990)
18. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorothms. MIT Press, Cambridge (1990)

19. Dor, D., Halperin, S., Zwick, U.: All pairs almost shortest paths. Siam Journal on Computing 29, 1740–1759 (2000)
20. Elkin, M.: Computing almost shortest paths. ACM Transactions on Algorithms (TALG) 1, 282–323 (2005)
21. Elkin, M., Peleg, D.: Strong inapproximability of the basic k-spanner problem. In: Proc. of 27th International Colloquium on Automata, Languages and Programming, pp. 636–648 (2000)
22. Elkin, M., Peleg, D.: $(1 + \epsilon, \beta)$-spanner construction for general graphs. SIAM Journal of Computing 33, 608–631 (2004)
23. Erdős, P.: Extremal problems in graph theory. In: Theory of Graphs and its Applications (Proc. Sympos. Smolenice,1963), pp. 29–36. Publ. House Czechoslovak Acad. Sci., Prague (1964)
24. Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a sparse table with $O(1)$ worst case time. Journal of Association of Computing Machinery 31, 538–544 (1984)
25. Pettie, S.: A new approach to all-pairs shortest paths on real-weighted graphs. Theoretical Computer Science 312, 47–74 (2004)
26. Peleg, D., Schaffer, A.A.: Graph spanners. Journal of Graph Theory 13, 99–116 (1989)
27. Roditty, L., Thorup, M., Zwick, U.: Deterministic constructions of approximate distance oracles and spanners. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 261–272. Springer, Heidelberg (2005)
28. Shoshan, A., Zwick, U.: All pairs shortest paths in undirected graphs with integer weights. In: Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS), pp. 605–615 (1999)
29. Thorup, M., Zwick, U.: Compact routing schemes. In: Proceedings of 13th ACM Symposium on Parallel Algorithms and Architecture, pp. 1–10 (2001)
30. Thorup, M., Zwick, U.: Approximate distance oracles. Journal of Association of Computing Machinery 52, 1–24 (2005)
31. Thorup, M., Zwick, U.: Spanners and emulators with sublinear distance errors. In: Proceedings of 17th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 802–809 (2006)
32. Zwick, U.: Exact and approximate distances in graphs - A survey. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 33–48. Springer, Heidelberg (2001)
33. Zwick, U.: All-pairs shortest paths using bridging sets and rectangular matrix multiplication. Journal of Association of Computing Machinery 49, 289–317 (2002)

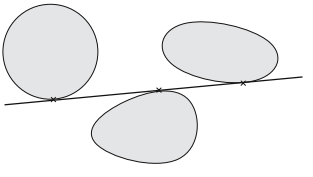
Line Transversals and Pinning Numbers

Otfried Cheong

Dept. of Computer Science, KAIST, Gwahangno 335, Yuseong-gu, Daejeon, Korea
otfried@kaist.edu

A *line transversal* to a family of convex objects in $\mathbb{R}^d$ is a line intersecting each member of the family. There is a rich theory of geometric transversals, see for instance the surveys of Danzer et al. [6], Eckhoff [7], Goodman et al. [8] and Wenger [11].

A classic result is Hadwiger's transversal theorem: a family $\mathcal{F}$ of (pairwise) disjoint compact convex sets in the plane has a line transversal if and only if there is an ordering of $\mathcal{F}$ such that each triple has a line transversal consistent with that ordering. The idea of Hadwiger's original proof is to uniformly shrink the sets until a triple has a unique transversal ℓ. The line ℓ is then necessarily an isolated transversal of the triple, that is, any perturbation of ℓ will miss a member of the triple. We call an isolated transversal ℓ to a family $\mathcal{F}$ a *pinned* transversal, and the family $\mathcal{F}$ a *pinning* of ℓ. The proof is now completed by the fact that whenever a family of compact convex sets in the plane pin a line ℓ, then three of the sets already pin ℓ, and the three sets that pin ℓ lie on alternating sides when considered in the order in which ℓ meets them (see the figure on the left).



Hadwiger's theorem does not generalize to more than two dimensions: For every n, there is a convex polytope K in $\mathbb{R}^3$ and an ordered family $\mathcal{F}$ of n translates of K such that $\mathcal{F}$ has no line transversal, but any subfamily of size $n-1$ has a line transversal consistent with the ordering on $\mathcal{F}$ [10].

There is one special case, though, in which Hadwiger's theorem can be generalized, namely when K is a (spherical) ball in $\mathbb{R}^d$. This was first shown for unit balls in $\mathbb{R}^3$ by Holmsen et al. [9], generalized to higher dimensions by Cheong et al. [4] and finally generalized to disjoint balls of arbitrary radius by Borcea et al. [2]. More precisely, a family $\mathcal{F}$ of (pairwise) disjoint balls in $\mathbb{R}^d$ has a line transversal if and only if there is an ordering of $\mathcal{F}$ such that each $2d$-tuple has a line transversal consistent with that ordering.

As in Hadwiger's original theorem, the idea of the proof is to shrink the balls uniformly until some $2d$-tuple $\mathcal{G}$ has a unique transversal ℓ. The line ℓ is then again necessarily an isolated transversal of the $2d$-tuple, that is, $\mathcal{G}$ pins ℓ. We then show that a family $\mathcal{G}$ of disjoint balls pins a line ℓ if and only if there is a subfamily $\mathcal{G}' \subset \mathcal{G}$ of size at most $2d-1$ that already pins ℓ. Finally, we use the fact that the set of transversals that intersect a family of disjoint balls in a given ordering is connected, and so ℓ must necessarily meet all the elements of $\mathcal{F}$.

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 44–46, 2009.

The connectedness of the space of transversals is a very strong property of spherical balls, and does not hold for more general objects, not even arbitrarily fat objects of constant description complexity. This explains why Hadwiger's theorem doesn't generalize.

The other ingredient of Hadwiger's original proof and our argument for spheres, however, does generalize. Let us say that a class of compact convex objects has *pinning number* k if whenever a family $\mathcal{F}$ of disjoint objects of this class pins a line ℓ, then a subfamily $\mathcal{G} \subset \mathcal{F}$ of size at most k already pins ℓ. So convex objects in the plane have pinning number three, and balls in $\mathbb{R}^d$ have pinning number $2d-1$.

In recent work [3], we showed that the bound $2d-1$ is the best possible bound on the pinning number of balls in $\mathbb{R}^d$, or in other words, there are minimal families of $2d-1$ balls that pin a line. This immediately implies a lower bound on the constant in the Hadwiger-type theorem for disjoint balls, answering the question posed by Danzer in the 1950s whether this constant must be increasing with the dimension [5].

Our proof of the lower bound is interesting in that we cannot directly exhibit such minimal families. Instead, we show that no family $\mathcal{F}$ of at most $2d-2$ balls can be a *stable pinning* of a line ℓ, that is, there is always a perturbation of $\mathcal{F}$ such that the perturbed family no longer pins ℓ. On the other hand, we show that there exist families of stable pinnings of size $2d-1$, implying that some perturbation of such a family must be minimal.

We prove the existence of stable pinnings by describing purely combinatorial patterns of halfspaces whose boundary contains the origin in $\mathbb{R}^{d-1}$. Whenever a family of disjoint convex objects projects along a line ℓ into such a *pinning pattern*, then it is automatically a pinning of ℓ; and since pinning patterns are purely combinatorial, it is then automatically a stable pinning of ℓ.

Incidentally, the argument also shows that in R^d, there are minimal pinning configurations of a line by disjoint balls of size $3, 5, 7, \ldots, 2d-1$.

In a second new result [1], we show that convex polytopes in $\mathbb{R}^3$ have bounded pinning number, at least under a mild assumption: if a family $\mathcal{F}$ of convex polytopes pins a line ℓ and ℓ touches each polytope in the interior of an edge, then there is a subfamily $\mathcal{G} \subset \mathcal{F}$ of at most six polytopes that already pins ℓ. The bound six is tight, as there are minimal pinning families of four, five, and six disjoint polytopes, and we can in fact completely characterize all pinning configurations of a line by disjoint polytopes.

This implies that a bounded pinning number is a strictly weaker property than a Hadwiger-type theorem for transversals: while the existence of a line transversal is not necessarily determined by a bounded subfamily, *locally* this is the case. It will be interesting to see how far the bounded pinning number generalizes. So far we have no counter-example that would show that arbitrary convex objects in, say, $\mathbb{R}^3$ do not have a bounded pinning number, and I conjecture that in fact their pinning number is bounded.

References

1. Aronov, B., Cheong, O., Goaoc, X., Rote, G.: Lines pinning lines (manuscript, 2008)
2. Borcea, C., Goaoc, X., Petitjean, S.: Line transversals to disjoint balls. Discrete & Computational Geometry 1-3, 158–173 (2008)
3. Cheong, O., Goaoc, X., Holmsen, A.: Lower bounds for pinning lines by balls (manuscript, 2008)
4. Cheong, O., Goaoc, X., Holmsen, A., Petitjean, S.: Hadwiger and Helly-type theorems for disjoint unit spheres. Discrete & Computational Geometry 1-3, 194–212 (2008)
5. Danzer, L.: Über ein Problem aus der kombinatorischen Geometrie. Archiv der Mathematik (1957)
6. Danzer, L., Grünbaum, B., Klee, V.: Helly's theorem and its relatives. In: Klee, V. (ed.) Convexity, Proc. of Symposia in Pure Math., pp. 101–180. Amer. Math. Soc. (1963)
7. Eckhoff, J.: Helly, Radon and Caratheodory type theorems. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Convex Geometry, pp. 389–448. North Holland, Amsterdam (1993)
8. Goodman, J.E., Pollack, R., Wenger, R.: Geometric transversal theory. In: Pach, J. (ed.) New Trends in Discrete and Computational Geometry. Algorithms and Combinatorics, vol. 10, pp. 163–198. Springer, Heidelberg (1993)
9. Holmsen, A., Katchalski, M., Lewis, T.: A Helly-type theorem for line transversals to disjoint unit balls. Discrete & Computational Geometry 29, 595–602 (2003)
10. Holmsen, A., Matoušek, J.: No Helly theorem for stabbing translates by lines in $\mathbb{R}^d$. Discrete & Computational Geometry 31, 405–410 (2004)
11. Wenger, R.: Helly-type theorems and geometric transversals. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete & Computational Geometry, 2nd edn., ch. 4, pp. 73–96. CRC Press LLC, Boca Raton (2004)

Algorithms for Computing Diffuse Reflection Paths in Polygons

Subir Kumar Ghosh[1], Partha Pratim Goswami[2], Anil Maheshwari[3], Subhas Chandra Nandy[4], Sudebkumar Prasant Pal[5], and Swami Sarvattomananda[6]

[1] School of Technology & Computer Science, Tata Institute of Fundamental Research, Mumbai 400005, India
ghosh@tifr.res.in
http://www.tcs.tifr.res.in/∼ghosh
[2] Institute of Radiophysics and Electronics, University of Calcutta, Kolkata 700009, India
ppg.rpe@caluniv.ac.in
http://www.irpel.org/index.htm
[3] School of Computer Science, Carleton University, Ottawa, Canada KIS 5B6
anil@scs.carleton.ca
http://www.scs.carleton.ca/∼maheshwa/
[4] Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata 700108, India
nandysc@isical.ac.in
http://www.isical.ac.in/∼nandysc/
[5] Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur 721302, India
spp@cse.iitkgp.ernet.in
http://www.facweb.iitkgp.ernet.in/∼spp/
[6] School of Mathematical Sciences, Ramakrishna Mission Vivekananda University, Belur 711202, India
shreesh@rkmvu.ac.in
http://www.rkmvu.ac.in/intro/academics/matsc

Abstract. Let s be a point source of light inside a polygon P of n vertices. A polygonal path from s to some point t inside P is called a *diffuse reflection path* if the turning points of the path lie on polygonal edges of P. We present three different algorithms for computing diffuse reflection paths from s to t inside P. For constructing such a path, the first algorithm uses a greedy method, the second algorithm uses a transformation of a minimum link path, and the third algorithm uses the edge-edge visibility graph of P. The first two algorithms are for polygons without holes, and they run in $O(n + k \log n)$ time, where k denotes the number of reflections in the path. The third algorithm is for both polygons with or without holes, and it runs in $O(n^2)$ time. The number of reflections in the path produced by this algorithm can be at most 3 times that of an optimal diffuse reflection path. The problem of computing a diffuse reflection path between two points inside a polygon has not been considered in the past.

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 47–58, 2009.

1 Introduction

In the last four decades, the problems of *direct* visibility have been investigated extensively [8]. Two points inside a polygon P are called *visible* (directly) if the line segment joining them lie totally inside P. The region of P directly visible from a point light source s inside P is called the *visibility polygon* of P from s (see Figure 1). Several efficient algorithms exist for computing visibility polygons under different conditions [8]. Here, we consider a problem of computing indirect visibility in P of n vertices, that arises due to multiple reflections inside P.

Assume that all edges of P reflect light like mirrors. It can be seen that some points of P, that are not directly visible or illuminated from s, may still become visible due to one or more reflections on the edges of P (see Figure 1). As per the standard law of reflection, reflection of a light ray at a point is called *specular* if the angle of incidence is the same as the angle of reflection. There is another type of reflection of light called *diffuse* reflection, where a light ray incident at a point is reflected in all possible interior directions. We assume that the light ray incident at a vertex is absorbed and not reflected.

Visibility with multiple reflections arises naturally in three dimensional scenarios where pixels of a screen are rendered to generate a realistic image. This is achieved by tracing the path of light backwards from each pixel through multiple reflections until a source of light is reached [6]. Clearly, light sources that can be reached through a smaller number of reflections would contribute more intensely, thereby making paths reachable by the minimum number of reflections is more important in illumination modeling. It is therefore worthwhile computing paths through which light arrives from a light source by the minimum number of reflections. Our concern in this paper is the computation of such a path between two points s and t within a polygon P with the minimum number of diffuse reflections. Whether this is an NP-hard problem remains an open question.

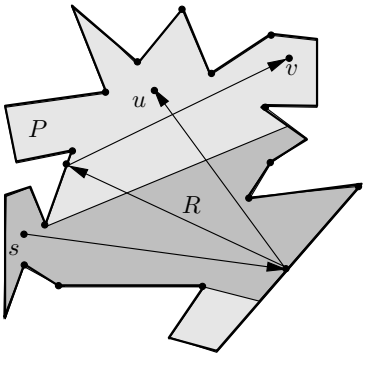


Fig. 1. The region R is directly visible from s. A ray from s reaches u after one specular reflection. A ray from s reaches v after two diffuse reflections.

Let us mention the previous results on visibility with multiple reflections. In [2], Aronov et al. investigated the region visible from a point source in a simple n-vertex polygon bounded by edges reflecting inwards, when at most one specular (or diffuse) reflection is permitted. For both specular as well as diffuse reflections, they established a tight $\Theta(n^2)$ worst-case geometric complexity bound and also designed an $O(n^2 \log^2 n)$ time algorithm for computing the region visible after at most one reflection. In [1], Aronov et al. addressed the more general problem where at most $k \geq 2$ specular reflections are permitted. They derived an $O(n^{2k})$ upper bound and an $\Omega((n/k)^{2k})$ worst-case lower bound on the geometric complexity of the region visible after at most a constant number k of *specular* reflections. They also designed an algorithm with $O(n^{2k} \log n)$ running time, for $k > 1$.

In contrast to the result of [1], Prasad et al. showed in [17], that the upper bound on the number of edges and vertices of the region visible due to at most k reflections improves to $O(n^{2\lceil (k+1)/2 \rceil + 1})$ when specular reflections are replaced by diffuse reflections. They also designed an $O(n^{2\lceil (k+1)/2 \rceil + 1} \log n)$ time algorithm for computing the visible region. They conjectured in [17] that the complexity of the region visible by at most k diffuse reflections is $\Theta(n^2)$. Note that this region contains blind spots or holes as shown in [16]. Recently, Aronov et al. [3] showed that the complexity of this region visible has an upper bound as low as $O(n^9)$. Bridging the gap between the $O(n^9)$ upper bound of Aronov at el. [3], and the $\Omega(n^2)$ lower bound in [17] remains an outstanding open problem.

In this paper, we present three different algorithms for computing diffuse reflection paths from s to t inside P. For constructing such a path, the first algorithm uses a greedy method, the second algorithm uses a transformation of a minimum link path, and the third algorithm uses the edge-edge visibility graph of P. The first two algorithms are for polygons without holes, and they run in $O(n+k \log n)$ time, where k denotes the number of reflections in the path. The third algorithm is for both polygons with or without holes, and it runs in $O(n^2)$ time. The number of turns in the path produced by this algorithm can be at most 3 times that of an optimal path, where an optimal path is a diffuse reflection path between s and t having the minimum number of turning points. In the next three sections, we present these algorithms. In Section 5, we conclude the paper with a few remarks.

2 Computing the Greedy Diffuse Reflection Path

In this section, we present an algorithm for computing a diffuse reflection path from s to t (denoted as $drp(s,t)$) inside a simple polygon P using greedy method. The algorithm runs in $O(n + k \log n)$ time, where k is the number of turning points in $drp(s,t)$.

Let $SP(u,v)$ denote the Euclidean shortest path between two points u and v inside P. Let $SP(s,t) = (u_0, u_1, \ldots, u_j)$, where $s = u_0$ and $t = u_j$. Extend the first edge u_0u_1 from u_1 till it meets the boundary of P at some point w_1 (see Figure 2). If w_1 is directly visible from t, then the diffuse reflection path from

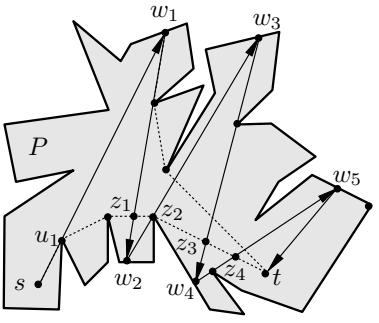


Fig. 2. The greedy diffuse path from s to t inside P

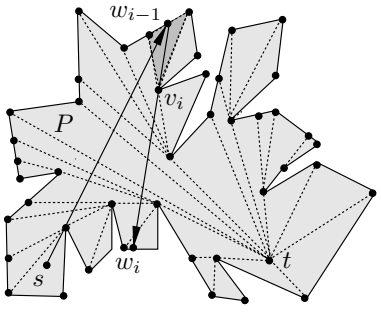


Fig. 3. The next vertex v_i of w_{i-1} on $SP(w_{i-1}, t)$ belongs to the triangle containing w_{i-1} in the shortest path map

s to t consists of two links sw_1 and w_1t. Otherwise, treating w_1 as s, compute the next link w_1w_2 of $drp(s,t)$ by extending the first edge of $SP(w_1,t)$ to the boundary of P. Repeat this process till w_k is computed such that w_k is directly visible from t. It can be seen that the greedy path $(sw_1, w_1w_2, \ldots, w_{k-1}w_k, w_kt)$ is $drp(s,t)$. The correctness of the algorithm follows from the following lemma.

Lemma 1. *The greedy diffuse reflection path* $(sw_1, w_1w_2, \ldots, w_{k-1}w_k,\ w_kt)$ *is a simple path.*

Proof. Observe that (i) every link $w_{i-1}w_i$ of the greedy path intersects $SP(s,t)$ at some point z_i, (ii) $w_{i-1}w_i$ passes through a distinct vertex v_i of P, where w_iv_i is the first edge of $SP(w_i,t)$, and (iii) for every z_i, the next intersection point z_{i+1} lies on $SP(z_i,t)$ (see Figure 2). Hence, the greedy path is simple and the number of links in the path can be at most n.

Let us analyze the time complexity of the algorithm. In order to compute $w_{i-1}w_i$, the algorithm finds the next vertex v_i of w_{i-1} in $SP(w_{i-1},t)$ and then extends $w_{i-1}v_i$ from v_i meeting the boundary of P at a point w_i. The vertex v_i can be located by computing $SP(w_{i-1},t)$, which can be done in $O(n)$ time by the algorithm of Lee and Preparata [13]. Then the point w_i can also be located in $O(n)$ time by traversing through the triangles in the triangulation of P [4]. Since each link $w_{i-1}w_i$ can be computed in $O(n)$ time and there can be at most n links, the entire greedy path can be computed in $O(n^2)$ time.

Instead of computing shortest paths repeatedly for locating the next vertex, the algorithm computes the shortest path tree rooted at t (denoted as $SPT(t)$) in $O(n)$ time by the algorithm of Hershberger [11], and then constructs the shortest path map by extending the edges of $SPT(t)$. It can be seen that the next vertex v_i of w_{i-1} is a vertex of the triangle in the shortest path map which contains w_{i-1} (see Figure 3). Once the triangle containing w_{i-1} is located, the

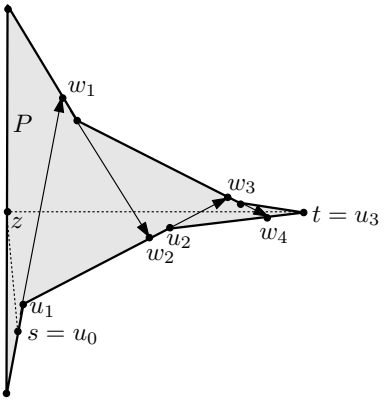


Fig. 4. Links sz and zt form an optimal path, whereas the greedy diffuse reflection path turns on all edges of P except three edges

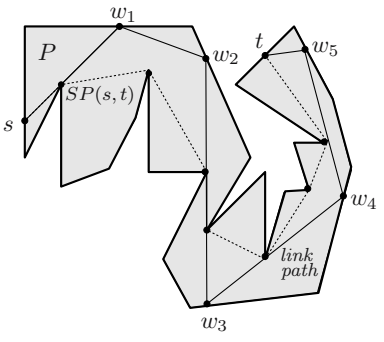


Fig. 5. The minimum link path (sw_1, $w_1w_2, w_2w_3, w_3w_4, w_4w_5, w_5t$) is also a diffuse reflection path as w_1, w_2, w_3, w_4 are boundary points

next vertex v_i is also located. Hence, the next vertex v_i can be located for each w_{i-1} in $O(\log n)$ time. Then the point w_i can be located by shooting a ray from w_{i-1} along $w_{i-1}v_i$, which takes $O(\log n)$ time after $O(n)$ time preprocessing [5]. Since the number of rays is bounded by the number of links in $drp(s,t)$, the greedy path can be computed in $O(n + k \log n)$ time.

Let us calculate the bound on the number of links of the greedy diffuse reflection path from s to t. Figure 4 shows that every link w_iw_{i+1} of the greedy path passes through a vertex of the polygonal edge containing w_i for all i. Moreover, except three edges, there is a turning point of the greedy path on every edge of P. Therefore, the number of links in the greedy path can be at most $n - 2$. On the other hand, the optimal path takes two links sz and zt to reach from s to t. Hence, the number of links in the greedy path can be at most $(n - 2)/2$ times that of an optimal path. We state the result in the following theorem.

Theorem 1. *The greedy diffuse reflection path from s to t can be computed in $O(n + k \log n)$ time, where k is the number of turning points in the path. The number of links in the path can be at most $(n - 2)/2$ times that of an optimal path.*

3 Computing a Diffuse Reflection Path Using a Minimum Link Path

In this section, we present an algorithm for transforming a minimum link path from s to t inside a simple polygon P into a diffuse reflection path ($sw_1, w_1w_2, \ldots, w_{k-1}w_k, w_kt$) in $O(n + k \log n)$ time. A minimum link path between two

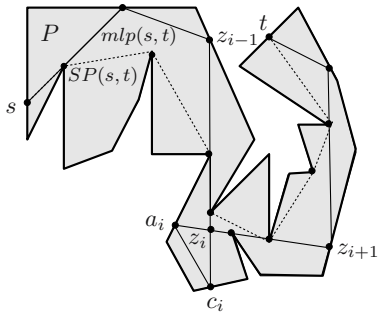


Fig. 6. Two links $z_{i-1}z_i$ and z_iz_{i+1} in $mlp(s,t)$ are replaced by three links $z_{i-1}c_i$, c_ia_i and a_iz_{i+1}

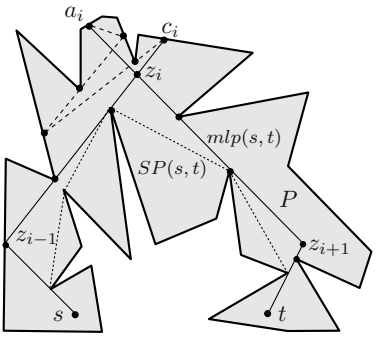


Fig. 7. The points a_i and c_i are connected by the greedy diffuse reflection path

points s and t (denoted as $mlp(s,t)$) is a path inside P having the minimum number of segments or links. We have the following observations.

Lemma 2. *Between s and t, the number of reflections k in any diffuse reflection path in P cannot be smaller than the number of turns m in any minimum link path.*

Lemma 3. *By Lemma 2, any diffuse reflection path between s and t can be at most k/m times the optimal diffuse reflection path.*

The algorithm first constructs $mlp(s,t)$ in $O(n)$ time using the algorithm of Ghosh [7]. If all turning points of $mlp(s,t)$ lie on edges of P, then $mlp(s,t)$ is $drp(s,t)$ (see Figure 5). Moreover, $drp(s,t)$ is an optimal path as it has the minimum number of turns or reflections. We have the following lemma.

Lemma 4. *If all turning points of a minimum link path lie on edges of P, then the path is an optimal diffuse reflection path.*

Let $(sz_1, z_1z_2, \ldots, z_{m-1}z_m, z_mt)$ be a minimum link path, where $z_1, z_2, \ldots, z_m$ are turning points. Consider the case when at least one turning point (say, z_i) is not lying on any edge of P (see Figure 6). Extend z_iz_{i+1} from z_i to the boundary of P meeting it at a point a_i. Similarly, extend z_iz_{i-1} from z_i to the boundary of P meeting it at a point c_i. If the segment a_ic_i lies inside P, then $(sz_1, z_1z_2, \ldots, z_{i-1}c_i, c_ia_i, a_iz_{i+1}, \ldots, z_{m-1}z_m, z_mt)$ is $drp(s,t)$. Otherwise, a_i and c_i are connected by a greedy link path as stated in the previous section to construct a diffuse reflection path (see Figure 7). If the minimum link path has turning points that are not lying on edges of P, then the above method is used for each such turning point to transform a minimum link path into a diffuse reflection path. Note that the greedy diffuse reflection paths are computed into disjoint regions of P, and are bounded by the links of the minimum link path [7], [8].

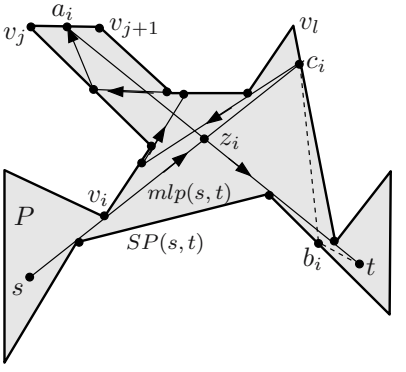


Fig. 8. The optimal path takes three links (or, two turns) to reach from s to t

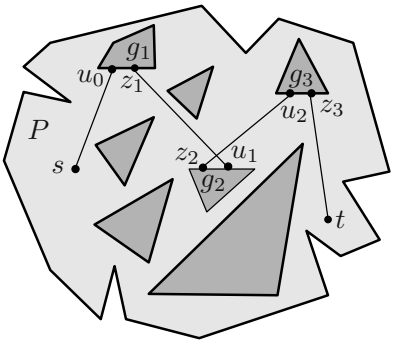


Fig. 9. A sequence of links has been constructed connecting pairs of weakly visible edges of P

Let us calculate the bound on the number of links in $drp(s,t)$. Figure 8 shows that the greedy diffuse reflection path from c_i to a_i passes through vertices of polygonal edges containing its turning points, and all turning points of the path lie on the clockwise boundary of P from v_i to v_l. Observe that this portion of the boundary from v_i to v_l can have at most $n-4-m$ vertices because the counter-clockwise boundary from v_i to v_l consists of at least four vertices of P around s and t, and the vertices of $SP(s,t)$ which is at least the number of links m of $mlp(s,t)$. In addition, no link of the greedy path can pass through vertices v_i and v_l as well as vertices of the edge v_jv_{j+1} containing a_i. So, the total number of links in the greedy path connecting c_i and a_i can have at most $n-m-8$ links. Therefore, $drp(s,t)$ can have at most $n-m-8+m=n-8$ links.

Let m' be the number of turning points of $mlp(s,t)$ not lying on the boundary of P. Since the optimal $drp(s,t)$ must take at least one additional link c_ib_i to cross $mlp(a_i,t)$ for every turning point of $mlp(s,t)$ not on the boundary of P, the optimal path must have at least $m+m'$ links. So, the number of links in $drp(s,t)$ can be at most $(n-8)/(m+m')$ times that of an optimal path. We state the result in the following theorem.

Theorem 2. *A minimum link path between s and t can be transformed into a diffuse reflection path from s to t in $O(n+k\log n)$ time, and the number of links in the path can be at most $(n-8)/(m+m')$ times that of an optimal path, where (i) k is the number of reflections in the diffuse reflection path, (ii) m is the number of links in the minimum link path, and (iii) m' is the number of turning points of the minimum link path not lying on the boundary of P.*

4 Computing a Diffuse Reflection Path Using the Visibility Graph

In this section, we present an $O(n^2)$ time algorithm for computing a diffuse reflection path from s to t inside a polygon P with or without holes using the

edge-edge visibility graph of P. The number of reflections in the path produced by the algorithm can be at most 3 times than that of an optimal $drp(s,t)$. The algorithm first finds a sequence of edges of P using BFS and then constructs a diffuse reflection path which reflects on these edges.

Let V_e denote the set of all edges of P. Two edges of P are said to be weakly visible if some internal point of one edge is visible from an internal point of the other edge. The edge-edge visibility graph G_e of P is a graph with nodes V_e and arcs between nodes that correspond to a weakly visible pair of edges in P [8], [15]. The algorithm starts by constructing the edge-edge visibility graph G_e of P. Then two nodes representing s and t are added in V_e. The node s (or, t) is connected by arcs in G_e to those nodes in V_e whose corresponding edges in P are partially or totally visible from s (respectively, t). We have the following observation.

Lemma 5. *Between s and t, the number of reflections in any diffuse reflection path in P cannot be smaller than the number of edges of P in the shortest path between s and t in G_e.*

Compute the shortest path from s to t in G_e using BFS. Let $g_1, g_2, \ldots, g_{k-1}$ be the sequence of edges of P corresponding to the nodes of V_e in the shortest path from s to t in G_e. Since g_i is weakly visible from g_{i+1}, for all i (see Figure 9), locate a pair of internal points $z_i \in g_i$ and $u_i \in g_{i+1}$, for all i, such that the segment z_iu_i lies inside P. Let u_0 be a point in g_1 visible from s. Let z_{k-1} be a point in g_{k-1} visible from t. So, a sequence of links $su_0, z_1u_1, \ldots, z_{k-2}u_{k_2}, z_{k-1}t$ has been constructed. If $z_i = u_{i-1}$, for all i (see Figure 10), then we have a diffuse reflection path $sz_1, z_1z_2, \ldots, z_{k-1}t$ with the minimum number of reflections. Otherwise, for every $z_i \neq u_{i-1}$, locate a point z_i' on an edge e_i of P such that all points of g_i are visible from z_i', and then add two links $u_{i-1}z_i'$ and $z_i'z_i$ to connect u_{i-1} with z_i (see Figure 11). The point z_i' can be located by extending the edge g_i to the nearest polygonal edge e_i and then choosing a point arbitrary close to the

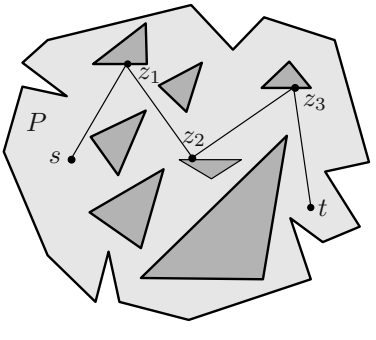


Fig. 10. The links connecting pairs of weakly visible edges of P has formed a diffuse reflection path

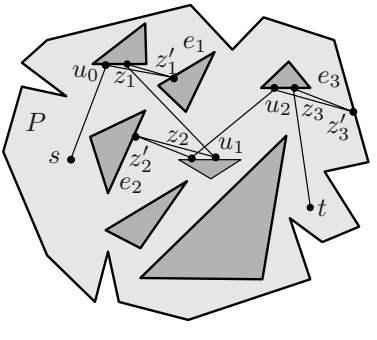


Fig. 11. Two additional links are constructed for every $z_i \neq u_{i-1}$ to form a diffuse reflection path

intersection point. Hence, $(su_0, u_0z'_1, z'_1z_1, z_1u_1, \ldots, u_{k-2}z'_{k-1}, z'_{k-1}z_{k-1}, z_{k-1}t)$ is $drp(s,t)$.

Observe that since the path can have at most $3k$ links and any optimal diffusion reflection path from s to t must have at least k links by Lemma 5, the number of reflections in the path can be at most 3 times than that of an optimal path.

Let us analyze the time complexity of the algorithm. The algorithm locates all pairs of weakly visible edges in P as follows. Using the algorithm of Ghosh and Mount [9], the algorithm first computes all visible pairs of vertices (say, E) of P in $O(n \log n + E)$ time. During the process of computation, the algorithm also constructs *funnel sequences* with edges as bases of the funnels. By traversing these funnel sequences, all pairs of weakly visible edges of P can be located. In addition, links connecting pairs of weakly visible edges of P can also be constructed using funnel sequences. The entire computation takes $O(n^2)$ time. By traversing through the funnel sequences again, edges g_1, g_2, $\ldots$, g_{k-1} can be extended to the respective nearest edges in P to locate points $z'_1, z'_2, \ldots, z'_{k-1}$ respectively. These points can be located in $O(n^2)$ time. Hence, the entire diffusion reflection path can be computed in $O(n^2)$ time. We summarize the result in the following theorem.

Theorem 3. *Using the edge-edge visibility graph of P, a diffuse reflection path from s to t can be computed in $O(n^2)$ time, and the number of reflections in the path can be at most three times than that of an optimal diffusion reflection path.*

Let us discuss the problem of choosing an appropriate point $z_i \in g_i$ on edges g_1, g_2, $\ldots$, g_{k-1} such that the segments $sz_1, z_1z_2, \ldots, z_{k-1}t$ lie inside P. Consider the case when P is a polygon without holes. Let I_1 be the set of all points of g_1 that are visible from s (see Figure 12). Similarly, let I_2 be the set of all points of g_2 that are visible from some point of I_1 . If the interval I_2 is not empty, then the next interval I_3 is again defined to be the set of all points of g_3 that are

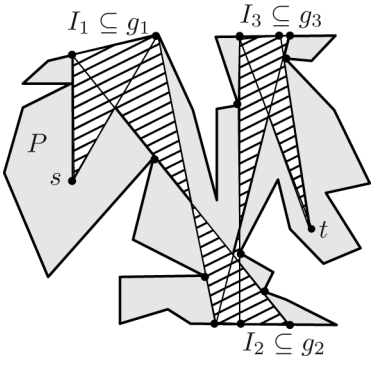


Fig. 12. The intervals I_1, I_2, I_3 on edges are not empty

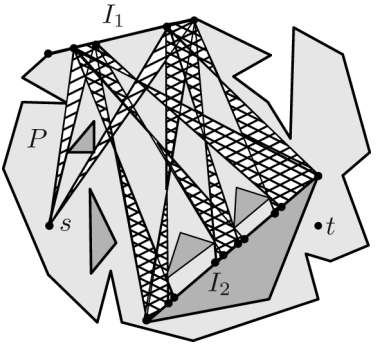


Fig. 13. There are two sub-intervals in I_1 due to one hole, and six sub-intervals in I_2 due to two holes

weakly visible from I_2. If all intervals $I_1, I_2, \ldots, I_{k-1}$ are non-empty, and there exists a point $z_{k-1} \in I_{k-1}$ such that z_{k-1} is visible from t, then for all i, there exists a point $z_i \in I_i$ that is visible from z_{i+1}. Therefore, locate a point $z_i \in I_i$ from z_{i+1}, for all i, such that $z_i z_{i+1}$ is a segment lying inside P. By Lemma 5, the path $(sz_1, z_1z_2, \ldots, z_{k-1}t)$ is an optimal diffuse reflection path from s to t. It can be seen that I_{i+1} can be computed from I_i in $O(n)$ time by computing the *hourglass* between g_i and g_{i+1} [8], [10]. Therefore, all intervals can be computed in $O(n^2)$ time. Hence, $(sz_1, z_1z_2, \ldots, z_{k-1}t)$ can be computed in $O(n^2)$ time. We have the following theorem.

Theorem 4. *Given a sequence of k edges of a polygon P without holes such that (i) the first and last edges are partially or totally visible from points s and t respectively, and (ii) every pair of consecutive edges in the sequence are weakly visible, a diffuse reflection path of k reflections using this sequence of edges from s to t can be computed, if such a path exists, in $O(n^2)$ time.*

Corollary 1. *If k is the smallest such sequence of edges in P, then the diffuse reflection path computed by the algorithm is optimal.*

Let us consider the other case when P is a polygon containing holes. Let I_1 be the set of all points of g_1 that are visible from s (see Figure 13). Since P contains holes, I_1 may consist of two or more disjoint sub-intervals. Again, let I_2 be the set of all points of g_2 that are visible from some point of sub-intervals of I_1. Observe that the number of sub-intervals in I_2 can be more than the number of sub-intervals in I_1 as P contains holes (see Figure 13). If I_2 has at least one sub-interval, compute sub-intervals of I_3. This process is repeated till sub-intervals of I_{k-1} on g_{k-1} are computed. Let $b_1, b_2, \ldots, b_{k-1}$ be a sequence of sub-intervals such that $b_i \in I_i$, for all i, and every point of b_i is visible from some point of b_{i-1}. So, a diffuse reflection path $(sz_1, z_1z_2, \ldots, z_{k-1}t)$, where $z_i \in b_i$, for all i, can be computed as stated earlier. However, the total number of sub-intervals on all edges in the sequence computed by the algorithm can be exponential. Therefore, the method of computing sub-intervals explicitly on all edges of a given sequence does not lead to any polynomial time algorithm for polygons with holes. Even after the union of overlapping sub-intervals is taken for every edge in the sequence, the total number of disjoint sub-intervals on all edges may still become exponential.

5 Concluding Remarks

We have presented algorithms for computing diffuse reflection paths from a light source s to a target point t inside P. As stated in the introduction, there are two types of reflections of light: diffuse and specular. So, it is natural to ask for a specular reflection path from s to t inside P. There is no known algorithm for this problem. Note that unlike diffuse reflection, it has been shown that a path of specular reflections may not always exist for all polygons and for all positions of s inside a polygon [12], [18].

Let $g_1, g_2, \ldots, g_{k-1}$ be a sequence of edges of P without holes such that g_i is weakly visible from g_{i+1}, for all i, and g_1 and g_{k-1} are visible from s and t respectively. Given a sequence of such edges, it is possible to find a specular reflection path (if it exists) from s to t passing through these edges in the given order as follows. Compute the interval $I_1 \subseteq g_1$ visible from s. Let s_1 be the position of the virtual source of s with respect to g_1. Let I_2 be the set of all points of g_2 such that for any point $z \in I_2$, (i) the line segment zs_1 intersects I_1, and (ii) zs_1 does not intersect the sides of the hourglass in P between g_1 and g_2. If the interval I_2 is empty, then there is no specular reflection path from s to t passing through the given sequence of edges. So, we assume that I_2 is not empty. Analogously, compute the corresponding intervals I_3 of g_3, I_4 of g_4,..., I_{k-1} of g_{k-1}. Then compute the interval $I'_{k-1} \subseteq I_{k-1}$ visible from t. Now a specular reflection path can be computed from s to t using these intervals in the reverse order. The algorithm runs in $O(n^2)$ time as all hourglasses can be computed in $O(n^2)$ time by the algorithm of Ghosh and Mount [9]. It can be seen that this result on specular reflections is analogous to Theorem 4.

Let us consider the problem of computing a minimum link path between two given points s and t inside a polygon P with holes [8], [14]. It can be seen from the last section that if segments connecting z_i with u_{i-1} are added, for all i, then the path $(su_0, u_0z_1, z_1u_1, \ldots, u_{k-2}z_{k-1}, z_{k-1}t)$ becomes a link path. Note that this sub-optimal algorithm is simpler than the optimal algorithm given by Mitchell et al. [14] which involves computing arrangements of line segments.

References

1. Aronov, B., Davis, A., Dey, T., Pal, S.P., Prasad, D.: Visibility with multiple reflections. Discrete & Computational Geometry 20, 61–78 (1998)
2. Aronov, B., Davis, A., Dey, T., Pal, S.P., Prasad, D.: Visibility with one reflection. Discrete & Computational Geometry 19, 553–574 (1998)
3. Aronov, B., Davis, A.R., Iacono, J., Yu, A.S.C.: The complexity of diffuse reflections in a simple polygon. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 93–104. Springer, Heidelberg (2006)
4. Chazelle, B.: Triangulating a simple polygon in linear time. Discrete & Computational Geometry 6, 485–524 (1991)
5. Chazelle, B., Edelsbrunner, H., Grigni, M., Guibas, L.J., Hershberger, J., Sharir, M., Snoeyink, J.: Ray shooting in polygons using geodesic triangulations. Algorithmica 12, 54–68 (1994)
6. Foley, J., van Dam, A., Feiner, S., Hughes, J., Phillips, R.: Introduction to Computer Graphics. Addison-Wesley, Reading (1994)
7. Ghosh, S.K.: Computing visibility polygon from a convex set and related problems. Journal of Algorithms 12, 75–95 (1991)
8. Ghosh, S.K.: Visibility algorithms in the plane. Cambridge University Press, Cambridge (2007)
9. Ghosh, S.K., Mount, D.M.: An output-sensitive algorithm for computing visibility graphs. SIAM Journal on Computing 20, 888–910 (1991)
10. Guibas, L.J., Hershberger, J.: Optimal shortest path queries in a simple polygon. Journal of Computer and System Science 39, 126–152 (1989)

11. Hershberger, J.: Finding the visibility graph of a polygon in time proportional to its size. Algorithmica 4, 141–155 (1989)
12. Klee, V.: Is every polygonal region illuminable from some point? American Mathematical Monthly 76, 180 (1969)
13. Lee, D.T., Preparata, F.P.: Euclidean shortest paths in the presence of rectilinear barriers. Networks 14, 393–415 (1984)
14. Mitchell, J.S.B., Rote, G., Woeginger, G.: Minimum-link paths among obstacles in the plane. Algorithmica 8, 431–459 (1992)
15. O'Rourke, J., Streinu, I.: The vertex edge visibility graph of a polygon. Computational Geometry: Theory and Applications 10, 105–120 (1998)
16. Pal, S.P., Brahma, S., Sarkar, D.: A linear worst-case lower bound on the number of holes in regions visible due to multiple diffuse reflections. Journal of Geometry 81, 5–14 (2004)
17. Prasad, D., Pal, S.P., Dey, T.: Visibility with multiple diffuse reflections. Computational Geometry: Theory and Applications 10, 187–196 (1998)
18. Tokarsky, G.T.: Polygonal rooms not illuminable from every point. American Mathematical Monthly 102, 867–879 (1995)

Shortest Gently Descending Paths

Mustaq Ahmed[1], Anna Lubiw[1,*], and Anil Maheshwari[2,**]

[1] D. R. C. School of Computer Science, University of Waterloo, ON, Canada
m6ahmed@uwaterloo.ca, alubiw@uwaterloo.ca
[2] School of Computer Science, Carleton University, ON, Canada
anil@scs.carleton.ca

Abstract. A path from s to t on a polyhedral terrain is *descending* if the height of a point p never increases while we move p along the path from s to t. We introduce a generalization of the shortest descending path problem, called the *shortest gently descending path (SGDP)* problem, where a path descends, but not too steeply. The additional constraint to disallow a very steep descent makes the paths more realistic in practice. We give two approximation algorithms (more precisely, FPTASs) to solve the SGDP problem on general terrains.

1 Introduction

A well-studied problem in computational geometry is to compute a shortest path on a polyhedral terrain. One variant of the problem for which good approximation algorithms have recently been found [3,14] is the *shortest descending path (SDP) problem*: given a polyhedral terrain, and points s and t on the surface, find a shortest path on the surface from s to t such that, as a point travels along the path, its elevation, or z-coordinate, never increases. In many applications of SDPs, we want a path that descends, but *not too steeply*. For example, when we ski down a mountain we avoid a too steep descent. In such cases, a very steep segment of a descending path should be replaced by "switchbacks" that go back and forth at a gentler slope, like the hairpin bends on a mountain road (Fig. 1). The *shortest gently descending path (SGDP)* problem combines two previously-studied problems: (i) to find SDPs; and (ii) to find shortest *anisotropic* paths, where there are different costs associated with traveling in different directions in a face. Our constraint that forbids a steep descent is an anisotropic constraint. At first glance it would seem that the entire SGDP problem is a special case of anisotropic paths, but results on anisotropic paths assume (e.g., Sun and Reif [17]) that there is a feasible path between any two points in a common face, a property that is violated when ascending paths are forbidden.

In this paper we combine techniques used for SDPs and for anisotropic paths and present two fully polynomial time approximation schemes (FPTASs) to solve the SGDP problem on a general terrain. We model the problem as a shortest

* Research supported by NSERC.
** Research supported by NSERC.

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 59–70, 2009.

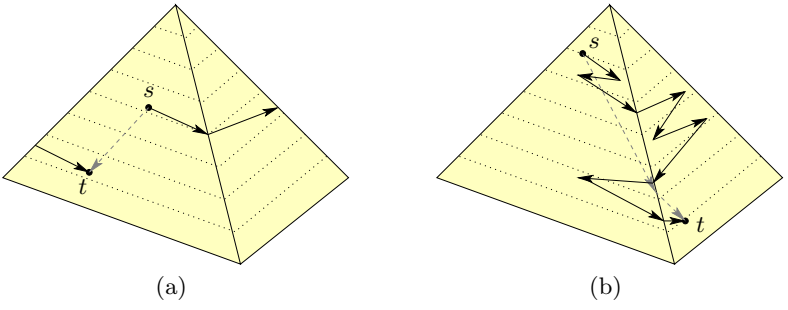


Fig. 1. Descending gently towards a steep direction

path problem in a graph whose nodes are Steiner points added along the edges of the terrain, with directed edges from higher to lower points in a common face, and edge weights corresponding to gently descending distances. Both the algorithms are simple, robust and easy to implement. We measure performance in terms of the number n of vertices of the terrain, the largest degree d of a vertex, the desired approximation factor ϵ, and a parameter X that depends on the geometry of the terrain and the measure of steepness (see Sect. 3). In our first algorithm, given a vertex s, we place Steiner points uniformly along terrain edges during an $O\left(\frac{n^2X}{\epsilon}\log\left(\frac{nX}{\epsilon}\right)\right)$ -time preprocessing so that we can determine a $(1+\epsilon)$-approximate SGDP from s to any point v in $O(nd)$ time if v is either a vertex of the terrain or a Steiner point, and in $O\left(n\left(d+\frac{X}{\epsilon}\right)\right)$ time otherwise. Our second algorithm places Steiner points in geometric progression along the edges to make the algorithm less dependent (than the first algorithm) on the slopes of the edges at the cost of slightly more dependency on n, see Theorem 2 for details. We can easily combine these two algorithms into a "hybrid" one: first check the edge inclinations of the input terrain, and then run whichever of these two algorithms ensures a better running time for that particular terrain.

The paper is organized as follows. In Sections 2 and 3 we mention related results and define a few terms. Section 4 establishes a few properties of an SGDP. Sections 5 and 6 give our approximation algorithms.

2 Related Work

The SDP problem was introduced by de Berg and van Kreveld [8], who gave an $O(n\log n)$ time algorithm to decide existence of a descending path between two points. Until recently the SDP problem has been studied in different restricted settings [2,13]. Two recent papers [3,14] give approximation algorithms for the problem on general terrains. Both papers use the Steiner point approach, i.e., the approach of discretizing the continuous space by adding Steiner points and approximating a shortest path through the space by a shortest path in the graph of Steiner points. Details of these algorithms appear in Ahmed et al. [1]. Another recent work on SDP [4] gives a full characterization of the bend angles

of an SDP, and points out the difficulty of finding an exact SDP on a general terrain (although the problem is not proved to be NP-hard).

The Steiner point approach has been used for other variants of shortest paths on terrains. One of them is the Weighted Region Problem [11]. See Aleksandrov et al. [6] for a brief survey of Steiner point algorithms for the problem, and Sun and Reif [18] for more recent work. One generalization of the Weighted Region Problem is finding a shortest anisotropic path [12], where the weight assigned to a region depends on the direction of travel. The weights in this problem capture, for example, the effect of gravity and friction on a vehicle moving on a slope. All the papers on this problem use the Steiner point approach [7,10,15,17]. As we mentioned before, these algorithms for anisotropic paths assume that every face f is *totally traversable*, i.e., there is a feasible path from any point to any other point in f. To be precise, Cheng et al. [7] assume that the (anisotropic) weight associated with a direction of travel is bounded by constants from both above and below, thus any direction of travel is feasible. (Moreover, the algorithm of Cheng et al. is for a subdivision of the plane, not for a terrain.) The rest of the papers [10,15,17] use the anisotropic weight model of Rowe and Ross [12] which allows switchback paths to "cover" any direction in f. The assumption that every face is totally traversable allows placing Steiner points in a face *independently from all other faces*. Sun and Reif [17, Sect. V] relax this assumption (i.e. the assumption that every face is totally traversable) but only in isolated faces. Thus, they can still rely on independent placement of Steiner points in a face. For both the SDP and the SGDP problems, ascending directions are unreachable in *every* face, which necessitates the use of a non-local strategy of placing Steiner points.

To obtain a better running time our algorithms use a variant of Dijkstra's algorithm, called the Bushwhack algorithm [16], to compute a shortest path in the graph of Steiner points. In such a graph, the Bushwhack algorithm improves the running time of Dijkstra's algorithm from $O(|V| \log |V| + |E|)$ to $O(|V| \log |V|)$.

3 Terminology

A terrain is a 2D surface in 3D space with the property that every vertical line intersects it in at most one point. For any point p in the terrain, $h(p)$ denotes the height of p, i.e., the z-coordinate of p. We consider a triangulated terrain, and add s as a vertex. The terrain has n vertices, and hence at most $3n$ edges and $2n$ faces by Euler's formula [9]. Let L be the length of the longest edge, h be the smallest distance of a vertex from a non-adjacent edge in the same face (i.e. the smallest 2D height of a triangular face), d be the largest degree of a vertex, and θ be the largest acute angle between a non-level edge and a vertical line.

In this paper, "edge" and "vertex" denote respectively a line segment of the terrain and an endpoint of an edge, "segment" and "node" denote respectively a line segment of a path and an endpoint of a segment, and "node" and "link" denote the corresponding entities in a graph. We assume that all paths are directed. In our figures dotted lines denote level lines.

A path P from s to t on the terrain is *descending* if the z-coordinate of a point p never increases while we move p along the path from s to t. Given an angle $\psi \in [0, \frac{\pi}{2})$, a line segment pq is *steep* if it makes an angle less than ψ with a vertical line. A path P is *gently descending* if P is descending, and no segment of P is steep. A downward direction in a face is called a *critical direction* if the direction makes an angle equal to ψ with a vertical line. (Note that only a *downward* direction can be a critical direction, although both upward and downward directions can be steep.) A gently descending path is called a *critical path* if each of its segments is in a critical direction. A critical path may travel through more than one face; inside a face it will zig-zag back and forth. We would like to replace steep descending segments by critical paths. This is sometimes possible, e.g. for a steep segment starting and ending at points interior to a face, but is not possible in general. The details are in Lemma 3, which uses the following terms. A vertex v in face f is *locally sharp in* f if v is either the higher endpoint of two steep edges or the lower endpoint of two steep edges of f. A vertex v is *sharp* if it is locally sharp in all its incident faces. Note that a sharp vertex is either the higher endpoint of all the edges incident to it, or the lower endpoint of all such edges. A sharp vertex is like a pinnacle from which you cannot descend gently.

4 Properties of an SGDP

Because our approximation algorithms use the Bushwhack algorithm which relies on the optimality of subpaths of a shortest path, we need to establish a similar property of an SGDP:

Lemma 1. *Any subpath of an SGDP is an SGDP.*

Another property that is crucial for our algorithm (and perhaps for *any* SGDP algorithm) is that any critical path in the terrain is an SGDP:

Lemma 2. *Any critical path from a point a to a point b in the terrain is an SGDP of length $(h(a) - h(b)) \sec \psi$.*

Proof. Any critical path P is a gently descending path. Since each segment of P makes an angle ψ with a vertical line, the length of P is $(h(a) - h(b)) \sec \psi$. Ignoring the terrain, *any* gently descending path from a to any point at height $h(b)$ has length at least $(h(a) - h(b)) \sec \psi$. So, P is an SGDP. □

We will now define the following notation to simplify our expressions involving the length of an SGDP. For any two points p and q in a common f, let $\|pq\| = |(h(p) - h(q))| \sec \psi$ when line pq is steep, and $\|pq\| = |pq|$ otherwise. Thus $\|pq\|$ is the length of an SGDP from p to q if one exists.

Observation 1. *For any three points p, q and r in a face f: (i) $\|pq\| = \|qp\|$, (ii) $\|pr\| \leq \|pq\| + \|qr\|$, and (iii) $|pq| \leq \|pq\| \leq |pq| \sec \psi$.*

Proof. The proof is obvious except for the second inequality of Case (iii), which follows from the inequalities $\sec\psi \geq 1$ and $|pq| \geq |h(p) - h(q)|$. □

Like other Steiner point approaches, our graph of Steiner points has a directed link (of appropriate cost) between two Steiner points a and b in a common face f such that the link represents an SGDP from a to b. However, unlike other well-studied shortest paths in terrains (e.g., SDPs and shortest paths in the Weighted Region Problem) where the shortest path represented by the link lies completely in f, the SGDP in our case may go through many other faces. For example, the critical path in Fig. 1(a) is an SGDP by Lemma 2, and it goes through four faces even though both s and t lie on a common face. It is not straightforward to determine if such an SGDP exists at all—we need this information during the construction of the graph. Moreover, in case an SGDP exists, we want to know the number of faces used by the path because our algorithm has to return the path in the terrain that corresponds to the shortest path in the graph. Lemma 3 below handles these issues:

Lemma 3. *Let a and b be two points in a face f with $h(a) \geq h(b)$.*

(i) If at least one of a and b is a sharp vertex, no gently descending path exists from a to b.
(ii) If neither a nor b is a locally sharp vertex in f, there exists an SGDP from a to b lying completely in f. Moreover, the SGDP is a critical path if ab is steep.
(iii) Otherwise, there exists an SGDP from a to b that uses at most $d+1$ faces, and is a critical path.

Proof. (i) If a [or b] is a sharp vertex, the segment ap [respectively pb] is steep for any point p in any face incident to a [respectively b]. So, no gently descending path exists from a to b.

(ii) If ab is not a steep segment, this is the SGDP. Otherwise, there are many critical paths in f from a to b, as shown Fig. 2(a). To be precise, we will trace one such path as follows. Since b is not a locally sharp vertex in f, there exists two points b_1 and b_2 on the boundary of f such that b_1b and b_2b are critical directions (Fig. 2(a)). Note that at most one of b_1 and b_2 may degenerate to point b. Now, because a is not a locally sharp vertex in f, we can follow a critical direction in f from a until we intersect either b_1b or b_2b or an edge of f. Let a_1 be the intersection point. Clearly, a_1 is not a locally sharp vertex in f, and therefore, if a_1 is neither on b_1b nor b_2b, we can again follow a critical direction from a_1 until we intersect either b_1b or b_2b or an edge of f. Let a_2 be this intersection point. We repeat this step until we reach a point a_k on either b_1b or b_2b. Thus we get the critical path $(a, a_1, a_2, \ldots, a_k, b)$, which is an SGDP by Lemma 2.

(iii) In this case, at least one of a and b is a locally sharp vertex in f, and therefore, ab is a steep line segment. If a is not a locally sharp vertex in f, let $a' = a$. Otherwise, let a' be an interior point of line segment ab such that no vertex of the terrain lies strictly in between the planes $z = h(a)$ and

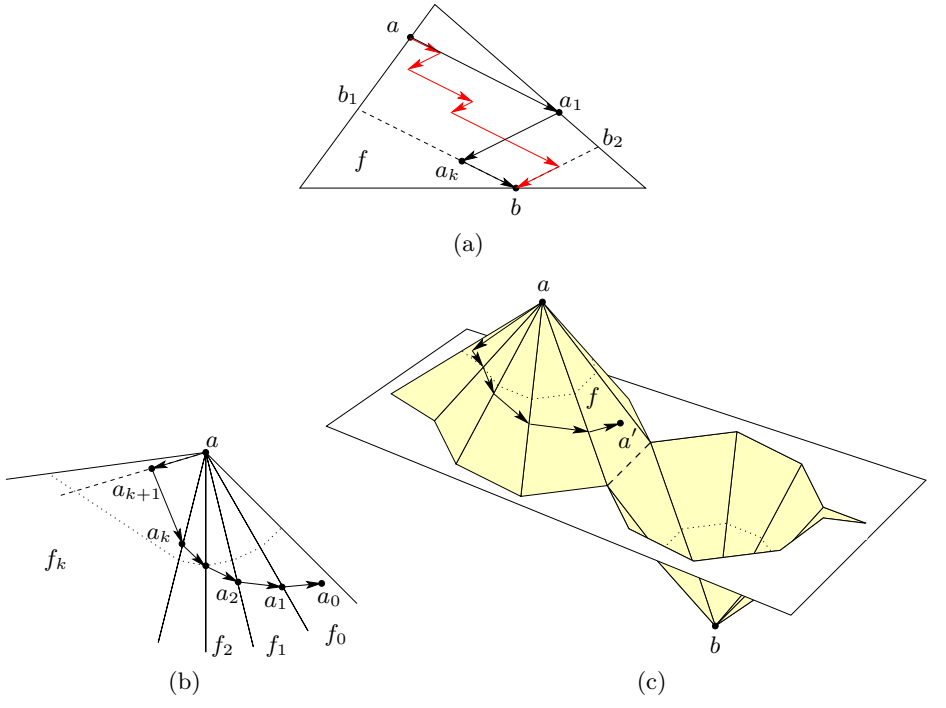


Fig. 2. (a) Two of the infinitely many critical paths from a to b when ab is a steep segment and neither a nor b is a locally sharp vertex in f. (b) A critical path from a vertex a that is locally sharp in $f = f_0$, but not in f_k. (c) An SGDP between two locally sharp vertices a and b in f that goes through $\Theta(d)$ other faces.

$z = h(a')$. We similarly define a point b', and make sure that $h(a') > h(b')$. Clearly $h(a) > h(a') > h(b') > h(b)$, and $a'b'$ is a steep line segment. By Case (ii) of the lemma there exists a critical path from a' to b' in f. We claim that there exists a critical path from a to a' through at most $\lfloor \frac{d}{2} \rfloor + 1$ faces when $a \neq a'$, and that there exists a critical path from b' to b through at most $\lfloor \frac{d}{2} \rfloor + 1$ faces when $b \neq b'$. Because face f is used by all these three subpaths, the whole path uses at most $d + 1$ faces. The proof then follows from Lemma 2. We will now prove the first claim. The proof for the second claim is similar, and hence omitted.

Since a is not a sharp vertex, it must have some incident face in which it is not locally sharp. Let $(f = f_0, f_1, f_2, \ldots, f_k)$ be (one of) the shortest sequence of faces around vertex a such that a is not a locally sharp vertex in f_k (Fig. 2(b)). Clearly, $k \leq \lfloor \frac{d}{2} \rfloor + 1$. We will build as follows a critical path backwards from a'. Let $a_0 = a'$. For each $i \in [0, k-1]$ in this order, since a is a locally sharp vertex in f_i, there is a point $a_{i+1} \in f_i \cap f_{i+1}$ such that $a_{i+1}a_i$ is a critical direction in f_i. The final point a_k lies on the edge between f_k and f_{k-1} which is a steep edge. Because a is not a locally sharp

vertex in f_k, and no other vertex of f_k lies above the plane $z = h(a_k)$, there exists an interior point $a_{k+1} \in f_k$ such that both aa_{k+1} and $a_{k+1}a_k$ are critical directions. Clearly the path $(a, a_{k+1}, a_k, \ldots, a_0 = a')$ is a critical path through at most $\lfloor \frac{d}{2} \rfloor + 1$ faces. □

It is easy to construct a terrain in which the number of faces in Case (iii) of the above lemma is exactly d, see Fig. 2(c) for an example (note that face f is a quadrilateral in the figure, and triangulating f by adding edge ab keeps both a and b in a common face). In fact, we can prove that such a path through $d + 1$ faces is impossible. We omit the proof here.

We would like the property of an SGDP that the path would visit a face at most once, because our method of approximating a path introduces some error each time the path crosses an edge. But unlike SDPs and shortest paths in the Weighted Region Problem, this property does *not* hold for an SGDP. For example, the two SGDPs in Fig. 1 visit a face twice. We can even make the SGDP in Fig. 1(b) visit a face infinitely many times, e.g., by making angle ψ arbitrarily close to $\frac{\pi}{2}$ so that the path spirals around the pyramid. We call an SGDP *ideal* if it crosses the interior of each face at most once. The good news is that we can "convert" any non-ideal SGDP into an ideal one—we will prove this claim in the following lemma:

Lemma 4. *If there is a gently descending path from s to t, there exists an ideal SGDP from s to t.*

Proof. Let P be an SGDP from s to t. Such a path exists because one gently descending path from s to t must be the shortest one. It suffices to show that if P visits the interior of a face f more than once, we can replace the portion between the first and the last visit by a shortcut that is gently descending, remains in face f, and has length no greater than the original. Let P_a [P_b] be the first [respectively last] path in $P \cap f$ that crosses the interior of face f. Let a [b] be the first [respectively last] point of P_a [respectively P_b]. Note that a and b are not locally sharp vertices in f. By Lemma 3(ii) there is an SGDP from a to b lying inside f. □

5 Approximation Using Uniform Steiner Points

5.1 Algorithm

In our first approximation algorithm, we preprocess the terrain by adding uniformly spaced Steiner points as follows. We take a set of level planes such that the distance between any two consecutive planes is at most $\delta = \frac{\epsilon h}{4n} \cos\theta \cos\psi$. We make sure that there is a level plane through every vertex. We then put Steiner points at all the intersection points of these planes with the non-level edges of the terrain. On the level edges, we add enough Steiner points so that consecutive points are at most $\delta \sec\theta$ units apart. We then construct a weighted

directed graph $G(V, E)$ in which each node in V represents either a Steiner point or a vertex, and there is a directed link $pq \in E$ if and only if all of the following are true:

(i) p and q lie in a common face,
(ii) $h(p) \geq h(q)$, and
(iii) neither p nor q is a sharp vertex.

By Lemma 3, there is an SGDP from p to q. The weight of link pq is the length of such an SGDP, i.e., $\|pq\|$. In the final step of our preprocessing we make a shortest path tree T rooted at s using the Bushwhack algorithm. The Bushwhack algorithm works for any distance metric that satisfies the following property: if e and e' are two edges of one face, and a and b are two points on edge e, then edge e' can be divided into two sub-intervals A and B, where points in A have a shorter path from a than from b and points in B have a shorter path from b than from a. This property holds for our distance metric, even though an SGDP connecting two points in face f may leave f. The proof follows from Sun and Reif [17], Lemmas 3 and 4, where they prove that the property holds for anisotropic paths.

Note that we are mentioning set E only to make the discussion easy. In practice, we do not construct E explicitly because the neighbors of a node $x \in V$ in the graph is determined *during* the execution of the Bushwhack algorithm.

To answer a query, we simply return the path from s to query point v in T if $v \in V$ and v is not a sharp vertex. If v is a sharp vertex, we return nothing since there is no SGDP from s to v. Otherwise, $v \notin V$. In this case, we find the node u among those in V lying in the face(s) containing v such that $h(u) \geq h(v)$, and the sum of $\|uv\|$ and the length of the path from s to u in T is minimum. Finally we return the corresponding path from s to v as an approximate SGDP.

There is an important issue regarding the path returned by our algorithm. It is a path in the graph augmented by vertex v. To obtain an actual path on the terrain we must replace each link ab in the path by an SGDP of the same length, which is possible by the definition of the links of the graph. Such an SGDP is not unique if ab is steep (Fig. 2(a)), but it is easy to compute one such path. In the case where neither a nor b is locally sharp in their common face f, we can even compute an SGDP with a minimum number of bends. Note, however, that the problem of minimizing the total number of bends in an SGDP is NP-hard because the hardness proof in Ahmed and Lubiw [5] applies directly to SGDPs.

5.2 Correctness and Analysis

For the proof of correctness, we show that an ideal SGDP P from s to any point v in the terrain is approximated by a path P' from s to v in the *augmented graph* $G_v(V_v, E_v)$ constructed as follows. We first add v to graph G, and then add to this graph a link uv with weight $\|uv\|$ for every $u \in V$ such that u and v lie in a common face, and $h(u) \geq h(v)$. As discussed above, graph path P' provides an SGDP of the same length from s to v on the terrain, which will complete the proof (further details below).

Let $\sigma_P = (s = p_0, p_1, p_2, \ldots, p_k, v = p_{k+1})$ be an ordered subsequence of the nodes in P such that (a) for each $i \in [0, k]$, the segments in-between p_i and p_{i+1} lie in a common face f_i, and (b) for each $i \in [0, k-1]$, the segment exiting from p_{i+1} does *not* lie in f_i. Note that p_i and p_{i+1} are two different boundary points of face f_i for all $i \in [0, k-1]$, and p_k and p_{k+1} are two different points of face f_k (p_{k+1} can be an interior point of f_k). For all $i \in [0, k]$, the part of P between p_i and p_{i+1} remains in f_i. Let e_i be an edge of the terrain through p_i for all $i \in [1, k]$ (e_i can be any edge through p_i if p_i is a vertex).

We construct a node sequence $P' = (s = p'_0, p'_1, p'_2, \ldots, p'_k, v = p'_{k+1})$ as follows: for each $i \in [1, k]$, let $p'_i = p_i$ if p_i is a vertex of the terrain; otherwise, let p'_i be the nearest point from p_i in $V \cap e_i$ such that $h(p'_i) \geq h(p_i)$. We will first prove that this node sequence defines a path in G_v.

Lemma 5. *For all $i \in [0, k]$, $h(p'_i) \geq h(p'_{i+1})$.*

Proof. We first claim that $h(p'_i) \geq h(p_{i+1})$. This claim follows from the facts that $h(p'_i) \geq h(p_i)$ by the definition of p'_i, and $h(p_i) \geq h(p_{i+1})$ as P is descending. Now consider the following two cases:

Case 1: $p'_{i+1} = p_{i+1}$ or e_{i+1} is a level edge. In this case, $h(p'_{i+1}) = h(p_{i+1})$. It follows from the inequality $h(p'_i) \geq h(p_{i+1})$ that $h(p'_i) \geq h(p'_{i+1})$.

Case 2: $p'_{i+1} \neq p_{i+1}$ and e_{i+1} is a non-level edge. In this case, there is either one or no point in e_{i+1} at any particular height. Let p''_{i+1} be the point in e_{i+1} such that $h(p''_{i+1}) = h(p'_i)$, or if no such point exists, let p''_{i+1} be the upper vertex of e_{i+1}. In the latter case, we can infer from the inequality $h(p'_i) \geq h(p_{i+1})$ that $h(p'_i) > h(p''_{i+1})$. Therefore we have $h(p'_i) \geq h(p''_{i+1})$ in both cases. Since $p''_{i+1} \in V \cap e_{i+1}$, the definition of p'_{i+1} implies that $h(p''_{i+1}) \geq h(p'_{i+1})$. So, $h(p'_i) \geq h(p'_{i+1})$.

Therefore, $h(p'_i) \geq h(p'_{i+1})$ for all $i \in [0, k]$. □

Lemma 6. *For all $i \in [0, k+1]$, p'_i is not a sharp vertex.*

Proof. None of $p'_0 = s$ and $p'_{k+1} = v$ are sharp vertices because both the segments sp_1 and $p_k v$ are gently descending. For each $i \in [1, k]$, if p'_i is a sharp vertex, then p'_i is either the unique topmost vertex or the unique bottommost vertex in all incident faces. Therefore, either $h(p'_{i-1}) < h(p'_i) > h(p'_{i+1})$, or $h(p'_{i-1}) > h(p'_i) < h(p'_{i+1})$. Both of these are impossible by Lemma 5. So p'_i is a not sharp vertex. □

Lemma 7. *Node sequence P' defines a path in G_v.*

Proof. It is sufficient to show that $p'_i p'_{i+1} \in E_v$ for all $i \in [0, k]$. For $i \in [0, k-1]$, since p'_i and p'_{i+1} are boundary points of face f_i by definition, $h(p'_i) \geq h(p'_{i+1})$ by Lemma 5, and neither p'_i nor p'_{i+1} is a sharp vertex by Lemma 6, it follows from the construction that $p'_i p'_{i+1} \in E \subseteq E_v$. The case $i = k$ is similar except that p'_k and $v = p'_{k+1}$ are points (i.e., not boundary points) in face f_k, and that $p'_k v \in E_v$. □

Lemma 8. *Our algorithm returns a $(1+\epsilon)$-SGDP.*

Proof. Let P and P' be respectively an ideal SGDP and a node sequence in G_v as described above. Our algorithm finds a shortest path P'' in G_v, which provides an SGDP of the same length. Since P' is a path in G_v (Lemma 7), the length of P'' is at most the length of P', and therefore it is sufficient to prove that the length of P' is at most $(1+\epsilon)$ times the length of P.

We first show that $\sum_{i=1}^{k} \|p_i p'_i\| < \frac{\epsilon h}{2}$. If $p_i \neq p'_i$, and e_i is a non-level edge, we have: $|h(p_i) - h(p'_i)| \leq \delta$ by construction, and $\frac{|h(p_i)-h(p'_i)|}{|p_i p'_i|} \geq \cos\theta$, which implies that $|p_i p'_i| \leq \delta \sec\theta$. If $p_i = p'_i$, or e_i is a level edge, $|p_i p'_i| \leq \delta \sec\theta$ in a trivial manner. Therefore, $\sum_{i=1}^{k} |p_i p'_i| \leq k\delta \sec\theta$. Since P is an ideal SGDP, $k < 2n$ (the number of faces), and hence, $\sum_{i=1}^{k} |p_i p'_i| < 2n\delta \sec\theta = \frac{\epsilon h \cos\psi}{2}$. Observation 1(iii) implies $\sum_{i=1}^{k} \|p_i p'_i\| \leq \sum_{i=1}^{k} (|p_i p'_i| \sec\psi) < \frac{\epsilon h \cos\psi \sec\psi}{2} = \frac{\epsilon h}{2}$.

The length of P' is equal to:

$$\begin{aligned}
\sum_{i=0}^{k} \|p'_i p'_{i+1}\| &\leq \sum_{i=0}^{k} \left(\|p'_i p_i\| + \|p_i p_{i+1}\| + \|p_{i+1} p'_{i+1}\|\right) \quad \text{(Observation 1(ii))} \\
&= \sum_{i=0}^{k} \|p_i p_{i+1}\| + 2\sum_{i=1}^{k} \|p_i p'_i\| \quad \text{(Observation 1(i))} \\
&< \sum_{i=0}^{k} \|p_i p_{i+1}\| + \epsilon h \ .
\end{aligned}$$

Assuming that P crosses at least one edge of the terrain (otherwise, both P' and P will have length $\|sv\|$), $h \leq \sum_{i=0}^{k} |p_i p_{i+1}| \leq \sum_{i=0}^{k} \|p_i p_{i+1}\|$ (Observation 1(iii)), and therefore, $\sum_{i=0}^{k} \|p'_i p'_{i+1}\| < (1+\epsilon) \sum_{i=0}^{k} \|p_i p_{i+1}\|$. So, the length of P' is at most $(1+\epsilon)$ times the length of P. □

Lemma 9. *Let $X = \frac{L}{h} \sec\theta \sec\psi$. Graph G has $O\left(\frac{n^2 X}{\epsilon}\right)$ nodes in total, and $O\left(\frac{nX}{\epsilon}\right)$ nodes along any edge of the terrain.*

Proof. (Idea) The proof is the same as that of Lemma 7 in Ahmed et al. [1], except that we use δ and X defined here. □

Theorem 1. *Let $X = \frac{L}{h} \sec\theta \sec\psi$. Given a vertex s, we can preprocess the terrain in $O\left(\frac{n^2 X}{\epsilon} \log\left(\frac{nX}{\epsilon}\right)\right)$ time after which we can determine a $(1+\epsilon)$-approximate SGDP from s to any query point v in: (i) $O(nd)$ time if v is a vertex or a Steiner point, and (ii) $O\left(n\left(d + \frac{X}{\epsilon}\right)\right)$ time otherwise.*

Proof. The approximation factor follows from Lemma 8.

The preprocessing time of our algorithm is the same as the running time of the Bushwhack algorithm, which is $O(|V| \log |V|) = O\left(\frac{n^2 X}{\epsilon} \log\left(\frac{nX}{\epsilon}\right)\right)$ by Lemma 9.

During the query phase, if v is a vertex or a Steiner point, the approximate path is in the tree T. Because the tree has height $O(n)$, it takes $O(n)$ time to

trace the path in the tree. Tracing the corresponding path in the terrain takes $O(nd)$ time by Lemma 3. The total query time is thus $O(nd)$ in this case. If v is neither a vertex nor a Steiner point, v is an interior point of a face or an edge of the terrain. The last intermediate node u on the path to v is a vertex or a Steiner point that lies on the boundary of a face containing v. If v is interior to a face [an edge], there are 3 [respectively 4] edges of the terrain on which u can lie. Thus there are $O\left(\frac{nX}{\epsilon}\right)$ choices for u by Lemma 9, and we try all of them to find the shortest approximate distance from s to v. Finally tracing the corresponding path in the terrain takes $O(nd)$ time by Lemma 3. The total query time in this case is $O\left(\frac{nX}{\epsilon}\right) + O(nd) = O\left(n\left(d + \frac{X}{\epsilon}\right)\right)$. □

Corollary 1. *If the answer to a query is the length of an SGDP (rather than the SGDP itself), the query times for Cases (i) and (ii) of Theorem 1 become* $O(1)$ *and* $O\left(\frac{nX}{\epsilon}\right)$ *respectively.*

6 Approximation Using Non-uniform Steiner Points

Our second approximation algorithm differs from the first one only in the way Steiner points are placed. We now place Steiner points in two phases. First, on every edge $e = v_1v_2$ we place Steiner points at points $p \in e$ such that $|pq| = \delta_1(1+\delta_2)^i$ for $q \in \{v_1, v_2\}$ and $i \in \{0, 1, 2, \ldots\}$, where $\delta_1 = \frac{\epsilon h}{6n}\cos\psi$ and $\delta_2 = \frac{\epsilon h}{6L}\cos\psi$. In the second phase we slice the terrain with a level plane through every Phase 1 Steiner point and every vertex, and add Steiner points at the points where these planes intersect the terrain.

Theorem 2. *Let* $X' = \frac{L}{h}\sec\psi$. *Given a vertex* s, *we can preprocess the terrain in* $O\left(\frac{n^2X'}{\epsilon}\log^2(\frac{nX'}{\epsilon})\right)$ *time after which we can determine a* $(1+\epsilon)$*-approximate SGDP from* s *to any point* v *in: (i)* $O(nd)$ *time if* v *is a vertex or a Steiner point, and (ii)* $O\left(nd + \frac{nX'}{\epsilon}\log(\frac{nX'}{\epsilon})\right)$ *time otherwise.*

Proof. (Idea) The proof is similar to that of Theorem 1, but we use slightly different versions of Lemmas 8 and 9, as is done in the proof of Theorem 2 in Ahmed et al. [1]. □

Corollary 2. *If the answer to a query is the length of an SGDP, the query times for Cases (i) and (ii) of Theorem 2 become* $O(1)$ *and* $O\left(\frac{nX'}{\epsilon}\log(\frac{nX'}{\epsilon})\right)$ *respectively.*

References

1. Ahmed, M., Das, S., Lodha, S., Lubiw, A., Maheshwari, A., Roy, S.: Approximation algorithms for shortest descending paths in terrains. CoRR, 0805.1401v1 [cs.CG] (May 2008)
2. Ahmed, M., Lubiw, A.: Shortest descending paths through given faces. In: Proceedings of the 18th Canadian Conference on Computational Geometry, pp. 35–38 (August 2006); accepted for publication in CCCG 2006 Special Issue of Computational Geometry: Theory and Applications

3. Ahmed, M., Lubiw, A.: An approximation algorithm for shortest descending paths. CoRR, 0705.1364v1 [cs.CG] (May 2007)
4. Ahmed, M., Lubiw, A.: Properties of shortest descending paths. In: The 17th Fall Workshop on Computational and Combinatorial Geometry, Hawthorne, New York (November 2007); Extended abstract
5. Ahmed, M., Lubiw, A.: Shortest anisotropic paths with few bends is NP-complete. In: The 18th Fall Workshop on Computational Geometry: Abstracts, Troy, New York, pp. 28–29 (October 2008)
6. Aleksandrov, L., Maheshwari, A., Sack, J.-R.: Determining approximate shortest paths on weighted polyhedral surfaces. J. ACM 52(1), 25–53 (2005)
7. Cheng, S.-W., Na, H.-S., Vigneron, A., Wang, Y.: Approximate shortest paths in anisotropic regions. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, Philadelphia, PA, USA, pp. 766–774. Society for Industrial and Applied Mathematics (2007)
8. de Berg, M., van Kreveld, M.J.: Trekking in the Alps without freezing or getting tired. Algorithmica 18(3), 306–323 (1997)
9. de Berg, M., van Kreveld, M.J., Overmars, M., Cheong, O.: Computational Geometry: Algorithms and Applications, 2nd edn. Springer, Berlin (2000)
10. Lanthier, M., Maheshwari, A., Sack, J.-R.: Shortest anisotropic paths on terrains. In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 524–533. Springer, Heidelberg (1999)
11. Mitchell, J.S.B., Papadimitriou, C.H.: The weighted region problem: finding shortest paths through a weighted planar subdivision. J. ACM 38(1), 18–73 (1991)
12. Rowe, N.C., Ross, R.S.: Optimal grid-free path planning across arbitrarily-contoured terrain with anisotropic friction and gravity effects. IEEE Trans. Robot. Autom. 6(5), 540–553 (1990)
13. Roy, S., Das, S., Nandy, S.C.: Shortest monotone descent path problem in polyhedral terrain. Comput. Geom. Theory Appl. 37(2), 115–133 (2007)
14. Roy, S., Lodha, S., Das, S., Maheshwari, A.: Approximate shortest descent path on a terrain. In: Proceedings of the 19th Canadian Conference on Computational Geometry, pp. 189–192 (August 2007)
15. Sun, Z., Bu, T.-M.: On discretization methods for approximating optimal paths in regions with direction-dependent costs. Inform. Process. Lett. 97(4), 146–152 (2006)
16. Sun, Z., Reif, J.H.: BUSHWHACK: An approximation algorithm for minimal paths through pseudo-euclidean spaces. In: Eades, P., Takaoka, T. (eds.) ISAAC 2001. LNCS, vol. 2223, pp. 160–171. Springer, Heidelberg (2001)
17. Sun, Z., Reif, J.H.: On finding energy-minimizing paths on terrains. IEEE Transactions on Robotics 21(1), 102–114 (2005)
18. Sun, Z., Reif, J.H.: On finding approximate optimal paths in weighted regions. J. Algorithms 58(1), 1–32 (2006)

All Farthest Neighbors in the Presence of Highways and Obstacles*

Sang Won Bae[1], Matias Korman[2], and Takeshi Tokuyama[2]

[1] Division of Computer Science, KAIST, Korea
swbae@tclab.kaist.ac.kr
[2] Graduate School of Information Sciences, Tohoku University, Sendai, 980-8579 Japan
{mati,tokuyama}@dais.is.tohoku.ac.jp

Abstract. We consider the problem of computing all farthest neighbors (and the diameter) of a given set of n points in the presence of highways and obstacles in the plane. When traveling on the plane, travelers may use highways for faster movement and must avoid all obstacles. We present an efficient solution to this problem based on knowledge from earlier research on shortest path computation. Our algorithms run in $O(nm(\log m + \log^2 n))$ time using $O(m + n)$ space, where the m is the combinatorial complexity of the environment consisting of highways and obstacles.

1 Introduction

Given a set S of n points in a space with a metric d, the *farthest neighbor* $f(s)$ of $s \in S$ is defined by $\arg\max_{p \in S} d(s, p)$. The value $\max_{s \in S} d(s, f(s))$ is called the *diameter* of S. The problem of computing the farthest point for every point of S is called the *all farthest neighbors problem* (AFNP).

The AFNP and diameter computation are classical problems in computational geometry, and it is clear that the diameter problem can be solved once we solve the AFNP. If we can evaluate the distance between given two points in $O(1)$ time, we can compute all farthest neighbors in $O(n^2)$ time in a naïve way. Also, if the distance is defined by the shortest path distance of a graph with n vertices, the all farthest neighbors problem can be solved by first solving the all-pairs-shortest-path problem and spending $O(n^2)$ additional time.

However, we can do better in several cases: If the space is the Euclidean plane, the diameter and also AFNP are computed in $O(n \log n)$ time by constructing the farthest Voronoi diagram.The diameter problem in the 3-dimensional space has also been well-studied, and can be solved in $O(n \log n)$ time [9].

In this paper, we consider the AFNP in metric spaces modeling urban transportation systems. The underlying space of our environment is the plane with

* Work by S.W. Bae was supported by the Brain Korea 21 Project. Work by M. Korman was supported by MEXT scolarship and CERIES GCOE project, MEXT Japan.

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 71–82, 2009.

the L_1 metric, also known as the Manhattan metric. In addition, we are given a set $\mathcal{O}$ of obstacles and a set $\mathcal{H}$ of vertical and horizontal highways in the plane. We assume that a traveler can use the highways to move faster and that (s)he should avoid the obstacles during traveling. In this situation, we want to compute paths of shortest travel time. This setting well reflects a city scene with a transportation system: consider a road system in a city. Areas that cars cannot trespass are considered as obstacles. We drive on avenues and streets, which are vertical and horizontal, respectively. We formally define this environment, shortest travel time paths, and a metric d induced by shortest paths, called the *generalized city metric*.

The generalized city metric space is a natural extension of two known realistic models; the plane with either polygonal obstacles or with highways. Shortest path computation under either of the models has been extensively studied: in the presence of polygonal obstacles, the optimal algorithm was introduced by Mitchell [10,11] applying the *continuous Dijkstra method*, and is extended to compute the Voronoi diagram with respect to the shortest path distance. Tha case in which only highways exist (named the *city metric*) was considered by Aichholzer *et al.* [3]. Also, the shortest path and the Voronoi diagram in this case can be computed in optimal time [5].

However, in the literature, only few results about farthest neighbors in the presence of highways or obstacles can be found. A brute-force way to solve the AFNP examines all the pairs of given points, spending quadratic time in the number of given points. As in the Euclidean case, the farthest Voronoi diagram can be used to solve the AFNP efficiently: once the diagram is computed, the farthest neighbor $f(s)$ of each $s \in S$ can be found in logarithmic time. When the obstacles are all axis-parallel rectangles on the L_1 plane, an implicit structure of the farthest Voronoi diagram can be constructed in $O(mn \log(m+n))$ time, where m is the total complexity of the given obstacles [6]. In the presence of m highways and no obstacle, an $O(nm \log^3(n+m))$ time algorithm for computing the farthest Voronoi diagram has been recently introduced [4]. Memory space for any algorithm that uses the FVD is $\Omega(nm)$ since it is known that the diagram can have $\Omega(nm)$ complexity [4]. To the best of our knowledge, there is no known algorithm to solve the AFNP in the presence of highways and obstacles, simultaneously.

In this paper, we adopt a different approach using the *shortest path map* and *segment dragging queries*, and solve the AFNP without building the farthest Voronoi diagram. Our algorithm runs in $O(nm(\log m + \log^2 n))$ time and uses linear space (i.e.: $O(n+m)$), which improves the previously best known $O(nm \log^3(n+m))$ running time for the city metric, reduces the total required memory and works in a more general environment.

We mention an application of our AFNP algorithm: the diameter is an important criterion to measure efficiency of a transportation system, and can be used for extending any such system. Ahn *et al.* [2] and Cardinal *et al.* [7] considered the problem of finding the optimal location of a highway in order to minimize the diameter of a given set of points in an empty transportation system (i.e.:

an environment with no highways or obstacles). Our AFNP algorithm combined with some optimization techniques allow us to generalize such results to locate a highway in a in a city with existing highways and obstacles (full details of such method are explained in a companion paper).

The precise definition of the generalized city metric and some preliminaries are given in Section 2. We first present an algorithm to solve the AFNP for the city metric in Section 3, and then consider the generalized city metric in Section 4. Finally, Section 5 concludes this paper with some remarks and open issues.

2 Preliminaries

In this section we introduce the formal definition of the generalized city metric and the Shortest Path Maps and Segment Dragging Queries, key elements of our AFNP algorithm.

2.1 City Metric and Generalized City Metric

A *highway* is a facility supporting faster movement. We represent a highway by a line segment on the plane. Each highway h is associated with a *speed* $\nu(h) > 1$. A traveler moves at speed $\nu(h)$ when moving along h, while the speed when not using any highway is 1. We deal with two kinds of highways; one is called a *freeway* allowing access through any point on it, and the other a *turnpike* accessible only through its two endpoints. An *obstacle* is a region that a traveler is not allowed to cross, and represented by a simple polygon.

Let $\mathcal{H}$ be a set of highways and $\mathcal{O}$ a set of disjoint obstacles in the L_1 plane. A feasible path is a rectilinear path avoiding all obstacles in $\mathcal{O}$. We let $\mathcal{F} := \mathbb{R}^2 \backslash \bigcup \mathcal{O}$ be the *free space*. For any feasible path π between two points in $\mathcal{F}$, we can measure the travel time of π since we know the speed of movement at any point on π and the length of any piece of π. We call a feasible path π connecting $s, t \in \mathcal{F}$ a *shortest (travel time) path* if π minimizes the travel time between s and t among all feasible s-t paths. We let $d(s, t)$ be the travel time of a shortest path between s and t. Then, d is a metric on $\mathcal{F}$ since d is based on shortest paths on $\mathcal{F}$. We call d the *generalized city metric induced by* $\mathcal{H}$ *and* $\mathcal{O}$. Through this paper, let m be the combinatorial complexity of the set of highways and obstacles.

One can find earlier research related to the generalized city metric: If $\mathcal{H} = \emptyset$, we have polygonal obstacles in the L_1 plane [10]. The case $\mathcal{O} = \emptyset$ and all highways are freeways was considered by Aichholzer *et al.* [3] and Bae *et al.* [5] (this metric is called the *city metric*). The case where we are given only turnpikes was considered by Ostrovsky-Berman [12]. Thus, by the generalized city metric, we deal with all the three kinds of objects in one environment. For simplicity in the explanation, we deal with only axis-parallel highways of equal speed ν but allow obstacle edges to have any orientation. Note that one can allow a constant number of speeds for the highways without much modification of our algorithms.

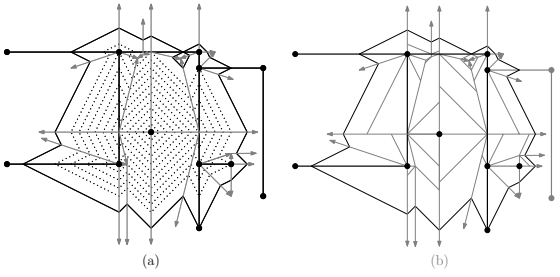


Fig. 1. (a) The wavefront propagation in the presence of freeways of speed 2. The wavefront $W(\delta)$ is divided into wavelets dragged by track rays (gray arrows). (b) The (partial) resulting shortest path map.

2.2 The Continuous Dijkstra Method and Shortest Path Maps

The *continuous Dijkstra method* is a conceptual algorithmic method to compute shortest paths from a given source $s \in \mathbb{R}^2$ to *every* other point. The output of the continuous Dijkstra method is called the *shortest path map* of a given source point $s \in \mathbb{R}^2$. For a fixed point s, a shortest path map for s is a subdivision of the plane into cells each of which is a set of points from which shortest paths to s are combinatorially equivalent. An implementation of the continuous Dijkstra method simulates the *wavefront propagation* from s: the *wavefront* $W(\delta)$ is defined as the set $\{p \in \mathbb{R}^2 \mid d(p, s) = \delta\}$ for any positive δ. In particular, under the L_1 metric, the wavefront $W(\delta)$ is expressed by a set of line segments, called *wavelets*.

In this environment, the wavelets have eight possible inclinations: $\pi/4$, $3\pi/4$, β, $\pi - \beta$, $\pi/2 + \beta$ and $\pi/2 - \beta$, where $\beta = \tan^{-1} 1/\nu$ [5]. Each wavelet is propagated in a certain direction along two *track rays* as δ increases. Each track ray of a wavelet is the locus of the endpoints of the wavelet and traces an edge of the resulting shortest path map. Figure 1 illustrates the wavefront propagation and how related the wavelets are to the resulting shortest path map. For more details, we refer to several technical papers implementing the continuous Dijkstra method [5,10,11,12].

2.3 Segment Dragging Queries

The segment dragging query problem is formulated as follows: Determine the next point "hit" by a query segment qq' when it is "dragged" along two rays. More formally, given three orientations θ, ϕ_l, and ϕ_r, we want to preprocess a set S of points for determining the first point hit by a query segment $q_l q_r$ of orientation θ when q_l slides in direction ϕ_l and q_r in direction ϕ_r in such a way that the segment being dragged remains parallel to θ. We call the locus of the endpoints of the dragged segment the *track rays*.

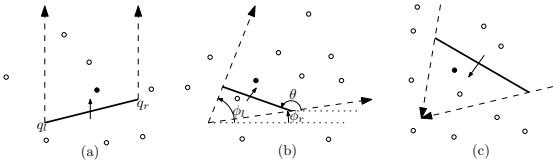


Fig. 2. Three types of segment dragging: parallel, out of a corner and into a corner

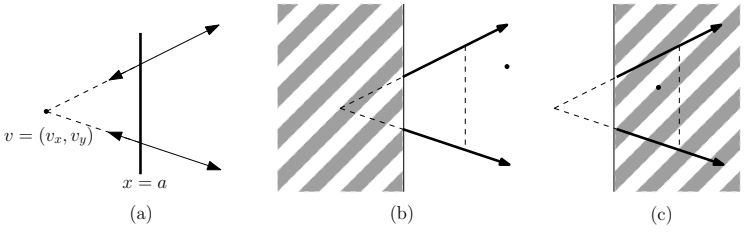


Fig. 3. (a) Queries $out(a, C)$ and $into(a, C)$ (b)(c) Oracle construction: after performing an out of the corner query we know in which side our solution lies. Depending on whether or not the reported point is inside the query range, we can discard either the right or the left halfplane, respectively.

This problem was first considered by Chazelle [8] in the particular case in which track rays were parallel. The generalization to three different types (parallel, dragging *out* of a corner and dragging *into* a corner, see Figure 2) was considered in [10]. For simplicity, we call each kind of query type (a), (b) or (c).

The first two cases can be handled in optimal time and space:

Lemma 1 (Chazelle [8] and Mitchell [10]). *One can preprocess a set S of n points into a data structure of $O(n)$ size in $O(n \log n)$ time that answers type (a) and (b) segment dragging queries in $O(\log n)$ time.*

To the authors' knowledge, no optimal way to handle type (c) queries is known. Simple techniques for each particular problem have been used in the literature to avoid those queries [10,5]. Here, we introduce a simple way to handle them with an additional logarithmic factor in the query time:

Lemma 2. *One can preprocess a set S of n points into a data structure of size $O(n)$ in $O(n \log n)$ time which answers into a corner segment dragging queries in $O(\log^2 n)$ time.*

Proof. Let θ, ϕ_l, and ϕ_r be three line orientations. Let C be the wedge with apex $v = (v_x, v_y)$ and bounding rays of orientations ϕ_l and ϕ_r. Let $out(a, C)$ and $into(a, C)$ be the dragging out of and into dragging queries with respect to the vertical query segment $C \cap \{x \geq a\}$ and $C \cap \{x \leq a\}$, respectively, (see Figure 3(a)). Without loss of generality, we can assume that the query segment is vertical (i.e.: $\theta = \pi/2$) and $v_x < a$.

We will give an oracle that, given $x_0 \in \mathbb{R}$, computes whether or not the point to report (if it exists) lies in the halfplane $\{x \leq x_0\}$ in $O(\log n)$ time. Combining the oracle with a binary search (in the x-coordinates of the points in S) allow us to answer type-(c) queries in $O(\log^2 n)$ time. Note that the only preprocessing needed is the sorted list of points and the structure to allow type-(b) range queries.

The oracle performs a single type (b) dragging query $out(x_0, C)$. If a point whose x coordinate is less than a is found (i.e.: the reported point is inside the wedge $C \cup \{x \geq x_0\}$), then the solution of $into(a, C)$ has an x-coordinate higher than x_0. Otherwise, the solution of $into(a, C)$ has an x-value smaller than x_0. (See Figure 3(b) and (c).)

3 AFNP in the Presence of Freeways

In this section, we first consider the AFNP under the city metric induced by a set of freeways. Let $S \subset \mathbb{R}^2$ be a set of n points and $\mathcal{H}$ be the set of m axis aligned freeways of speed $\nu > 1$. Bae *et al.* [5] obtained the first optimal algorithm for computing the shortest path map of a fixed point s by applying the continuous Dijkstra method:

Lemma 3 (Bae *et al.* [5]). *Given m freeways, the shortest path map $\mathcal{SPM}_s$ for a given source $s \in \mathbb{R}^2$ under the city metric can be computed in $O(m \log m)$ time with $O(m)$ space.*

They also showed several properties of the shortest path map $\mathcal{SPM}_s$ obtained by their algorithm. We can rephrase them as follows:

Lemma 4 (Bae *et al.* [5]). *Let Θ be the set of all possible inclinations for wavelets and Φ be the set of all possible orientations of track rays of wavelets under a city metric. Then, we have $|\Theta| = 6$, $|\Phi| \leq 24$. Moreover, the orientation of any edge of $\mathcal{SPM}_s$ is in Φ and each cell of $\mathcal{SPM}_s$ is x-monotone or y-monotone.*

These properties of $\mathcal{SPM}_s$ allow us to find the farthest neighbor $f(s)$ of $s \in S$ efficiently:

Theorem 1. *Given m axis parallel freeways of equal speed and n points, all farthest neighbors and the diameter among n points under the city metric can be computed in $O(nm(\log m + \log^2 n))$ time using $O(n)$ space.*

Proof. The pseudo-code of our algorithm can be seen in Figure 4: after preprocessing P to allow segment dragging queries for $\theta \in \Theta$ and $\phi_l, \phi_r \in \Phi$, we compute $f(s)$ for each $s \in S$ independently as follows: we construct the shortest path map $\mathcal{SPM}_s$ using the algorithm of Bae *et al.* [5]. Also, we divide each cell C of $\mathcal{SPM}_s$ into trapezoids: if C is x-monotone (resp. y-monotone) we cut C by vertical (resp. horizontal) lines through each vertex on the boundary of C. Consider the resulting subdivision: each cell is a trapezoid whose edges have inclinations in Φ by Lemma 4 and our construction of the trapezoids.

Preprocess S for segment dragging queries
for each $s \in S$ **do**
  compute $\mathcal{SPM}_s$ and decompose it into trapezoids
  for each trapezoid τ **do**
    Find $f_\tau(s) = \arg\max_{p \in S \cap \tau} d(s,p)$ by segment dragging queries
  end for
end for

Fig. 4. AFNP pseudo-code

Let τ be any such trapezoid; without loss of generality we can assume that τ comes from an x-monotone cell C of $\mathcal{SPM}_s$. Now, we describe how to find the point $f_\tau(s) \in S \cap \tau$ that maximizes the distance $\max_{p \in S \cap \tau} d(s,p)$: consider the set $W_\tau(\delta) := \{q \in \tau \mid d(s,q) = \delta\}$ (that is, the intersection of the wavefront $W(\delta)$ and τ): since τ is completely included in a cell C of $\mathcal{SPM}_s$, the shortest path topology to s is the same for any point $q \in \tau$. Thus $W_\tau(\delta)$ is either a line segment or an empty set for any $\delta > 0$. Moreover, if $W_\tau(\delta)$ is a segment, its slope must be in Θ, since it is a portion of a wavelet by Lemma 4.

Now, consider sweeping τ by $W_\tau(\delta)$ as δ decreases. Let $p_\tau \in S \cap \tau$ be the first point hit by $W_\tau(\delta)$, if $S \cap \tau \neq \emptyset$. Any other point $q \in S \cap \tau$ will have *smaller* distance to s and therefore can be ignored, and thus $f_\tau(s) = p_\tau$. We sweep τ by three consecutive segment dragging queries: first find the v of τ that is farthest away from s. We then perform a type (b) dragging query that originates from that vertex v. The point reported by the query either is p_τ or lies out of τ. If the point reported is outsie τ, we perform a type (a) segment dragging query along the vertical sides of τ. Similarly, we perform a type (c) dragging query if no point has been found in the second query. Figure 5 shows these three consecutive segment dragging queries. We repeat this procedure for all trapezoids τ and then use that $f(s) = \max_\tau f_\tau(s)$.

Preprocessing takes $O(n \log n)$ time and needs $O(n)$ space since Θ and Φ are of constant size by Lemma 4. Building $\mathcal{SPM}_s$ (and its further decomposition into trapezoids) can be done $O(m \log m)$ time and $O(m)$ space using the algorithm of Bae *et al.* [5]. At most three segment dragging queries are needed to report $f_\tau(s)$,

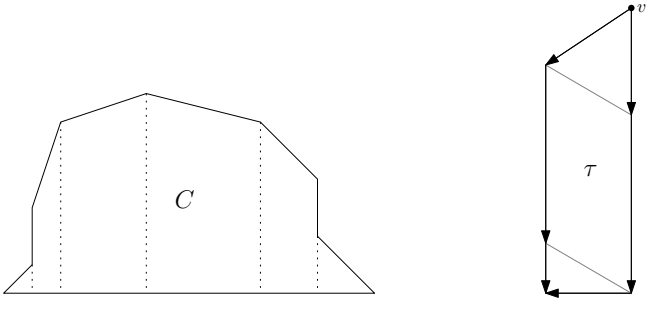


Fig. 5. A cell C of $\mathcal{SPM}_p$ is divided into trapezoids and each trapezoid τ is swept by three consecutive queries

and iterating for all the trapezoids of the $\mathcal{SPM}_s$ takes $O(m \log^2 n)$ time in total. Since we do this procedure for all $s \in S$, the total time is $O(nm(\log m + \log^2 n))$.

4 Algorithm for the Generalized City Metric

In this section we will generalize the previous algorithm to work in a generalized city metric. First we will consider the case in which only obstacles exist and then focus in the general case.

4.1 All Farthest Neighbors in the Presence of Obstacles

For simplicity in the explanation, we assume the set $\mathcal{O}$ are mutually disjoint simple polygons with total complexity m Throughout this section, we let V be the set of vertices of obstacles in $\mathcal{O}$. Mitchell [10] gave an optimal algorithm to construct $\mathcal{SPM}_s$ in the presence of obstacles on the L_1 plane:

Lemma 5 (Mitchell [10]). *Given a set $\mathcal{O}$ of polygonal obstacles, the $\mathcal{SPM}_s$ of a source $s \in \mathcal{F}$ under d induced by $\mathcal{O}$ on the L_1 plane can be computed in $O(m \log m)$ time and $O(m)$ space. Moreover, $\mathcal{SPM}_s$ fulfills the following properties:*

- *Any obstacle vertex $v \in V$ is a vertex of the map $\mathcal{SPM}_s$.*
- *Each edge e of $\mathcal{SPM}_s$ is a portion of an edge of an obstacle in $\mathcal{O}$ or has slope $\phi \in \Phi$, where Φ is defined in Lemma 4.*
- *Each cell of $\mathcal{SPM}_s$ is x-monotone or y-monotone.*
- *Let v be a vertex on the boundary of a cell C of $\mathcal{SPM}_s$ minimizing $d(v, s)$ among points in C. Then, one of two incident edges to v is vertical or horizontal.*

This algorithm is also based on the continuous Dijkstra paradigm. In the presence of obstacles, the inclinations of wavelets are $\pi/4$ or $3\pi/4$. However, in this case we do not have a constant bound on the number of directions for the track rays of wavelets due to the fact that obstacles have m edges with free orientations. Mitchell adapted the segment dragging query method in order to cope with the growth of directions; as before, track directions are fixed but if the dragged segment encounters an obstacle edge, it changes the colliding track ray to slide along that edge (see Figure 6).

Lemma 6 (Mitchell [10]). *Given a set $\mathcal{O}$ of obstacles with m vertices V and a set S of n points in the free space, one can preprocess $\mathcal{O}$ and S into a data structure of $O(m + n)$ size in $O((m + n) \log(m + n))$ time that answers the segment dragging query of type (a) or (b) in $O(\log(m + n))$ time to report the point in $V \cup S$ hit first by the dragged segment, if any.*

Using this modified segment dragging query, we can solve the AFNP using the same approach as before:

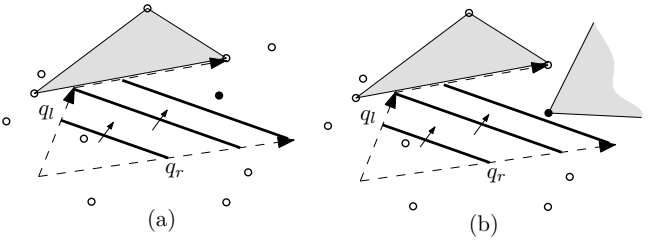


Fig. 6. Segment dragging queries in the presence of obstacles. The small circles depict candidate points to be reported (the dark points are the ones to report by the query segment $q_l q_r$). The reported point is either (a) a point in S or (b) an obstacle vertex in V.

Theorem 2. *Given obstacles $\mathcal{O}$ with m vertices V and a set S of n points in the free space $\mathcal{F}$, all farthest neighbors and the diameter among S can be computed in $O(nm(\log m + \log^2 n))$ time with $O(n + m)$ space.*

Proof. First, we preprocess $\mathcal{O}$ and S for segment dragging queries of type (a) and (b) by Lemma 6, and preprocess S for type (c) queries by Lemma 2. For $s \in S$, we compute the shortest path map $\mathcal{SPM}_s$ in $O(m \log m)$ time and divide each cell C of $\mathcal{SPM}_s$ into trapezoids by the third and the fourth properties in Lemma 5: let v be the nearest vertex to s among all vertices on the boundary of C. If there is a vertical (or horizontal) edge incident to v, we cut C by horizontal (resp. vertical) lines through each vertex on the boundary of C. As done in Section 3, we find the farthest point $f_\tau(s) \in S \cap \tau$ to s among the obstacles $\mathcal{O}$ for each such trapezoid τ.

Consider a trapezoid τ. Without loss of generality, we assume that τ comes from a cell of $\mathcal{SPM}_s$ cut by vertical lines. As in the freeway case we perform three consecutive queries of type (b), (a) and (c) in order as in Figure 5. However, each segment dragging query may end with an obstacle vertex $v \in V$ before encountering a point $p \in S$ or a vertex of τ since we have preprocessed S plus V for segment dragging queries in the presence of obstacles. In that case, we ignore v and do another segment dragging after v. Since the first two queries (of type (a) and (b)) sweep the interior of τ, vertices $v \in V$ encountered during these two queries always lie on the boundary of τ by the first property in Lemma 5. Thus, these two queries in τ are performed in time $O(k \log(m + n))$, where k is the number of obstacle vertices on the boundary of τ. Observe that the trapezoidal decomposition of $\mathcal{SPM}_s$ is indeed a complete subdivision of $\mathcal{F}$. This implies that summing k for all trapezoids τ is at most twice the number of obstacle vertices. Therefore, the cost of the first two queries is bounded by $O(m \log(m + n))$.

The query of type (c) is performed at last. By the fourth property of Lemma 5 and our construction of trapezoids, one track ray is horizontal and the other is in Φ. Since we have processed only S for segment dragging queries of type (c), during this query we do not encounter any obstacle vertex.

4.2 Combining Freeways, Turnpikes and Obstacles

We are now ready for combining all three kinds of transportation objects. The combination of freeways and obstacles was considered by Bae *et al.* [5] in the $\mathcal{SPM}$ computation: the shortest path map can be computed in $O(m \log m)$ time where freeways and obstacles with complexity m are given. Thus, Lemma 3 extends to the combined environment by freeways and obstacles. Moreover, the algorithm can in general compute the Voronoi diagram of k *weighted points* in $O((m+k)\log(m+k))$ time and $O(m+k)$ space [5]. (The distance of any $q \in \mathbb{R}^2$ to a weighted point p is measured as $d(q,p) + w(p)$, where $w(p)$ denotes the weight of p). Furthermore, the resulting diagram has information about shortest paths to the nearest point in such a way that each region is subdivided into cells, where all the points in each cell have the combinatorially equivalent shortest paths; That is, the resulting diagram is a multi-source shortest path map.

Lemma 7. *The $\mathcal{SPM}_s$ for any $s \in \mathcal{F}$ under the generalized city metric induced by a set $\mathcal{H}$ of highways and a set $\mathcal{O}$ of obstacles can be computed in $O(m \log m)$ time using $O(m)$ space.*

Proof. Let $\mathcal{H}_{\text{free}} \subseteq \mathcal{H}$ be the set of freeways in $\mathcal{H}$, and d' be the generalized city metric induced by the freeways $\mathcal{H}_{\text{free}}$ and the obstacles $\mathcal{O}$. We fix a source $s \in \mathcal{F}$. Let P be the set of all endpoints of the turnpikes in $\mathcal{H}_{free}$, and $w(p) := d(p,s)$ be the *weight* of each $p \in P$. Also, let $P_s := P \cup \{s\}$ and $w(s) = 0$. Now, consider the Voronoi diagram $\mathcal{V}_{d'}(P_s)$ of weighted points P_s under d'.

We show that $\mathcal{V}_{d'}(P_s)$ coincides with a shortest path map $\mathcal{SPM}_s$ for s under d. Take any shortest path π from $a \in \mathcal{F}$ to s. If π uses no turnpike, we simply have $d(a,s) = d'(a,s)$. Otherwise, let $p \in P$ be the first entrance of a turnpike used by π. Then, we have $d(a,s) = d(a,p) + d(p,s) = d'(a,p) + d(p,s) = d'(a,p) + w(p)$. Hence, in general, $d(a,s) = d'(a,p) + w(p)$ for some $p \in P_s$. For any point a in the Voronoi region of $p \in P_s$ and any other $q \in P_s$, we have $d'(a,p) + w(p) \leq d'(a,q) + w(q)$ and thus $d(a,s) = d'(a,p) + w(p)$.

Therefore, we are done by computing $\mathcal{V}_{d'}(P_s)$. However, we do not know the value $w(p) = d(p,s)$ at the beginning. Here, we introduce a trick to resolve this problem by *lazy evaluation*. The algorithm computing $\mathcal{V}_{d'}(P_s)$ also applies the continuous Dijkstra method, and simulates the wavefront propagation, where the wavefront $W(\delta)$ is defined as $W(\delta) := \{a \in \mathcal{F} \mid \min_{p \in P_s}\{d'(a,p) + w(p)\} = \delta\}$. We let $P_s(\delta) := \{p \in P_s \mid d(p,s) \leq \delta\}$ and $W'(\delta) := \{a \in \mathcal{F} \mid \min_{p \in P_s(\delta)}\{d'(a,p) + d(p,s)\} = \delta\}$. Observe that $W(\delta) = W'(\delta)$ and $d(a,p) = \min_{p \in P_s(\delta)}\{d'(a,p) + d(p,s)\}$ for any $a \in \mathcal{F}$ by the above argument.

When $\delta < d(p,s)$, no wavelets from p is propagated out. We do the following when the wavefront $W'(\delta)$ hits an endpoint p of a turnpike h whose other endpoint is p': if p has not been assigned its weight, we assign the weight $w(p) = \delta$. Similarly, if p' has not been assigned its weight, we assign the weight $w(p') = \delta + \frac{l}{\nu(h)}$, where l and $\nu(h)$ are the length and the speed of h, respectively. Note that the value of $w(p')$ might not be the same as $d(p',s)$ but we have $w(p') \geq d(p',s)$. If $w(p') < d(p',s)$, the wavelet $W'(\delta')$ will hit p' at $\delta' = d(p',s)$ before $w(p')$ thus the wrong weight assignement will be detected. Note that this

corresponds to the case in which highway h can be ignored (i.e.: the Voronoi regions of p and p' are empty, therefore, we can ignore p'. Hence, by our lazy evaluation of the weight, whether it is correct or not, the wavefront $W(\delta)$ is maintained correctly and the algorithm builds $\mathcal{V}_{d'}(P_s)$ properly.

As shown in the proof of Lemma 7, the shortest path map $\mathcal{SPM}_s$ under d coincides with a Voronoi diagram in the presence of freeways and obstacles only. Hence, $\mathcal{SPM}_s$ inherits the properties of shortest path maps shown in the previous sections:

Lemma 8. *The $\mathcal{SPM}_s$ computed by the algorithm in the proof of Lemma 7 fulfills the following properties:*

- *Any obstacle vertex $v \in V$ is a vertex of $\mathcal{SPM}_s$.*
- *Each edge e of $\mathcal{SPM}_s$ is a portion of an edge of an obstacle in $\mathcal{O}$ or has slope $\phi \in \Phi$, where Φ is defined in Lemma 4.*
- *Each cell C of $\mathcal{SPM}_s$ is x-monotone or y-monotone.*
- *Let C be a cell of $\mathcal{SPM}_s$ and v be a vertex on the boundary of C minimizing $d(v, s)$ among points in C. Then, one of two incident edges to v is vertical (or horizontal) and C is y-monotone (resp. x-monotone).*

Finally, we can prove the general case:

Theorem 3. *All farthest neighbors and the diameter among n points in the free space $\mathcal{F}$ under the generalized city metric induced by a set of highways and obstacles can be computed in $O(nm(\log m + \log^2 n))$ time using $O(n + m)$ space.*

Proof. The algorithm is almost the same as those introduced in Theorems 1 and 2. Let S be the given set of n points. We preprocess S and the obstacle vertices V for segment dragging queries as in Lemmas 6 and 2. Then, we compute the shortest path map $\mathcal{SPM}_s$ for $s \in S$ using Lemma 7 and subdivide each cell of $\mathcal{SPM}_s$ into trapezoids as done in Theorems 1 and 2.For each trapezoid τ, we perform consecutive segment dragging queries to find $f_\tau(s)$, the farthest point from s in $S \cap \tau$. As in Theorem 2, the number of segment dragging queries is bounded by $O(m)$ in total, thus the theorem is shown.

5 Concluding Remarks

We considered the problem of finding the farthest neighbor of each point in a set S of n points under metrics defined by shortest paths in the plane. Under conventional metrics like the Euclidean or the L_1 plane, this problem is easy as discussed in the beginning: In the L_1 plane, the problem can be solved even in linear time.The problem we considered poses the additional difficulty that evaluating the distance between two points constant time computable and is no known nice geometry like the convex hull. Thus, finding a lower bound of such problem is be an interesting open issue. As a progress on this question, Cardinal *et al.* [7] proved an $\Omega(n \log n)$ lower bound in computing a diametral pair on the

L_1 plane with one turnpike. We conjecture that $\Omega(nm \log(n+m))$ is the right bound for the generalized city metric, since it is known that the farthest Voronoi diagram can have $\Omega(nm)$ combinatorial complexity [4,6].

Constructing an optimal structure for the type (c) segment dragging queries is another challenge. In our algorithms, the segment dragging query is the most frequently called subroutine. Thus, improving its would automatically improve our algorithms. The basic idea of our approach can be extended to any metric where the wavefront is a set of line segments (such as the L_∞ metric or the fixed orientation metric [10]). This methodology can be generalized to other metrics, provided that there is a way to perform dragging queries of the wavefront shape.For example, under the Euclidean metric, the segments transform into ars of circles and thus we need "circular-arc" dragging queries. Those queries can be performed in $O(n^{1/2+\epsilon})$ time using $O(n)$ space, after $O(n \log n)$ preprocessing [1]. Thus, given a set of obstacles and turnpikes, we can solve the AFNP in the Euclidean plane $O(nm(n^{1/2+\epsilon} + \log m))$ time using the Ostrovsky-Berman algorithm [12] for computing $\mathcal{SPM}$.

References

1. Agarwal, P.K., Matoušek, J.: On range searching with semialgebraic sets. Discrete Comput. Geom. 11(1), 393–418 (1994)
2. Ahn, H.-K., Alt, H., Asano, T., Bae, S.W., Brass, P., Cheong, O., Knauer, C., Na, H.-S., Shin, C.-S., Wolff, A.: Constructing optimal highways. In: Proc. 13th Comput., Australasian Theory Sympos. (CATS), Ballarat, Australia. CRPIT, vol. 65, pp. 7–14. ACS (2007)
3. Aichholzer, O., Aurenhammer, F., Palop, B.: Quickest paths, straight skeletons, and the city Voronoi diagram. In: Proc. 18th Annu. ACM Sympos. Comput. Geom., pp. 151–159 (2002)
4. Bae, S.W., Chwa, K.-Y.: The farthest city Voronoi diagram. In: Proc. of the First Meeting of AAAC (2008)
5. Bae, S.W., Kim, J.-H., Chwa, K.-Y.: Optimal construction of the city Voronoi diagram. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 183–192. Springer, Heidelberg (2006)
6. Ben-Moshe, B., Katz, M.J., Mitchell, J.S.B.: Farthest neighbors and center points in the presence of rectangular obstacles. In: Proc. 17th Annu. ACM Sympos. Comput. Geom., pp. 164–171 (2001)
7. Cardinal, J., Collette, S., Hurtado, F., Langerman, S., Palop, B.: Moving walkways, escalators, and elevators. CoRR, abs/0705.0635 (2007)
8. Chazelle, B.: An algorithm for segment dragging and its implementation. Algorithmica 3, 205–221 (1988)
9. Clarkson, K.L., Shor, P.W.: Applications of random sampling in computational geometry. Discrete Comput. Geom. 4, 387–421 (1989)
10. Mitchell, J.S.B.: L_1 shortest paths among polygonal obstacles in the plane. Algorithmica 8, 55–88 (1992)
11. Mitchell, J.S.B.: Shortest paths among obstacles in the plane. Internat. J. Comput. Geom. Appl. 6(3), 309–331 (1996)
12. Ostrovsky-Berman, Y.: The transportation metric and related problems. Inform. Process. Lett. 95, 461–465 (2005)

Improved Algorithm for a Widest 1-Corner Corridor

Gautam K. Das[1], Debapriyay Mukhopadhyay[2], and Subhas C. Nandy[3]

[1] TOR ANUMANA Technologies Private Limited, Kolkata - 700020, India
[2] Rebaca Technologies Private Limited, Kolkata 700 091, India
[3] Indian Statistical Institute, Kolkata - 700108, India

Abstract. Given a set P of n points on a 2D plane, the 1-corner empty corridor is a region inside the convex hull of P which is bounded by a pair of links; each link is an unbounded trapezium bounded by two parallel half-lines, and it does not contain any point of P. We present an improved algorithm for computing the widest empty 1-corner corridor that runs in $O(n^3 \log^2 n)$ time and $O(n^2)$ space. This improves the time complexity of the best known algorithm for the same problem by a factor of $\frac{n}{\log n}$ [4].

1 Introduction

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points on the Euclidean plane. A corridor $C = (\ell', \ell'')$ is an open region in the plane bounded by a pair of parallel straight lines ℓ' and ℓ'' that intersects the boundary of the convex hull of P. The width of the corridor is the perpendicular distance between the bounding lines ℓ' and ℓ''. The widest empty corridor problem was first proposed in the context of robot path planning [6] where the objective was to find the widest straight route avoiding obstacles. An algorithm was given in the same paper that runs in $O(n^2)$ time using $O(n)$ space. If the insertion and deletion of points are allowed, then the dynamic maintenance of the widest empty corridor can be done using $O(n^2)$ space data structure, where the worst case time complexity of handling each insertion/deletion of point is $O(n \log n)$ [7]. Studies have been made for computing the widest empty k-dense corridor problem, where the robot can tolerate collision with at most k obstacles [2,7,9,10].

Sometimes it is observed that the widest empty corridor may not be enough to transport some object of bigger width. This motivates to allow angle turns. Cheng [1] studied this generalization considering L-shaped corridor, which is the concatenation of two perpendicular links. A *link* $L = (\ell', \ell'')$ is an unbounded trapezoid containing no point of P; its two parallel sides are the two half-lines ℓ' and ℓ'' originating from two points p and p' respectively, and one of its other two sides is defined by the line segment $s(L) = [p, p']$; the fourth side is unbounded. The half-lines ℓ' and ℓ'' are said to be the *legs* of link L, and $s(L)$ is called the *base* of L. The width of link L is the perpendicular distance of ℓ' and ℓ''. Diaz-Banez and Hurtado [3] proposed an $O(n^2)$ time algorithm for locating an obnoxious 1-corner polygonal chain anchored at two given points.

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 83–92, 2009.

Recently, Diaz-Banez et al. [4] introduced the widest empty 1-corner corridor problem, as stated below.

Definition 1. *The 1-corner corridor $C = (L_1, L_2)$ is the union of two links $L_1 = (\ell'_1, \ell''_1)$ and $L_2 = (\ell'_2, \ell''_2)$ with disjoint interior, and sharing a common base, i.e., $s(L_1) = s(L_2)$.*

The legs ℓ'_1 and ℓ'_2 (resp. ℓ''_1 and ℓ''_2) define the outer (resp. inner) boundary of the corridor C. Thus, C is a region bounded by an outer boundary containing a convex corner, and an inner boundary that contains a concave corner with respect to the interior of the corridor. The *width* of a 1-corner corridor C, denoted by $w(C)$, is the smaller of the widths of two links. The *angle* $\alpha(C)$, $0 < \alpha(C) \leq \pi$, of the 1-corner corridor $C = (L_1, L_2)$ is the angle determined by the half-lines ℓ'_1 and ℓ'_2 (or equivalently ℓ''_1 and ℓ''_2). A 1-corner corridor C is *empty* if it does not properly contain any point in P, and it partitions the plane into two unbounded regions, each containing at least one of the points in P. Diaz-Banez et al. [4] proposed two algorithms for computing the widest empty 1-corner corridor. The time and space complexities of the deterministic one are $O(n^4 \log n)$ and $O(n)$ respectively. For a given ϵ (> 0), the other algorithm produces a solution having width greater than $(1 - \epsilon)w^*$ in $O(\frac{n \log n}{\sqrt{\epsilon}} + \frac{n^2}{\epsilon})$ time, where w^* is the width of the widest empty 1-corner corridor. If $\alpha = \frac{\pi}{2}$, then the corresponding corridor is an L-shaped corridor. The available algorithm for computing the widest empty L-shaped corridor runs in $O(n^3)$ time and $O(n^3)$ space [1].

In this paper, we present an efficient algorithm for computing the widest empty 1-corner corridor. The time and space complexities of our proposed algorithm are $O(n^3 \log^2 n)$ and $O(n^2)$ respectively. This improves the existing result on the time complexity of the problem by a factor of $\frac{n}{\log n}$ [4].

2 Preliminaries

Let $P = \{p_1, p_2, \ldots, p_n\}$ be the set of n points in 2-dimensional plane, and $CH(P)$ be the convex hull of P. An empty corridor is a pair of parallel lines passing through $CH(P)$ and containing no member of P. A corridor is *locally widest* if each boundary contains at least one point of P and it is not possible to increase its width by performing rotations of the legs around that point. Theorem 1 states a key property of widest empty corridors.

Theorem 1. *[7] Let C be a widest empty corridor through $CH(P)$, with bounding lines ℓ' and ℓ''. Then one of the following conditions must hold:*

(i) there are points $p_i, p_k \in P$ such that ℓ' passes through p_i, ℓ'' passes through p_k, and ℓ' and ℓ'' are perpendicular to the line passing through p_i and p_k, or
(ii) One of the lines, say ℓ', passes through two points $p_i, p_j \in P$ and ℓ'' passes through a different member $p_k \in P$.

From now onwards, we shall restrict our search among the locally widest corridors that satisfy conditions (i) or (ii) stated in Theorem 1. We will use the

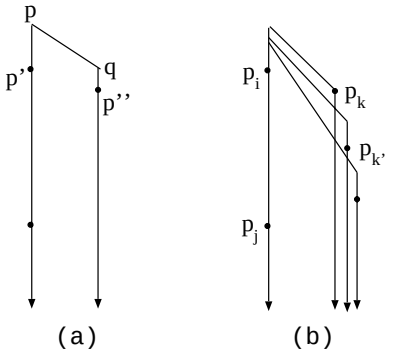


Fig. 1. Demonstration of (a) *1-ended corridor*, and (b) its combinatorial complexity

term *corridor* to refer a locally widest corridor. We will call a widest corridor C satisfying condition (i) (resp. condition (ii)) a *type-1* (resp. *type-2*) corridor. It can be shown that given a set P of n points, the worst case numbers of *type-1* and *type-2* corridors are $O(n)$ and $O(n^2)$ respectively [7]. We now define the *1-ended* corridor, which will be very useful in designing our algorithm.

Definition 2. *An 1-ended corridor is a link $L = (\ell', \ell'')$ whose legs ℓ' and ℓ'' contain at least one point of P, and it is locally widest in the sense that, if p' and p'' lie on ℓ' and ℓ'' respectively, then L is widest among all the links defined by p' and p''. This 1-ended corridor is said to be bounded above (resp. below) if the interior of L is to the right (resp. left) side of the ray $\overrightarrow{pq}$, where $[p, q]$ defines the base of L (see Figure 1(a)).*

The *1-ended* corridors can be classified into two types, namely *type-1* or *type-2* depending on whether (i) both ℓ' and ℓ'' contain one point of P, or (ii) one of the half-lines ℓ' and ℓ'' contains two points and the other one contains one point of P.

Lemma 1. *[4] If $p_i, p_j, p_k \in P$ are the three points on a link (ℓ'_1, ℓ''_1) of the 1-ended locally widest type-2 corridor such that $p_i, p_j \in \ell'_1$ and $p_k \in \ell''_1$, then $\Delta p_i p_k p_j$ is an acute angle triangle. In other words, each angle of the triangle $\Delta p_i p_k p_j$ is less than $\frac{\pi}{2}$.*

Thus, while searching for the widest empty *type-2* corridor, we consider those triples of points (p_i, p_j, p_k) which form a locally widest corridor and $\Delta p_i p_j p_k$ is an acute angle triangle.

Given a pair of parallel half-lines ℓ' and ℓ'', there may exist an infinite number of 1-ended corridors depending on the choice of the base. But, these are considered to be topologically the same in the sense that the parallel sides of both of them are bounded by ℓ' and ℓ''. Thus, for a pair of half-lines ℓ' and ℓ'', there may exist at most two topologically different *1-ended corridors*, one of them is bounded above and other one is bounded below. Thus, if a given pair of points

$p_i, p_j \in P$ can define a set of topologically same *type-1 1-ended* corridors, these are considered to be the same *type-1 1-ended* corridor defined by (p_i, p_j). Similarly, if a triple of points (p_i, p_j, p_k) defines a set of topologically same *type-2 1-ended* corridors, these are considered to be the same *type-2 1-ended* corridor defined by (p_i, p_j, p_k).

Figure 1(b) demonstrates that the combinatorial bound stated for the *type-2* corridors are not valid for the *type-2 1-ended* corridors, since for a pair of points (p_i, p_j) on ℓ', there may exist several members in P that can define ℓ'' to form a topologically different *type-2 1-ended corridor*. A trivial upper bound on the number of such *type-2 1-ended* corridors is $O(n^3)$. Similarly, a trivial upper bound on the number of topologically different *type-1 1-ended* corridors is $O(n^2)$.

A *1-corner corridor* is the union of two *1-ended corridors* (see Figure 2(c)). A 1-corner corridor is said to be *locally widest* if each of its four legs contain at least one point of P, and its width is maximum among all such 1-corner corridors defined by those four points of P. From now onwards, we will use the term 1-corner corridor to refer to the locally widest 1-corner corridor.

3 Computation of 1-Ended Corridors

As a preprocessing step, we shall compute each pairs of points (p_i, p_k) that can define a *type-1 1-ended* corridor, and each triple of points (p_i, p_j, p_k) that can define a *type-2 1-ended* corridor. For each $p_i \in P$, we sort the points in $P \setminus \{p_i\}$ in an array D_i of size $n-1$ in anti-clockwise order around the point p_i. The time needed for computing the array D_i, for all $i = 1, 2, \ldots, n$, is $O(n^2)$ [8].

Note that, each pair of points $p_i, p_k \in P$ can give birth to at most two *type-1 1-ended* corridors as follows. Join $s = [p_i, p_k]$, and define a half-strip R using two parallel half-lines ℓ' and ℓ'' that pass through p_i and p_k respectively, and are perpendicular to s. Two such half-strips are possible. Each of these two half-strips can define *type-1 1-ended* corridors if the region inside it does not contain any point in P (see Figure 2(a)). Using an $O(n^2)$ space data structure for triangular range counting query [5], this can be tested in $O(\log n)$ time. Though the number of such *type-1 1-ended* corridors is infinite, the parallel half-lines of all of them is defined by ℓ' and ℓ''. Thus, we can consider them to belong in the same class. We create an array L_{type_1} to store each pair of points that can define a class of *type-1 1-ended* corridor defined by that pair of points. This needs $O(n^2 \log n)$ time.

We now consider the computation of *type-2 1-ended* corridors. Again observe that for a given triple (p_i, p_j, p_k), we can get an infinite number of *type-2 1-ended* corridors by varying the third (non-parallel) side. But this set of corridors will be treated as a single class $C^2_{(ij)k}$ (say). We consider each pair of points (p_i, p_j), and describe the procedure for computing each point $p_k \in P$ that can define the class $C^2_{(ij)k}$ of *type-2 1-ended* corridors with one leg, say ℓ', containing (p_i, p_j), and the other leg ℓ'' containing p_k. First we rotate the coordinate axes such that ℓ' becomes vertical. Without loss of generality, assume that p_i is above p_j on

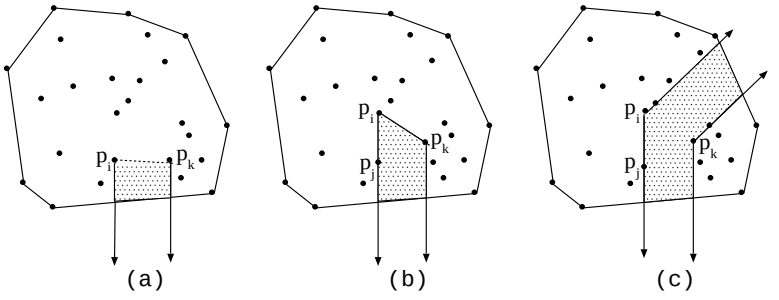


Fig. 2. Examples of (a) *type-1 1-ended* corridor, (b) *type-2 1-ended* corridor, and (c) *1-corner* corridor

this vertical line. Assuming the new coordinate axes, we can classify the *type-2 1-ended* corridors with (p_i, p_j) on ℓ' into the following four types.

C_{LA}^{ij}: The subset of P, each of whose members p_k defines a *type-2 1-ended* corridor with left boundary defined by the line ℓ' containing (p_i, p_j), right boundary ℓ'' containing the point p_k, and the corridors are bounded above.

C_{LB}^{ij}: The subset of P, each of whose members p_k defines a *type-2 1-ended* corridor with left boundary defined by the line ℓ' containing (p_i, p_j), right boundary ℓ'' containing the point p_k, and the corridors are bounded below.

C_{RA}^{ij}: The subset of P, each of whose members p_k defines a *type-2 1-ended* corridor with right boundary defined by the line ℓ' containing (p_i, p_j), left boundary ℓ'' containing the point p_k, and the corridors are bounded above.

C_{RB}^{ij}: The subset of P, each of whose members p_k defines a *type-2 1-ended* corridor with right boundary defined by the line ℓ' containing (p_i, p_j), left boundary ℓ'' containing the point p_k, and the corridors are bounded below.

We now enumerate the set of points in C_{LA}^{ij}. The sets C_{LB}^{ij}, C_{RA}^{ij}, and C_{RB}^{ij} can be found in a similar manner. Let ℓ_{ir} and ℓ_{jr} be the half-lines perpendicular to ℓ' drawn at p_i and p_j respectively, and to the right side of ℓ'. If the right boundary ℓ'' of a *type-2 1-ended* corridor is defined by a point p_k, then (i) p_k lies in the half-strip R_{ij} bounded by ℓ_{ir} and ℓ_{jr} (since $\Delta p_i p_j p_k$ is an acute angle triangle by Lemma 1) and (ii) the region bounded by ℓ', ℓ'', and the line passing through p_i and p_k within $CH(P)$ is empty (see Figure 2(b)).

We sort the set of points in P lying in the half-strip R_{ij} with respect to their distances from ℓ', and store them in a list S. Consider a new list $\hat{S}$ with the maximum number of points in S such that the members in $\hat{S}$ have the same order as that in S, and for each two consecutive points $p_k, p_{k'} \in \hat{S}$, if p_k is closer to ℓ' than $p_{k'}$ then p_k is closer to ℓ_{ir} than $p_{k'}$. In other words, the points in $\hat{S}$ forms a stair inside the half-strip R_{ij} (see Figure 1(b)). We consider each point $p_k \in \hat{S}$, and test whether the triple (p_i, p_j, p_k) can form a *type-2 1-ended* corridor as follows:

We draw a line ℓ'' parallel to ℓ' and passing through p_k. If the region inside the link formed by ℓ', ℓ'' and the line segment $[p_i, p_k]$ is empty, then triple (p_i, p_j, p_k) is a member of the set C^{ij}_{LA}, and the equivalence class of *type-2 1-ended* corridors defined by p_i, p_j, p_k is denoted by $C^2_{(ij)k}$.

As mentioned earlier, the emptiness of this half-strip can be tested using the triangular range counting query algorithm [5] in $O(\log n)$ time. Thus, for each pair of points (p_i, p_j) the computation of all the members in C^{ij}_{LA} needs $O(n \log n)$ time in the worst case. Therefore, the worst case time complexity of this preprocessing phase for computing all the *type-2 1-ended* corridors is $O(n^3 \log n)$. Thus, we have the following result:

Lemma 2. *The worst case time complexity of the preprocessing phase is* $O(n^3 \log n)$.

4 Algorithm for the Widest 1-Corner Corridor

In this section, we describe in detail the method of computing the *1-corner* corridors with the available *type-2 1-ended* corridors. Next, we show that a similar method works for finding the widest *1-corner* corridor with the available *type-1 1-ended* corridors.

Without loss of generality, we assume that no two points in P lie on the same vertical line. We consider each point $p_i \in P$ separately, and consider a vertical line ℓ_i through p_i. We rotate ℓ_i anti-clockwise until it hits a point $p_j \in P$. Thus, $\ell' = (p_i, p_j)$ is defined. This also defines the stair of points $\hat{S}$ as mentioned in the earlier section. Now we choose a point $p_k \in \hat{S}$ to define ℓ'' of a *type-2 1-ended* corridor $C^2_{(ij)k}$. Let P_L and P_R be the subset of P to the left and right of $\ell' = \ell_i$ respectively. We first consider P_R to compute the members in C^{ij}_{LA} with ℓ' as one of its legs. Let $\mathcal{A}$ be the arrangement of the dual lines corresponding to the points in P_R. Each vertex of $\mathcal{A}$ corresponds to an empty corridor among the points in P_R. We can compute the width of all the corridors corresponding to the vertices in $\mathcal{A}$ in $O(n^2 \log n)$ time [7].

But, not all the empty corridors among the points in P_R can be joined with the *1-ended corridor* $C^2_{(ij)k}$ to get an *1-corner corridor*. In order to pick up the *valid* ones, consider the convex hull H_{ijk} of all the points above the line segment $[p_i, p_k]$ in the strip defined by ℓ' and ℓ''. If an empty corridor among the points in P_R lies below the convex hull H_{ijk}, then it can be joined to get an *1-corner corridor*, and hence will be referred to as a *valid* corridor (see Figure 3(a)).

In order to get these *valid* corridors, we need to consider the dual of the lower hull of the convex polygon H_{ijk}. It is a monotone polychain $H^L_{ijk} = \{h_1, h_2, \ldots, h_m\}$, where $h_i, i = 1, 2, \ldots, m$ are the edges of the polychain. The empty corridor corresponding to a vertex of $\mathcal{A}$ that lies below H^L_{ijk} is not a *valid* corridor for joining with $C^2_{(ij)k}$. Thus, we need to consider only the vertices of $\mathcal{A}$ that lie above H^L_{ijk} (see Figure 3(b)) and identify one for which the corresponding corridor is of maximum width. These vertices of $\mathcal{A}$ are called the *valid*

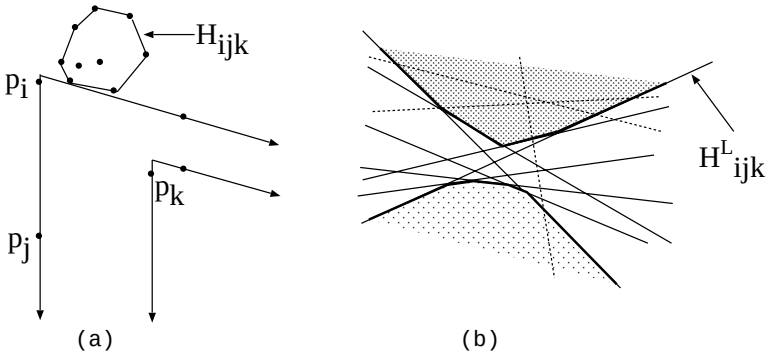


Fig. 3. Computation of type-2 1-corner corridor

vertices. The data structure for maintaining the arrangement $\mathcal{A}$ for the dual lines of P_R is as follows:

We consider the dual line λ corresponding to each member in P_R, and attach a height-balanced tree with it. Its leaf nodes correspond to the corridors corresponding to the vertices of $\mathcal{A}$ that appear on λ. Its each internal node contains the maximum width among the corridors rooted at that node. This computation needs $O(n)$ time for each members in P_R. The *valid* vertices of each dual line λ are obtained as follows:

We test the intersection of λ with the polychain H^L_{ijk}. This may give rise to one of the following cases: (i) λ does not intersect H^L_{ijk}; here none of the vertices on λ is a *valid* vertex, (ii) λ intersects H^L_{ijk} at exactly one point; here all the vertices on λ which are above H^L_{ijk} are the valid vertices, and (iii) λ intersects H^L_{ijk} at exactly two points; here all the vertices that lie inside the intersected segment of λ are valid vertices. In order to compute the intersection point(s) of λ with H^L_{ijk}, we apply a binary search on the vertices of H^L_{ijk} as follows:

Choose the middle-most vertex v of H^L_{ijk}, and consider the edges adjacent to it. Shoot two rays ρ_1 and ρ_2 towards left and right respectively from v along these two edges.

If none of them intersect λ, then Case (i) takes place.

If only one of these rays intersects λ, then Case (ii) appears. Let ρ_1 intersects λ. This implies that the intersection(s) of λ with H^L_{ijk} (if any) are in the left side of v. We apply the same procedure of choosing the middle-most one having the vertices of the H^L_{ijk} at the left side of v. Again, any one of these three cases may appear.

If both ρ_1 and ρ_2 intersect λ, then Case (iii) must appear. We can apply binary search in both the left and right part of H^L_{ijk} with respect to v to identify these two intersection points. Note that, this case appears only once throughout the binary search.

After getting the intersection points (if exists), the vertex on λ having maximum width can be obtained by applying an upward traversal from two appropriate

leaves up to their least common ancestor in the tree attached to λ. Thus, the widest among the *valid* corridors on λ can be obtained in $O(\log n)$ time. This implies that the maximum width *valid* corridor can be computed in $O(n \log n)$ time for a given triple of points (p_i, p_j, p_k).

Next, we need to join this widest empty *valid* corridor with one of the 1-ended corridors whose one boundary passes through the points $p_i, p_j \in P$. Without loss of generality, assume that the leg is vertical. The points defining the other leg are stored in $\hat{S}$ in order. The highest point in $\hat{S}$ defines the narrowest *1-ended corridor*, and as we go down, the width of the *1-ended corridor* increases.

We choose the middle-most element $p_k \in \hat{S}$ that defines the *1-ended corridor* $C^2_{(ij)k}$. We compute H^L_{ijk} of the convex hull H_{ijk}, and then we compute the widest *valid* corridor C among the points in P_R with respect to H^L_{ijk}. If the width of C matches with $C^2_{(ij)k}$, it is recorded; otherwise we apply binary search in $\hat{S}$ to choose a point above or below p_k depending on whether the width of C is less/greater than that of $C^2_{(ij)k}$. This search procedure stops when two consecutive members in $\hat{S}$ are considered. Finally, the widest *1-corner* corridor with one boundary passing through (p_i, p_j) is reported. Thus, we have the following result.

Lemma 3. *The widest empty 1-corner corridor with (p_i, p_j) in one of its legs can be obtained in $O(n \log^2 n)$ time.*

After processing p_j, we rotate ℓ_i in the anti-clockwise direction until it hits another point $p_{j'}$ in $P_L \cup P_R$. In either case, p_j goes to P_L. In addition, if $p_{j'} \in P_L$ (resp. P_R) then $p_{j'}$ leaves P_L (resp. P_R). We can use D_i to get the point $p_{j'}$ in $O(1)$ time. Again we compute the array $\hat{S}$ of points that defines *type-2 1-ended* corridors. As P_R gets changed, the arrangement $\mathcal{A}$ will also be changed. The deletion of p_j and the arrival of $p_{j'}$ cause deletion of some old vertices of $\mathcal{A}$ and insertion of some new vertices in the updated $\mathcal{A}$. Using the method of dynamically maintaining the corridors [9], the incremental time needed for updating $\mathcal{A}$ is $O(n \log n)$. The updates of the data structures attached to the dual lines in the revised set P_R can also be done in the same amount of time. Finally, the reporting of the widest *1-corner corridor* in the changed environment needs another $O(n \log^2 n)$ time (see Lemma 3). Thus, processing of all the members in D_i needs a total of $O(n^2 \log^2 n)$ time in the worst case. Thus, we have the following result:

Lemma 4. *The worst case time and space complexities of computing the widest* 1-corner corridor *of C^{ij}_{LA} with the available* type-2 1-ended *corridors are $O(n^3 \log^2 n)$ and $O(n^2)$ respectively.*

Proof. The pivotal point p_i can be any one of the members in P, and the above argument says that the processing of each such p_i as pivotal point needs $O(n^2 \log^2 n)$ time. The space complexity follows from the fact that maintaining the arrangement $\mathcal{A}$ needs $O(n^2)$ space. □

Using the same method we can compute the widest *1-corner* corridor of type C^{ij}_{LB}, C^{ij}_{RA}, and C^{ij}_{RB} respectively.

Next, we show that the same technique can be applied for computing the widest 1-corner corridor with the available *type-1 1-ended* corridors.

For a given point p_i, let us consider a vertical line ℓ passing through p_i. Draw a half-line ℓ' perpendicular to ℓ at p_i, and to the right-side of ℓ. We compute the arrangement $\mathcal{A}$ of the dual lines of the points in P_R that are to the right of ℓ. We use the same data structure to store the arrangement $\mathcal{A}$. We rotate the lines ℓ and ℓ' in same speed around p_i in anti-clockwise order until a point p_k is reached by either ℓ or ℓ'. Here two cases may arise.

If ℓ hits p_k, then we update the arrangement $\mathcal{A}$ by inserting or deleting the dual line corresponding to p_k as described above. This needs $O(n \log n)$ time.

On the other hand, if p_k is reached by ℓ' and $[p_i, p_k]$ defines a valid *type-1 1-ended* corridor C^1_{ik}, then we compute the convex hull H_{ik} with the points above the line segment $[p_i, p_k]$ inside the strip $L_1 = (\ell', \ell'')$, where ℓ' and ℓ'' are the two parallel legs of the *type-1 1-ended* corridor C^1_{ik}. Next, we compute the widest among the *valid corridors* with respect to the lower chain of the convex hull H^L_{ik}. As described earlier, this needs $O(n \log n)$ time in the worst case.

Thus, processing p_i needs $O(n^2 \log n)$ time in the worst case. Now, we have the following result.

Lemma 5. *The worst case time complexity of computing the widest* 1-corner corridor *with the available* type-1 1-ended *corridors is* $O(n^3 \log n)$.

Proof. The pivotal point p_i can be any one of the members in P. The above arguments say that the processing of each such p_i as pivotal point needs $O(n^2 \log n)$ time. Thus the lemma follows. □

Lemmata 3, 4 and 5 lead to the final theorem of this work as stated below:

Theorem 2. *The worst case time and space complexities of computing the widest* 1-corner corridor *are* $O(n^3 \log^2 n)$ *and* $O(n^2)$ *respectively.*

References

1. Chen, S.-W.: Widest empty L-shaped corridor. Information Processing Letters 58, 277–283 (1996)
2. Chattopadhyay, S., Das, P.P.: The k-dense corridor problem. Pattern Recognition Letters 11, 463–469 (1990)
3. Diaz-Banez, J.M., Hurtado, F.: Computing obnoxious 1-corner polygonal chains. Computers and Operations Research 33, 1117–1128 (2006)
4. Diaz-Banez, J.M., Lopez, M.A., Sellares, J.A.: On finding a widest empty 1-corner corridor. Information Processing Letters 98, 199–205 (2006)
5. Goswami, P.P., Das, S., Nandy, S.C.: Triangular range counting query in 2D and its application in finding k nearest neighbors of a line segment. Computational Geometry Theory and Applications 29, 163–175 (2004)

6. Houle, M.E., Maciel, A.: Finding the widest empty corridor through a set of points. In: Toussaint, G. (ed.) Snapshots of Computational and Discrete Geometry, Technical Report SOCS-88.11, School of Computer Science, McGill University (1988)
7. Janardan, R., Preparata, F.P.: Widest-corridor problem. Nordic J. Computing 1, 231–245 (1994)
8. Lee, D.T., Ching, Y.T.: The Power of Geometric Duality Revisited. Information Processing Letter 21, 117–122 (1985)
9. Nandy, S.C., Harayama, T., Asano, T.: Dynamically maintaining the widest k-dense corridor. Theoretical Computer Science 255, 627–639 (2001)
10. Shin, C.-S., Shin, S.Y., Chwa, K.-Y.: The widest k-dense corridor problem. Information Processing Letters 68, 25–31 (1998)

Maximum Neighbour Voronoi Games*

Md. Muhibur Rasheed[1,**], Masud Hasan[2], and M. Sohel Rahman[2,3]

[1] Department of Computer Science
University of Texas at Austin, Austin, Texas 78712, USA
muhibur@cs.utexas.edu
[2] Department of CSE, BUET, Dhaka-1000, Bangladesh
{masudhasan,msrahman}@cse.buet.ac.bd
http://www.buet.ac.bd/cse
[3] Department of Computer Science
King's College London, Strand, London WC2R 2LS, England
http://www.dcs.kcl.ac.uk/adg

Abstract. Recently several researchers have studied the competitive facility location problem in the form of Voronoi games, where each of the two players places n points with the target of maximizing total Voronoi area of its sites in the Voronoi diagram of $2n$ points. In this paper we address this problem by introducing Voronoi games *by neighbours* where the basic objective of an optimal playing strategy is to acquire more neighbors than the opponent. We consider several variations of this game, and for each variation we either give a winning strategy, if it exists, or show how the game ends in a tie.

Keywords: Competitive facility location, Delaunay triangulation, Voronoi diagram, Voronoi games.

1 Introduction

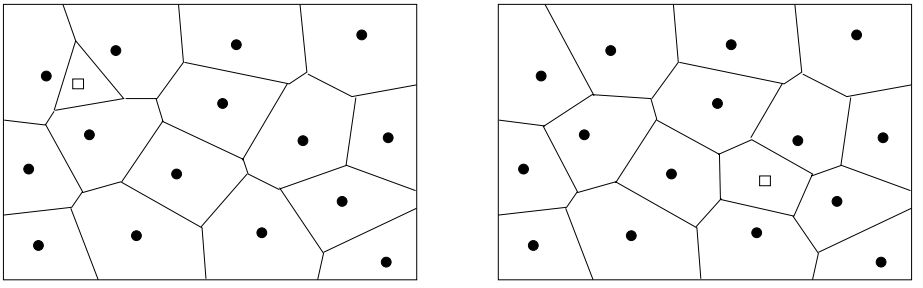
Consider a city having multiple shops of same type. It is common to assume that customers prefer to go to the shop which is geographically closest to them (provided that the service is similar in quality). Thus, Voronoi diagram of the shops perfectly model the customers affiliation with a shop, where the customers within a Voronoi region are affiliated with the corresponding shop.

Now imagine that you are the owner of a shopping chain and you are planning to set up shops in a city where at the moment you do not have any shops but one of your competitors shopping chain has already set up numerous shops. So, your objective would be to put the new shop in a location such that its Voronoi region is as large as possible. The problem of finding such a location falls, in general, under the *competitive facility location problem* [1,2,5,6,8,9]. If you have more than one shop, or more competitively, the same number of shops that your competitor has, then your objective would be to put your shops such that the total Voronoi area of your shops is at least as large as compared to your opponent's shops. This variation of the competitive facility location problem is known as *Voronoi games*.

* This research was carried out in the Department of CSE, BUET as part of the M.Sc. Engg. thesis of the first author.

** On leave from the Department of CSE, BUET, Dhaka-1000, Bangladesh.

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 93–104, 2009.



(a) (b)

Fig. 1. A new site becoming neighbour to different number of existing sites

Now consider the above competitiveness in a different way. Suppose, you know that your products are at least as good as the opponent's products and you also know that if you set up a shop, many customers would be tempted to come to your shop rather than going to your opponent's ones. Your objective is to entice customers away from as many shops of your opponent as possible. For example, Figure 1(a) shows a new shop (represented as a rectangle) which takes customers away from three existing shops and Figure 1(b) shows another placement where the new shop takes customers away from five existing shops. Therefore, to cause the maximum interference to the business of your opponent you should choose to place your shops such that each of your shops gets as many shops of the opponent as *neighbors* as possible. In this paper, we introduce this competitive facility location problem in several variations and refer to them in general as *Voronoi games by neighbors*.

1.1 Related Works

It started with the problem of finding the locations to place shops of two shopping chains along a highway such that the total area (according to Voronoi diagram in 1D) of the newly placed shops of each shopping chain is maximized [1,2]. The authors in [1,2] formulated the problem in terms of an n-round Voronoi game where the two players take n turns to place their sites. At each turn, firstly, the first player places a site and then the second player follows suit. The wining criteria of a player in this game is to get a total Voronoi region that is more than that of the opponent. In [1,2], the problem was considered along a line and a circle and it was proved that the second player always wins but the first player can keep the wining margin as small as possible.

For a two dimensional playing area, the problem is still open, but several variations have been studied. Dehne, Klein and Seidel [8] first addressed the problem in 2D. They studied how a single new point can be placed in an existing n-site Voronoi diagram so that the Voronoi area of the new site is maximized. They formulated the area of the new site as a function of its location. They proved that if the (wouldbe) neighbors of the new sites are in convex positions then there can be exactly one maxima of the function and that is where the new site should be placed. However, they could not solve the problem where the neighbors are in general positions.

Cheong, Efrat and Har-Peled [5] presented some approximate solutions to the problem. They gave an agorithm which approximates the Voronoi region of the new site to $(1-\delta)$ times the optimal, for any $\delta > 0$, and runs in $O(n/\delta^2 + n \log n)$ time.

Cheong et. al. [6] addressed the problem in terms of one round Voronoi game, where the first player places all of its sites at once and then the second player places its sites. According to their formulation, two players place their sites inside a unit square region and try to maximize the total Voronoi region of their sites. They proved that, for sufficiently large n, the second player can always place his sites in such a way that the sum of the area of their Voronoi regions is at least $1/2 + \alpha$ times the total playing area, for any $\alpha > 0$, and thus found a winning strategy for the second player. Fekete and Meijer [9] extended the results of [6] to find winning strategies even when the playing area is not a unit square. They showed that the second player wins only if $n \geq 3$ and $\rho > \sqrt{2}/n$ or if $n = 2$ and $\rho > \sqrt{3}/2$, where ρ represents the aspect ratio of the playing area. They also proved that if the playing area is a polygon with holes, then finding a winning strategy for the second player is NP-hard. To the best of our knowledge, competitive facility location problems have not been studied in terms of optimal number of Voronoi neighbours.

One thing to be noted here is that Voronoi games in terms of area (as discussed above) significantly depend upon the bounding area in which the games are to be played. However, as we will see, for Voronoi games by neighbours no such restriction is required.

1.2 Our Results

In this paper we introduce the competitive facility location problem in terms of Voronoi games by neighbours. The game is between two players: Player1 and Player2. We consider the game in one round. At first, Player1 places his n sites. Then, Player2 places his n sites. The result is measured in the Voronoi diagram of $2n$ sites. Each Player has the target of *optimizing* the *number of Voronoi neighbours*. We study several variations of the game which are based on different criteria for optimality, different types of neighborship, and different motivation (Figure 2). For each variation, we give a wining strategy, if it exists, or show how the game ends in a tie.

	Criterion
1	To get all of the opponent's sites as neighbours
2	To get more opponent's sites as neighbours than what the opponent gets from our sites
3	To achieve Criterion 2 considering non-distinctness[a] of the neighbourship
4	To achieve Criterion 2 avoiding self-neighbourship[b]
5	To Achieve Criterion 4 considering non-distinctness[a] of the neighbourship

[a] If a site is neighbour to k sites, then its neighbourship is counted as k (as opposed to one).
[b] Self-neighbours are neighbours from the same player.

Fig. 2. The different variations of Voronoi Games in this paper

1.3 Outline

We will mostly work on the Delaunay triangulation and Delaunay circles of the sites. Since we consider one round games, we will follow an incremental process: after Player1 places all his n sites, Player2 will place his sites one by one as necessary. It will require an incremental update of the Delaunay triangulation to see how the addition of a new site changes the neighbourship relation among the sites.

In order to get Player1's sites as neighbours "efficiently" by the sites of Player2, we will select edges from the Delaunay triangulation of Player1's sites and then insert Player2's sites inside the adjacent triangles of those selected edges. However, for Player2 to win, those edges should be selected carefully so that the optimal neighbourship criterion holds. For that we will use the edges of a maximum matching in the Delaunay triangulation.

We will first study, in Section 2, the basics of Voronoi diagram, Delaunay triangulation, their incremental update, including the update of the neighbourship relation among the sites, and matching in a graph. Then, in Section 3, we will precisely formulate the variations considered in this paper. In Section 4, we will give our main results. Finally, we conclude in Section 5 with some directions of future work.

2 Preliminaries

Given a set of n points P in the plane, a *Voronoi diagram* [3,7] $\mathcal{V}$ of P is the subdivision of the plane into n regions, one for each point in P, such that any point x in the plane lies in the region corresponding to a point y in P iff the distance of x from y is smaller than its distance from any other site in P. The points of P are called the *Voronoi sites*, or simply *sites*, and their regions are called the *Voronoi regions* of $\mathcal{V}$. Voronoi regions of $\mathcal{V}$ meet at edges and vertices called *Voronoi edges* and *Voronoi vertices*, respectively, of $\mathcal{V}$. Two Voronoi regions are called *neighbours* of each other if they share a Voronoi edge. The Voronoi diagram $\mathcal{V}$ is a connected plane graph where the number of Voronoi edges, Voronoi vertices and Voronoi regions are $O(n)$ each.

Throughout this paper, we assume that the points of P are in general position; so no three sites are collinear and no four sites cocircular. In general position, the maximum degree of a Voronoi vertex of $\mathcal{V}$ is three.

The *Delaunay triangulation* [3,7] $\mathcal{D}$ of P is defined in terms of the components of $\mathcal{V}$. It is the straight line dual graph of $\mathcal{V}$, where the vertices are the sites of P and two vertices are connected by a *Delaunay edge* iff their corresponding Voronoi regions are neighbours in $\mathcal{V}$. Two sites defining a Delaunay edge are also called *neighbours* of each other.

Similar to $\mathcal{V}$, $\mathcal{D}$ is also a plane graph with linear number of Delaunay edges and linear number of *faces*. Note that the boundary of $\mathcal{D}$ is the convex hull $\mathcal{H}$ of the points of P. Since the points of P are in general position, faces of $\mathcal{D}$, possibly except the outer face, are triangles and are called *Delaunay triangles*. The circumcircle of a Delaunay triangle is called its *Delaunay circle*. One important characterization of a Delaunay triangulation is that three sites u, v, w forms a Delaunay triangle iff their circumcircle does not contain any other site. Observe that even if the outer face of $\mathcal{D}$ is a triangle, it is not a Delaunay triangle by the above characterization.

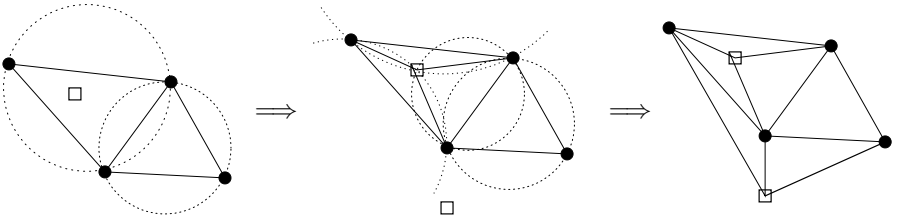


Fig. 3. Illustration of inserting a new site

2.1 Inserting a New Site

The phenomenon of placing a new site in the existing Delaunay triangulation and the resulting change in neighbourship of the sites is well studied in incremental construction of Delaunay triangulation [3,7]. Let $\triangle(u, v, w)$ be a Delaunay triangle consisting of the sites u, v and w. We denote by $C(u, v, w)$ the corresponding Delaunay circle passing through u, v and w. We now present the following lemmas and the corollary which will play central role in the rest of the paper. Also see Fig. 3.

Lemma 1. *If a new site p is placed inside a Delaunay triangle $\triangle(u, v, w)$, then in the resulting triangulation, u, v and w become neighbours of q.*

A proof of this lemma can for example be found in [3] [7, Page194].

Lemma 2. *If a new site p is placed outside the Delaunay circle $C(u, v, w)$, then in the resulting Delaunay triangulation, $\triangle(u, v, w)$ remains a Delaunay triangle.*

Proof. After inserting p, the circle $C(u, v, w)$ still remains empty of other sites. So, $\triangle(u, v, w)$ remains a Delaunay triangle. □

Corollary 1. *Let q_1 be a site of $\mathcal{D}$ but not a point of $\mathcal{H}$. Adding a new site q_2 outside all Delaunay circles of $\mathcal{D}$ can not make q_2 a neighbour of q_1.*

Proof. Clearly, q_2 is placed outside the convex hull $\mathcal{H}$. If it becomes neighbour of q_1, then one edge in $\mathcal{H}$ is no more a Delaunay edge. But since q_2 is placed outside $\mathcal{H}$, by Lemma 2, all existing Delaunay triangles in $\mathcal{D}$ exist in the resulting Delaunay triangulation, which contradicts that an edge of $\mathcal{H}$ is no more a Delaunay edge. □

2.2 Maximum Matching

A *matching* $\mathcal{M}$ in a graph G is a subset of the edges of G such that any vertex of G has at most one incident edge in $\mathcal{M}$ [11]. A matching is *maximum* if it has maximum possible cardinality among all matching. A matching $\mathcal{M}$ is *perfect* if every vertex has an incident edge in $\mathcal{M}$. So the size of a perfect matching is $n/2$, where n is the number of vertices in the graph. There exist efficient polynomial time algorithms to compute a maximum matching in a graph. While $n/2$ is the upper bound of a maximum matching for general graphs, not all classes of graphs have a perfect matching. One such graph class is the triangulated planar graphs [4], where each face, including the outer face, is a triangle. However, Biedl et.al. [4] presented the following lemma that gives a lower bound on the size of a matching in a triangulated planar graph.

Lemma 3. [4] *Any triangulated planar graph with $n \geq 10$ vertices has a matching of size at least $\frac{n+4}{3}$.*

In what follows we denote the i^{th} site placed by Player1 as p_i and the i^{th} site placed by Player2 as q_i. In the figures, the sites placed by Player1 will be represented by solid circles and the sites placed by Player2 will be represented by rectangles.

3 The Games

We consider the following five games.

Variation 1: Maximizing opponent neighbors. This variation is the simplest one and is more like an optimization problem. In this case, at first, Player1 places all his n sites. Now, the wining criteria for Player2 is to get all n sites of Player1 as neighbours by placing minimum number of its sites. For this game we show that, to win, Player2 needs to place only at most $\frac{2n+2}{3}$ of his sites. In Figure 4, we present an example of Variation 1, where Player2 can win by using only one site.

Variation 2: Distinct opponent neighbors. In this variation, like Variation 1, each player wants to get all sites of the opponent as neighbours; but at the same time, he also wants as many of its own sites not to be neighbours of the opponent as possible. The motivation lies in more competitiveness in the competitive facility location problem: here, beside the primary motivation of interfering opponents business as much as possible, the secondary motivation is to ensure that self business is interfered as less as possible by the opponent. More formally, suppose that after both players place n sites, the sites of Player1 get in total N_1 sites of Player2 as their neighbors and the sites of Player2 get in total N_2 sites of Player1 as neighbours. Then Player1 wins if $N_1 > N_2$, player two wins if $N_2 > N_1$, and the game ends in a tie if $N_1 = N_2$. For this game we show that Player2 always wins. See Figure 4, where $N_1 = 3$ and $N_2 = 5$, so Player2 wins.

Variation 3: Non-distinct opponent neighbors. Next we consider the above gaming criteria combined with the *non-distinctness* of N. That means, we take into account the number of *times* a particular site becomes neighbours to opponent's sites. Again, the motivation of counting non-distinctness of N follows from the competitive facility location problem where a competitor may want to interfere opponent's sites as many

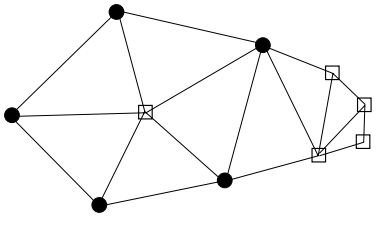


Fig. 4. A (common) example for all five variations. As Variation 1, 2, 4 and 5, Player2 wins here. As a Variation 3, it is a tie.

times as possible (irrespective of how many sites are interfered) by its own sites. If a site x_i of playerX gets $N(x_i)$ opponent sites as its neighbor, then the total number of *non-distinct* opponent sites as neighbors of PlayerX is $N_x = \sum_{i=1}^{n} |N(x_i)|$. So, Player1 wins if $N_1 > N_2$, Player2 wins if $N_2 > N_1$, and it is a tie if $N_1 = N_2$. We show that this game always ends in a tie. See Figure 4, where $N_1 = 8$ and $N_2 = 8$; so the game is a tie.

Variation 4: Distinct opponent and self neighbors. In this variation, the winning criterion involves the new concept of self-neighbourship. When a site gets another site belonging to the same player as its neighbor, they become *self neighbors*. We consider this variation, because in real life situation, a competitor may also want not to entice customer away from its own shops. The winning criterion is defined as follows. Suppose that N_1 (N_2) is the number of distinct sites of Player2 (Player1) which are neighbors of Player1 (Player2). Also suppose that M_1 (M_2) is the number of distinct sites of Player1 (Player2) which are self-neighbors. Then the score of Player1 is S_1 = N_1 - M_1 and the score of Player2 is S_2 = N_2 - M_2. So, if $S_1 > S_2$, Player1 wins; if $S_1 < S_2$, Player2 wins, and otherwise, the game ends in a tie. We show that Player2 wins here. See Figure 4, where N_1 = 3, M_1 = 5, N_1 = 5 and M_1 = 4, so, Player2 wins.

Variation 5: Non-distinct opponent and self neighbors. Our last winning criterion is to add the non-distinctness with the criteria of Variation 4. More specifically, if the neighbourship meaning of N_1, N_2, M_1 and M_2 are the same as above but the count is non-distinct and if $S_1 = N_1 - M_1$ and $S_2 = N_2 - M_2$, then Player1 wins if $S_1 > S_2$, Player2 wins if $S_1 < S_2$, and otherwise it is a tie. For example, See Figure 4 where $N_1 = N_2 = 8$ and $M_1 = M_2 = 10$, so the game is a tie. For this variation too, we show that Player2 can always win.

4 Playing the Games

Since we are considering only one round games, Player1 has to place his points without any knowledge whatsoever about how Player2 is going to place his sites. On the other hand, Player2 has complete knowledge of the positions of the sites placed by Player1. This gives Player2 an advantage which enables him to effectively implement winning strategies for almost all the variations.

4.1 Variation 1: Maximizing Opponent Neighbors

Since Player1 can never know in advance where Player2 will place his sites, it is not possible to formulate a strategy for Player1. On the contrary, several strategies can be found for Player2. We first present a very simple lemma which implicitly presents us with an easy strategy for Player2.

Lemma 4. *It is possible for Player2 to get all n sites of Player1 as neighbors.*

Proof. Consider a site p_i of Player1. Let $\triangle(p_i, x, y)$ be an adjacent Delaunay triangle of p_i (x and y can be of any player). Player2 will place q_i inside $\triangle(p_i, x, y)$. By Lemma 1, p_i will become a neighbour of q_i. □

In the rest of this section, we investigate whether Player2 can achieve all n sites of Player1 as neighbors using less than n sites of its own. In particular, we will establish an upper bound on the number of sites required by Player2 to achieve the above goal. The main result we achieve is the following theorem.

Theorem 1. *It is possible for Player2 to get all n sites of Player1 as neighbors by using at most $\frac{2n+2}{3}$ sites.*

Proof. Assume that Player1 has placed all of his n sites. Now imagine that 3 other sites x, y, z, not belonging either to Player1 or Player2, placed far enough and outside all existing Delaunay circles such that in the resulting Delaunay triangulation $\mathcal{D}$ of $n' = n+3$ sites, the convex hull is the triangle containing x, y, z. By Lemma 1, the neighbourships among the n sites of Player2 do not change in the resulting triangulation. Moreover, we place these three sites in such a way that they do not become part of circle through four points. So the resulting triangulation is a triangulated planar graph and we find a maximum matching $\mathcal{M}$ of this triangulation. By Lemma 3, for $n' \geq 10$ (i.e., $n \geq 7$), $\mathcal{M}$ has at least $\frac{n'+4}{3}$ edges. For each edge $e = (u, v)$ of $\mathcal{M}$, Player2 adds one new site inside an adjacent triangle of e. By Lemma 1, both u and v become neighbours of q. In this way by placing $\frac{n'+4}{3}$ sites, Player2 gets $2 \times \frac{n'+4}{3}$ sites of Player1 as neighbours. For each of the remaining $n' - 2 \times \frac{n'+4}{3} = \frac{n'-8}{3}$ sites of Player1, Player2 places one site as mentioned in Lemma 4 to get it as a neighbour. So in total Player2 needs $\frac{n'+4}{3} + \frac{n'-8}{3} = \frac{2n'-4}{3} = \frac{2n+2}{3}$ sites to get all n points of Player1 as neighbours.

For $1 \leq n \leq 4$, Player2 can place one site and for $n = 5, 6$, Player2 can place two sites that get all n sites of Player2 as neighbour. These can be verified by a case by case drawing of $\mathcal{D}$, which we omit due to space constraint. □

4.2 Variation 2: Distinct Opponent Neighbors

From Theorem 1, we already know that it is possible for Player2 to get all sites of Player1 as neighbors. So, to win this variation, Player2 only need to ensure that at least one of his sites does not become neighbour to any of the sites of Player1. In what follows, this will be referred to as *hiding a site from the opponent*.

Lemma 5. *To hide a site from all the existing sites in $\mathcal{D}$, two extra new self sites are always sufficient.*

Proof. Let the site to be hidden be c and the two sites that will hide this site be a and b. With only one existing site d, a, b and c can be placed trivially: place them such that all of a, b, c and d are in the resulting convex hull and (c, d) is not a Delaunay edge. So, assume that there are two or more existing sites. Also assume that $\mathcal{D}$ and $\mathcal{H}$ mean for the existing sites. With only two exisiting sites, $\mathcal{D}$ and $\mathcal{H}$ are simply the line segment between them. Let q and r be two sites on $\mathcal{H}$. Let x be a point outside $\mathcal{H}$ such that $\mathcal{D} \cup \triangle(q, x, r)$ is also convex (see Figure 5(a)). Now place a and b on qx and rx, respectively, at an ε distance away from x. Put c inside the triangle $\triangle(a, b, x)$ and close to x. For sufficiently small ε, in the resulting Delaunay triangulation, any circle through c and a site other than a, b will contain a or b inside of it. That means, c can not form a Delaunay triangle avoiding a and b, further implying that c have only two neighbours, a and b. □

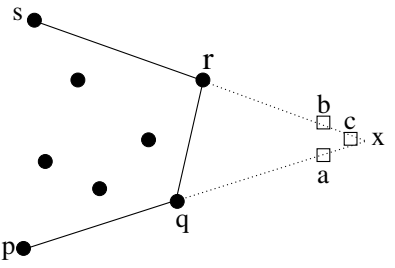


Fig. 5. Hiding a site

Corollary 2. *Suppose that Player2 needs m sites to get all sites of Player1 as neighbors. Then it needs $m + 3$ sites to hide at least one of its own site from Player1.*

Theorem 2. *Player2 can always get more distinct opponent sites as neighbors than Player1 if $n \geq 11$.*

Proof. By Theorem 1 and Corollary 2, total number of sites required for Player2 is $\frac{2n+2}{3} + 3$. Clearly we must have:
$n \geq \frac{2n+2}{3} + 3$
$\Rightarrow n \geq 11$
And hence the result follows. □

For $n < 3$, Player2 does not have sufficient sites to hide one site and the game ends in a tie. For $4 \leq n \leq 6$, by Theorem 1 Player2 has three extra sites, and one of them can be hidden by Lemma 5. So Player2 wins for these cases. For $n = 3$ and for $7 \leq n \leq 10$, a case by case analysis can give a winning strategy for Player2. We leave that exploration for the full version of this paper.

4.3 Variation 3: Non-distinct Opponent Neighbors

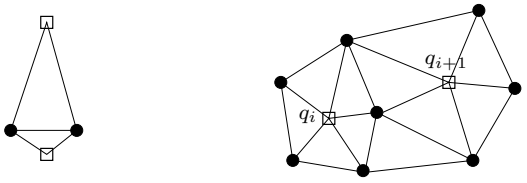
In this variation, we show that the game always ends in a tie.

Theorem 3. *The game in Variation 3 always ends in a tie.*

Proof. Note that if a site p_i of Player1 gets m neighbors, then those m sites of Player2 will get p_i as their neighbors for a total of m times. Applying the same logic for every site placed by Player2, we can see that the total number of non-distinct neighbors from the opponent sites is the same for both players. □

4.4 Variation 4: Distinct Opponent and Self Neighbors

Recall that, in this variation, the two players need not only to maximize opponent sites as neighbors but also to minimize self-neighbourships. In what follows, we will modify the strategy applied for Variation 1 to get a winning strategy for Player2 for this variation.



(a) (b)

Fig. 6. Winning the Variation 4. (a) Wining when $n = 2$. (b) Isolating Player2 sites when $n \geq 3$.

Theorem 4. *If $n \geq 2$, then Player2 can always win in Variation 4. For $n = 1$, the game ends in a tie.*

Proof. For $n = 1$, we get $N_1 = N_2 = 1$ and $M_1 = M_2 = 0$, so the game is a tie. If $n = 2$, Player2 wins by placing its sites as shown in Figure 6(a).

Now assume that $n \geq 3$. So, $\mathcal{D}$ has at least one triangle. Our strategy is to ensure that some sites of Player2 remain completely surrounded by sites of Player1 and thus avoid self-neighbourship among themselves and at the same time each Player1 site becomes neighbour of at least another Player1 site and thus Player1 suffers plenty of self-neghbourship. (See Figure 6(b)). We do that as follows.

We will use a modified version of the idea in the proof of Theorem 1. This time we do not add three sites of Player2 as the bounding triangle. Then we place the Player2 sites one by one as follows. For the very first time, let $e = (u, v)$ be an edge of $\mathcal{D}$. We add one new site q_1 inside an adjacent triangle of e. By Lemma 1, both u and v become neighbours of q_1. However, there may be other sites of Player1 that also become neighbours of q_1. We mark all those sites of Player1 that become neighbours of q_1.

Now at each step, until all sites of Player1 are marked we do the following. Let p be an unmarked site of Player1. The fact that p is not a neighbour of any Player2 site implies that if we place new site q_i at p, then q_i will not be neighbour of any Player2 site. In fact, a small neighbourhood of p can be treated as a *safe region* for q_i, where by "safe" means not becoming neighbour of any site of Player2. We add q_i within any triangle adjacent to p and within that small and safe neighbourhood of p. We mark p and any other sites of Player1 that become neighbour of q_i.

Assume that k sites of Player2 have been placed by the above procedure. Note that none of these k sites is in the resulting convex hull. The remaining $n-k$ sites of Player2 are placed all together far enough and outside any existing Delaunay circles so that by Corollary 1 they do not become neighbours of the k sites of Player2.

At this moment we claim that every site of Player1 is a neighbour of at least one other site of Player1. For if a site p of Player1 did not have another site of Player1 as neighbour, then p would be a point of at least one triangle whose other two sites are of Player2 and are thus self-neighbours. But that is a contradiction, because our placement ensures that Player2 sites can not be neighbours of each other.

Now we come to the analysis. We have ensured three things: (i) Player2 gets all sites of Player1 as neighbours, so $N_2 = n$; (ii) all n sites of Player1 suffer self-neighbourship, so $M_1 = n$; and (iii) since the size of $\mathcal{M}$ is at least one, $k \geq 1$, which implies that at least one site of Player2 is not neighbour of other sites of player2, so

$M_2 \leq n-1$. Moreover, for $n-k \geq 3$, among the $n-k$ sites of Player2 that are placed far apart, we can hide at least one site by using two or more self sites (by Lemma 5). So $N_1 \leq n$. Therefore, Player2 wins. □

4.5 Variation 5: Non-distinct Opponent and Self Neighbors

Recall from Section 4.3 that no winning strategy exist for variation 3 where the criterion was to get more non-distinct opponent sites as neighbors. So, in this variation too, for both Player1 and Player2 the number of non-distinct opponent sites as neighbor is equal, i.e. $N_1 = N_2$. But, the "number of non-distinct self sites as neighbor" can be different, i.e., M_1 and M_2 can be different, and that difference will decide who wins this variation of the game.

We will use the idea of Variation 4 with slight modification. We will first get all n sites of Player1 as neighbours by the sites of Player2 as described in Theorem 4. After that, we will, however, not simply place the remaining sites of Player2 far apart, because that may cause a large number of self-neighbourship among those sites (e.g., they may become uniformly close to each other and form a dense cluster with higher number of self-neighbourship.) We will place those sites uniformly around and in the outside of all existing Delaunay circles in such a way that they are the only sites in the resulting convex hull and each site in that convex hull has only two self-neighbours, which are the two adjacent sites in the convex hull.

Theorem 5. *Player2 always wins for variation 5.*

Proof. By Theorem 4 and by counting non-distinctness, $M_1 \geq 2n$. By Corollary 1, self-neighbourship of Player2 are only among the $n-k$ sites that form the final convex hull. So, $M_2 \leq 2(n-k)$. Since $k \geq 1$, $M_2 \leq 2(n-1)$. Therefore, Player2 wins. □

5 Conclusion

In this paper we have addressed the problem of competitive facility location by introducing Voronoi games by neighbours where the basic objective of an optimal playing strategy is to acquire more neighbors than the opponent. We have considered several variations of this game in one round, and for each variation we either have given a winning strategy, if it exists, or have shown how the game ends in a tie. There remain several issues that may be addressed for future research as follows.

1. For Variation 1 of the game, we have proved an upper bound for the number of sites of Player2 required to get all sites of Player1 as neighbours. We believe this upper bound is not tight. In particular we have the following conjecture.

   Conjecture 1. It is possible for Player2 to get all n sites of Player1 as neighbors by using at most $\lceil \frac{n}{3} \rceil$ sites.

   To prove Conjecture 1, essentially, the idea is as follows. Suppose, p_j, p_k and p_l are three existing sites. Our strategy is to place a new site q_i inside a cell of the arrangement and thereby making all of p_j, p_k and p_l its neighbours. Before placing

the next site q_{i+1}, we remove p_j, p_k, p_l and their corresponding Delaunay edges from the triangulation and choose the another three sites for q_{i+1}. We continue to do the above until all sites have become neighbors. Clearly, the main task is to prove that such a site can always be found, which we have been able to prove for almost all the cases. There exists, however, some pathological cases [10] for which we can not get such sites readily and hence further investigation through a case by case analysis is required.

2. We have assumed that the service provided by each facility provider were equivalent and the receivers always affiliated with the provider which was geometrically closest. But there can be situations when some service providers can be for preferred by the receivers even if they are geometrically further. In such cases, we can model them using weighted Voronoi diagrams. Therefore, it will be interesting to investigate the games for the weighted Voronoi diagram.

The first author in this thesis [10] have studied these games in n-round, where for most cases the game ends in a tie. It would be interesting to see variations for n-round games where players have wining strategies.

References

1. Ahn, H.-K., Cheng, S.-W., Cheong, O., Golin, M., van Oostrum, R.: Competitive facility location: the Voronoi game. Theor. Comput. Sci. 310(1-3), 457–467 (2004)
2. Ahn, H.-K., Cheng, S.-W., Cheong, O., Golin, M.J., van Oostrum, R.: Competitive facility location along a highway. In: Wang, J. (ed.) COCOON 2001. LNCS, vol. 2108, pp. 237–246. Springer, Heidelberg (2001)
3. Aurenhammer, F., Klein, R.: Voronoi diagrams. In: Sack, J.-R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 201–290. North-Holland, Amsterdam (2000)
4. Biedl, T., Demaine, E.D., Duncan, C.A., Fleischer, R., Kobourov, S.G.: Tight bounds on maximal and maximum matchings. In: Eades, P., Takaoka, T. (eds.) ISAAC 2001. LNCS, vol. 2223, pp. 308–319. Springer, Heidelberg (2001)
5. Cheong, O., Efrat, A., Har-Peled, S.: Finding a guard that sees most and a shop that sells most. Discrete Comput. Geom. 37(4), 545–563 (2007)
6. Cheong, O., Har-Peled, S., Linial, N., Matoušek, J.: The one-round Voronoi game. In: SCG 2002: Proceedings of the 18th Annual Symposium on Computational Geometry, pp. 97–101. ACM, New York (2002)
7. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications, 2nd edn. Springer, Heidelberg (2000)
8. Dehne, F.K.H.A., Klein, R., Seidel, R.: Maximizing a voronoi region: The convex case. In: Bose, P., Morin, P. (eds.) ISAAC 2002. LNCS, vol. 2518, pp. 624–634. Springer, Heidelberg (2002)
9. Fekete, S.P., Meijer, H.: The one-round Voronoi game replayed. Comput. Geom. Theory Appl. 30(2), 81–94 (2005)
10. Rasheed, M.M.: Voronoi neighbors: Optimization, Variation and Games, MSc Thesis, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh (2008)
11. West, D.B.: Introduction to Graph Theory, 2nd edn. Pearson Education, London (2002)

On Exact Solutions to the Euclidean Bottleneck Steiner Tree Problem*

Sang Won Bae, Chunseok Lee, and Sunghee Choi

Division of Computer Science, KAIST, Korea
{swbae,stonecold,sunghee}@tclab.kaist.ac.kr

Abstract. We study the Euclidean bottleneck Steiner tree problem: given a set P of n points in the Euclidean plane, called terminals, find a Steiner tree with at most k Steiner points such that the length of the longest edge in the tree is minimized. This problem is known to be NP-hard even to approximate within ratio $\sqrt{2}$. We focus on finding exact solutions to the problem for a small constant k. Based on geometric properties of optimal location of Steiner points, we present an $O(n \log n)$ time exact algorithm for $k = 1$ and an $O(n^2)$ time algorithm for $k = 2$. Also, we present an $O(n \log n)$ time exact algorithm to the problem for a special case where there is no edge between Steiner points.

1 Introduction

A *bottleneck Steiner tree* (BST) (also known as a *min-max Steiner tree*) is a Steiner tree with the length of the longest edge minimized. We consider the *Euclidean bottleneck Steiner tree problem* with unknown k Steiner points in $\mathbb{R}^2$, described as follows:

*Problem 1 (*EuclidBST*).* Given n points, called *terminals*, in the Euclidean plane and a positive integer k, find a Steiner tree spanning all terminals and at most k Steiner points that minimizes the length of the longest edge.

Unlike the classical Steiner tree problem where the total length of the Steiner tree is minimized, this problem asks a Steiner tree where the maximum of the edge lengths is minimized and the Steiner points in the resulting tree can be chosen in the whole plane $\mathbb{R}^2$. These problems, and their variations, have some known applications in VLSI layout [3], multifacility location, and wireless communication network design [11].

The EuclidBST problem is known to be NP-hard to approximate within ratio $\sqrt{2}$ [11]. The best known upper bound on approximation ratio is 1.866 by Wang and Li [12]. For the special case of this problem where there should be no edge connecting any two Steiner points in the optimal solution, Li et al. [7] present a $(\sqrt{2} + \epsilon)$-factor approximation algorithm with inapproximability within $\sqrt{2}$.

* Work by S.W. Bae was supported by the Brain Korea 21 Project. Work by C. Lee and S. Choi was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MOST) (No. R01-2007-000-20865-0).

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 105–116, 2009.

There has been some effort on devising an exact algorithm for finding the locations of k Steiner points. Of course, it is impossible to get a polynomial time algorithm unless P=NP since the problem is NP-hard. Thus, many researchers considered the following problem [5, 10].

*Problem 2 (*EuclidBST-FT*).* Given n terminals in the Euclidean plane and a topology tree T of n terminals and k Steiner points, find a bottleneck Steiner tree with topology T spanning all terminals.

Once we have an exact algorithm to this problem, we can find an exact solution to EuclidBST by enumerating all valid topology trees; this number is roughly bounded by $(n + k)!$ [5]. To the best of our knowledge, there is no known exact algorithm to EuclidBST even for a single Steiner point $k = 1$, or even to the EuclidBST-FT problem. The first algorithms for EuclidBST-FT can be found in Elzinga et al. [4] and Love et al. [8], which are based on nonlinear optimization but do not give an exact solution. The decision version of EuclidBST-FT asks whether there exists a Steiner tree with a given topology T such that its maximum edge length does not exceed a given parameter $\lambda > 0$. Sarrafzadeh and Wong [10] presented an $O((n + k) \log(n + k))$ time algorithm for this decision problem. Indeed, one can get a simple $(1 + \epsilon)$-approximation to EuclidBST-FT using the decision algorithm in a binary-search fashion [5].

In the rectilinear case where each edge of Steiner trees should be rectilinear, a quadratic-time exact algorithm for finding a bottleneck Steiner tree with a given topology is known by Ganley and Salowe [5]; they also described the difficulty of the Euclidean case.

In this paper, we present an $O(n \log n)$ time algorithm to the EuclidBST problem when $k = 1$ and an $O(n^2)$ time algorithm when $k = 2$. Our approach is rather from a geometric point of view: we reveal several useful properties of Euclidean bottleneck Steiner trees and optimal locations for Steiner points using geometric knowledge. Among them is an interesting connection between the optimal location of Steiner points and the *smallest color spanning disk*, a smallest disk containing at least one point of each color when we are given a set of colored points [1]. Also, we present an $O(n \log n)$ time algorithm for any constant k to a special case of EuclidBST where there is no edge between Steiner points; the hidden constant is essentially exponential in k. We remark that the running times of our algorithms are all polynomial in n; any exact solution to EuclidBST-FT seems hardly to yield an exact algorithm running in time polynomial in n due to the number of possible topologies [5]. Moreover, our observations can be naturally extended to any metric on $\mathbb{R}^2$; for example, the rectilinear case.

The paper is organized as follows: Section 2 presents an exact algorithm for a single Steiner point. Next, we show several helpful properties of optimal solutions for any k, and make use of them to devise the algorithms for the case without edges between Steiner points and for the case of $k = 2$ in Section 3. Finally, Section 4 concludes the paper.

2 Exact Algorithm for a Single Steiner Point

Let $P \subset \mathbb{R}^2$ be a set of n points; we call each point in P a *terminal*. A *(Euclidean) bottleneck spanning tree* of P is a spanning tree of P such that the length of a longest edge is minimized. We call the length of a longest edge in a bottleneck spanning tree of P the *bottleneck of the set* P, denoted by $b(P)$. Note that $b(P)$ is dependent only on the set P, not on how to connect the points in P. Our problem is to find the optimal location $q \in \mathbb{R}^2$ of a Steiner point such that the bottleneck $b(P \cup \{q\})$ of any bottleneck spanning tree of $P \cup \{q\}$ is minimized. We start with a *(Euclidean) minimum spanning tree* $MST(P)$ of given points P. Obviously, $MST(P)$ is a bottleneck spanning tree of P. Since computing a minimum or bottleneck spanning tree of a given set can be done easily, our problem can be viewed as finding an optimal location of q to minimize the bottleneck of $P \cup \{q\}$.

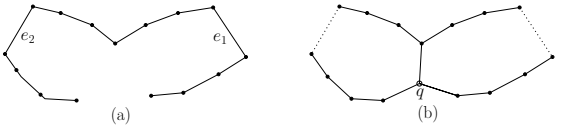


Fig. 1. (a) $MST(P)$ and (b) $MST(P \cup \{q\})$. By a single Steiner point q, the longest edge e_1 and the second longest edge e_2 are removed from $MST(P)$.

Let $e_1, \ldots, e_{n-1}$ be the edges of $MST(P)$ in the order that their lengths are not increasing. Obviously, $|e_1| = b(P) \geq b(P \cup \{q\})$, where $|e|$ denotes the length of an edge (or a segment) e. In order to have the strict inequality $b(P) > b(P \cup \{q\})$, we must be able to remove the longest edge e_1 of $MST(P)$ after adding q. Then, q would connect two points in P as a substitution of e_1. Sometimes, q with new edges incident to it can replace the second longest edge e_2 simultaneously with e_1. But it is obvious that effort for removing e_3 without removing e_2, or in general removing e_i without removing e_{i-1} is useless in reducing the bottleneck since the length of the longest edge is lower bounded by $|e_{i-1}| \geq |e_i|$. Thus, in $MST(P \cup \{q\})$, we lose all of $e_1, \ldots, e_c$ from $MST(P)$, where some positive integer $c < n$. Also, note that $MST(P)$ might have many longest edges, and thus all of them must be removed to have the strictly better bottleneck. The number does not exceed $n - 1$ but we could really have that many number of longest edges in $MST(P)$. Fortunately, the following observations allow us to focus on a constant number of edges to be removed.

Lemma 1. *There exists a bottleneck Steiner tree of P with a single Steiner point such that each edge belongs to $MST(P)$ or is incident to the Steiner point q.*

Proof. Suppose that a bottleneck Steiner tree $MST(P \cup \{q\})$ has an edge connecting two terminals $p_1, p_2 \in P$ which does not belong to $MST(P)$. Then, there exists a path π from p_1 to p_2 in $MST(P)$ which is not a single edge connecting p_1 and p_2. By optimality of $MST(P)$, the length of each edge on π is at most $|p_1 p_2|$. This implies that we can replace the edge between p_1 and p_2 in

$MST(P \cup \{q\})$ by the original path π connecting p_1 and p_2 in $MST(P)$ without increasing the length of the longest edge contained in the resulting Steiner tree. □

Due to Lemma 1, we assume that every newly added edge in our optimal solution $MST(P \cup \{q\})$ from $MST(P)$ is incident to q.

Lemma 2. *For any $q \in \mathbb{R}^2$, $b(P \cup \{q\}) \geq |e_5|$. Therefore, there exists a bottleneck Steiner tree of P with a single Steiner point such that it contains the edges $e_5, e_6, \ldots, e_{n-1}$ and thus the degree of the Steiner point is at most 5.*

Proof. Assume to the contrary that $b(P \cup \{q\}) < |e_5|$. This means that $MST(P \cup \{q\})$ does not contain $e_1, \ldots, e_5$ any more. This also implies that q has at least 6 incident edges in $MST(P \cup \{q\})$ by Lemma 1.

Now, we construct another tree T on $P \cup \{q\}$: We add e_j again to $MST(P \cup \{q\})$ for $j \geq 6$, if it has been removed from $MST(P)$, and delete the same number of edges incident to q to make the resulting graph a tree. Then, q has exactly 6 incident edges in T. Let $N = \{p_1, \ldots, p_6\}$ be the set of the 6 points in P, which are adjacent to q in T, in clockwise order at q. Observe that there exists an integer a with $1 \leq a \leq 6$ such that the angle $\angle p_a q p_{a+1}$ is at most $60°$. By simple trigonometry, $|p_a p_{a+1}|$ is at most $|qp_a|$ or $|qp_{a+1}|$, that is, $|p_a p_{a+1}| \leq \max\{|qp_a|, |qp_{a+1}|\}$. Also, note that we have $|p_i p_{i+1}| \geq |e_5|$ for any i, since each p_i belongs to a different subtree after removing the five longest edges $e_1, \ldots, e_5$ from $MST(P)$, and that $|qp_i| \leq b(P \cup \{q\}) < |e_5|$ by the assumption. Hence, we have

$$|e_5| \leq |p_a p_{a+1}| \leq \max\{|qp_a|, |qp_{a+1}|\} \leq b(P \cup \{q\}) < |e_5|,$$

a contradiction.

Furthermore, the equality holds only when the six points N form a regular hexagon centered at q. Hence, if q has six or more incident edges, we can just remove some while keeping the longest length in the resulting tree. Thus, the lemma is shown. □

By the above lemma, we remove at most four longest edges from $MST(P)$, and also newly added edges are all incident to q. Thus, we have only four possibilities: $e_1, \ldots, e_c$ are removed from $MST(P)$, where $c = 1, \ldots, 4$. Our algorithm is summarized as follows: (1) Remove $e_1, \ldots, e_c$ from $MST(P)$. Then, we have a forest containing $c+1$ subtrees $T_1, \ldots, T_{c+1}$. (2) Find a smallest disk containing at least one point from each subtree T_i. (This also minimizes the maximum length of edges incident to q.)

In the second step, we indeed ask a solution to the *smallest color spanning disk*: letting $C := \{1, 2, \ldots c+1\}$ be a color set, we color each point of T_i by color $i \in C$. This problem can be solved in $O(cn \log n)$ time by computing the *farthest color Voronoi diagram* [1, 6]: given a collection $\mathcal{C} = \{P_1, \ldots, P_c\}$ of c sets of colored points, define the *distance to a color $i \in C$* as $d(x, i) := \min_{p \in P_i} |xp|$. Then, the farthest color Voronoi diagram $FCVD(\mathcal{C})$ is the farthest Voronoi diagram of the colors under the distance to colors. Like the standard (Euclidean) farthest Voronoi diagram, $FCVD(\mathcal{C})$ has an application to finding

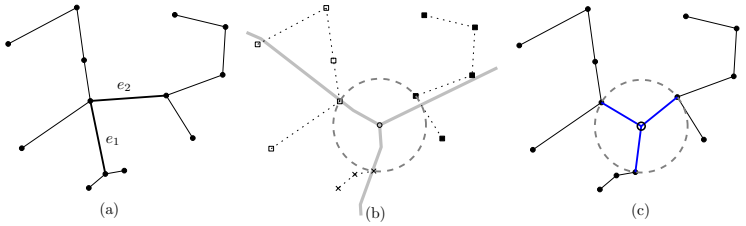


Fig. 2. Illustration of the algorithm for $c = 2$. (a) For a given set P of points. compute $MST(P)$ and remove e_1 and e_2. (b) Color T_i using different colors, and compute the FCVD (gray thick segments) to find the smallest color spanning disk (gray dashed circle). (c) Finally, locate the Steiner point at the center of the disk and add necessary edges to complete the resulting tree.

```
1: procedure 1-EuclidBST(P)                          ▷ P: a set of points in ℝ²
2:     Compute MST(P)                                ▷ minimum spanning tree
3:     Sort the edges of MST(P) by their lengths
4:     Let e_1, ..., e_{n-1} be the edges in the order
5:     for 1 ≤ c ≤ 4 do
6:         Remove e_1, ..., e_c from MST(P) to get c + 1 subtrees T_1, ..., T_{c+1}
7:         Compute FCVD(C) where C := {T_1, ..., T_{c+1}}
8:         Traverse FCVD(C) to find the center of a smallest color-spanning disk D
9:         if If the radius of D is smaller than |e_c| then
10:            q ← the center of D
11:        else
12:            return q
13:        end if
14:    end for
15:    return q
16: end procedure
```

Fig. 3. Exact algorithm to EuclidBST for $k = 1$

a *center* minimizing the maximum distance to all colors, which is exactly the center of a smallest color spanning disk. Also, such a center is located on a vertex or an edge of $FCVD(\mathcal{C})$. The combinatorial complexity of $FCVD(\mathcal{C})$ is known to be $O(cn)$, where n is the total number of points contained in $\mathcal{C}$.

More precisely, we regard each T_i as a set of points with color i: let $\mathcal{C} := \{T_i \mid 1 \leq i \leq c+1\}$, and build $FCVD(\mathcal{C})$ to compute the center of a smallest color spanning disk, which is the candidate of an optimal location of the Steiner point. Repeating this procedure for $c = 1, \ldots, 4$ gives us four candidates of an optimal location of the Steiner point.

Theorem 1. *Given a set P of n points in the plane, a Euclidean bottleneck Steiner tree with a single Steiner point can be computed in $O(n \log n)$ time with $O(n)$ space, and this time bound is worst-case tight in the algebraic decision tree model.*

Proof. The time and space complexity of the algorithm is easily checked. To prove the lower bound of the problem, consider the *maximum gap* problem:

*Problem 3 (*MaxGap*).* Given n real numbers in $\mathbb{R}$, find the maximum difference between two consecutive numbers when they are sorted.

This problem has an $\Omega(n \log n)$ lower bound in the algebraic decision tree model by Ben-Or [2]; a proof can be found in the book by Preparata and Shamos [9]. For any instance of the maximum gap problem, we transform the given numbers into the points in $\mathbb{R}^2$ along the x-axis. Any algorithm to the Euclidean bottleneck Steiner tree problem locates a Steiner point between two points with the maximum gap. Thus, two points adjacent to the Steiner point are from the two consecutive numbers defining the maximum gap. Hence, finding the optimal location of a single Steiner point needs at least $\Omega(n \log n)$ steps in the algebraic decision tree model. □

3 Towards Optimal Location of Multiple Steiner Points

In this section, we extend the above discussion for a single Steiner point to multiple $k > 1$ Steiner points. First, Lemma 2 extends to the following.

Lemma 3. *Let k be the number of allowed Steiner points in an instance of the Euclidean bottleneck Steiner tree problem. Then, $b(P \cup \{q_1, \ldots, q_k\}) \geq |e_{4k+1}|$ for any k points $q_1, \ldots, q_k \in \mathbb{R}^2$ and there exists a Euclidean bottleneck Steiner tree in which each Steiner point has degree at most 5.*

Proof. Our proof is by induction on k. The case when $k = 1$ is proven by Lemma 2. Suppose that there exists a smallest positive integer $k' > 1$ such that $b(P \cup Q) < |e_{4k+1}|$ where $Q = \{q_1, \ldots, q_{k'}\}$ is a set of optimal locations of k' Steiner points. Let T be the resulting Steiner tree of vertices $P \cup Q$. By the assumption, T does not contain $e_1, \ldots, e_{4k'+1}$ of $MST(P)$. Also, we can assume that each q_i is of degree at most 5 in T by Lemma 2: consider the problem instance where we are given $P \cup Q \setminus \{q_i\}$ as terminals and want to locate one Steiner point. On the other hand, consider the problem instance where we are given $P \cup \{q_i\}$ as terminals and we want to locate $k' - 1$ Steiner points. By the assumption, an optimal location of $k' - 1$ Steiner points removes at most $4k' - 4$ edges of $\{e_1, \ldots, e_{4k'+1}\}$. This means that q_i removes at least 5 edges among them and $T - q_i$ is a forest consisting of at least 6 subtrees, implying that the degree of q_i in T should be at least 6, a contradiction. Hence, there does not exist such k' and the lemma is true. □

In this section, we introduce the following problem, namely the *Euclidean bottleneck Steiner tree with fixed topology on subtrees*.

*Problem 4 (*EuclidBST-FT-ST*).* Given a set P of n points (terminals) in the plane, positive integers k and c with $c \leq 4k + 1$, and a topology tree $\mathcal{T}$ on the subtrees T_i and k Steiner points, find an optimal location of k Steiner points to obtain a Euclidean bottleneck Steiner tree with the given topology $\mathcal{T}$.

Let $V := \{v_1, \ldots, v_{c+1}, s_1, \ldots, s_k\}$ be the vertex set of $\mathcal{T}$, where v_i represents T_i and s_j represents a Steiner point. Each Steiner point does not have degree 1 but

can have degree 2 in the bottleneck Steiner tree [10]. Together with Lemma 3, we can restrict $\mathcal{T}$ so that each s_j has degree between 2 and 5.

3.1 A Special Case Without Edges between Two Steiner Points

The idea of the single Steiner point location can be used to solve a special case of the EuclidBST, where the resulting Steiner tree should have no edge between two Steiner points; for fixed c with $k \leq c \leq 4k$, we remove $e_1, \ldots, e_c$ from $MST(P)$ to get $c+1$ subtrees $T_1, \ldots, T_{c+1}$. Then, for a fixed topology $\mathcal{T}$ on all T_i and k Steiner points as vertices, each Steiner point s_j is located at the center of the smallest color spanning disk of $\{T_i \mid T_i \text{ adjacent to } s_j \text{ in } \mathcal{T}\}$. Hence, for a given c and a topology tree $\mathcal{T}$ as input of EuclidBST-FT-ST, we can find an optimal location of k Steiner points Q in $O(cn \log n)$, when we do not allow edges between Steiner points in $\mathcal{T}$.

For each $k \leq c \leq 4k$, enumerating all such topologies gives us an exact solution to EuclidBST. We now count the number of possible topologies $\mathcal{T}$ without edges between two Steiner points.

Lemma 4. *There are at most $O((4k)!k!/6^k)$ possible topology trees as input of* EuclidBST-FT-ST*, where there is no edge between two Steiner points, for any positive integers k and c with $k \leq c \leq 4k$.*

Proof. Since each Steiner point has degree at most 5 and $c \leq 4k$ by Lemma 3, we can roughly count the number of possible topologies by choosing 5 subtrees for each Steiner point. For the first Steiner point s_1, it has at most $\binom{4k+1}{5}$ number of possibilities to choose five subtrees to connect. For the second one s_2, one of the five subtrees chosen by s_1 should be connected to s_2 without loss of generality; thus the number of its possibilities is at most $5 \cdot \binom{4k-4}{4}$. In this way, s_i has $(4i-3) \cdot \binom{4(k-i)+4}{4}$ possibilities to choose five subtrees. Then, we have at most

$$\binom{4k+1}{5} \cdot 5 \binom{4k-4}{4} \cdots (4i-3) \binom{4(k-i)+4}{4} \cdots (4k-3) \binom{4}{4} = O((4k)!k!/6^k)$$

possible topology trees for any k and $c \leq 4k$, where there is no edge between Steiner points. ◻

Note that there was no known exact algorithm even for this special case of the Euclidean bottleneck Steiner tree problem. Thus, it is worth noting the following theorem.

Theorem 2. *There is an $O((4k)!k!k^2/6^k \cdot n \log n)$ time algorithm to compute an optimal Euclidean bottleneck Steiner tree with no edge between Steiner points when we are given n terminals in the plane and allow k Steiner points.*

Proof. As discussed above, EuclidBST-FT-ST can be solved in $O(cn \log n)$ time. And for fixed c, we have $O((4k)!k!/6^k)$ possible topology trees by Lemma 4 and it is easy to see that each enumeration can be done in $O(1)$ time. A rough calculation results in $\sum_{c=k}^{4k} O((4k)!k!/6^k \cdot cn \log n) = O((4k)!k!k^2/6^k \cdot n \log n)$. ◻

The running time appeared in Theorem 2 is exponential in k but polynomial in n. Thus, our algorithm runs in $O(n \log n)$ time for any constant k. This is remarkable since there was no known exact algorithm polynomial in n for the bottleneck Steiner tree problem for any k even for the rectilinear case.

In order to allow edges between Steiner points, we need more geometric observations. In the following subsection, we show some properties of optimal locations of Steiner points.

3.2 Properties of the Optimal Solutions

Consider an optimal location of k Steiner points and its resulting Euclidean Steiner tree; the optimal location of $s_i \in V$ is denoted by $q_i \in \mathbb{R}^2$ and the resulting Steiner tree is denoted by T^*, where $Q := \{q_1, \ldots, q_k\}$. Let N_i be the set of points in $P \cup Q$ that are adjacent to q_i in T^*, r_i be the length of the longest edge incident to q_i in T^*, and D_i be the disk centered at q_i with radius r_i. By local optimization, we can force each D_i to have two or three points in N_i on its boundary. Note that a disk is said to be *determined by three points* or *by two diametral points* if the three points lie on the boundary of the disk or if the two points define the diameter of the disk, respectively.

Lemma 5. *There exists an optimal location of k Steiner points, $q_1, \ldots, q_k$ such that (1) D_i is determined by two diametral points or by three points in N_i, each of which belongs to different components T_j and that (2) if there is a terminal $p \in T_j \cap N_i$ on the boundary of D_i, then there is no other terminal $p' \in T_j$ contained in the interior of D_i.*

Proof. We first prove that there exists an optimal solution satisfying (1). See Figure 4. Suppose that $q_1, \ldots, q_k$ are an optimal location of k Steiner points and that there is D_i which does not satisfy (1). Then, we have two possibilities: there are no three nor two points in N_i determining D_i, or two in $P \cap N_i$ determining D_i belong to the same T_j for some j. We can simply discard the latter case since it makes a cycle in the resulting Steiner tree; thus, this is the former case. If there is only one point $p \in N_i$ on the boundary of D_i, we can continuously move q_i in direction towards p while maintaining N_i to lie inside of the corresponding disk D_i, until D_i hits another point $p' \in N_i$. Observe that the radius of D_i decreases during this process, and two points p and p' lie on the boundary of D_i in the end.

Now, we can assume that we have at least two points p, p' on the boundary of D_i. If p and p' define the diameter of D_i, we are done. Hence, we are left with the case when the smaller angle $\angle pq_ip'$ at q_i is strictly less than 180°. We move q_i along the *bisecting line*[1] between p and p' in the direction that the smaller angle at q_i increases. Then, the smaller angle $\angle pq_ip'$ increases continuously and the radius of D_i decreases continuously, locally. We stop moving q_i when the boundary of D_i touches a third point $p'' \in N_i$ or when the angle $\angle pq_ip'$ reaches exactly 180°. Also, observe that the radius of D_i decreases during this second movement of q_i.

[1] The bisecting line between two points p and p' is the set of equidistant points in $\mathbb{R}^2$ from the two points, that is, the line $\{x \in \mathbb{R}^2 \mid |xp| = |xp'|\}$.

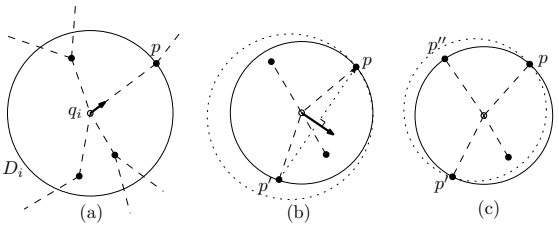


Fig. 4. Illustration for the proof of Lemma 5. Black dots are those in N_i around q_i. (a) When there is only one point $p \in N_i$ on the boundary of D_i, (b) we can move q_i in direction towards p so that the boundary of D_i touches another point p' and further (c) in direction along the bisecting line between p and p' so that three points p, p', and p'' determines D_i. During this process, the radius of D_i is subsequently decreasing.

From now, suppose that $q_1, \ldots, q_k$ is an optimal solution satisfying (1) but not (2), so that there is D_i such that $p, p' \in D_i \cap T_j$ and p lies on the boundary of D_i. Then, we can discard the farther one p from q_i and we connect q_i to p', instead, and perform the above process to make D_i satisfy (1). Therefore, the lemma is shown. □

Thus, we can assume that our optimal location Q of Steiner points satisfies the above properties. We call the two or the three points in N_i determining D_i the *determinators* of D_i. If the determinators of D_i are all terminals, the position of q_i is rather fixed by Lemma 5; we call such q_i *solid*. Otherwise, if at least one of the determinators of D_i is a Steiner point, then q_i is called *flexible*. The following is an immediate observation.

Lemma 6. *Suppose that there is an edge in T^* between two Steiner points $q_i, q_j \in Q$. If q_i is flexible and q_j is a determinator of D_i, then $r_j \geq r_i = |q_i q_j|$.*

3.3 Exact Algorithm for $k = 2$

Based on the properties observed above, we present an exact algorithm for computing the Euclidean bottleneck Steiner tree with two Steiner points.

Lemma 7. *An optimal location $\{q_1, q_2\}$ of two Steiner points fall into one of the following cases:*
(1) There is no edge between q_1 and q_2, that is, $q_1 \notin N_2$ and $q_2 \notin N_1$.
(2) $q_1 \in N_2$, and both q_1 and q_2 are solid.
(3) $q_1 \in N_2$, and one of q_1 and q_2 is solid and the other is flexible.
(4) $q_1 \in N_2$, and both q_1 and q_2 are flexible.

Given $c \leq 8$ and a topology tree $\mathcal{T}$ as above, if $\mathcal{T}$ has no edge between s_1 and s_2, then we can find an optimal solution as in Theorem 2. Thus, here we focus on Cases (2)–(4) of Lemma 7.

In the case when $\mathcal{T}$ has the edge between s_1 and s_2, other edges incident to s_1 or s_2 cover all the v_i in $\mathcal{T}$. Let $C := \{1, 2, \ldots, c+1\}$ and $C_j \subset C$ be defined as

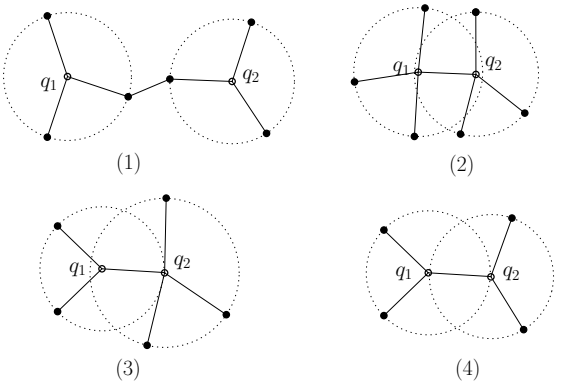


Fig. 5. Illustration to four cases in Lemma 7. Small black circles and black dots represent Steiner points and terminals, respectively. D_1 and D_2 are depicted as dotted circles centered at q_1 and q_2.

$C_j := \{i \in C \mid v_i \text{ adjacent to } s_j\}$, for $j = 1, 2$. Obviously, C_1 and C_2 are disjoint and $C_1 \cup C_2 = C$. Let $\mathcal{C}_j := \{T_i \mid i \in C_j\}$ be the two sets of colored points induced from the subtrees T_i.

Consider an optimal location $Q = \{q_1, q_2\}$. If q_1 is solid, then D_1 is a color spanning disk of $\{T_i \mid i \in C_1\}$ and is determined by two or three points from different T_i by Lemma 5; that is, q_1 lies on a vertex or an edge of the farthest color Voronoi diagram $FCVD(\mathcal{C}_1)$ of $\mathcal{C}_1$. (This process is almost the same as done for the case of $k = 1$.) Hence, the number of possible positions of q_1, provided that q_1 is solid, is $O(|C_1|n)$.

Note that finding the location q_1 of a solid Steiner point is independent from finding q_2 even if q_2 is flexible. Thus, even if q_1 is solid and q_2 is flexible, then we can just compute the exact location of q_1 as above, and next compute q_2 by adding one more colored point $\{q_1\}$ into $\mathcal{C}_2$ and performing the above procedure as if q_2 were solid. Hence, solutions in Cases (2) and (3) can be checked in $O(cn \log n)$ time.

What remains is the case when both q_1 and q_2 are flexible. In this case, we have $r_1 = r_2$ by Lemma 6 since q_1 is a determinator of D_2 and vice versa. The other determinators of D_1 except q_2 are of course terminals in P. There are two cases; D_1 has three determinators or two. In case of two terminals p_1 and p_2, q_1 lies on an edge of $FCVD(\mathcal{C}_1)$ determined by p_1 and p_2 since $|p_1q_1| = |p_2q_1| = r_1$. In the latter case where one terminal p lies on the boundary of D_1, q_1 is the midpoint on segment p_1q_2.

Thus, we can find an optimal location in Case (4) in $O(c^2n^2)$ time as follows:

1. Choose $p_1 \in T_i \in \mathcal{C}_1$ and $p_2 \in T_j \in \mathcal{C}_2$. Let x_1 and x_2 be two points on segment p_1p_2 such that $|p_1x_1| = |x_1x_2| = |x_2p_2| = \frac{1}{3}|p_1p_2|$. Test whether x_1 lies in the cell of T_i in $FCVD(\mathcal{C}_1)$ and x_2 lies in the cell of T_j in $FCVD(\mathcal{C}_2)$. If this test is passed, $\{x_1, x_2\}$ is a candidate of an optimal location $\{q_1, q_2\}$ of two Steiner points where both D_1 and D_2 have only one terminal on each of their boundaries.

2. Choose $p_1 \in T_i \in \mathcal{C}_1$ and an edge e from $FCVD(\mathcal{C}_2)$. Note that e is a portion of the bisecting line of two points $p \in T_j$ and $p' \in T_{j'}$ for some $j, j' \in C_2$. Then, we find a point $x_2 \in e$ such that $2|x_2p| = |x_2p_1|$, if any, and let x_1 be the midpoint on segment x_2p_1. Test whether x_1 lies in the cell of T_i in $FCVD(\mathcal{C}_1)$. If the test is passed, $\{x_1, x_2\}$ is a candidate of an optimal location $\{q_1, q_2\}$ of two Steiner points where D_1 has one terminal and D_2 has two on its boundary, respectively. The case where two terminals lie on the boundary of D_1 and one lies on the boundary of D_2 can be handled in a symmetric way.
3. Choose an edge e_1 from $FCVD(\mathcal{C}_1)$ and e_2 from $FCVD(\mathcal{C}_2)$. Let $p_1 \in T_i$ and $p'_1 \in T_{i'}$ for $i, i' \in C_1$ be the terminals such that e_1 is from the bisecting line between p_1 and p'_1. And let $p_2 \in T_j$ and $p'_2 \in T_{j'}$ for $j, j' \in C_2$ be the terminals such that e_2 is from the bisecting line between p_2 and p'_2. Then, find two points $x_1 \in e_1$ and $x_2 \in e_2$ such that $|p_1x_1| = |x_1x_2| = |x_2p_2|$, if any. Then, $\{x_1, x_2\}$ is a candidate of an optimal location $\{q_1, q_2\}$ of two Steiner points where two terminals lie on the boundary of each of D_1 and D_2.

Consequently, we can find an optimal location q_1, q_2 by examining all pairs of terminals and edges of $FCVD(\mathcal{C}_1)$ and $FCVD(\mathcal{C}_2)$. Since the complexity of these diagrams is bounded by $O(cn)$, $O(c^2n^2)$ time is sufficient to compute an optimal location of two Steiner points in Case (4) of Lemma 7. To find an optimal bottleneck Steiner tree with two Steiner points, we enumerate all possible topologies with two Steiner points and repeat the above process for each $1 \leq c \leq 8$. Finally, we conclude the following theorem.

Theorem 3. *Given a set P of n points in the plane, a Euclidean bottleneck Steiner tree with two Steiner points can be exactly computed in $O(n^2)$ time with $O(n)$ space.*

4 Concluding Remarks

We presented exact algorithms for the Euclidean bottleneck Steiner tree problem when the number of allowed Steiner points is one or two. In doing so, we revealed an interesting relation between the optimal location of Steiner points and the farthest color Voronoi diagram. Surprisingly, any exact algorithm to EuclidBST or EuclidBST-FT is still unknown for three or more Steiner points, even that with time super-polynomial in n or k. However, it seems not so simple to compute an optimal location of Steiner points when there are many flexible Steiner points. It might need more geometric or combinatorial observations on situations, which have not been discovered yet.

One remarkable fact is that our approach can be naturally extended to other metric spaces, such as the rectilinear case (the L_1 metric); our proofs are not very dependent on the Euclidean geometry. For example, in the rectilinear case, observe that there exists a rectilinear bottleneck Steiner tree where each Steiner point is of degree at most 7 and the farthest color Voronoi diagram with respect to the L_1 metric, which has been also well understood [6], could be used to build an algorithmic block as we did for the Euclidean case.

References

1. Abellanas, M., Hurtado, F., Icking, C., Klein, R., Langetepe, E., Ma, L., Palop, B., Sacristán, V.: The farthest color Voronoi diagram and related problems. Technical Report 002, Institut für Informatik I, Rheinische Fridrich-Wilhelms-Universität Bonn (2006)
2. Ben-Or, M.: Lower bounds for algebraic computation trees. In: Proc. 15th Annu. ACM Sympos. Theory Comput (STOC), pp. 80–86. ACM, New York (1983)
3. Chiang, C., Sarrafzadeh, M., Wong, C.: A powerful global router: based on Steiner min-max trees. In: Proc. IEEE Int. Conf. CAD, pp. 2–5 (1989)
4. Elzinga, J., Hearn, D., Randolph, W.: Minimax multifacility location with Euclidean distances. Transport. Sci. 10, 321–336 (1976)
5. Ganlet, J.L., Salowe, J.S.: Optimal and approximate bottleneck Steiner trees. Oper. Res. Lett. 19, 217–224 (1996)
6. Huttenlocher, D.P., Kedem, K., Shrir, M.: The upper envelope of Voronoi surfaces and its applications. Discrete Comput. Geom. 9, 267–291 (1993)
7. Li, Z.-M., Zhu, D.-M., Ma, S.-H.: Approximation algorithm for bottleneck Steiner tree problem in the Euclidean plane. J. Comput. Sci. Tech. 19(6), 791–794 (2004)
8. Love, R., Wesolowsky, G., Kraemer, S.: A multifacility minimax location problem with Euclidean distances. J. Prod. Res. 11, 37–45 (1973)
9. Preparata, F.P., Shamos, M.I.: Computational Geometry. Springer, Heidelberg (1985)
10. Sarrafzadeh, M., Wong, C.: Bottleneck Steiner trees in the plane. IEEE Trans. Comput. 41(3), 370–374 (1992)
11. Wang, L., Du, D.-Z.: Approximations for a bottleneck Steiner tree problem. Algorithmica 32, 554–561 (2002)
12. Wang, L., Li, Z.: An approximation algorithm for a bottleneck k-Steiner tree problem in the Euclidean plane. Inform. Process. Lett. 81, 151–156 (2002)

Colinear Coloring on Graphs*

Kyriaki Ioannidou and Stavros D. Nikolopoulos

Department of Computer Science, University of Ioannina
GR-45110 Ioannina, Greece
{kioannid, stavros}@cs.uoi.gr

Abstract. Motivated by the definition of linear coloring on simplicial complexes, recently introduced in the context of algebraic topology, and the framework through which it was studied, we introduce the colinear coloring on graphs. We provide an upper bound for the chromatic number $\chi(G)$, for any graph G, and show that G can be colinearly colored in polynomial time by proposing a simple algorithm. The colinear coloring of a graph G is a vertex coloring such that two vertices can be assigned the same color, if their corresponding clique sets are associated by the set inclusion relation (a clique set of a vertex u is the set of all maximal cliques containing u); the colinear chromatic number $\lambda(G)$ of G is the least integer k for which G admits a colinear coloring with k colors. Based on the colinear coloring, we define the χ-colinear and α-colinear properties and characterize known graph classes in terms of these properties.

Keywords: Colinear coloring, chromatic number, chordal graphs, threshold graphs, quasi-threshold graphs, algorithms, complexity.

1 Introduction

A *colinear coloring* of a graph G is a coloring of its vertices such that two vertices are assigned different colors, if their corresponding clique sets are not associated by the set inclusion relation; a *clique set* of a vertex u is the set of all maximal cliques in G containing u. The colinear chromatic number $\lambda(G)$ of G is the least integer k for which G admits a colinear coloring with k colors.

Motivated by the definition of linear coloring on simplicial complexes associated to graphs, first introduced by Civan and Yalçin [5] in the context of algebraic topology, we studied linear colorings on simplicial complexes which can be represented by a graph. In particular, we studied the linear coloring problem on a simplicial complex, namely independence complex $\mathcal{I}(G)$ of a graph G. The independence complex $\mathcal{I}(G)$ of a graph G can always be represented by a graph and, more specifically, is identical to the complement graph $\overline{G}$ of the graph G; indeed, the facets of $\mathcal{I}(G)$ are exactly the maximal cliques of $\overline{G}$. The outcome of this study was the definition of the colinear coloring of a graph G; the colinear

* This research is co-financed by E.U.-European Social Fund (75%) and the Greek Ministry of Development-GSRT (25%).

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 117–128, 2009.

coloring of a graph G is a coloring of G such that for any set of vertices taking the same color, the collection of their clique sets can be linearly ordered by inclusion. Note that, the two definitions cannot always be considered as identical since not in all cases a simplicial complex can be represented by a graph; such an example is the neighborhood complex $\mathcal{N}(G)$ of a graph G. Recently, Civan and Yalçin [5] studied the linear coloring of the neighborhood complex $\mathcal{N}(G)$ of a graph G and proved that the linear chromatic number of $\mathcal{N}(G)$ gives an upper bound for the chromatic number $\chi(G)$ of the graph G. This approach lies in a general framework met in algebraic topology.

In the context of algebraic topology, one can find much work done on providing boundaries for the chromatic number of an arbitrary graph G, by examining the topology of the graph through different simplicial complexes associated to the graph. This domain was motivated by Kneser's conjecture, which was posed in 1955, claiming that "if we split the n-subsets of a $(2n + k)$-element set into $k + 1$ classes, one of the classes will contain two disjoint n-subsets" [9]. Kneser's conjecture was first proved by Lovász in 1978, with a proof based on graph theory, by rephrasing the conjecture into "the chromatic number of Kneser's graph $KG_{n,k}$ is $k + 2$" [10]. Many more topological and combinatorial proofs followed the interest of which extends beyond the original conjecture [13]. Although Kneser's conjecture is concerned with the chromatic numbers of certain graphs (Kneser graphs), the proof methods that are known provide lower bounds for the chromatic number of any graph [11]. Thus, this initiated the application of topological tools in studying graph theory problems and more particularly in graph coloring problems [6].

The interest to provide boundaries for the chromatic number $\chi(G)$ of an arbitrary graph G through the study of different simplicial complexes associated to G, which is found in algebraic topology bibliography, drove the motivation for defining the colinear coloring on the graph G and studying the relation between the chromatic number $\chi(G)$ and the colinear chromatic number $\lambda(\overline{G})$. We show that for any graph G, $\lambda(\overline{G})$ is an upper bound for $\chi(G)$. The interest of this result lies on the fact that we present a colinear coloring algorithm that can be applied to any graph G and provides an upper bound $\lambda(\overline{G})$ for the chromatic number of the graph G, i.e. $\chi(G) \leq \lambda(\overline{G})$; in particular, it provides a proper vertex coloring of G using $\lambda(\overline{G})$ colors. Additionally, recall that a known lower bound for the chromatic number of any graph G is the clique number $\omega(G)$ of G, i.e. $\chi(G) \geq \omega(G)$. Motivated by the definition of perfect graphs, for which $\chi(G_A) = \omega(G_A)$ holds $\forall A \subseteq V(G)$, it was interesting to study those graphs for which the equality $\chi(G) = \lambda(\overline{G})$ holds, and even more those graphs for which this equality holds for every induced subgraph.

In this paper, we first introduce the colinear coloring of a graph G and study the relation between the colinear coloring of $\overline{G}$ and the proper vertex coloring of G. We prove that, for any graph G, a colinear coloring of $\overline{G}$ is a proper vertex coloring of G and, thus, $\lambda(\overline{G})$ is an upper bound for $\chi(G)$, i.e. $\chi(G) \leq \lambda(\overline{G})$. We present a colinear coloring algorithm that can be applied to any graph G. Motivated by these results and the Perfect Graph Theorem [7], we study those

graphs for which the equality $\chi(G) = \lambda(\overline{G})$ holds for every induce subgraph and characterize known graph classes in terms of the χ-colinear and the α-colinear properties. A graph G has the *χ-colinear property* if its chromatic number $\chi(G)$ equals to the colinear chromatic number $\lambda(\overline{G})$ of its complement graph $\overline{G}$, and the equality holds for every induced subgraph of G, i.e. $\chi(G_A) = \lambda(\overline{G}_A), \forall A \subseteq V(G)$; a graph G has the *α-colinear property* if its stability number $\alpha(G)$ equals to its colinear chromatic number $\lambda(G)$, and the equality holds for every induced subgraph of G, i.e. $\alpha(G_A) = \lambda(G_A), \forall A \subseteq V(G)$. Note that the *stability number* $\alpha(G)$ of a graph G is the greatest integer r for which G contains an independent set of size r. We show that the class of threshold graphs is characterized by the χ-colinear property and the class of quasi-threshold graphs is characterized by the α-colinear property.

2 Preliminaries

Let G be a finite undirected graph with no loops or multiple edges. We denote by $V(G)$ and $E(G)$ the vertex set and edge set of G. The subgraph of a graph G induced by a subset S of vertices of G is denoted by $G[S]$.

An edge is a pair of distinct vertices $x, y \in V(G)$, and is denoted by xy if G is an undirected graph and by $\overrightarrow{xy}$ if G is a directed graph. For a set $A \subseteq V(G)$ of vertices of the graph G, the subgraph of G *induced* by A is denoted by G_A. Additionally, the cardinality of a set A is denoted by $|A|$. The set $N(v) = \{u \in V(G) : uv \in E(G)\}$ is called the *open neighborhood* of the vertex $v \in V(G)$ in G, sometimes denoted by $N_G(v)$ for clarity reasons. The set $N[v] = N(v) \cup \{v\}$ is called the *closed neighborhood* of the vertex $v \in V(G)$ in G.

The greatest integer r for which a graph G contains an independent set of size r is called the *independence number* or otherwise the *stability number* of G and is denoted by $\alpha(G)$. The cardinality of the vertex set of the maximum clique in G is called the *clique number* of G and is denoted by $\omega(G)$. A *proper vertex coloring* of a graph G is a coloring of its vertices such that no two adjacent vertices are assigned the same color. The *chromatic number* $\chi(G)$ of G is the least integer k for which G admits a proper vertex coloring with k colors. For the numbers $\omega(G)$ and $\chi(G)$ of an arbitrary graph G the inequality $\omega(G) \leq \chi(G)$ holds. In particularly, G is a *perfect graph* if the equality $\omega(G_A) = \chi(G_A)$ holds $\forall A \subseteq V(G)$.

Next, definitions of some graph classes mentioned throughout the paper follow. A graph is called a *chordal graph* if it does not contain an induced subgraph isomorphic to a chordless cycle of four or more vertices. A graph is called a *co-chordal graph* if it is the complement of a chordal graph [7]. A hole is a chordless cycle C_n if $n \geq 5$; the complement of a hole is an antihole. A graph G is a *split graph* if there is a partition of the vertex set $V(G) = K + I$, where K induces a clique in G and I induces an independent set. Split graphs are characterized as $(2K_2, C_4, C_5)$-free. *Threshold graphs* are defined as those graphs where stable subsets of their vertex sets can be distinguished by using a single linear inequality. Threshold graphs were introduced by Chvátal and Hammer [4] and characterized

as $(2K_2, P_4, C_4)$-free. *Quasi-threshold* graphs are characterized as the (P_4, C_4)-free graphs and are also known in the literature as trivially perfect graphs [7], [12]. For more details on basic definitions in graph theory refer to [2], [7].

3 Colinear Coloring on Graphs

In this section we define the colinear coloring of a graph G, and we prove some properties of the colinear coloring of G. It is worth noting that these properties have been also proved for the linear coloring of the neighborhood complex $\mathcal{N}(G)$ in [5].

Definition 1. *Let G be a graph and let $v \in V(G)$. The clique set of a vertex v is the set of all maximal cliques of G containing v and is denoted by $\mathcal{C}_G(v)$.*

Definition 2. *Let G be a graph and let k be an integer. A surjective map $\kappa : V(G) \to \{1, 2, \ldots, k\}$ is called a k-colinear coloring of G if the collection $\{\mathcal{C}_G(v) : \kappa(v) = i\}$ is linearly ordered by inclusion for all $i \in \{1, 2, \ldots, k\}$, where $\mathcal{C}_G(v)$ is the clique set of v, or, equivalently, for two vertices $v, u \in V(G)$, if $\kappa(v) = \kappa(u)$ then either $\mathcal{C}_G(v) \subseteq \mathcal{C}_G(u)$ or $\mathcal{C}_G(v) \supseteq \mathcal{C}_G(u)$. The least integer k for which G is k-colinear colorable is called the colinear chromatic number of G and is denoted by $\lambda(G)$.*

Next, we study the colinear coloring on graphs and its association to the proper vertex coloring. In particular, we show that for any graph G the colinear chromatic number of $\overline{G}$ is an upper bound for $\chi(G)$.

Proposition 1. *Let G be a graph. If $\kappa : V(G) \to \{1, 2, \ldots, k\}$ is a k-colinear coloring of $\overline{G}$, then κ is a coloring of the graph G.*

Proof. Let G be a graph and let $\kappa : V(G) \to \{1, 2, \ldots, k\}$ be a k-colinear coloring of $\overline{G}$. From Definition 2, we have that for any two vertices $v, u \in V(G)$, if $\kappa(v) = \kappa(u)$ then either $\mathcal{C}_{\overline{G}}(v) \subseteq \mathcal{C}_{\overline{G}}(u)$ or $\mathcal{C}_{\overline{G}}(v) \supseteq \mathcal{C}_{\overline{G}}(u)$ holds. Without loss of generality, assume that $\mathcal{C}_{\overline{G}}(v) \subseteq \mathcal{C}_{\overline{G}}(u)$ holds. Consider a maximal clique $C \in \mathcal{C}_{\overline{G}}(v)$. Since $\mathcal{C}_{\overline{G}}(v) \subseteq \mathcal{C}_{\overline{G}}(u)$, we have $C \in \mathcal{C}_{\overline{G}}(u)$. Thus, both $u, v \in C$ and therefore $uv \in E(\overline{G})$ and $uv \notin E(G)$. Hence, any two vertices assigned the same color in a k-colinear coloring of $\overline{G}$ are not neighbors in G. Concluding, any k-colinear coloring of $\overline{G}$ is a coloring of G. □

It is therefore straightforward to conclude the following.

Corollary 1. *For any graph G, $\lambda(\overline{G}) \geq \chi(G)$.*

In Figure 1 we depict a colinear coloring of the well known graphs $2K_2$, C_4 and P_4, using the least possible colors, and show the relation between the chromatic number $\chi(G)$ of each graph $G \in \{2K_2, C_4, P_4\}$ and the colinear chromatic number $\lambda(\overline{G})$.

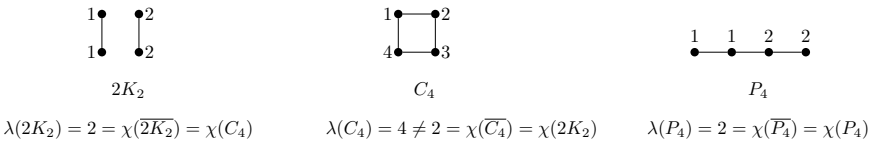


Fig. 1. Illustrating a colinear coloring of the graphs $2K_2$, C_4 and P_4 with the least possible colors

Proposition 2. *Let G be a graph. A coloring $\kappa : V(G) \to \{1, 2, \ldots, k\}$ of G is a k-colinear coloring of G if and only if either $N_G[u] \subseteq N_G[v]$ or $N_G[u] \supseteq N_G[v]$ holds in G, for every $u, v \in V(G)$ with $\kappa(u) = \kappa(v)$.*

Proof. Let G be a graph and let $\kappa : V(G) \to \{1, 2, \ldots, k\}$ be a k-colinear coloring of G. We will show that either $N_G[u] \subseteq N_G[v]$ or $N_G[u] \supseteq N_G[v]$ holds in G for every $u, v \in V(G)$ with $\kappa(u) = \kappa(v)$. Consider two vertices $v, u \in V(G)$, such that $\kappa(u) = \kappa(v)$. Since κ is a colinear coloring of G, we have either $\mathcal{C}_G(u) \subseteq \mathcal{C}_G(v)$ or $\mathcal{C}_G(u) \supseteq \mathcal{C}_G(v)$ holds. Without loss of generality, assume that $\mathcal{C}_G(u) \subseteq \mathcal{C}_G(v)$. We will show that $N_G[u] \subseteq N_G[v]$ holds in G. Assume the opposite. Thus, a vertex $z \in V(G)$ exists, such that $z \in N_G[u]$ and $z \notin N_G[v]$ and, thus, $zu \in E(G)$ and $zv \notin E(G)$. Now consider a maximal clique C in G which contains z and u. Since $zv \notin E(G)$, it follows that $v \notin C$. Thus, there exists a maximal clique C in G such that $C \in \mathcal{C}_G(u)$ and $C \notin \mathcal{C}_G(v)$, which is a contradiction to our assumption that $\mathcal{C}_G(u) \subseteq \mathcal{C}_G(v)$. Therefore, $N_G[u] \subseteq N_G[v]$ holds in G.

Let G be a graph and let $\kappa : V(G) \to \{1, 2, \ldots, k\}$ be a coloring of G. Assume now that either $N_G[u] \subseteq N_G[v]$ or $N_G[u] \supseteq N_G[v]$ holds in G, for every $u, v \in V(G)$ with $\kappa(u) = \kappa(v)$. We will show that the coloring κ of G is a k-colinear coloring of G. Without loss of generality, assume that $N_G[u] \subseteq N_G[v]$ holds in G, and we will show that $\mathcal{C}_G(u) \subseteq \mathcal{C}_G(v)$. Assume the opposite. Thus, a maximal clique C exists in G, such that $C \in \mathcal{C}_G(u)$ and $C \notin \mathcal{C}_G(v)$. Consider now a vertex $z \in V(G)$ $(z \neq v)$, such that $z \in C$ and $zv \notin E(G)$. Such a vertex exists since C is maximal in G and $C \notin \mathcal{C}_G(v)$. Thus, $zv \notin E(G)$ and either $zu \in E(G)$ or $z = u$, which is a contradiction to our assumption that $N_G[u] \subseteq N_G[v]$. □

4 An Algorithm for Colinear Coloring

In this section we present a polynomial time algorithm for colinear coloring which can be applied to any graph G, and provides an upper bound for $\chi(G)$. Although we have introduced colinear coloring through Definition 2, in our algorithm we exploit the property proved in Proposition 2, since the problem of finding all maximal cliques of a graph G is not polynomially solvable on general graphs. Before describing our algorithm, we first construct a directed acyclic graph (DAG) D_G of a graph G, which we call *DAG associated to the graph G*, and we use it in the proposed algorithm.

The DAG D_G associated to the graph G. Let G be a graph. We first compute the closed neighborhood $N_G[v]$ of each vertex v of G and, then, we

construct the following directed acyclic graph D, which depicts all inclusion relations among the vertices' closed neighborhoods: $V(D) = V(G)$ and $E(D) = \{\overrightarrow{xy} \ : \ x, y \in V(D) \text{ and } N_G[x] \subseteq N_G[y]\}$, where $\overrightarrow{xy}$ is a directed edge from x to y. In the case where the equality $N_G[x] = N_G[y]$ holds, we choose to add one of the two edges so that the resulting graph D is acyclic. To achieve this, we consider a partition of the vertex set $V(G)$ into the sets $S_1, S_2, \ldots, S_\ell$, such that for any $i \in \{1, 2, \ldots, \ell\}$ vertices x and y belong to a set S_i if and only if $N_G[x] = N_G[y]$. For vertices x and y belonging to the same set S_i we add the edge $\overrightarrow{xy}$ if and only if $x < y$. For vertices x and y belonging to different sets S_i and S_j respectively, we add the edge $\overrightarrow{xy}$ if and only if $N_G[x] \subset N_G[y]$. It is easy to see that the resulting graph D is unique up to isomorphism.

Additionally, it is easy to see that D is a transitive directed acyclic graph. Indeed, by definition D is constructed on a partially ordered set of elements $(V(D), \leq)$, such that for some $x, y \in V(D)$, $x \leq y \Leftrightarrow N_G[x] \subseteq N_G[y]$. Throughout the paper we refer to the constructed directed acyclic graph as the DAG associated to the graph G and denote it by D_G.

The algorithm for colinear coloring. The proposed algorithm computes a colinear coloring and the colinear chromatic number of a graph G. The algorithm works as follows:

(i) `compute` the closed neighborhood set of every vertex of G and, then, find the inclusion relations among the neighborhood sets and construct the DAG D_G associated to the graph G.
(ii) `find` a minimum path cover $\mathcal{P}(D_G)$, and its size $\rho(D_G)$, of the transitive DAG D_G (e.g. see [1],[8]).
(iii) `assign` a color $\kappa(v)$ to each vertex $v \in V(D_G)$, such that vertices belonging to the same path of $\mathcal{P}(D_G)$ are assigned the same color and vertices of different paths are assigned different colors; this is a surjective map $\kappa : V(D_G) \to [\rho(D_G)]$.
(iv) `return` the value $\kappa(v)$ for each vertex $v \in V(D_G)$ and the size $\rho(D_G)$ of the minimum path cover of D_G; κ is a colinear coloring of G and $\rho(D_G)$ equals the colinear chromatic number $\lambda(G)$ of G.

Correctness of the algorithm. Let G be a graph and let D_G be the DAG associated to the graph G, which is unique up to isomorphism. Consider the value $\kappa(v)$ for each vertex $v \in V(D_G)$ returned by the algorithm and the size $\rho(D_G)$ of a minimum path cover of D_G. We show that the surjective map $\kappa : V(D_G) \to [\rho(D_G)]$ is a colinear coloring of the vertices of G, and prove that the size $\rho(D_G)$ of a minimum path cover $\mathcal{P}(D_G)$ of the DAG D_G is equal to the colinear chromatic number $\lambda(G)$ of the graph G.

Proposition 3. *Let G be a graph and let D_G be the DAG associated to the graph G. A colinear coloring of the graph G can be obtained by assigning a particular color to all vertices of each path of a path cover of the DAG D_G. Moreover, the size $\rho(D_G)$ of a minimum path cover $\mathcal{P}(D_G)$ of the DAG D_G equals to the colinear chromatic number $\lambda(G)$ of the graph G.*

Proof. Let G be a graph, D_G be the DAG associated to G, and let $\mathcal{P}(D_G)$ be a minimum path cover of D_G. The size $\rho(D_G)$ of the DAG D_G, equals to the minimum number of directed paths in D_G needed to cover the vertices of D_G and, thus, the vertices of G. Now, consider a coloring $\kappa : V(D_G) \to \{1, 2, \ldots, k\}$ of the vertices of D_G, such that vertices belonging to the same path are assigned the same color and vertices of different paths are assigned different colors. Therefore, we have $\rho(D_G)$ colors and $\rho(D_G)$ sets of vertices, one for each color. For every set of vertices belonging to the same path, their corresponding closed neighborhood sets can be linearly ordered by inclusion. Indeed, consider a path in D_G with vertices $\{v_1, v_2, \ldots, v_m\}$ and edges $\overrightarrow{v_i v_{i+1}}$ for $i \in \{1, 2, \ldots, m\}$. From the construction of D_G, it holds that $\forall i, j \in \{1, 2, \ldots, m\}$, $\overrightarrow{v_i v_j} \in E(D_G) \Leftrightarrow N_G[v_i] \subseteq N_G[v_j]$. In other words, the corresponding neighborhood sets of the vertices belonging to a path in D_G are linearly ordered by inclusion. Thus, the coloring κ of the vertices of D_G gives a colinear coloring of G. This colinear coloring κ is optimal, uses $k = \rho(D_G)$ colors, and gives the colinear chromatic number $\lambda(G)$ of the graph G. Indeed, suppose that there exists a different colinear coloring $\kappa' : V(D_G) \to [k']$ of G using k' colors, such that $k' < k$. For every color given in κ', consider a set consisted of the vertices assigned that color. It is true that for the vertices belonging to the same set, their neighborhood sets are linearly ordered by inclusion. Therefore, these vertices can belong to the same path in D_G. Thus, each set of vertices in G corresponds to a path in D_G and, additionally, all vertices of G (and therefore of D_G) are covered. This is a path cover of D_G of size $\rho'(D_G) = k' < k = \rho(D_G)$, which is a contradiction since $\mathcal{P}(D_G)$ is a minimum path cover of D_G. Therefore, we conclude that the colinear coloring $\kappa : V(D_G) \to [\rho(D_G)]$ is optimal and, hence, $\rho(D_G) = \lambda(G)$. □

Complexity of the algorithm. Let G be a graph, $V(G) = n$, $E(G) = m$, and let D_G be the DAG associated to the graph G. Step (i) of the algorithm, which includes the construction of the DAG D_G, takes $O(nm)$ time. In particular, it takes $O(nm)$ time to compute the closed neighborhood set of every vertex of G, $O(nm)$ time to find the inclusion relations among the neighborhood sets, and $O(n+m)$ time to construct the DAG D_G. Note that, we only need to check pairs of vertices that are connected by an edge in G. Step (ii) computes a minimum path cover in the transitive DAG D_G; the problem is known to be polynomially solvable, since it can be reduced to the maximum matching problem in a bipartite graph formed from the transitive DAG [1]. The maximum matching problem in a bipartite graph takes $O((m + n)\sqrt{n})$ time, due to an algorithm by Hopcroft and Karp [8]. Finally, both Steps (iii) and (iv) can be executed in $O(n)$ time. Therefore, the complexity of the algorithm is $O(nm + n\sqrt{n})$.

5 Graphs Having the χ-Colinear and α-Colinear Properties

In Section 3 we showed that for any graph G, the colinear chromatic number $\lambda(\overline{G})$ of $\overline{G}$ is an upper bound for the chromatic number $\chi(G)$ of G, i.e. $\chi(G) \leq$

$\lambda(\overline{G})$. Recall that a known lower bound for the chromatic number of G is the clique number $\omega(G)$ of G, i.e. $\chi(G) \geq \omega(G)$. Motivated by the Perfect Graph Theorem [7], in this section we exploit our results on colinear coloring and we study those graphs for which the equality $\chi(G) = \lambda(\overline{G})$ holds for every induced subgraph. The outcome of this study was the definition of the following two graph properties and the characterization of known graph classes in terms of these properties.

- **χ-colinear property.** A graph G has the χ-colinear property if for every induced subgraph G_A of the graph G, $\chi(G_A) = \lambda(\overline{G}_A)$, $A \subseteq V(G)$.
- **α-colinear property.** A graph G has the α-colinear property if for every induced subgraph G_A of a graph G, $\alpha(G_A) = \lambda(G_A)$, $A \subseteq V(G)$.

Next, we show that the class of threshold graphs is characterized by the χ-colinear property and the class of quasi-threshold graphs is characterized by the α-colinear property. We also show that any graph that has the χ-colinear property is perfect; actually, we show that any graph that has the χ-colinear property is a co-chordal graph, and any graph that has the α-colinear property is a chordal graph. We first give some definitions and show some interesting results.

Definition 3. *The edge uv of a graph G is called actual if neither $N_G[u] \subseteq N_G[v]$ nor $N_G[u] \supseteq N_G[v]$. The set of all actual edges of G will be denoted by $E_\alpha(G)$.*

Definition 4. *A graph G is called quasi-threshold if it has no induced subgraph isomorphic to a C_4 or a P_4 or, equivalently, if it contains no actual edges.*

More details on actual edges and characterizations of quasi-threshold graphs through a classification of their edges can be found in [12]. The following result directly follows from Definition 3 and Proposition 2.

Proposition 4. *Let $\kappa : V(G) \rightarrow \{1, 2, \ldots, k\}$ be a k-colinear coloring of the graph G. If the edge $uv \in E(G)$ is an actual edge of G, then $\kappa(u) \neq \kappa(v)$.*

Based on Definition 3, the χ-colinear property and Proposition 5.1, we prove the following result.

Proposition 5. *Let G be a graph and let F be the graph such that $V(F) = V(G)$ and $E(F) = E(G) \cup E_\alpha(\overline{G})$. The graph G has the χ-colinear property if $\chi(G_A) = \omega(F_A)$, $\forall A \subseteq V(G)$.*

Proof. Let G be a graph and let F be a graph such that $V(F) = V(G)$ and $E(F) = E(G) \cup E_\alpha(\overline{G})$, where $E_\alpha(\overline{G})$ is the set of all actual edges of $\overline{G}$. By definition, G has the χ-colinear property if $\chi(G_A) = \lambda(\overline{G}_A)$, $\forall A \subseteq V(G)$. It suffices to show that $\lambda(\overline{G}_A) = \omega(F_A)$, $\forall A \subseteq V(G)$. From Definition 2, it is easy to see that two vertices which are not connected by an edge in $\overline{G}_A$ belong necessarily to different cliques and, thus, they cannot receive the same color in a colinear coloring of $\overline{G}_A$. In other words, the vertices which are connected by an

edge in G_A cannot take the same color in a colinear coloring of $\overline{G}_A$. Moreover, from Proposition 4 vertices which are endpoints of actual edges in $\overline{G}_A$ cannot take the same color in a colinear coloring of $\overline{G}_A$.

Next, we construct the graph F_A with vertex set $V(F_A) = V(G_A)$ and edge set $E(F_A) = E(G_A) \cup E_\alpha(\overline{G}_A)$, where $E_\alpha(\overline{G}_A)$ is the set of all actual edges of $\overline{G}_A$. Every two vertices in F_A, which have to take a different color in a colinear coloring of $\overline{G}_A$ are connected by an edge. Thus, the size of the maximum clique in F_A equals to the size of the maximum set of vertices which pairwise must take a different color in $\overline{G}_A$, i.e. $\omega(F_A) = \lambda(\overline{G}_A)$ holds for all $A \subseteq V(G)$. Concluding, G has the χ-colinear property if $\chi(G_A) = \omega(F_A)$, $\forall A \subseteq V(G)$. □

Taking into consideration Proposition 5 and the structure of the edge set $E(F) = E(G) \cup E_\alpha(\overline{G})$ of the graph F, it is easy to see that $E(F) = E(G)$ if $\overline{G}$ has no actual edges. Actually, this will be true for all induced subgraphs, since if G is a quasi-threshold graph then G_A is also a quasi-threshold graph for all $A \subseteq V(G)$. Thus, $\chi(G_A) = \omega(F_A)$, $\forall A \subseteq V(G)$. Therefore, the following result holds.

Corollary 2. *Let G be a graph. If $\overline{G}$ is quasi-threshold, then G has the χ-colinear property.*

From Corollary 2 we obtain a more interesting result.

Proposition 6. *Any threshold graph has the χ-colinear property.*

Proof. Let G be a threshold graph. It has been proved that an undirected graph G is a threshold graph if and only if G and its complement $\overline{G}$ are quasi-threshold graphs [12]. From Corollary 2, if $\overline{G}$ is quasi-threshold then G has the χ-colinear property. Concluding, if G is threshold, then $\overline{G}$ is quasi-threshold and thus G has the χ-colinear property. □

However, not any graph that has the χ-colinear property is a threshold graph. Indeed, Chvátal and Hammer [4] showed that threshold graphs are $(2K_2, P_4, C_4)$-free and, thus, the graphs P_4 and C_4 have the χ-colinear property but they are not threshold graphs (see Figure 1). We note that the proof that any threshold graph G has the χ-colinear property can be also obtained by showing that any coloring of a threshold graph G is a colinear coloring of $\overline{G}$ by using Proposition 2, the basic set theory property that $N_G(u) = V(G) \backslash N_{\overline{G}}[u]$, Corollary 1 and the property that $N(u) \subseteq N[v]$ or $N(v) \subseteq N[u]$ for any two vertices u, v of G. However, Proposition 5 and Corollary 2 actually give us a stronger result since the class of quasi-threshold graphs is a superclass of the class of threshold graphs.

The following result is even more interesting, since it shows that any graph that has the χ-colinear property is a perfect graph.

Proposition 7. *Any graph that has the χ-colinear property is a co-chordal graph.*

Proof. Let G be a graph that has the χ-colinear property. It has been showed that a co-chordal graph is $(2K_2, antihole)$-free [7]. To show that any graph G that has

the χ-colinear property is a co-chordal graph we will show that if G has a $2K_2$ or an *antihole* as induced subgraph, then G is does not have the χ-colinear property. Since by definition a graph G has the χ-colinear property if the equality $\chi(G_A) = \lambda(\overline{G}_A)$ holds for every induced subgraph G_A of G, it suffices to show that the graphs $2K_2$ and *antihole* do not have the χ-colinear property.

The graph $2K_2$ does not have the χ-colinear property, since $\chi(2K_2) = 2 \neq 4 = \lambda(C_4)$; see Figure 1. Now, consider the graph $G = \overline{C}_n$ which is an antihole of size $n \geq 5$. We will show that $\chi(G) \neq \lambda(\overline{G})$. It follows that $\lambda(\overline{G}) = \lambda(C_n) = n \geq 5$, i.e. if the graph $\overline{G} = C_n$ is to be colored colinearly, every vertex has to take a different color. Indeed, assume that a colinear coloring $\kappa : V(G) \rightarrow \{1, 2, \ldots, k\}$ of $\overline{G} = C_n$ exists such that for some $u_i, u_j \in V(G)$, $i \neq j$, $1 \leq i, j \leq n$, $\kappa(u_i) = \kappa(u_j)$. Since u_i, u_j are vertices of a hole, their neighborhoods in $\overline{G}$ are $N[u_i] = \{u_{i-1}, u_i, u_{i+1}\}$ and $N[u_j] = \{u_{j-1}, u_j, u_{j+1}\}$, $2 \leq i, j \leq n-1$. For $i = 1$ or $i = n$, $N[u_1] = \{u_n, u_2\}$ and $N[u_n] = \{u_{n-1}, u_1\}$. Since $\kappa(u_i) = \kappa(u_j)$, from Proposition 2 we obtain that one of the inclusion relations $N[u_i] \subseteq N[u_j]$ or $N[u_i] \supseteq N[u_j]$ must hold in $\overline{G}$. Obviously this is possible if and only if $i = j$, for $n \geq 5$; this is a contradiction to the assumption that $i \neq j$. Thus, no two vertices in a hole take the same color in a colinear coloring. Therefore, $\lambda(\overline{G}) = n$. It suffices to show that $\chi(G) < n$. It is easy to see that for the antihole $\overline{C}_n$, $deg(u) = n - 3$, for every vertex $u \in V(G)$. Brook's theorem [3] states that for an arbitrary graph G and for all $u \in V(G)$, $\chi(G) \leq max\{d(u)+1\} = (n-3)+1 = n-2$. Therefore, $\chi(G) \leq n-2 < n = \lambda(\overline{G})$. Thus the antihole $\overline{C}_n$ does not have the χ-colinear property.

We have showed that the graphs $2K_2$ and *antihole* do not have the χ-colinear property. It follows that any graph that has the χ-colinear property is $(2K_2, antihole)$-free and, thus, any graph that has the χ-colinear property is a co-chordal graph. □

Although any graph that has the χ-colinear property is co-chordal, the reverse is not always true. For example, the graph G in Figure 2 is a co-chordal graph but it does not have the χ-colinear property. Indeed, $\chi(G) = 4$ and $\lambda(\overline{G}) = 5$. It is easy to see that this graph is also a split graph. Moreover, not any graph that has the χ-colinear property is a split graph, since the graph C_4 has the χ-colinear property but it is not a split graph. However, there exist split graph which have the χ-colinear property; an example is the graph C_3. Recall that a graph G is a *split graph* if there is a partition of the vertex set $V(G) = K + I$, where K induces a clique in G and I induces an independent set; split graphs are characterized as $(2K_2, C_4, C_5)$-free graphs.

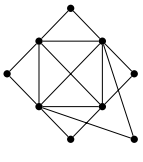
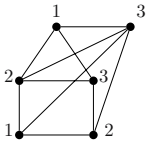
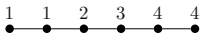


Fig. 2. A graph G which is a split graph but it does not have the χ-colinear property, since $\chi(G) = 4$ and $\lambda(\overline{G}) = 5$



$\overline{P_6}$ and $\chi(\overline{P_6}) = 3$



P_6 and $\lambda(P_6) = 4$

Fig. 3. Illustrating the graph $\overline{P_6}$ which does not have the χ-colinear property, since $\chi(\overline{P_6}) \neq \lambda(P_6)$

We have proved that graphs that satisfy the χ-colinear property do not contain a $2K_2$ or an *antihole*. Note that, since $\overline{C_5} = C_5$ and also the chordless cycle C_n is $2K_2$-free for $n \geq 6$, it is easy to see that graphs that have the χ-colinear property are *hole*-free. In addition, the graph $\overline{P_6}$ does not have the χ-colinear property (see Figure 3). Thus, we obtain the following result.

Proposition 8. *If a graph G satisfies the χ-colinear property, then G is a $(2K_2, antihole, \overline{P_6})$-free graph.*

Since graphs having the χ-colinear property are perfect, it follows that any graph G having the χ-colinear property satisfies $\chi(G_A) = \omega(G_A) = \alpha(\overline{G_A})$, $\forall A \subseteq V(G)$. Therefore, the following result holds.

Proposition 9. *A graph G has the α-colinear property if and only if the graph $\overline{G}$ has the χ-colinear property.*

From Corollary 1, and Proposition 9 we obtain the following characterization.

Proposition 10. *The graphs that are characterized by the α-colinear property are those graphs G for which the colinear chromatic number achieves its theoretical lower bound in every induced subgraph of G.*

From Corollary 2 and Proposition 9 we can obtain the following result.

Proposition 11. *Any quasi-threshold graph has the α-colinear property.*

From Propositions 8 and 9 we obtain that graphs that are characterized by the α-colinear property are $(C_4, hole, P_6)$-free. Therefore, the following result holds.

Proposition 12. *Any graph that has the α-colinear property is a chordal graph.*

Although any graph that has the α-colinear property is chordal, the reverse is not always true, i.e. not any chordal graph graph has the α-colinear property. For example, the complement $\overline{G}$ of the graph illustrated in Figure 2 is a chordal graph but it does not have the α-colinear property. Indeed, $\alpha(\overline{G}) = 4$ and $\lambda(\overline{G}) = 5$. It is easy to see that this graph is also a split graph. Moreover, not any graph having the α-colinear property is a split graph, since the graph $2K_2$ has the α-colinear property but it is not a split graph. However, there exist split graphs that have the α-colinear property; an example is the graph C_3.

6 Concluding Remarks

In this paper we introduced the colinear coloring on graphs, proposed a colinear coloring algorithm that can be applied to any graph G, and defined two graph properties, namely the χ-colinear and α-colinear properties. An interesting question would be to study the graphs that are characterized completely by the χ-colinear or the α-colinear property. In addition, it would be interesting to study the relation between the colinear chromatic number and other coloring numbers such as the harmonious number and the achromatic number on classes of graphs.

References

1. Boesch, F.T., Gimpel, J.F.: Covering the points of a digraph with point-disjoint paths and its application to code optimization. J. of the ACM 24, 192–198 (1977)
2. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. SIAM, Philadelphia (1999)
3. Brooks, R.L.: On colouring the nodes of a network. Proc. Cambridge Phil. Soc. 37, 194–197 (1941)
4. Chvátal, V., Hammer, P.L.: Aggregation of inequalities for integer programming. Ann. Discrete Math. I, 145–162 (1977)
5. Civan, Y., Yalçin, E.: Linear colorings of simplicial complexes and collapsing. J. Comb. Theory A 114, 1315–1331 (2007)
6. Csorba, P., Lange, C., Schurr, I., Wassmer, A.: Box complexes, neighborhood complexes, and the chromatic number. J. Comb. Theory A 108, 159–168 (2004)
7. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, New York (1980); annals of Discrete Mathematics, 2nd edn., vol. 57. Elsevier (2004)
8. Hopcroft, J., Karp, R.M.: A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J. Computing 2, 225–231 (1973)
9. Kneser, M.: Aufgabe 300. Jahresbericht der Deutschen Mathematiker-Vereinigung 58, 2 (1955)
10. Lovász, L.: Kneser's conjecture, chromatic numbers and homotopy. J. Comb. Theory A 25, 319–324 (1978)
11. Matoušek, J., Ziegler, G.M.: Topological lower bounds for the chromatic number: a hierarchy. Jahresbericht der Deutschen Mathematiker-Vereinigung 106, 71–90 (2004)
12. Nikolopoulos, S.D.: Recognizing cographs and threshold graphs through a classification of their edges. Inform. Proc. Lett. 74, 129–139 (2000)
13. Ziegler, G.M.: Generalised Kneser coloring theorems with combinatorial proofs. Inventiones mathematicae 147, 671–691 (2002)

Recursive Generation of 5-Regular Planar Graphs

Mahdieh Hasheminezhad[1], Brendan D. McKay[2,⋆], and Tristan Reeves[3,⋆⋆]

[1] Department of Computer Science
Faculty of Mathematics and Computer Science
Amirkabir University of Technology
Tehran, Iran
m.hashemi@aut.ac.ir
[2] Department of Computer Science
Australian National University
ACT 0200, Australia
bdm@cs.anu.edu.au
[3] Polytopia Systems Pty. Ltd.
19 Colbeck Street
Mawson, ACT, 2607, Australia

Abstract. We describe for the first time how the 5-regular simple planar graphs can all be obtained from an elementary family of starting graphs by repeatedly applying a few local expansion operations. The proof uses an innovative amalgam of theory and computation. By incorporating the recursion into the canonical construction path method of isomorph rejection, a generator of non-isomorphic embedded 5-regular planar graphs is obtained with time complexity $O(n^2)$ per isomorphism class.

Keywords: pentangulation, planar, graph, map, quintic, 5-regular, 5-valent.

1 Introduction

On account of Euler's formula, a simple planar graph has average degree less than 6, and therefore a regular simple planar graph can have degree at most 5. However, although much is known about the structure of 3-regular and 4-regular simple planar graphs, there is little literature on the 5-regular case.

Our aim in this paper is to present a recursive construction of all connected 5-regular simple planar graphs. The proof employs an innovative combination of human and computational logic.

By a *connected 5-regular simple planar graph (CSPG5)* we mean a connected 5-regular simple graph embedded on the sphere. We do not distinguish an outer face. The dual of a CSPG5 is a connected planar graph of minimum degree at least 3, with each face bounded by 5 edges, having the additional property that

⋆ Corresponding author.

⋆⋆ Research carried out at the Australian National University, 2003-4.

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 129–140, 2009.

no two faces share more than one edge of their boundaries. The dual is not necessarily simple.

Two planar graphs are regarded as the same if there is an embedding-preserving isomorphism (possibly reflectional) between them. That is, we are not concerned with abstract graph isomorphisms.

Let $\mathcal{C}$ be a class of planar graphs, $\mathcal{S}$ a subset of $\mathcal{C}$, and $\mathcal{F}$ a set of mappings from subsets of $\mathcal{C}$ to the power set $2^{\mathcal{C}}$. We say that $(\mathcal{S}, \mathcal{F})$ *recursively generates* $\mathcal{C}$ if for every $G \in \mathcal{C}$ there is a sequence $G_1, G_2, \ldots, G_k = G$ in $\mathcal{C}$ where $G_1 \in \mathcal{S}$ and, for each i, $G_{i+1} \in F(G_i)$ for some $F \in \mathcal{F}$. In many practical examples including that in this paper, there is some nonnegative integral graph parameter (such as the number of vertices) which is always increased by mappings in $\mathcal{F}$, and $\mathcal{S}$ consists of those graphs which are not in the range of any $F \in \mathcal{F}$. In this case, we refer to the elements of $\mathcal{F}$ as *expansions*, their inverses as *reductions*, and the graphs in $\mathcal{S}$ as *irreducible*. In this circumstance, $(\mathcal{S}, \mathcal{F})$ recursively generates $\mathcal{C}$ if every graph in $\mathcal{C} - \mathcal{S}$ is reducible.

Recursive generation algorithms for many classes of planar graphs have appeared in the literature. Expansions usually take the form of replacing some small subgraph by a larger subgraph. We mention the examples of 3-connected [12], 3-regular [5], minimum degree 4 [1], 4-regular [4,10], minimum degree 5 [3], and fullerenes (minimum degree 5 and maximum degree 6) [7]. Such construction theorems can be used to prove properties of graph classes by induction as well as to produce actual generators for practical use. Conspicuously missing from this list is the class of 5-regular planar graphs, which is somewhat harder than the others. We are aware only of a partial result [9]. In this paper we will fill this gap.

The numbers of isomorphism types of CSPG5s of order up to 36 appear in Table 1.

Table 1. Counts of connected 5-regular simple planar graphs of small order

vertices	faces	connectivity 1	connectivity 2	connectivity 3	total
12	20	0	0	1	1
14	23	0	0	0	0
16	26	0	0	1	1
18	29	0	0	1	1
20	32	0	0	6	6
22	35	0	0	14	14
24	38	0	2	96	98
26	41	0	11	518	529
28	44	5	113	3917	4035
30	47	53	1135	29821	31009
32	50	573	11383	240430	252386
34	53	5780	110607	1957382	2073769
36	56	55921	1054596	16166596	17277113

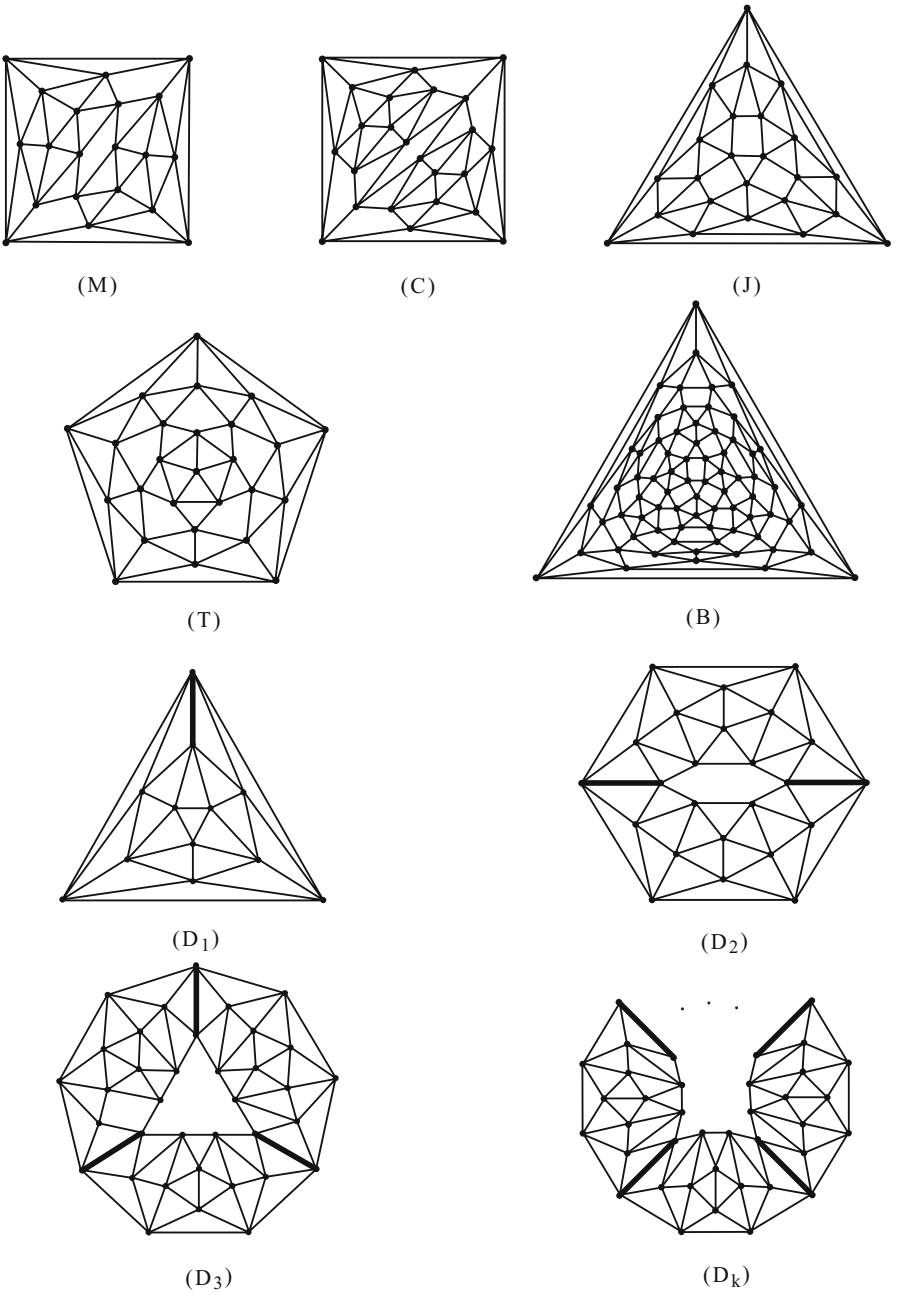


Fig. 1. Irreducible graphs, including the infinite sequence $D_1, D_2, \ldots$

Our starting set $\mathcal{S}$ consists of the 5 graphs M, C, J, T, B and the infinite family $\{D_i \mid i \geq 1\}$ described in Figure 1.

Our main result employs 6 expansions, each of which involve replacing a small subgraph by a larger subgraph. We define them via their corresponding reductions, as shown in Figure 2. In interpreting the figure, the following rules apply:

1. The 5-regular graph resulting from a reduction must be connected and simple.
2. The vertices of degree 1 shown in Figure 2 as $A, B, \ldots$ are defined by their cyclic order around the given vertices of degree 5. However they need not be distinct.
3. The vertices of degree 5 shown in Figure 2 have no adjacencies apart from those shown.
4. Each reduction includes its mirror image. (For example, the mirror image of A_2 inserts edges CD, BE, AF, HG.)

Let $\mathcal{F}$ be the set of expansions inverse to the reductions $\{A_1, A_2, B, C_1, C_2, C_3\}$ shown in Figure 2. We can now state our main result.

Theorem 1. *The class of all CSPG5s is generated by* $(\mathcal{S}, \mathcal{F})$.

To prove the theorem, we need to show that every CSPG5 not in $\mathcal{S}$ is reducible by one of the reductions $\mathcal{R} = \{A_1, A_2, B, C_1, C_2, C_3\}$. We abbreviate this to "$\mathcal{R}$-reducible". The structure of the remainder of the paper is as follows. In Section 2 we show that CSPG5s with a cut-vertex are $\mathcal{R}$-reducible, and in the following section we do the same for graphs of connectivity 2, partly with computer assistance. In Section 4, we first show that 3-connected CSPG5s with a separating 3-cycle are $\mathcal{R}$-reducible. Then we show that 3-connected CSPG5s not in $\{M, C, J, T, B, D_1, D_2\}$ are $\mathcal{R}'$-reducible, where $\mathcal{R}'$ is $\mathcal{R}$ with an additional reduction D added. Finally, we show that the extra reduction D is unnecessary if D_i $(i \geq 3)$ are added to the starting set. This will complete the proof of Theorem 1.

2 Cut-Vertices

In this section we consider the case that the graph has a cut-vertex. Reductions will be specified according to the labelling in Figure 2. In the case of A and B reductions, we can also specify the mirror image with a notation like, for example, $A_2^R(v, w)$.

Lemma 1. *Every CSPG5 with a cut-vertex is $\mathcal{R}$-reducible.*

Proof. Take such a cut-vertex v incident with an end-block. Three cases can occur, as illustrated in Figure 3(a–c), where the end-block is drawn on the left. In case (a), reduction $A_1(v, w)$ applies. (Multiple edges are impossible, and the reduced graph is connected because end-blocks are connected. We will generally omit such detail in our description.)

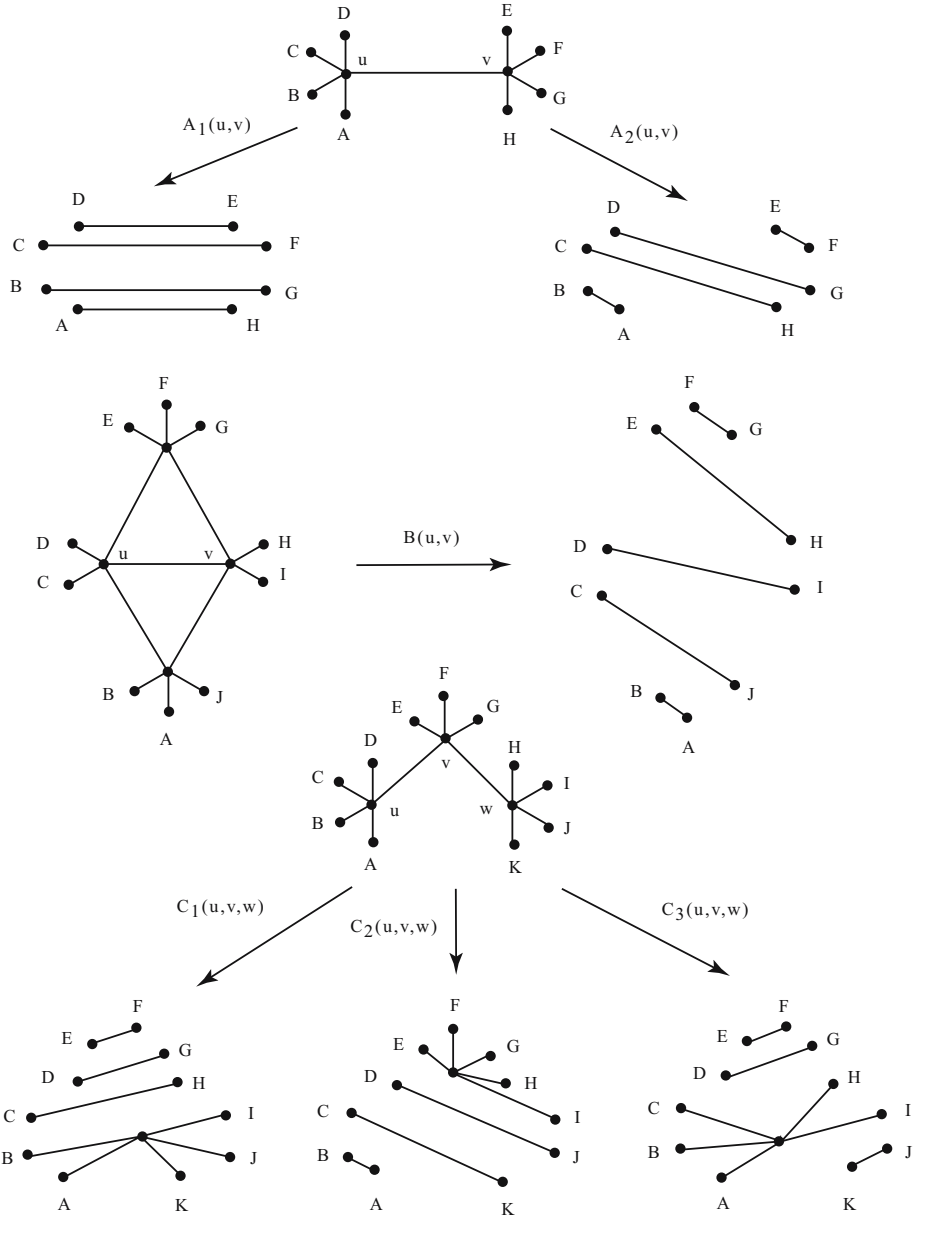


Fig. 2. Reductions A_1, A_2, B, C_1, C_2 and C_3 (subject to rules 1–4)

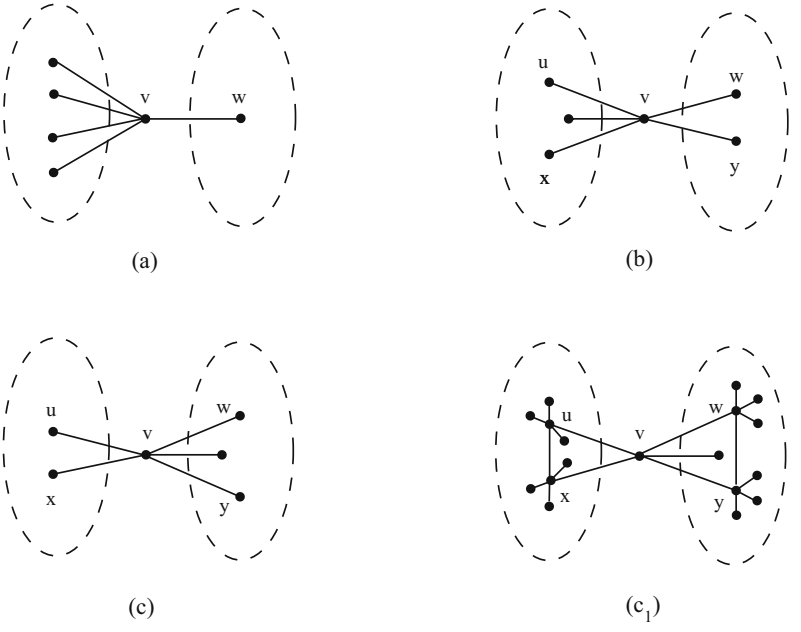


Fig. 3. Possible cases for a 1-cut

In case (c), either reduction $C_1(u, v, w)$ or $C_1(x, v, y)$ applies unless the situation in Figure 3(c_1) occurs. However, in this case $C_1(y, v, x)$ applies. In case (b), either reduction $C_1(u, v, w)$ or $C_1(x, v, y)$ applies. A situation mirror to that in Figure 3(c_1) cannot occur since the left side of v is an end-block.

3 2-Vertex Cuts

In this section we show that 2-connected CSPG5s with a 2-cut are $\mathcal{R}$-reducible.

Lemma 2. *Every 2-connected CSPG5 with a cut consisting of two edges or an edge and a vertex is $\mathcal{R}$-reducible.*

Proof. The two possibilities are illustrated in Figure 4.

In the case of cuts of two edges, we can apply $A_1(x, y)$ unless $x_i = w$ and $y_i = z$ for some i. If $x_1 = w$ and $y_1 = z$, $C_2(w, x, y)$ can be applied instead. If $x_2 = w$ and $y_2 = z$, either $A_2^R(w, x)$ or $C_3(x_1, x, w)$ can be applied. The other two cases are equivalent to these.

Now consider the case of a cut consisting of an edge and a vertex. If $y = d$, connectivity implies that $y_4 = v$. If $y_1 = e$ and $x = c$, the reduction $C_2(e, y, x)$ applies; otherwise $A_2(v, y)$ applies. If $y \neq d$, and also $y \neq e$ (which is equivalent), we consider whether x is the same as a, b or c. The case $x = b$ implies a two-edge cut, which is treated above, and $x = c$ is equivalent to $x = a$. If $x = a$, apply $C_1(e, v, c)$, while if $x \neq a, b, c$ apply $A_1(x, y)$.

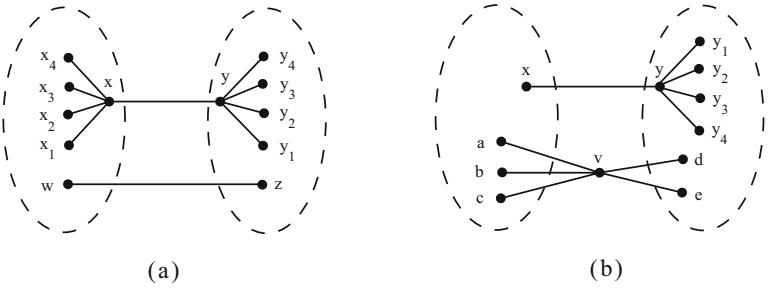


Fig. 4. Cuts of two edges or one edge and a vertex

Lemma 3. *Every 2-connected CSPG5 with a cut of two adjacent vertices is $\mathcal{R}$-reducible.*

Proof. The only possibility not covered by Lemma 2 is shown in Figure 5(a). We start by noting that $b = f$, $d = h$, $a = e$ or $c = g$ imply a 2-cut covered by Lemma 2. Therefore, either $C_1(a, x, b)$ or $C_1(b, x, a)$ applies unless either $a = g$ and $h = b$, or $d = f$ and $c = e$. We now divide the argument according to which of those two situations occurs.

If $a = g$, $h = b$, $d = f$ and $c = e$, either $A_2(c, x)$ or $A_2(c, y)$ applies. If $a = g$, $h = b$, $c \neq e$ and $d = f$, $C_3(a, y, b)$ or $C_3(a, x, b)$ applies.

Suppose $a = g$, $h = b$, $c \neq e$ and $d \neq f$, as shown in Figure 5(b). If $f_4 \neq d$, $C_1(f, y, x)$ applies. If $f_4 = d$ and $d_1 \neq b$, $C_3(f, y, x)$ applies, whereas if $f_4 = d$ and $f_1 \neq b$, $C_3(d, y, x)$ applies. Therefore, from now on we assume that $f_4 = d$ and $d_1 = f_1 = b$. If $d_2 \neq b_2$, $A_2(f, b)$ applies, whereas if $f_2 \neq b_2$, $A_2(d, b)$ applies. If $d_2 = f_2 = b_2$, $A_2(y, f)$ applies, and if $f_4 \neq d_4$ then $B(f, b)$ applies.

If $a \neq g$, $h = b$, $c = e$ and $d = f$, one of $A_2^R(x, d)$ and $A_2^R(y, d)$ applies.

The remaining case is that $a \neq g$, $h \neq b$, $c = e$ and $d = f$. One of $A_2(y, d)$, $A_2(x, d)$, $A_2(y, c)$ and $A_2(x, c)$ applies unless we have the situation that appears in Figure 5(c). Removal of the cut $\{x, y\}$ clearly results in exactly 2 components, so assume that (i) no other types of adjacent 2-cuts are present (since we already considered them above), and (ii) the component to the right of $\{x, y\}$ is a smallest of the components resulting from a cut of 2 adjacent vertices.

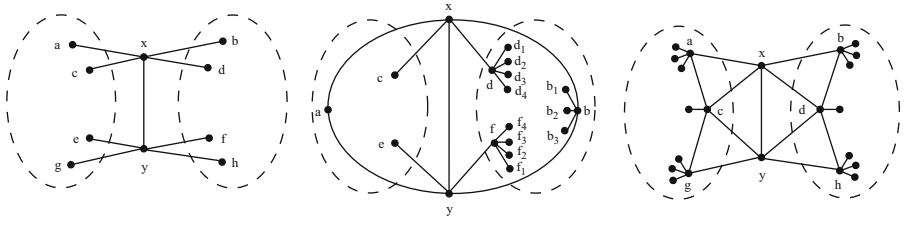


Fig. 5. Remaining case of cuts of 2 adjacent vertices

Although this case can be completed by hand, it is time-consuming and complicated so a computer program was employed. The initial configuration shown in the figure was expanded one vertex at a time. At each step, the program had an induced subgraph, some of whose vertices had additional edges whose other endpoint was not yet constructed. Such "incomplete edges" are distinct (since the subgraph is induced) but their endpoints might coincide. Expanding the subgraph consisted of choosing an incomplete edge (choosing the oldest on the right side of the cut proved a good heuristic), adding a new vertex to its incomplete end, then deciding all the additional adjacencies of the new vertex to the previous induced subgraph. Those sets of adjacencies that implied a 1-cut, a 2-cut of two adjacent vertices to the right of $\{x, y\}$, or a reduction in $\mathcal{R}$, were rejected. This expansion process finished after less than one second, never making an induced subgraph larger than 18 vertices. This completes the proof of the lemma.

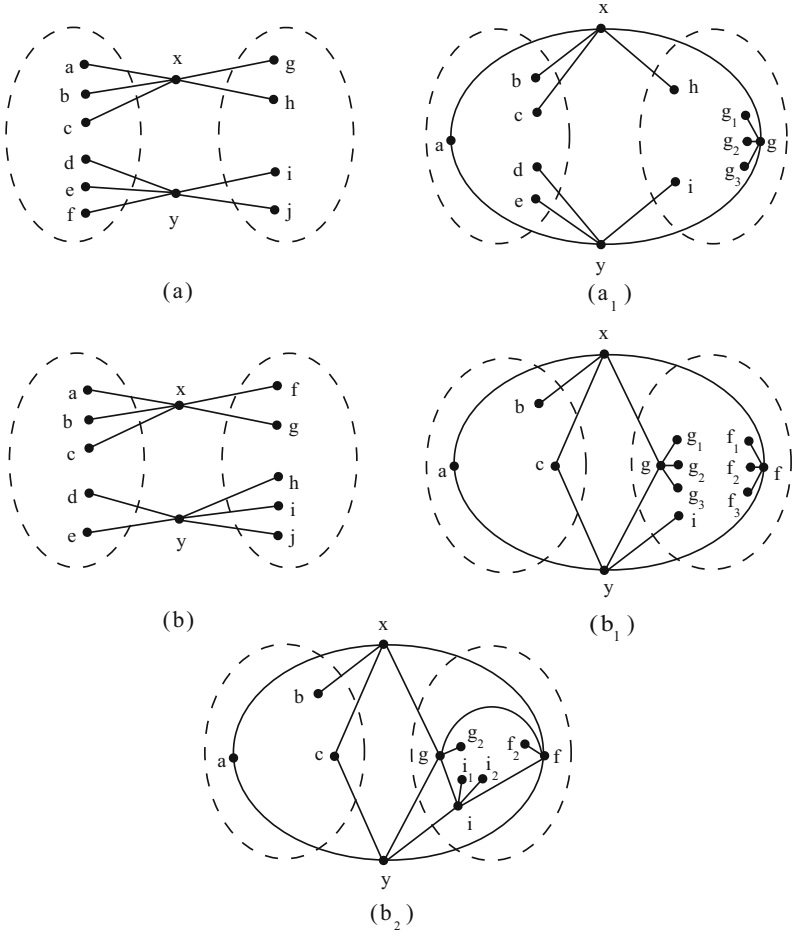


Fig. 6. Remaining cases of cuts of 2 non-adjacent vertices

Lemma 4. *Every 2-connected CSPG5 with a cut of two non-adjacent vertices is $\mathcal{R}$-reducible.*

Proof. The two cases not covered by Lemma 2 are shown in Figure 6(a,b).

Consider case (a) first. If $g \neq j$ or $a \neq f$, then $C_1(g, x, a)$ applies. If $g = j$ and $a = f$, we have the situation of Figure 6(a_1). If $g_3 \neq i$, $A_2(x, g)$ applies, whereas if $g_1 \neq h$, $A_2(y, g)$ applies. If $g_3 = i$ and $g_1 = h$, then $C_1(h, x, c)$ applies.

Now consider case (b). If $a \neq e$ or $f \neq j$, $C_1(f, x, a)$ applies, while if $a = e$, $f = j$ and either $c \neq d$ or $g \neq h$, $C_1(g, x, c)$ applies. This leaves the case that $a = e$, $f = j$, $c = d$ and $g = h$, as in Figure 6(b_1). In that case we find that $C_3(c, y, g)$ applies if $g_1 \neq f$, $A_2(x, f)$ applies if $g_1 = f$ and $i \neq f_3$, and $A_2^R(x, g)$ applies if $g_1 = f$, $i = f_3$ and $g_3 \neq i$. The remaining situation is as shown in Figure 6(b_2). We find that $A_2(y, i)$ applies if $g_2 \neq i_1$ and $C_3(g_2, g, f)$ applies of $g_2 = i_1$ (since $i_2 = f_2$ would imply that $\{i_2, f_2\}$ is a type of 2-cut that was already considered).

4 Completion of the Proof

A *separating 3-cycle* is a 3-cycle which is not the boundary of a face.

Lemma 5. *Every 3-connected CSPG5 with a separating 3-cycle is $\mathcal{R}$-reducible.*

Proof. By the symmetry between the inside and outside of the 3-cycle, two cases can occur as shown in Figure 7. In case (a), 3-connectivity requires x, y, z to be distinct, so $C_2(v, u, x)$ applies (if the reduced graph is disconnected then x is a cut-vertex). In case (b), 3-connectivity requires $y \neq z$. If $x \neq z$ and $t \neq y$, $C_2(w, v, y)$ applies, while if $x = z$ we must have $t \neq y$ by connectivity, so $A_2^R(u, x)$ applies.

To complete the proof of Theorem 1 we will first show that 3-connected CSPG5s without separating 3-cycles, other than a finite set of CSPG5s, are $\mathcal{R}'$-reducible. Here $\mathcal{R}' = \mathcal{R} \cup \{D\}$, where D is the additional reduction shown in Figure 8.

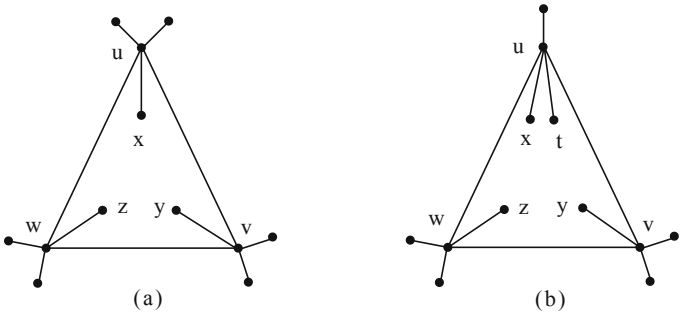


Fig. 7. Cases for a separating 3-cycle

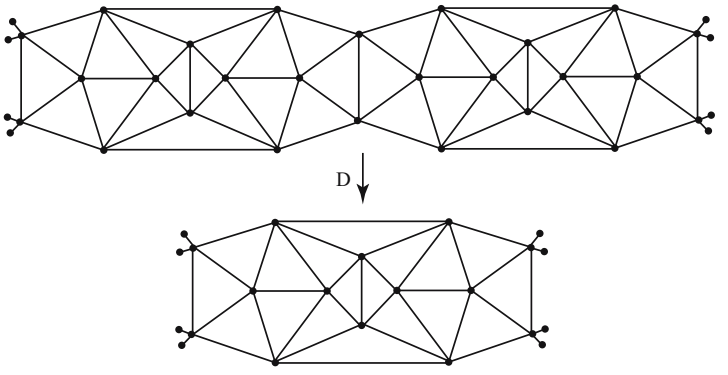


Fig. 8. The reduction D

Together with the results of Sections 2-4, this shows that all CSPG5s other than elements of $\mathcal{S}$ are $\mathcal{R}'$-reducible. We will then argue that reduction D is not actually required, thereby proving Theorem 1.

Lemma 6. *Every 3-connected CSPG5 is $\mathcal{R}'$-reducible, except for M, C, J, T, B, D_1, D_2.*

Proof. The proof of this lemma is tedious and was carried out by a computer program similar to that described in Lemma 3.

Since the average face size of a CSPG5 is greater than 3 (except for the dodecahedron), there is a face of size at least 4. Therefore, we grew induced subgraphs starting with a 4-face, and then starting with a path of 4 vertices on the boundary of a larger face. In the latter case, we forbade 4-faces since they were already covered by the former case. As the induced subgraph was grown, we rejected those that implied cuts of size less than 3, separating 3-cycles (on account of Lemma 5), or $\mathcal{R}'$-reductions.

It is easy to see that the result of applying an $\mathcal{R}'$-reduction to a 3-connected CSPG5 results in a connected graph. So in all cases the program does not need to verify connectivity.

The program completed execution in 21 seconds. In total, 39621 induced subgraphs were found which did not evidently have connectivity problems or $\mathcal{R}'$-reductions. These had at most 72 vertices. Of these subgraphs, 23 were regular but all of these were isomorphic to one of M, C, J, T, B, D_1, D_2. This completes the proof.

Proof (of Theorem 1). According to Lemmas 1–6, every CSPG5 is reducible by a reduction in $\mathcal{R}'$, except for the graphs M, C, J, T, B, D_1, D_2. Now consider the smallest CSPG5 G, not in the above list, that is not $\mathcal{R}$-reducible. Let G' be the result of reducing G by a D reduction. Since D reductions preserve regularity, simplicity and connectivity, G' is a CSPG5. Moreover, any reduction in $\mathcal{R}$ that applies to G' must also apply to G. Therefore, G' contradicts the minimality of G unless G' is one of M, C, J, T, B, D_1, D_2. Of these possibilities, only D_2 has the

configuration that results from a D reduction and the only graph which reduces to it is D_3. Arguing in the same manner produces the sequence $D_4, D_5, \ldots$. This completes the proof of the theorem.

As a partial check of the theorem, we found an $\mathcal{R}$-reduction for each of the 19.6 million graphs listed in Table 1, apart from the known irreducible graphs. These were made very slowly using a modified version of the program `plantri` [2]. The present, very much faster, algorithm will be incorporated into `plantri` in due course.

5 Concluding Remarks

Theorem 1 can be used in conjunction with the method of [11] to produce a generator of non-isomorphic simple 5-regular planar graphs. Briefly the method works as follows. For each graph G, one expansion is attempted from each equivalence class of expansions under the automorphism group of G. If the new larger graph is H, then H is accepted if the reduction inverse to the expansion by which H was constructed is equivalent under the automorphism group of H to a "canonical" reduction of H; otherwise it is rejected. The essential algorithmic requirements are computation of automorphism groups and canonical labelling, which can be done in linear time [6,8]. The number of reductions can be applied to one graph is clearly $O(n)$, so by [11, Theorem 3], the amortised time per output graph is at most $O(n^2)$. This does not reflect the likely practical performance; as with all the graph classes mentioned in [2], a careful use of heuristics is likely to make the amortised time per graph approximately constant within the range of sizes for which examination of all the graphs is plausible.

References

1. Batagelj, V.: An improved inductive definition of two restricted classes of triangulations of the plane. Combinatorics and Graph Theory, Banach Center Publications, 25, PWN - Polish Scientific Publishers, Warsaw, pp. 11–18 (1989)
2. Brinkmann, G., McKay, B.D.: Fast generation of planar graphs. MATCH Commun. Math. Comput. Chem. 58, 323–357 (2007), `http://cs.anu.edu.au/~bdm/plantri`
3. Brinkmann, G., McKay, B.D.: Construction of planar triangulations with minimum degree 5. Discrete Math. 301, 147–163 (2005)
4. Broersma, H.J., Duijvestijn, A.J.W., Göbel, F.: Generating all 3-connected 4-regular planar graphs from the octahedron graph. J. Graph Theory 17, 613–620 (1993)
5. Butler, J.W.: A generation procedure for the simple 3-polytopes with cyclically 5-connected graphs. Can. J. Math. 26, 686–708 (1974)
6. Fontet, M.: Linear algorithms for testing isomorphism of planar graphs. In: Proceedings Third Colloquium on Automata, Languages, and Programming, pp. 411–423 (1976)

7. Hasheminezhad, M., Fleischner, H., McKay, B.D.: A universal set of growth operations for fullerenes. Chem. Phys. Lett. 464, 118–121 (2008)
8. Hopcroft, J.E., Wong, J.K.: Linear time algorithm for isomorphism of planar graphs. In: 6th Annual ACM Symposium on Theory of Computing, Seattle, Washington, pp. 172–184 (1974)
9. Kanno, J., Kriesell, M.: A generating theorem for 5-regular simple planar graphs. I. Congr. Numerantium 185, 127–143 (2007)
10. Lehel, J.: Generating all 4-regular planar graphs from the graph of the octahedron. J. Graph Theory 5, 423–426 (1981)
11. McKay, B.D.: Isomorph-free exhaustive generation. J. Algorithms 26, 306–324 (1998)
12. Tutte, W.T.: A theory of 3-connected graphs. Nederl. Akad. Wetensch. Proc. Ser. A 64, 441–455 (1961)

Efficient Enumeration of Ordered Trees with k Leaves (Extended Abstract)

Katsuhisa Yamanaka[1], Yota Otachi[2], and Shin-ichi Nakano[2]

[1] Graduate School of Information Systems, The University of Electro-Communications, 1-5-1 Chofugaoka, Chofu, Tokyo 182-8585, Japan
yamanaka@is.uec.ac.jp
[2] Department of Computer Science, Gunma University, 1-5-1 Tenjin-cho, Kiryu, Gunma 376-8515, Japan
{otachi@comp.,nakano@}cs.gunma-u.ac.jp

Abstract. In this paper, we give a simple algorithm to generate all ordered trees with exactly n vertices including exactly k leaves. The best known algorithm generates such trees in $O(n - k)$ time for each, while our algorithm generates such trees in $O(1)$ time for each in worst case.

Keywords: graph, algorithm, ordered tree, enumeration, family tree.

1 Introduction

It is useful to have the complete list of objects for a particular class. One can use such a list to search for a counter-example to some conjecture, to find the best object among all candidates, or to experimentally measure an average performance of an algorithm over all possible inputs.

Many algorithms to generate all objects in a particular class, without repetition, are already known [1,2,11,13,12,15,16,17,20,26,28]. Many excellent textbooks have been published on the subject [4,6,10,25].

Trees are the most fundamental models frequently used in many areas, including searching for keys, modeling computation, parsing a program, etc. From the point of view, a lot of enumeration algorithms for trees are proposed [2,11,15,17,18,23,26], and a great textbook has been published by Knuth [7]. Also, enumeration algorithms for some subclasses of trees are known [5].

A *rooted* tree means a tree with one designated "root" vertex. Note that there is no ordering among the children of each vertex. Beyer and Hedetniemi [2] gave an algorithm to generate all rooted trees with n vertices. Their algorithm is the first one to generate all rooted trees in $O(1)$ time per tree on average, and based on the level sequence representation. Li and Ruskey [11] also gave an algorithm to generate all such trees, and showed that it was easily modified to generate restricted classes of rooted trees. The possible restrictions are (1) upper bound on the number of children, and (2) lower and upper bounds on the height of a rooted tree.

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 141–150, 2009.

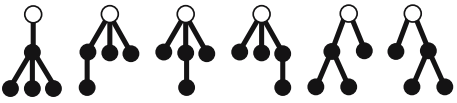


Fig. 1. All rooted ordered trees with 5 vertices including 3 leaves

A tree without the root vertex is called a *free tree*. Due to the absence of the root vertex, the generation of nonisomorphic free trees is a more difficult problem. Wright et al. [26], and Li and Ruskey [11] gave algorithms to generate all free trees in $O(1)$ time per tree on average, then Nakano and Uno [17] improved the running time to $O(1)$ time in worst case. Also they generalized the algorithm to generate all "colored" trees [18], where a colored tree is a tree in which each vertex has a color.

An *ordered* tree means a rooted tree with a left-to-right ordering specified for the children of each vertex. An algorithm to generate all ordered trees has been proposed by Nakano [15]. He also gave a method to generate non-rooted ordered trees in [15]. Sawada [23] handled enumeration problem for similar but different class of trees, called *circular*-ordered trees. A circular-ordered tree is a rooted tree with a circular ordering specified for the children of each vertex. Sawada [23] gave algorithms to generate circular-ordered trees and non-rooted ones in $O(1)$ time per tree on average.

In this paper, we wish to generate all ordered trees with exactly n vertices including exactly k leaves. See Fig. 1 for examples.

Let $S_{n,k}$ be the set of ordered trees with exactly n vertices including exactly k leaves. For instance there are six ordered trees with exactly 5 vertices including 3 leaves, as shown in Fig. 1 in which the root vertices are depicted by white circles, and $|S_{5,3}| = 6$. Such trees are one of the most natural subclasses of trees and are researched extensively, including enumeration [15,19], counting [24, p.237] and random generation [14].

The number of trees in $S_{n,k}$ is known as the Narayana number [24, p.237] as follows:

$$|S_{n,k}| = \frac{\binom{n-2}{k-1}\binom{n-1}{k-1}}{k}.$$

Two algorithms to generate all trees in $S_{n,k}$ are already known. Pallo [19] gave an algorithm to generate each tree in $S_{n,k}$ in $O(n-k)$ time on average. Also, Nakano's algorithm in [15] generates each tree in $S_{n,k}$ in $O(n-k)$ time on average.

By combining an algorithm to generate all ordered trees with specified degree sequence [8,9,22,29, etc], and a slightly modified version of an algorithm to generate all integer partitions into $(n-k)$ parts [3,21,27,30, etc], one can design an algorithm to generate all trees in $S_{n,k}$. Although such algorithm may generate each tree in $O(1)$ time in worst case, the algorithm is very complicated.

In this paper, we give a simple and efficient algorithm to generate all trees in $S_{n,k}$. Our algorithm generates each tree in $S_{n,k}$ in $O(1)$ time in worst case. The main idea of our algorithms is as follows. For some graph enumeration problems

(biconnected triangulation [12], triconnected triangulations [16], plane graphs [28] and ordered trees [15]) we can define a simple tree structure among the graphs, called the family tree, in which each vertex corresponds to each graph to be enumerated. In this paper, we design a cleverer family tree than the one in [15].

The rest of the paper is organized as follows. Section 2 gives some definitions. Section 3 defines the family tree among trees in $S_{n,k}$. Section 4 gives a simple algorithm to generate all trees in $S_{n,k}$.

2 Definitions

In this section, we give some definitions.

Let G be a connected graph with n vertices. In this paper, all graphs are unlabeled. The *degree* of a vertex v, denoted by $d(v)$, is the number of neighbors of v in G. A *tree* is a connected graph with no cycle. A *rooted* tree is a tree with one vertex r chosen as its *root*. For each vertex v in a rooted tree, let $UP(v)$ be the unique path from v to r. If $UP(v)$ has exactly k edges then we say *the depth* of v is k. *The parent* of $v \neq r$ is its neighbor on $UP(v)$, and *the ancestors* of $v \neq r$ are the vertices on $UP(v)$ except v. The parent of r and the ancestors of r are not defined. We say if v is the parent of u then u is a *child* of v, and if v is an *ancestor* of u then u is a *descendant* of v. A *leaf* is a vertex having no child. If a vertex is not a leaf, then it is called an *inner* vertex.

An *ordered tree* is a rooted tree with a left-to-right ordering specified for the children of each vertex. For an ordered tree T with the root r, let $LP(T) = (l_0(= r), l_1, l_2, \ldots, l_p)$ be the path such that l_i is the leftmost child of l_{i-1} for each i, $1 \leq i \leq p$, and l_p is a leaf of T. We call $LP(T)$ *the leftmost path* of T, and l_p *the leftmost leaf* of T. Similarly, let $RP(T) = (r_0(= r), r_1, r_2, \ldots, r_q)$ be the path such that r_i is the rightmost child of r_{i-1} for each i, $1 \leq i \leq q$, and r_q is a leaf of T. We call $RP(T)$ *the rightmost path* of T, and r_q *the rightmost leaf* of T.

3 The Family Tree

Let $S_{n,k}$ be the set of all ordered trees with exactly n vertices including exactly k leaves. In this section, we define a tree structure among the trees in $S_{n,k}$ in which each vertex corresponds to a tree in $S_{n,k}$.

We need some definitions.

The root tree, denoted by $R_{n,k}$, of $S_{n,k}$ is the tree consisting of the leftmost path $(l_0(= r), l_1, \ldots, l_{n-k})$ and $k-1$ leaves attaching at vertex l_{n-k-1}. See Fig. 2 for an example.

Then we define *the parent tree*, denoted by $P(T)$, of each tree T in $S_{n,k} \backslash \{R_{n,k}\}$ as follows. Let l_p and r_q be the leftmost leaf and rightmost leaf in T. We have two cases.

Fig. 2. The root tree $R_{7,4}$

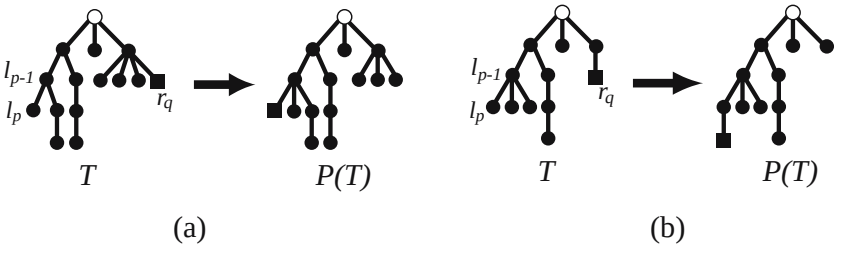


Fig. 3. Examples of the parents in (a) Case 1 and (b) Case 2

Case 1: r_{q-1} has two or more children.

$P(T)$ is the tree obtained from T by (1) removing r_q, then (2) attaching a new leaf to l_{p-1} as the leftmost child of l_{p-1}. See Fig. 3(a) for an example. The removed and attached vertices are depicted by boxes.

Case 2: r_{q-1} has only one child r_q.

$P(T)$ is the tree obtained from T by (1) removing r_q, then (2) attaching a new leaf to l_p. See Fig. 3(b) for an example.

Note that $P(T)$ is also in $S_{n,k}$.

T is called a *child tree* of $P(T)$. If T is a child tree in Case 1, then T is called *Type 1 child*, otherwise, T is *Type 2 child*.

Lemma 1. *For any* $T \in S_{n,k} \setminus \{R_{n,k}\}$, $P(T) \in S_{n,k}$ *holds.*

Given a tree T in $S_{n,k} \setminus \{R_{n,k}\}$, by repeatedly finding the parent tree of the derived tree, we can have the unique sequence $T, P(T), P(P(T)), \ldots$ of trees in $S_{n,k}$ which is called *the removing sequence* of T. See Fig. 4 for an example. in which each solid line corresponds to the relation with Case 1, and each dashed line corresponds to the relation with Case 2.

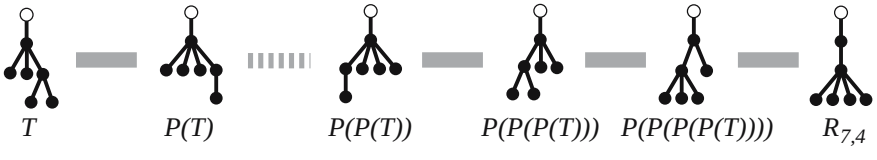


Fig. 4. The sequence of trees

Lemma 2. *The removing sequence ends up with the root tree $R_{n,k}$.*

Proof. Let T be a tree in $S_{n,k} \setminus \{R_{n,k}\}$. Let $LP(T) = (l_0, l_1, \ldots, l_p)$ be the leftmost path of T. We define two functions $f(T)$ and $g(T)$ as follows. We define that $f(T) = |LP(T)|$. Let $c_1, c_2, \ldots, c_a$ be the children of l_{p-1} from left to right. We choose the minimum i such that c_i is an inner vertex. Then we define that $g(T) = i - 1$. For convenience, if there is no such vertex, then we define that $g(T) = a$. Note that $1 \leq f(T) \leq n - k + 1$ and $1 \leq g(T) \leq k$ for any T in $S_{n,k}$.

Now we define a potential function $p(T) = (f(T), g(T))$. It is not difficult to see that $p(T) = (n - k + 1, k)$ if and only if $T = R_{n,k}$. Suppose that T_1 and T_2 are two trees in $S_{n,k}$ such that $T_1 \neq T_2$. We denote $p(T_1) < p(T_2)$ if (1) $f(T_1) < f(T_2)$ or (2) $f(T_1) = f(T_2)$ and $g(T_1) < g(T_2)$.

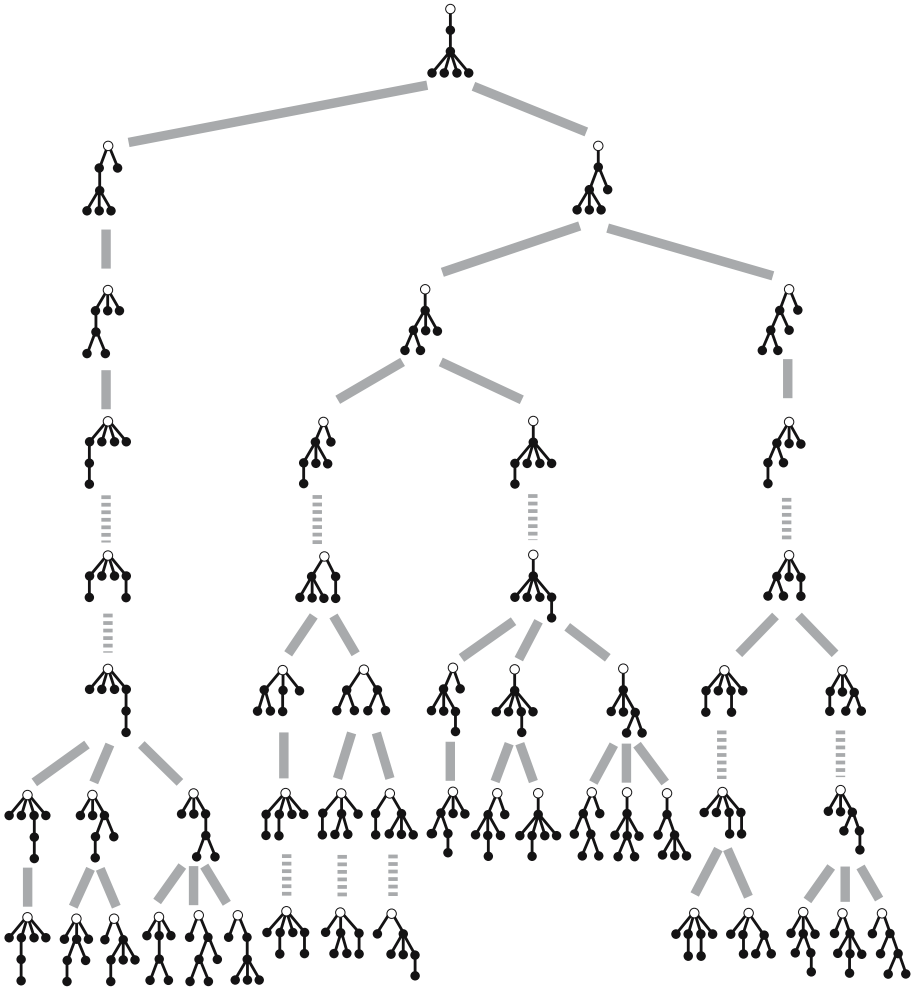


Fig. 5. The family tree $T_{7,4}$

Next we show that $p(T) < p(P(T))$. Suppose T is a Type 1 child of $P(T)$ (see Fig. 3(a)). In this case, we have $f(T) = f(P(T))$ and $g(T) + 1 = g(P(T))$. Thus $p(T) < p(P(T))$ holds. If T is a Type 2 child of $P(T)$, we always have $f(T) + 1 = f(P(T))$. Thus $p(T) < p(P(T))$ holds.

Therefore, by repeatedly finding the parent of the derived tree, we eventually obtain $R_{n,k}$ on which the potential is maximized. This completes the proof. □

By merging removing sequences we can have *the family tree* $T_{n,k}$ of $S_{n,k}$ such that the vertices of $T_{n,k}$ correspond to the trees in $S_{n,k}$ and each edge correspond to the relation between some T and $P(T)$. See Fig. 5 for an example.

4 Algorithm

Let $S_{n,k}$ be the set of ordered trees with exactly n vertices including exactly k leaves. This section gives our algorithm to generate all trees in $S_{n,k}$ by traversing $T_{n,k}$.

Given $S_{n,k}$ we can construct $T_{n,k}$ by the definition, possibly with huge space and much running time. However, how can we construct $T_{n,k}$ efficiently only given two integers n, k? Our idea [12,15,16,28] is by reversing the procedure finding the parent tree as follows.

If $k = 1$, $S_{n,k}$ includes only one element which is the path with $n - 1$ edges, then generation is trivial. Also if $k = n - 1$, $S_{n,k}$ includes only the star of n vertices. Therefore, from now on we assume $1 < k < n - 1$.

Let $T \in S_{n,k}$. Let $LP(T) = (l_0(= r), l_1, \ldots, l_p)$ be the leftmost path of T, and l_p the leftmost leaf of T. Let $RP(T) = (r_0(= r), r_1, \ldots, r_q)$ be the rightmost path of T, and r_q the rightmost leaf of T. We denoted by $T[r_i]$, $0 \leq i \leq q$, the tree obtained from T by (1) removing the leftmost leaf and (2) attaching a new leaf to r_i as the rightmost child of r_i.

Now we explain an algorithm to generate all child trees of the given tree T in $S_{n,k}$. We have the following two cases.

Case 1: T is the root tree $R_{n,k}$.

Each $T[r_i]$, $0 \leq i \leq q - 2$, is a child of T, since $P(T[r_i]) = T$. Since $T[r_{q-1}]$ is isomorphic to the root tree $R_{n,k}$ in $S_{n,k}$, $T[r_{q-1}]$ is not a child tree of T. Since $P(T[r_q]) \neq T$, $T[r_q]$ is not a child of T.

Thus T has $q - 1$ of Type 1 children and no Type 2 child.

Case 2: T is not the root tree.

If l_{p-1} has two or more children, and the second child of l_{p-1} from left is not a leaf, then T has no child tree, since if T is the parent of some tree then the second child of l_{p-1} from left is a leaf (Case 1), or l_{p-1} has only one child and it is a leaf (Case 2). See Fig. 3. Now we assume otherwise. We have the following two subcases.

Case 2(a): l_{p-1} has two or more children.

Let T' be the tree obtained from T by removing l_p. Then T' has $k - 1$ leaves. Thus we should add a new vertex to T' so that in the resulting graph the number of leaves increases by one. The detail is as follows.

Each $T[r_i]$, $0 \leq i \leq q-1$, is a child tree of T, since $P(T[r_i]) = T$. On the other hand, $T[r_q]$ is not a child tree of T, since $P(T[r_q]) \neq T$.

Thus T has q of Type 1 children and no Type 2 child.

Case 2(b): l_{p-1} has only one child, which is l_p.

Any $T[r_i]$, $0 \leq i \leq q-1$, is not a child tree of T. $T[r_q]$ is the child tree of T. Thus T has exactly one Type 2 child.

The above case analysis gives the following algorithm.

```
    Procedure find-all-child-trees(T)
    begin
01    Output T
      {Output the difference from the previous tree.}
02    Let l_p and r_q be the leftmost leaf and the rightmost leaf of T.
03    Let RP(T) = (r_0(= r), r_1, r_2, ..., r_q) be the rightmost path of T.
04    if l_{p-1} has two or more children and the second child of l_{p-1} from left
      is not a leaf then
05      return
06    if l_{p-1} has two or more children then
07      for i = 0 to q - 1
08        find-all-child-trees(T[r_i])     {Case 2(a)}
09    else
10      find-all-child-trees(T[r_q])     {Case 2(b)}
    end

    Algorithm find-all-trees(n, k)
    begin
01    Output R_{n,k}
02    T = R_{n,k}
03    Let RP(T) = (r_0(= r), r_1, r_2, ..., r_q) be the rightmost path.
04    for i = 0 to q - 2
05      find-all-child-trees(T[r_i]) {Case 1}
    end
```

We have the following theorem.

Theorem 1. *The algorithm uses $O(n)$ space and runs in $O(|S_{n,k}|)$ time, where $|S_{n,k}|$ is the number of ordered trees with exactly n vertices including exactly k leaves.*

Proof. To construct $T[r_i]$ from T, our algorithm needs the references to the leftmost leaf l_p and the rightmost path of T. Each can be updated as follows. In Case 2(a) the second child of l_{p-1} from left becomes the leftmost leaf of $T[r_i]$. In Case 2(b) the parent l_{p-1} of l_p becomes the leftmost leaf of $T[r_i]$. In both cases the rightmost path is updated to the path from the newly added vertex to its root. Thus we can maintain in $O(1)$ time the leftmost leaf and the rightmost path. □

The algorithm generates all trees in $S_{n,k}$ in $O(|S_{n,k}|)$ time. Thus the algorithm generates each tree in $O(1)$ time "on average." However, after generating a tree corresponding to the last vertex in a large subtree of $T_{n,k}$, we have to merely return from the deep recursive call without outputting any tree. This may take much time. Therefore, the next tree cannot be generated in $O(1)$ time in worst case.

However, a simple modification [17] improves the algorithm to generate each tree in $O(1)$ time in worst case. The algorithm is as follows.

Procedure find-all-children2(T, *depth*)
{ T is the current tree, and *depth* is the depth of the recursive call.}
begin
01 **if** *depth* is even **then**
02 Output T {before outputting its child trees.}
03 Generate child trees by the method in the first algorithm, and recursively call **find-all-children2** for each child tree.
04 **if** *depth* is odd **then**
05 Output T {after outputting its child trees.}
end

One can observe that the algorithm generates all trees so that each tree can be obtained from the preceding one by tracing at most three edges of $T_{n,k}$. Note that if tree T corresponds to a vertex v in $T_{n,k}$ with odd depth, then we may need to trace three edges to generate the next tree. Otherwise we need to trace at most two edges to generate the next tree. Note that each tree is similar to the preceding one, since it can be obtained with at most three operations. Therefore we have the following theorem.

Theorem 2. *One can generate ordered trees with exactly n vertices including exactly k leaves in $O(1)$ time for each in worst case.*

5 Conclusion

In this paper, we have given an efficient algorithm to generate all ordered trees with exactly n vertices including exactly k leaves.

We defined a cleverer family tree than the one in [15]. By traversing the family tree, our algorithm generates all trees in $S_{n,k}$ in $O(1)$ time for each in worst case.

References

1. Avis, D., Fukuda, K.: Reverse search for enumeration. Discrete Appl. Math. 65(1-3), 21–46 (1996)
2. Beyer, T., Hedetniemi, S.M.: Constant time generation of rooted trees. SIAM J. Comput. 9(4), 706–712 (1980)
3. Fenner, T.I., Loizou, G.: A binary tree representation and related algorithms for generating integer partitions. The Computer J. 23(4), 332–337 (1980)

4. Goldberg, L.: Efficient algorithms for listing combinatorial structures. Cambridge University Press, New York (1993)
5. Kikuchi, Y., Tanaka, H., Nakano, S., Shibata, Y.: How to obtain the complete list of caterpillars. In: Warnow, T.J., Zhu, B. (eds.) COCOON 2003. LNCS, vol. 2697, pp. 329–338. Springer, Heidelberg (2003)
6. Knuth, D.: The art of computer programming. Generating all tuples and permutations, vol. 4, fascicle 2. Addison-Wesley, Reading (2005)
7. Knuth, D.E.: The art of computer programming. Generating all trees, history of combinatorial generation, vol. 4, fascicle 4. Addison-Wesley, Reading (2006)
8. Korsh, J.F., LaFollette, P.: Multiset permutations and loopless generation of ordered trees with specified degree sequence. Journal of Algorithms 34(2), 309–336 (2000)
9. Korsh, J.F., LaFollette, P.: Loopless generation of trees with specified degrees. The Computer Journal 45(3), 364–372 (2002)
10. Kreher, D.L., Stinson, D.R.: Combinatorial algorithms. CRC Press, Boca Raton (1998)
11. Li, G., Ruskey, F.: The advantages of forward thinking in generating rooted and free trees. In: Proc. 10th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA 1999), pp. 939–940 (1999)
12. Li, Z., Nakano, S.: Efficient generation of plane triangulations without repetitions. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 433–443. Springer, Heidelberg (2001)
13. McKay, B.D.: Isomorph-free exhaustive generation. J. Algorithms 26(2), 306–324 (1998)
14. Muramatsu, T., Nakano, S.: A random generation of plane trees with exactly k leaves. IEICE Transaction on Fundamentals J90-A(12), 940–947 (2007) (in Japanese)
15. Nakano, S.: Efficient generation of plane trees. Inf. Process. Lett. 84(3), 167–172 (2002)
16. Nakano, S.: Efficient generation of triconnected plane triangulations. Comput. Geom. Theory and Appl. 27(2), 109–122 (2004)
17. Nakano, S., Uno, T.: Constant time generation of trees with specified diameter. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 33–45. Springer, Heidelberg (2004)
18. Nakano, S., Uno, T.: Generating colored trees. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 249–260. Springer, Heidelberg (2005)
19. Pallo, J.: Generating trees with n nodes and m leaves. International Journal of Computer Mathematics 21(2), 133–144 (1987)
20. Read, R.C.: Every one a winner or how to avoid isomorphism search. Annuals of Discrete Mathematics 2, 107–120 (1978)
21. Reingold, E.M., Nievergelt, J., Deo, N.: Combinatorial Algorithms. Prentice-Hall, Englewood Cliffs (1977)
22. Ruskey, F., van Baronaigien, D.R.: Fast recursive algorithms for generating combinatorial objects. Congressus Numerantium 41, 53–62 (1984)
23. Sawada, J.: Generating rooted and free plane trees. ACM Transactions on Algorithms 2(1), 1–13 (2006)
24. Stanley, R.P.: Enumerative combinatorics, vol. 2. Cambridge University Press, Cambridge (1999)
25. Wilf, H.S.: Combinatorial algorithms: An update. SIAM, Philadelphia (1989)
26. Wright, R.A., Richmond, B., Odlyzko, A., McKay, B.D.: Constant time generation of free trees. SIAM J. Comput. 15(2), 540–548 (1986)

27. Yamanaka, K., Kawano, S., Kikuchi, Y., Nakano, S.: Constant time generation of integer partitions. IEICE Trans. Fundamentals E90-A(5), 888–895 (2007)
28. Yamanaka, K., Nakano, S.: Listing all plane graphs. In: Nakano, S.-i., Rahman, M. S. (eds.) WALCOM 2008. LNCS, vol. 4921, pp. 210–221. Springer, Heidelberg (2008)
29. Zaks, S., Richards, D.: Generating trees and other combinatorial objects lexicographically. SIAM J. Comput. 8(1), 73–81 (1979)
30. Zoghbi, A., Stojmenović, I.: Fast algorithms for generating integer partitions. Int. J. Comput. Math. 70, 319–332 (1998)

Generating All Triangulations of Plane Graphs (Extended Abstract)

Mohammad Tanvir Parvez[1], Md. Saidur Rahman[1], and Shin-ichi Nakano[2]

[1] Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka-1000, Bangladesh
[2] Department of Computer Science, Gunma University, Japan
{tanvirparvez,saidurrahman}@cse.buet.ac.bd, nakano@cs.gunma-u.ac.jp

Abstract. In this paper, we deal with the problem of generating all triangulations of plane graphs. We give an algorithm for generating all triangulations of a triconnected plane graph G of n vertices. Our algorithm establishes a tree structure among the triangulations of G, called the "tree of triangulations," and generates each triangulation of G in $O(1)$ time. The algorithm uses $O(n)$ space and generates all triangulations of G without duplications. To the best of our knowledge, our algorithm is the first algorithm for generating all triangulations of a triconnected plane graph; although there exist algorithms for generating triangulated graphs with certain properties. Our algorithm for generating all triangulations of a triconnected plane graph needs to find all triangulations of a convex polygon. We give an algorithm to generate all triangulations of a convex polygon P of n vertices in time $O(1)$ per triangulation, where the vertices of P are numbered. Our algorithm for generating all triangulations of a convex polygon also improves previous results; existing algorithms need to generate all triangulations of convex polygons of less than n vertices before generating the triangulations of a convex polygon of n vertices. Finally, we give an algorithm for generating all triangulations of a convex polygon P of n vertices in time $O(n^2)$ per triangulation, where vertices of P are not numbered.

Keywords: Polygon; Triangulation; Graph; Plane Graph; Genealogical Tree.

1 Introduction

In this paper, we consider the problem of generating all triangulations of plane graphs. Such triangulations have many applications in Computational Geometry [4],[13], VLSI floorplanning [15], and Graph Drawing [9]. The main challenges in finding algorithms for generating all triangulations are as follows. Firstly, the number of such triangulations is exponential in general, and hence listing all of them requires huge time and computational power. Secondly, generating algorithms produce huge outputs, and the output dominates the running time. For this reason reducing the amount of output is essential. Thirdly, checking for any repetitions must be very efficient. Storing the entire list of objects generated

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 151–164, 2009.

so far will not be efficient, since checking each new object with the entire list to prevent repetition would require huge amount of memory and overall time complexity would be very high.

There have been a number of methods for generating combinatorial objects. Classical method algorithms first generate combinatorial objects allowing duplications, but output only if the object has not been output yet. These methods require huge space to store the list and a lot of time to check duplications. Orderly methods algorithms [8] need not to store the list of objects generated so far, they output an object only if it is a canonical representation of an isomorphism class. Reverse search method algorithms [1] also need not to store the list. The idea is to implicitly define a connected graph H such that the vertices of H correspond to the objects with the given property, and the edges of H correspond to some relation between the objects. By traversing an implicitly defined spanning tree of H, one can find all the vertices of H, which correspond to all the graphs with the given property. The main feature of our algorithm for generating all triangulations of a plane graph G is that, we define a tree structure among the triangulations of G and generate the triangulations of G in the order they are present in that tree. Thus our algorithm need not find any "graph of triangulations" of G from which it is necessary to find a spanning tree. Also, our algorithm generates each triangulation of G in $O(1)$ time.

There are some well known results for triangulating simple polygons and finding bounds on the number of operations required to transform one triangulation into another [3,6,14]. Researchers also have focused their attention on generating triangulated polygons and graphs with certain properties. Hurtado and Noy [5] built a tree of triangulations of convex polygons with any number of vertices. Their construction is primarily of theoretical interests; also all the triangulations of convex polygons with number of vertices up to n need to be found before finding the triangulations of a convex polygon of n vertices. Also, in [5] the authors did not discuss the time complexity of their method for generating the tree of triangulations of convex n-gons. Li and Nakano [7] gave an algorithm to generate all biconnected "based" plane triangulations with at most n vertices. Their algorithm generates all graphs with some properties without duplications. Here also, the biconnected "based" plane triangulations of n vertices are generated after the biconnected based plane triangulations of less than n vertices are generated. Hence, if we need to generate the triangulations of a convex polygon or a plane graph of exactly n vertices, existing algorithms generate all the triangulations of convex polygons or plane graphs with less than n vertices. This is not an efficient way of generation.

There exists an optimal algorithm for encoding and generating planar triangulations [12]. There also exists work [2] concerning the generation of triangulations of n points in the plane based on a tree of triangulations, and on a lexicographic way of generating triangulations, with $O(\log \log n)$ time complexity per triangulation.

We now give the idea behind our algorithm for generating all triangulations of a triconnected plane graph G. Consider Figure 1. For a particular triconnected

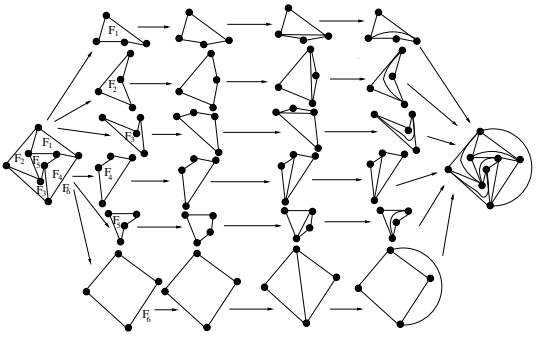


Fig. 1. Illustration of the algorithm for generating the triangulations of a triconnected plane graph

plane graph G, we treat each face of G as a convex polygon and find triangulations of those convex polygons. These triangulations of the convex polygons correspond to particular triangulations of the faces of the graph G. Combining the triangulations of the faces of G gives us a particular triangulation of G. Therefore, to generate all the triangulations of G we need to triangulate those intermediate convex polygons in all possible ways and combine the triangulations efficiently so as to find all the triangulations of G.

Therefore, in this paper, we also give an algorithm to generate all triangulations of a convex polygon P of n vertices in $O(1)$ time per triangulation, where the vertices of P are numbered sequentially. Such triangulations are called *labeled triangulations* of P. We also give the idea to generate all *unlabeled triangulations* of a convex polygon P of n vertices in time $O(n^2)$ per triangulation, where the vertices of P are not numbered.

The rest of the paper is organized as follows. Section 2 gives some definitions. Section 3 gives the outline of the algorithm for generating all triangulations of a triconnected plane graph. Section 4 deals with generating all labeled triangulations of convex polygon of n vertices. Section 5 deals with generating all unlabeled triangulations of a convex polygon of n vertices. Finally, Section 6 is the conclusion.

2 Preliminaries

In this section, we define some terms used in this paper.

Let $G = (V, E)$ be a connected simple graph with vertex set V and edge set E. An edge connecting vertices v_i and v_j in V is denoted by (v_i, v_j). The *degree* of a vertex v is the number of edges incident to v in G. The *connectivity* $\kappa(G)$ of a graph G is the minimum number of vertices whose removal results in a disconnected graph or a single vertex graph. A graph is *k-connected* if $\kappa(G) \geq k$.

A graph G is *planar* if it can be embedded in the plane so that no two edges intersect geometrically except at a vertex to which they are both incident. A

plane graph is a planar graph with fixed embedding in the plane. A plane graph divides the plane into connected regions called *faces*. The unbounded face is called *outer face* and the other faces are called *inner faces*. A plane graph is called a *plane triangulation* if each face boundary contains exactly three edges.

A *polygon* is the region of a plane bounded by a finite collection of line segments forming a simple closed curve. Each line segment of the closed curve is called a *side* or an *edge* of the polygon. A point joining two consecutive sides of a polygon is called a *vertex* of the polygon. A polygon is called simple if it does not cross itself. The set of points in the plane enclosed by a simple polygon forms the *interior* of the polygon, the set of points on the polygon itself forms its *boundary*, and the set of points surrounding the polygon forms its *exterior*. We say two vertices x and y of polygon P are visible to each other if and only if the *closed line segment* xy is nowhere exterior to the polygon P; i.e, $xy \subseteq P$. We say x has clear visibility to y if $xy \subseteq P$ and xy does not touch any vertex or edge of P. A *diagonal* of a polygon P is a line segment between two of its vertices x and y that are clearly visible to each other.

A simple polygon is *convex* if, given any two points on its boundary or in its interior, all points on the line segment drawn between them are contained in the polygon's boundary or interior. Let the vertices of a convex polygon P are labeled $v_1, v_2, \cdots, v_n$ counterclockwise. We represent P by listing its vertices as $P = \langle v_1, v_2, \cdots, v_n \rangle$. A diagonal (v_i, v_j) divides the polygon into two polygons: $\langle v_i, v_{i+1}, \cdots, v_j \rangle$ and $\langle v_j, v_{j+1}, \cdots, v_i \rangle$. A decomposition of a polygon into triangles by a set of non-intersecting diagonals is called a *triangulation* of the polygon. In a triangulation T, the set of diagonals is maximal, that means, every diagonal not in T intersects some diagonal in T. The sides of triangles in the triangulation are either the diagonals or the sides of the polygon. We say a vertex y is *visible* from a vertex x in a triangulation T of a convex polygon P if there exists a diagonal (x, y) of P in T. In this paper, we represent each triangulation T of a convex polygon P by listing its diagonals. Given the list of diagonals, we can uniquely construct the corresponding triangulation.

There is a natural correspondence between a cycle C of n vertices and a convex polygon P of n vertices. Each triangulation of C also corresponds to a triangulation of P.

3 Triangulations of a Triconnected Plane Graph

In this section we give an algorithm for generating all triangulations of a triconnected plane graph G of n vertices. Our idea is to define a parent-child relationship among the triangulations of G such that all the triangulations of G form a tree structure. Our algorithm generates the triangulations of G in the order they are present in that tree, called "tree of triangulations", without storing or building the entire tree at once in the memory.

Assume that G has k faces, arbitrarily labeled $F_1, F_2, \cdots, F_k$. For each face F_i of G, there is a convex polygon P_i *associated* with F_i, where the number of vertices of P_i and F_i is the same and the vertices of P_i are labeled similarly

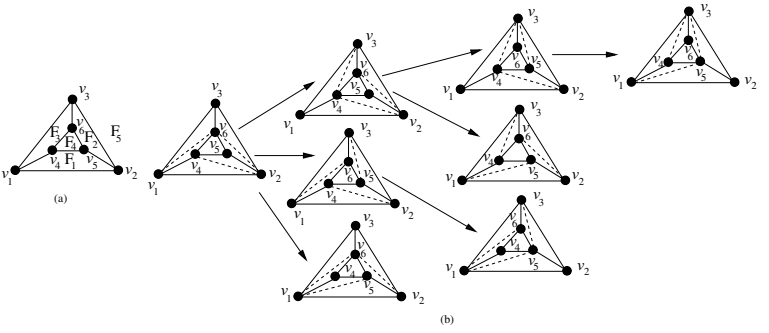


Fig. 2. Illustration of (a) a triconnected plane graph G with five faces and (b) genealogical tree $\mathcal{T}(G)$ of G

to the vertices of F_i. Since a particular triangulation of the face F_i, denoted $T(F_i)$, corresponds to a triangulation of P_i, denoted $T(P_i)$, triangulating the face F_i in all possible ways is equivalent to triangulating the convex polygon P_i in all possible ways. This is true since a plane graph is a data structure where the ordering (clockwise or counterclockwise) of the edges incident to a vertex is preserved and the outer face of the graph is fixed. Therefore, our algorithm for generating all triangulations of a triconnected plane graph needs to find all the triangulations of a convex polygon, the details of that is given in Section 4.

Let T be a triangulation of a convex polygon P of n vertices. To generate a new triangulation from T, we use the following operation. Let (v_i, v_j) be a shared diagonal of two adjacent triangles of T which form a convex quadrilateral $\langle v_q, v_i, v_r, v_j \rangle$. If we remove the diagonal (v_i, v_j) from T and add the diagonal (v_q, v_r), we get a new triangulation T'. The operation above is well known as *flipping*. Therefore, we *flip* the edge (v_i, v_j) to generate a new triangulation T', which we denote by $T(v_i, v_j)$.

Similarly, the operation of flipping an edge of the triangulation $T(F_i)$ is defined as the flipping of the corresponding diagonal of the triangulation $T(P_i)$. Therefore, to generate new triangulations of the plane graph G from an existing triangulation T of G, we flip some edges of T. In our algorithm, we define the parent-child relationship among the triangulations of G in such a way that every triangulation of G, except the root triangulation, is generated from its parent by a single edge flip. Such a tree of triangulations of the triconnected plane graph G is called a *genealogical tree* and denoted by $\mathcal{T}(G)$. The genealogical tree of the triconnected plane graph G of Figure 2(a) is shown in Figure 2(b).

This definition of flipping requires G to be triconnected. This is because, if G has a cut set of two vertices, then some flip operations may introduce multi-edges. If G is triconnected then any flip operation will generate a new triangulation of G. Note that, while generating new triangulations from an existing triangulation T of G, the edges of the graph G can not be flipped. Therefore, for a triangulation T of G, we need to classify edges of T as flippable and non-flippable. We introduce the related concepts below.

Let T be a triangulation of a convex polygon P of n vertices. The diagonals of T which can be flipped to generate new triangulations of P are called *generating diagonals* of T. In Section 4, we describe the way to find the generating diagonals of T. The set of generating diagonals of T is called the *generating set*. The triangulation $T(F_i)$ of the face F_i of G has a generating set of edges equivalent to the generating set of the triangulation $T(P_i)$ of the convex polygon P_i. Therefore, to generate new triangulations of G, we flip an edge from the generating set of a face F_i of G.

3.1 Finding the Root

In this section we describe the procedure for finding the root triangulation of the genealogical tree $\mathcal{T}(G)$ of a triconnected plane graph G of n vertices.

Let F_i be a face of G. We can represent F_i as a list of vertices on the boundary of F_i. We choose a vertex v_j on the boundary of F_i arbitrarily and call it the *root vertex* of F_i. Let P_i be the convex polygon associated with F_i. Then v_j is also called the root vertex of P_i. Consider the triangulation $T(P_i)$ of P_i where all the diagonals of $T(P_i)$ are incident to the root vertex v_j. This triangulation $T(P_i)$ of P_i gives us a triangulation of the face F_i of G, which we denote by $T(F_i)$. Once all the faces of G are triangulated in that way, we get a triangulation T of the graph G itself. In our algorithm, such a triangulation T of G is taken as the root triangulation T_R of the genealogical tree $\mathcal{T}(G)$. Note that, the choice of the root triangulation T_R will depend on the way the root vertices are chosen.

The procedure for finding T_R and corresponding generating sets are as follows. We traverse the face F_i of G to find the generating set of $T(F_i)$, denoted by C_i, using the doubly connected adjacency list representation of G [9]. Face F_i can be traversed in time proportional to the number of vertices on the boundary of it. Assume that we traverse the face F_i clockwise starting at vertex v_j and take v_j as the root vertex of P_i. Let v_k, v_l and v_m be three consecutive vertices on the boundary of F_i, where $v_k \neq v_j$ and $v_m \neq v_j$. We add the edge (v_j, v_l) to the generating set C_i of $T(F_i)$. We now have the following lemma, the proof of which is omitted here.

Lemma 1. *Let G be a triconnected plane graph of n vertices. Then the root triangulation T_R of the genealogical tree $\mathcal{T}(G)$ of G can be found in $O(n)$ time.*

3.2 The Algorithm

In this section, we give the details of the algorithm for generating all triangulations of a triconnected plane graph G of n vertices.

Assume that the triconnected plane graph G has k faces. For a particular triangulation T of G, we generate new triangulations of G from T as follows. We generate all new triangulations T' that can be generated from T by flipping the generating edges of the triangulation $T(F_j)$, where F_j is a face of G. We say that the face F_j is an *eligible face* for T to generate new triangulations of G from T. We find the eligible faces for T using the following criteria. Assume

that we have generated T from its parent T'' by flipping a generating edge of $T''(F_i)$, where F_i is a face of G. Then all the faces F_j of G, $1 \leq j \leq i$ are eligible for the triangulation T. This simple condition for eligibility ensures that each triangulation of G is generated exactly once.

We now have the following algorithm for generating all triangulations of a triconnected plane graph G with k faces.

Procedure find-all-child-triangulations(T, i)
{T is the current triangulation and F_i is an eligible face of T}
begin
1 Let E_G be the set of generating edges of $T(F_i)$;
2 **if** E_G is empty **then return** ;
3 **for** each edge $e \in E_G$
4 Flip e to find a new triangulation T';
5 output T';
{For T', all the faces F_j, $1 \leq j \leq i$, are eligible}
6 **for** $j = 1$ **to** i
7 **find-all-child-triangulations**(T', j);
end;
Algorithm find-all-triangulations(G, k)
{The triconnected plane graph G has k faces}
begin
Label the faces $F_1, F_2, \cdots, F_k$ arbitrarily;
Find root triangulation T_R of $\mathcal{T}(G)$;
output *root* T_R;
{For the root T_R, all the faces of G are eligible}
for $i = 1$ **to** k
find-all-child-triangulations(T_R, i);
end.

The correctness of the algorithm **find-all-triangulations** depends on the correct finding of the generating set of the triangulation $T(F_i)$ of face F_i of G (Step 1). We also have to ensure that flipping the edges in the generating set of $T(F_i)$ generates all the children of $T(F_i)$ without duplications. Flipping an edge of $T(F_i)$ is equivalent to flipping a diagonal of the triangulation $T(P_i)$ of the convex polygon P_i associated with F_i. Therefore, we need to prove that for a triangulation T of a convex polygon P: (1) flipping the generating diagonals of T generates all the child triangulations of T without duplications and (2) the number of generating diagonals in any child triangulation of T is less than T. We prove both of these in Section 4.

The time and space complexity of the algorithm **find-all-triangulations** can be found as follows. From Lemma 1, finding the root triangulation T_R takes $O(n)$ time. To find the time required to generate each new triangulation T' from a triangulation T of G, note that the difference between the representations of T' and T can be found in the triangulation of only one face, say F_i (Steps 3 - 5). Assume that face F_i has the triangulation $T(F_i)$ in T and $T'(F_i)$ in T'. Now,

to get the representation of T', all we need to do is to find the representation of $T'(F_i)$ from the representation of $T(F_i)$. Equivalently, the problem reduces to the following. Let T and T' be two triangulations of a convex polygon P and T' is generated from T by flipping a generating diagonal of T. Then, how can we find the representation and the generating set of T' from T efficiently? Section 4 shows that this can be done in $O(1)$ time.

To find the space complexity, note that, we can represent a triangulation T of G by listing its edges only. Therefore, it takes only $O(n)$ space to store a triangulation T. The height of the tree $\mathcal{T}(G)$ is bounded by the number of edges in T_R (since we may need to flip each generating edge T_R once to generate a triangulation of G), which is linear in n. The algorithm **find-all-triangulations** needs to store (1) the representation and generating set of the current triangulation T and (2) the information of the path from the root to the current node of the tree. This implies that the space complexity of the entire algorithm can be reduced to $O(n)$.

We need to address one last point. While generating the triangulations of a plane graph G, we have to triangulate the outer face also. But this poses no problem; since the outer face of G can be treated just like an inner face. Therefore we have the following theorem.

Theorem 1. *The algorithm* ***find-all-triangulations*** *generates all the triangulations of a triconnected plane graph G of n vertices in time $O(1)$ per triangulation, with $O(n)$ space complexity.*

4 Labeled Triangulations of a Convex Polygon

In this section, we give an algorithm to generate all labeled triangulations of a convex polygon P of n vertices. Here we also define a unique parent for each triangulation of P so that it results in a tree structure among the triangulations of P, with a suitable triangulation as the root. Figure 3 shows such a genealogical tree of a convex polygon of six vertices. Once such a parent-child relationship of P is established, we can generate all the triangulations of P using the relationship. We need not to build or to store the entire tree of triangulations at once, rather we generate each triangulation in the order it appears in the tree structure.

We denote the genealogical tree of P by $\mathcal{T}(P)$. Let T be a triangulation of P, in which all the diagonals of T are incident to vertex v_1. We regard T as the *root* T_r of $\mathcal{T}(P)$ and call v_1 as the root vertex of P. Note that the choice of root vertex is arbitrary in finding T_r. We can take any of the vertices of P other than v_1 as the root vertex to find a root triangulation T_r. With this definition of the root vertex of P, we describe the labeled triangulations of P as *rooted triangulations*.

Note that, in the root T_r of $\mathcal{T}(P)$, every vertex of P is visible from the root vertex v_1. We say that the root vertex v_1 has *full vision* in T_r. Obviously, in a non-root triangulation T of P, vertex v_1 does not have the full vision. The reason is that T has some "blocking diagonals" which are blocking some vertices of P from being visible from the root vertex v_1. A diagonal (v_i, v_j) of a triangulation T of P

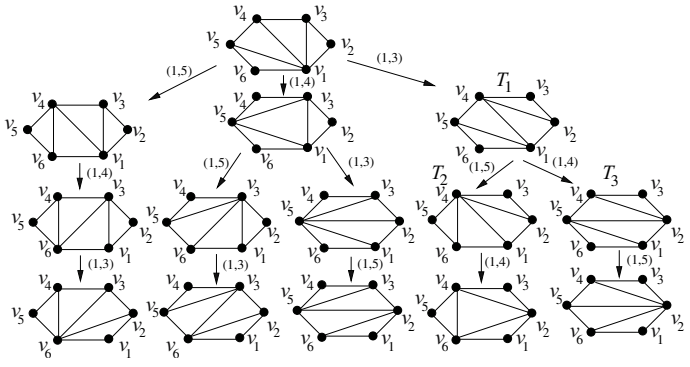


Fig. 3. Genealogical tree $\mathcal{T}(P)$ for a convex polygon P of six vertices

is a *blocking diagonal* of T if both v_i and v_j are adjacent to the root vertex of P. We say that the root vertex of P has a *blocked vision* in a non-root triangulation T of P. The following lemma characterizes the non-root triangulations of P.

Lemma 2. *Each triangulation T of a convex polygon $P = \langle v_1, v_2, \cdots, v_n \rangle$ has at least one blocking diagonal, if T is not the root of $\mathcal{T}(P)$.*

Proof. Let v_j be the vertex of P such that (v_1, v_k) is a diagonal of T, for all $k \geq j$. Then there exists a vertex v_i such that $i < j$ and (v_i, v_j) is a diagonal of T (choose i to be the minimum). Otherwise, all diagonals of T would be incident to v_1 and T would be the root of $\mathcal{T}(P)$. Since T is a triangulation of P, (v_1, v_i, v_j) is a triangle, and hence (v_i, v_j) is a blocking diagonal. □

Suppose we flip a diagonal (v_1, v_j) of T to generate a new triangulation T'. Let $(v_b, v_{b'})$, $b < b'$ be the newly found diagonal in T'. Obviously $(v_b, v_{b'})$ is a blocking diagonal of T'. Similarly, if we flip a blocking diagonal of T to generate T', the newly found diagonal will be non-blocking, incident to vertex v_1 in T'.

4.1 Child-Parent Relationship

Our idea of defining a parent-child relationship is that the parent of a triangulation T, denoted by $P(T)$, must have a "clearer vision" than T. Let T and T' be two triangulations of P. We say that T' has a *clearer vision* than T if the number of vertices visible from v_1 in T' is greater than the number of vertices visible from v_1 in T. We can easily get a triangulation T' from T, where T' has a clearer vision than T, by flipping a blocking diagonal $(v_b, v_{b'})$ of T. We say that the triangulation T' is the *parent* of T if the diagonal $(v_b, v_{b'})$ is the "leftmost blocking diagonal" of T. The diagonal $(v_b, v_{b'})$, $b < b'$, of T is the *leftmost blocking diagonal* of T if no other blocking diagonal of T is incident to a higher indexed vertex than $v_{b'}$ in T.

The above definition of the parent of a triangulation T of P ensures that we can always find a unique parent of a non-root triangulation T of P. From

Lemma 2, a non-root triangulation T of P has at least one blocking diagonal, and from those blocking diagonals of T we choose the one which is leftmost and we flip that diagonal to find the unique parent $P(T)$ of T. Based on the above parent-child relationship, the following lemma claims that every triangulation of a convex polygon P of n vertices is present in the genealogical tree $\mathcal{T}(P)$.

Lemma 3. *For any triangulation T of a convex polygon $P = \langle v_1, v_2, \cdots, v_n \rangle$, there is a unique sequence of flipping operations that transforms T into the root T_r of $\mathcal{T}(P)$.*

Proof. Let T be a triangulation other than the root of $\mathcal{T}(P)$. Then according to Lemma 2, T has at least one blocking diagonal. Let $(v_b, v_{b'})$ be the leftmost blocking diagonal of T. We find the parent $P(T)$ of T by flipping the leftmost blocking diagonal of T. Since flipping a blocking diagonal of T results in a diagonal incident to vertex v_1 in the new triangulation, $P(T)$ has one more diagonals incident to v_1 than T. Now, if $P(T)$ is the root, then we stop. Otherwise, we apply the same procedure to $P(T)$ and find its parent $P(P(T))$. By continuously applying this process of finding the parent, we eventually generate the root triangulation T_r of $\mathcal{T}(P)$. □

Lemma 3 ensures that there can be no omission of triangulations in the genealogical tree $\mathcal{T}(P)$ of a convex polygon P of n vertices. Since there is a unique sequence of operations that transforms a triangulation T of P into the root T_r of $\mathcal{T}(P)$, by reversing the operations we can generate that particular triangulation, starting at the root.

4.2 Generating the Children of a Triangulation in $\mathcal{T}(P)$

In this section we describe the method for generating the children of a triangulation T in $\mathcal{T}(P)$.

To find the parent $P(T)$ of the triangulation T, we flip the leftmost blocking diagonal of T. That means $P(T)$ has fewer blocking diagonals than T. Therefore, the operation for generating the children of T must increase the number of blocking diagonals in the children of T. Intuitively if we flip a diagonal (v_1, v_j) of T, which is incident to vertex v_1 in T, and generate a new triangulation T', then T' contains one more blocking diagonal than T. We call all such diagonals (v_1, v_j) as the *candidate diagonals* of T.

Note that, flipping a candidate diagonal of T may not always preserve the parent-child relationship described in Section 4.1. For example, we generate the triangulation of Figure 4(b) by flipping the candidate diagonal (v_1, v_3) of the triangulation of Figure 4(a). The leftmost blocking diagonal of the triangulation of Figure 4(b) is (v_4, v_6); therefore the parent of the triangulation of Figure 4(b) is the triangulation of Figure 4(c), not the triangulation of Figure 4(a).

Therefore to keep the parent-child relationship unique, we flip a candidate diagonal (v_1, v_j) of T to generate a new triangulation T' if only if flipping (v_1, v_j) of T results in the leftmost blocking diagonal of T'. We call such a candidate diagonal (v_1, v_j) of T as a *generating diagonal*. The generating diagonals of a

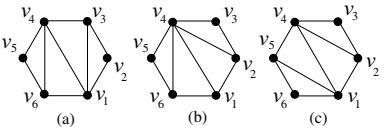


Fig. 4. Illustration of a flipping that does not preserve parent-child relationship

triangulation T of P can be found as follows. Let $(v_b, v_{b'})$ be the leftmost blocking diagonal of a triangulation T of a convex polygon P of n vertices. Then (v_1, v_j) is a generating diagonal of T if $j \geq b$. If T has no blocking diagonal then all diagonals of T are generating diagonals. Thus all the diagonals of the root T_r of $\mathcal{T}(P)$ are generating diagonals. All other candidate diagonals of T are called *non-generating*. We call the set of generating diagonals of a triangulation T as *generating set* C of T. We now have the following lemmas, the proofs are omitted here.

Lemma 4. *Let (v_1, v_j) be a generating diagonal of a triangulation T of a convex polygon P of n vertices. Then flipping (v_1, v_j) in T results in the leftmost blocking diagonal of $T(v_1, v_j)$.*

Lemma 5. *Let T be a triangulation of a convex polygon P of n vertices. Let $T(v_1, v_j)$ be the triangulation generated by flipping the diagonal (v_1, v_j) of T. Then T is the parent of $T(v_1, v_j)$ in the genealogical tree $\mathcal{T}(P)$ if and only if (v_1, v_j) is a generating diagonal of T.*

According to Lemma 4 and 5, if the generating set C of a triangulation T is non-empty, then we can generate each of the children of T in $\mathcal{T}(P)$ by flipping a generating diagonal of T. Therefore, the number of children of a triangulation T in $\mathcal{T}(P)$ will be equal to the cardinality of the generating set. Thus, the following lemma holds.

Lemma 6. *The number of children of a triangulation T of a convex polygon P is equal to the number of diagonals in the generating set of T. The root of $\mathcal{T}(P)$ has the maximum number of children.*

4.3 The Representation of a Triangulation in $\mathcal{T}(P)$

In this section we describe a data structure that we use to represent a triangulation T and that enables us to generate each child triangulation of T in constant time.

For a triangulation T of P, we maintain three lists: L, C and O to represent T completely. Here L is the list of diagonals of T and C is the generating set of T. For each diagonal (v_1, v_j) in the generating set C of T, we maintain a corresponding *opposite pair* $(v_o, v_{o'})$, such that $\langle v_1, v_o, v_j, v_{o'} \rangle$ is a convex quadrilateral of T. Note that, $o < j$ and $o' > j$. O is the list of such opposite pairs.

Since we generate triangulations of P starting with the root T_r, we find the representation of T_r first. The diagonals of T_r are listed in L in counterclockwise

order. That is, for T_r, $L = \{(v_1, v_{n-1}), (v_1, v_{n-2}), \cdots (v_1, v_3)\}$. The generating set C is exactly similar to the list L of T_r: $C = \{(v_1, v_{n-1}), (v_1, v_{n-2}), \cdots, (v_1, v_4), (v_1, v_3)\}$. Corresponding list of opposite pairs is $O = \{(v_{n-2}, v_n), (v_{n-3}, v_{n-1}), \cdots, (v_3, v_5), (v_2, v_4)\}$; that means, (v_{j-1}, v_{j+1}) is the opposite pair of (v_1, v_j) in T_r, $3 \leq j \leq n-1$.

Let $T(v_1, v_j)$ be a child triangulation of T in $\mathcal{T}(P)$ generated from T by flipping the diagonal (v_1, v_j) of T. Let $(v_b, v_{b'})$ be the blocking diagonal which appears in $T(v_1, v_j)$ after flipping (v_1, v_j) of T. The list L of $T(v_1, v_j)$ can be found easily from the representation of T by removing (v_1, v_j) from the list L of T and adding $(v_b, v_{b'})$ to it. Note that one can easily find the blocking diagonal $(v_b, v_{b'})$ of T', since $(v_b, v_{b'})$ is the opposite pair of (v_1, v_j) in the representation of T.

4.4 The Algorithm

In this section we describe the steps to generate all triangulations of P.

Let $v_{j_1}, v_{j_2}, \cdots, v_{j_k}$, $j_1 > j_2 > \cdots > j_k$, be the sequence of k vertices of a triangulation T of P such that $(v_1, v_{j_1}), (v_1, v_{j_2}), \cdots, (v_1, v_{j_k})$ are the diagonals of T and each of the diagonals $(v_1, v_{j_i}), 1 \leq i \leq k$, is a generating diagonal of T. Then, T has a generating set $C = \{(v_1, v_{j_1}), (v_1, v_{j_2}), \cdots, (v_1, v_{j_k})\}$ of k generating diagonals, for $0 \leq k \leq n-3$. For T_r, $C = \{(v_1, v_{n-1}), (v_1, v_{n-2}), \cdots, (v_1, v_4), (v_1, v_3)\}$. For each diagonal (v_1, v_j) of T, we keep an opposite pair $(v_o, v_{o'})$ in T. O is the set of such pairs. For T_r, $O = \{(v_{n-2}, v_n), (v_{n-3}, v_{n-1}), \cdots, (v_3, v_5), (v_2, v_4)\}$ as shown in Section 4.3. We find the sets C and O of a child T' of T by updating the lists C and O of T while we generate T'.

We now describe a method for generating the children of a triangulation T in $\mathcal{T}(P)$. We have two cases based on whether T is the root of $\mathcal{T}(P)$ or not.

Case 1. T is the root of $\mathcal{T}(P)$. In this case, all the diagonals of T are generating diagonals and there are a total of $n-3$ such diagonals in T. Any of these generating diagonals of T can be flipped to generate a child triangulation of T. For example, the root of the genealogical tree in Figure 3 has three generating diagonals; thus it has three children as shown in Figure 3.

Case 2. T is not the root of $\mathcal{T}(P)$. Let $(v_b, v_{b'})$ be the leftmost blocking diagonal of T. Consider a diagonal (v_1, v_j) of T. If $j \geq b$, then (v_1, v_j) is a generating diagonal of T. Therefore, according to Lemma 5, $T(v_1, v_j)$ is a child of T in $\mathcal{T}(P)$. Thus, for all diagonals (v_1, v_j) of T such that $j \geq b$, a new triangulation is generated by flipping (v_1, v_j).

If $j < b$, then (v_1, v_j) is a non-generating diagonal of T and according to Lemma 5, we can not flip (v_1, v_j) to generate a new triangulation from T.

Based on the case analysis above, we can generate all triangulations of P. Therefore, we have the following theorem. The proof is omitted here.

Theorem 2. *Given a convex polygon P of n vertices, we can generate all the triangulations of P in $O(1)$ time per triangulation, without duplications and omissions. The space complexity of the algorithm is $O(n)$.*

5 Generating Unlabeled Triangulations

In this section we give the idea for generating all unlabeled triangulations of a convex polygon P of n vertices.

Let T be a triangulation of P where the vertices of P are labeled sequentially from v_1 to v_n. A *labeled degree sequence* $\langle d_1, d_2, \cdots, d_n \rangle$ of T is the sequence of degrees of the vertices, where d_i is the degree of v_i in the graph associated with T. It can be shown that T can be represented uniquely by its labeled degree sequence. Our algorithm for generating unlabeled triangulations of P is based on the following two facts.

Fact 7. *Let T and T' be two triangulations of a convex polygon P of n vertices, which are rotationally equivalent to each other. Then, by rotating the labeled degree sequence of T, we get the labeled degree sequence of T'.*

Fact 8. *Let T and T' be two triangulations of a convex polygon of n vertices, which are mirror images of each other. Let T has the labeled degree sequence $\langle d_1, d_2, \cdots, d_n \rangle$. Then the labeled degree sequence of T' is $\langle d_n, d_{n-1}, \cdots, d_2, d_1 \rangle$.*

The details of our algorithm are omitted here. We only give the following theorem.

Theorem 3. *For a convex polygon P of n vertices, all the triangulations of P can be found in time $O(n^2)$ per triangulation, where the vertices of P are not numbered. The space complexity is $O(n)$.*

6 Conclusion

In this paper we gave an algorithm to generate all triangulations of triconnected plane graph G of n vertices in $O(1)$ time per triangulation with linear space complexity. We also gave an algorithm to generate all triangulations of a convex polygon of n labeled vertices in time $O(1)$ per triangulation with $O(n)$ space complexity. The performance of the algorithms can be further improved by using parallel processing. Our algorithm also works for biconnected graphs, but may produce multi-edges occasionally. Finally, we described a method to eliminate any rotational and mirror repetitions while generating all triangulations of a convex polygon P, when the vertices of P are not numbered.

References

1. Avis, D., Fukuda, K.: Reverse Search for Enumeration. Discrete Applied Mathematics 6, 82–90 (1996)
2. Bespamyatnikh, S.: An Efficient Algorithm for Enumeration of Triangulations. Comput. Geom. Theory Appl. 33, 271–279 (2002)
3. Chazelle, B.: Triangulating a Polygon in Linear Time. Discrete Comput. Geom. 6, 485–524 (1991)

4. de Berg, M., van Krevald, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications. Springer, Heidelberg (2000)
5. Hurtado, F., Noy, M.: Graph of Triangulations of a Convex Polygon and Tree of Triangulations. Computational Geometry 13, 179–188 (1999)
6. Komuro, H., Nakamoto, A., Negami, S.: Diagonal Flips in Triangulations on Closed Surfaces with Minimum Degree at Least 4. Journal of Combinatorial Theory, Series B 76, 68–92 (1999)
7. Li, Z., Nakano, S.: Efficient Generation of Plane Triangulations without Repetitions. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 433–443. Springer, Heidelberg (2001)
8. Mckay, B.D.: Isomorph-free Exhaustive Generation. Journal of Algorithms 26, 306–324 (1998)
9. Nishizeki, T., Rahman, M.S.: Planar Graph Drawing. World Scientific, Singapore (2004)
10. Nakano, S., Uno, T.: More Efficient Generation of Plane Triangulations. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 273–292. Springer, Heidelberg (2004)
11. Nakano, S., Uno, T.: Constant Time Generation of Trees with Specified Diameter. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 33–45. Springer, Heidelberg (2004)
12. Poulalhon, D., Schaeffer, G.: Optimal Coding and Sampling of Triangulations. Algorithmica 46, 505–527 (2006)
13. O'Rourke, J.: Computational Geometry in C, 2nd edn. Cambridge University Press, Cambridge (1998)
14. Sleator, D., Tarjan, R., Thurston, W.: Rotation Distance, Triangulations, and Hyperbolic Geometry. J. Amer. Math. Soc. 1, 647–681 (1988)
15. Sait, S.M., Youssef, H.: VLSI Physical Design Automation: Theory and Practice. World Scientific, Singapore (1999)

Recognition of Unigraphs through Superposition of Graphs (Extended Abstract)

Alessandro Borri, Tiziana Calamoneri, and Rossella Petreschi

Department of Computer Science
Univ. of Rome "Sapienza" - Italy
via Salaria 113, 00198 Roma, Italy
alessandroborri@gmail.com, {calamo,petreschi}@di.uniroma1.it

Abstract. Unigraphs are graphs uniquely determined by their own degree sequence up to isomorphism. In this paper a structural description for unigraphs is introduced: vertex set is partitioned into three disjoint sets while edge set is divided into two different classes. This characterization allows us to design a linear time recognition algorithm that works recursively pruning the degree sequence of the graph. The algorithm detects two particular graphs whose superposition generates the given unigraph.

1 Introduction

Unigraphs [6, 7, 9] are graphs uniquely determined by their own degree sequence up to isomorphism. In [15], unigraphs are characterized by a unique decomposition into particular components (see Section 2). This result, though very interesting from a structural point of view, does not seem to immediately lead to an efficient recognition algorithm; nevertheless, the author, in a private communication, observed that it is somehow possible to restrict to unigraphs the algorithm presented in [14] that decomposes arbitrary graphs using results from [15]. To the best of our knowledge, the only published linear time algorithm for recognizing unigraphs is in [8] and works exploiting Ferrer diagrams. On the contrary, it is possible to find several linear recognition algorithms for subclasses of unigraphs, as matrogenic, split matrogenic and threshold graphs [2, 3, 5, 10, 11, 12, 13, 16].

In this paper we generalize to unigraphs the pruning algorithm designed for matrogenic graphs in [11] providing a new recognition algorithm for the whole class of unigraphs.

The algorithm is linear and has a completely different approach with respect to [8], although works on the degree sequence, too. In particular, it partitions the vertex set and the edge set into three and two disjoint sets, respectively, detecting two particular graphs whose superposition generates the given unigraph. This superposition allows us to interpret in a simplified way the unigraph's structure. Indeed, also the algorithm for the recognition of matrogenic graphs [11] provides a similar superposition that is exploited for solving other problems (e.g.

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 165–176, 2009.

the $L(2,1)$-labeling [1]). It is in the conviction of the authors that the results presented in this paper will be useful for solving such problems, that are NP-hard for general graphs, polynomially solvable for subclasses of unigraphs and still unknown for unigraphs.

This paper is organized as follows: next two sections recall some known results and definitions useful in Section 4 for introducing our characterization for unigraphs. In Section 5 a linear time recognition algorithm justified by the previous characterization is described. Section 6 addresses some conclusions.

2 Preliminaries and Notations

We consider only finite, simple, loopless, connected graphs $G = (V, E)$, where V is the vertex set of G with cardinality n and E is the edge set of G with cardinality m. Where no confusion arises, we will call $G = (V, E)$ simply G.

Let $DS(G) = \delta_1, \delta_2, \ldots, \delta_n$ be the degree sequence of a graph G sorted by non-increasing values: $\delta_1 \geq \delta_2 \geq \ldots \geq \delta_n \geq 0$. We call *boxes* the equivalence classes of vertices in G under equality of degree. In terms of boxes the degree sequence can be compressed as $d_1^{m_1}, d_2^{m_2}, \ldots d_r^{m_r}, d_1 > d_2 > \ldots > d_r \geq 0$, where d_i is the degree of the m_i vertices contained in box $B_i(G)$, $1 \leq m_i \leq n$. We call a box *universal (isolated)* if it contains only universal (isolated) vertices, where a vertex $x \in V$ is called *universal* (*isolated*) if it is adjacent to all other vertices of V (no other vertex in V); if x is a universal (isolated) vertex, then its degree is $d(x) = n - 1$ ($d(x) = 0$).

A graph I induced by subset $V_I \subseteq V$ is called *complete* or *clique* if any two distinct vertices in V_I are adjacent in G, *stable* if no two vertices in V_I are adjacent in G.

A graph G is said to be *split* if there is a partition $V = V_K \cup V_S$ of its vertices such that the induced subgraphs K and S are complete and stable, respectively [4].

A set M of edges is a *perfect matching of dimension* h of X onto Y if and only if X and Y are disjoint subsets of vertices with the same cardinality h and each edge is incident to exactly one vertex $x \in X$ and to one vertex $y \in Y$, and different edges must be incident to different vertices.

The complement of a perfect matching of dimension h is called *hyperoctahedron of dimension* h.

An *antimatching of dimension* h of X onto Y is a set A of edges such that $M(A) = X \times Y - A$ is a perfect matching of dimension h of X onto Y.

A graph G is *threshold (matrogenic)* if and only if it does not contains the forbidden configuration of Fig. 1.a (1.b) as induced subgraph [2, 5].

The complement of a threshold graph/matrogenic graph/unigraph is still a threshold graph/matrogenic graph/unigraph.

Given a graph G, if its vertex set V can be partitioned into three disjoint sets V_K, V_S, and V_C such that every vertex in V_C is adjacent to every vertex in V_K and to no vertex in V_S, then the subgraph induced by V_C is called *crown*.

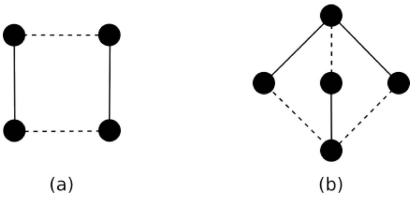


Fig. 1. Forbidden configurations for: a. threshold graphs, b. matrogenic graphs; — shows a present edge, - - shows an absent edge

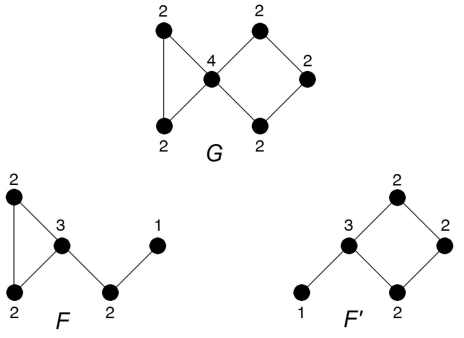


Fig. 2. A unigraph whose induced subgraphs are not unigraphs: F and F' are both subgraphs of G and have the same degree sequence even if are not isomorphic

A property $\mathcal{P}$ of a graph G is said *hereditary* if it holds for every subgraph of G. Thresholdness and matrogenicity are hereditary properties, while unigraphicity is not, as Fig. 2 shows.

Definition 1. *[14] Given a split graph $F = (V_K \cup V_S, EF)$ and a simple graph $H = (VH, EH)$, their* composition *is a graph $G = (V, E) = F \circ H$ obtained as the disjoint union $F \cup H$ plus the edge set of the complete bipartite graph with parts V_K and VH.*

Theorem 1. *[15] An n vertex graph G, given through its degree sequence $DS(G) = \delta_1, \delta_2, \ldots, \delta_n$, is* decomposable *as $F \circ H$, where F is a split graph and H is a simple graph, if and only if there exist nonnegative integers p and q such that*

$$0 < p + q < n, \qquad \sum_{i=1}^{p} \delta_i = p(n - q - 1) + \sum_{i=n-q+1}^{n} \delta_i$$

and the degree sequences of F and H are $\delta_1 - n + p + q, \ldots, \delta_p - n + p + q$, $\delta_{n-q+1}, \ldots, \delta_n$ and $\delta_{p+1} - p, \ldots, \delta_{n-q} - p$, respectively. If p and q do not exist, G is said indecomposable.

Let us conclude this Section introducing the concept of inverse of a given split graph. If $G = (V_K \cup V_S, E)$ is a split graph, its *inverse* G^I is obtained from

G by deleting the set of edges $\{\{a_1, a_2\} : a_1, a_2 \in V_K\}$ and adding the set of edges$\{\{b_1, b_2\} : b_1, b_2 \in V_S\}$; in other words, the inverse of a split graph swaps the roles of the clique and the stable set. Then, the inverse of a threshold graph/split matrogenic graph/split unigraph is still a threshold graph/split matrogenic graph/split unigraph.

In the next section we present two theorems characterizing matrogenic graphs and unigraphs, respectively.

3 Two Known Characterization Theorems

The following theorem characterizes matrogenic graphs as the superposition of a *black* and a *red graph*, B and R.

Theorem 2. *[11] A graph G is matrogenic if and only if its vertices can be partitioned into three disjoint sets V_K, V_S and V_C such that:*

(i) *$V_K \cup V_S$ induces a split matrogenic graph, in which K is the clique and S is the stable set;*

(ii) *V_C induces a crown that is a perfect matching or a hyperoctahedron or a penthagon (i.e. the chordless cycle C_5);*

(iii) *the edges of G can be colored red and black so that:*

- *a. the red partial graph is the union of the subgraph induced by V_C and of vertex-disjoint pieces $P_i, i = 1, \ldots, z$. Each piece is either a stable graph N_j, belonging either to K or to S; or a matching M_r of dimension h_r of $K_r \subseteq V_K$ onto $S_r \subseteq V_S, r = 1, \ldots \mu$; or an antimatching A_t of dimension h_t of $K_t \subseteq V_K$ onto $S_t \subseteq V_S, t = 1, \ldots \alpha$;*
- *b. the linear ordering $P_1, ..., P_z$ is such that each vertex in V_K belonging to P_i is not linked to any vertex in V_S belonging to P_j , $j = 1, ..., i-1$, but is linked by a black edge to every vertex in V_S belonging to P_j, $j = i+1, ..., z$. Furthermore, any edge connecting either two vertices in V_K or a vertex in V_K and a vertex in V_C is black.*

For the sake of completeness, we observe that the black graph cited in Theorem 2 is a threshold graph according to one of the equivalent definitions presented in [10].

Theorem 3. *[15] The unigraphs are all graphs of the form $G_1 \circ \ldots \circ G_c \circ G$, where:*

- *$c \geq 0$ if $G \neq \emptyset$ and $c \geq 1$ otherwise;*
- *For each $i = 1, \ldots, c$, either G_i or $\overline{G_i}$ or G_i^I or $\overline{G_i^I}$, is one of the following split unigraphs:*

$$K_1, \quad S_2(p_1, q_1; \ldots; p_t, q_t), \quad S_3(p, q_1; q_2), \quad S_4(p, q);$$

- *If $G \neq \emptyset$, either G or $\overline{G}$ is one of the following non split unigraphs:*

$$C_5, \quad mK_2, m \geq 2, \quad U_2(m, s), \quad U_3(m).$$

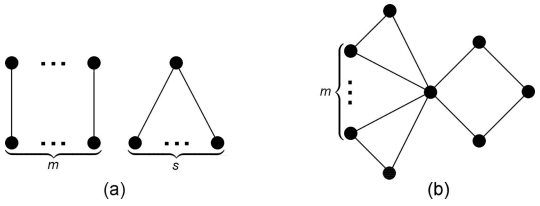


Fig. 3. a. $U_2(m, s)$; b. $U_3(m)$

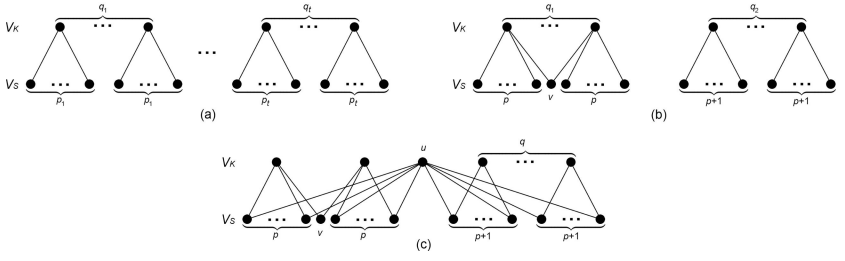


Fig. 4. a. $S_2(p_1, q_1; \ldots; p_t, q_t)$; b. $S_3(p, q_1; q_2)$; c. $S_4(p, q)$. For the sake of clearness, the edges of the clique connecting vertices of V_K are omitted.

All graphs cited in the previous theorem are said *indecomposable*, and they are defined as follows.

$K_1$is a singleton; C_5 is a pentagon and mK_2 is a perfect matching of dimension m.

$U_2(m, s)$: the disjoint union of perfect matching mK_2 and star $K_{1,s}$,for $m \geq 1, s \geq 2$ (see Fig. 3.a).

$U_3(m)$: for $m \geq 1$, fix a vertex in each component of the disjoint union of the chordless cycle C_4 and m triangles K_3, and merge all these vertices in one (see Fig. 3.b).

$S_2(p_1, q_1; \ldots; p_t, q_t)$: add all the edges connecting the centers of l nonisomorphic arbitrary stars K_{1,p_i}, $i = 1, \ldots, t$, each one occurring q_i times, where $p_i, q_i, t \geq 1$, $q_1 + \ldots + q_t = l \geq 2$ (see Fig. 4.a).

$S_3(p, q_1; q_2)$: take a graph $S_2(p, q_1; p + 1, q_2)$ where $p \geq 1$, $q_1 \geq 2$ and $q_2 \geq 1$. Add a new vertex v to the stable part of the graph and add the set of q_1 edges $\{\{v, w\} : w \in V_K, \deg_{V_S}(w) = p\}$ (see Fig. 4.b).

$S_4(p, q)$: take a graph $S_3(p, 2; q)$, $q \geq 1$. Add a new vertex u to the clique part, connecting it by the edges with each vertex except v (see Fig. 4.c).

4 Unigraphs as Superposition of Red and Black Graphs

In this section we present a characterization of unigraphs in terms of superposition of a red and a black graph. This result generalizes the one holding for matrogenic graphs (cf. Theorem 2). Nevertheless, it is to notice that the proof

of the following theorem is not a straightforward generalization of the proof presented in [11], as the latter one is based on the heredity of matrogenicity while this property does not hold for unigraphs.

Theorem 4. *A graph G is a unigraph if and only if its vertex set can be partitioned into three disjoint sets V_K, V_S and V_C such that:*

1. *$V_K \cup V_S$ induces a split unigraph F in which K is the clique and S is the stable set;*
2. *V_C induces a crown H and either H or $\overline{H}$ is one of the following graphs:*

$$C_5, \quad mK_2, m \geq 2, \quad U_2(m,s), \quad U_3(m);$$

3. *the edges of G can be colored red and black so that:*
   a. *the red partial graph is the union of H and of vertex-disjoint pieces $P_i, i = 1, \ldots, z$. Each piece P_i (or $\overline{P_i}$, or P_i^I or $\overline{P_i^I}$) is one of the following graphs:*

$$K_1, \quad S_2(p_1, q_1; \ldots; p_t, q_t), \quad S_3(p, q_1; q_2), \quad S_4(p, q),$$

   considered without the edges in the clique;
   b. *The linear ordering $P_1, \ldots, P_z$ is such that each vertex in V_K belonging to P_i is not linked to any vertex in V_S belonging to P_j, $j = 1, ..., i-1$, but is linked by a black edge to every vertex in V_S belonging to P_j, $j = i+1, \ldots, z$. Furthermore, any edge connecting either two vertices in V_K or a vertex in V_K and a vertex in V_C is black.*

Proof. Let us prove the 'if' part, first. Items (1) and (2) and the ordering (3).b. identify the decomposition in $F \circ H$ where F is a split graph and H is an indecomposable unigraph. Let us now consider graph F, i.e. $G - H$. For item (3).a F is the union of vertex-disjoint pieces connected by the black threshold graph; it follows that $F = F_1 \circ \ldots \circ F_c$, where each F_i is a piece P_i plus the black edges in the clique part. This is the crucial point that allows us to bypass the lack of heredity of unigraphicity. So Theorem 3 holds and G is a unigraph.

Concerning the 'only if' part, observe that the edges added during the composition operation, together with the edges in the cliques of the split components, induce a threshold graph. As G is a unigraph, items (1) and (2) derive from this observation and from Theorem 3 , while item (3) is obtained from the following coloring operation: color black all edges coming from the composition operation, all edges induced by V_K and all edges connecting V_K and V_C; color red all other edges. The elimination of the edges from the clique of the red pieces in item (3) is necessary for avoiding to color these edges both red and black.

In Fig. 5.a a unigraph is depicted , and its red and black partial graphs are highlighted in Fig. 5.b and 5.c, respectively.

Observe that Theorem 1 guarantees that the first p and the last q elements in the degree sequence of a unigraph individuate the first indecomposable component of the ordered sequence of compositions. As we will show in the following,

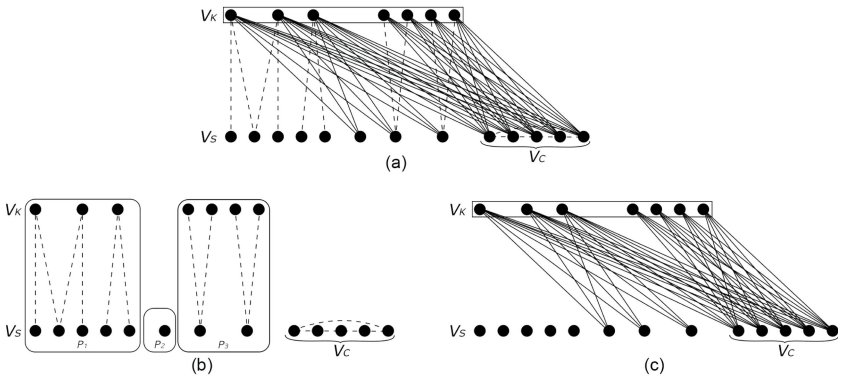


Fig. 5. a. A unigraph; b. its red graph; c. its black graph

we are able to characterize the degree sequence associated to each indecomposable unigraph. This characterization, together with the analysis of the previous theorem, allows us to naturally design a pruning algorithm for the class of unigraphs.

5 A Linear Time Recognition Algorithm for Unigraphs

In this section, we design a new algorithm for recognizing unigraphs that works pruning the degree sequence $d_1^{m_1}, \ldots, d_r^{m_r}$ of a given graph G. At each step, the algorithm finds an indecomposable split component F of G checking the first p and the last q boxes, according to part (3).a. of Theorem 4. The algorithm proceeds on the pruned graph $G - F$ and iterates this step until either G is recognized to be a unigraph or some contradiction is highlighted.

To do that, we need to list each indecomposable non-split (let it be H) and split (let it be F) graph and to exhibit the rules that must be respected by the first p and the last q boxes of the degree sequence $d_1^{m_1}, \ldots, d_r^{m_r}$ corresponding to graph G. Due to the lack of space, the complete list of conditions will be provided in an extended version of the paper. In order to follow the algorithm, it is enough to know that each condition is called with the name of the graph it detects.

ALGORITHM. `Pruning-Unigraphs`
INPUT: a graph G by means of its degree sequence $d_1^{m_1}, \ldots, d_r^{m_r}$
OUTPUT: a red/black edge coloring if G is an unigraph, "failure" otherwise.

$imax \leftarrow 1; imin \leftarrow r; n \leftarrow \sum_{j=imax}^{imin} m_j;$
REPEAT
Step 1 (non split indecomposable component, i.e. crown)
IF $imax = imin$ AND (COND. K_2 OR COND.$\overline{K_2}$ OR COND. C_5)
THEN color by red all edges of the crown;

$n \leftarrow 0$;
ELSE
IF $imax = imin - 1$ AND (COND. U_2 OR COND. U_3 OR COND. $\overline{U_2}$ OR COND. $\overline{U_3}$)
THEN color by red all edges of the crown;
$n \leftarrow 0$;
ELSE

Step 2 (universal or isolated box)
IF COND. U
THEN FOR $i = imax + 1$ TO $imin$ DO
$d_i \leftarrow d_i - m_{imax}$;
$imax \leftarrow imax + 1$;
$n \leftarrow n - m_{imax}$;
ELSE
IF COND. I
THEN $imin \leftarrow imin - 1$;
$n \leftarrow n - m_{imin}$;
ELSE

Step 3 (split indecomposable components)
IF COND. S_3 OR COND. $\overline{S_3^I}$ OR COND. $\overline{S_4}$ OR COND. S_4^I
THEN color by red all the edges of the split component but the edges of its clique;
FOR $i = imax + 1$ TO $imin - 2$ DO
$d_i \leftarrow d_i - m_{imax}$;
$imax \leftarrow imax + 1$;
$imin \leftarrow imin - 2$;
$n \leftarrow n - m_{imax} - m_{imin} - m_{imin-1}$;
ELSE
IF COND. $\overline{S_3}$ OR COND. S_3^I OR COND. S_4 OR COND. $\overline{S_4^I}$
THEN color by red all the edges of the split component but the edges of its clique;
FOR $i = imax + 2$ TO $imin - 1$ DO
$d_i \leftarrow d_i - m_{imax} - m_{imax+1}$;
$imax \leftarrow imax + 2$;
$imin \leftarrow imin - 1$;
$n \leftarrow n - m_{imax} - m_{imax+1} - m_{imin}$;
ELSE
IF $m_{imax} \leq m_{imin}$ AND there exists an $x \geq 1$ s.t. COND. S_2 OR COND. $\overline{S_2^I}$
THEN color by red all the edges of the split component but the edges of its clique;
FOR $i = imax + x$ TO $imin - 1$ DO
$d_i \leftarrow d_i - m_{imax} - \ldots - m_{imax+x-1}$;
$imax \leftarrow imax + x$;
$imin \leftarrow imin - 1$;
$n \leftarrow n - m_{imax} - \ldots - m_{imax+x-1} - m_{imin}$;
ELSE
IF $m_{imax} > m_{imin}$ AND there exists an $x \geq 1$ s.t. COND. $\overline{S_2}$ OR COND. S_2^I
THEN color by red all the edges of the split component but the edges of its clique;
FOR $i = imax + 1$ TO $imin - x$ DO
$d_i \leftarrow d_i - m_{imax}$;
$imax \leftarrow imax + 1$;
$imin \leftarrow imin - x$;

$n \leftarrow n - m_{imax} - m_{imin-x+1} - \ldots - m_{imin}$;
ELSE
Step 4 (G is not an unigraph)
RETURN "failure";
UNTIL $n = 0$;
color by black all the uncolored edges of G;
RETURN the red/black edge coloring of G.

Theorem 5. *Algorithm* `Pruning-Unigraphs` *returns "success" if and only if G is a unigraph.*

Proof. Checking the first p and the last q boxes of the degree sequence allows one to recognize the split component F, if it exists. Item (3).a. of Theorem 4 identifies all the components of F for a unigraph and the conditions univocally determine each component and indicate the values of p and q for each component in terms of boxes. Conditions of steps 1, 2, 3, 4 are mutually exclusive, so at each iteration of the algorithm only one step may occur. Item (3).b. of Theorem 4 guarantees that the pruning operation can be iteratively applied. Indeed, the algorithm, at each step, red-colors and eliminates a complete piece P_i and eliminates all the the edges connecting P_i to P_j, for $j = i + 1, \ldots, z$ that will be colored by black at the end; hence, if the original graph G is a unigraph, for the reduced graph $G - P_i$ Theorem 4 holds and hence it is a unigraph, too. Finally, the crown, if it exists, is specified by item (2) of Theorem 4 and by the conditions; it is red-colored. The correctness of the algorithm follows.

Theorem 6. *Algorithm* `Pruning-Unigraphs` *runs in $O(n)$ time.*

Proof. Each indecomposable component P_i involves a certain number of boxes $r_i, 1 \leq r_i \leq r$ and $\sum r_i = r$, where the sum is performed over all the found indecomposable components. Observe that all the indecomposable components, except S_2 (and its complement, its inverse and the inverse of its complement) involve a constant number of boxes (either 1 or 2 or 3). From the other hand, determining x, and hence recognizing S_2, takes time $\Theta(x)$. Since the recognition of component S_2 is executed as last possibility, it follows that the recognition of each indecomposable component P_i, involving r_i boxes, always takes $\Theta(r_i)$ time. Also the update of the degree sequence can be run in the same time provided that a clever implementation is performed. Indeed, in Algorithm `Pruning-Unigraphs`, it is not necessary to update at each step the degree sequence as we do in order to highlight the pruning technique: instead, it is enough to keep an integer variable *prune* and to check conditions on ($d_i -$ *prune*) instead of d_i in order to guarantee that no additional time is used to prune the sequence. Hence the algorithm recognizes if G is a unigraph in time $\Theta(\sum r_i) = \Theta(r) = O(n)$.

In the following, an example of execution of the algorithm is provided.

Let $16^3, 12^4, 9^5, 5^2, 3^1, 2^1, 1^4$ be the degree sequence of an input graph G (depicted in Fig. 6). On this sequence, COND. S_3=true, hence $B_1 \cup B_6 \cup B_7$ induce $S_3(1, 2; 1)$ (see Fig. 7).

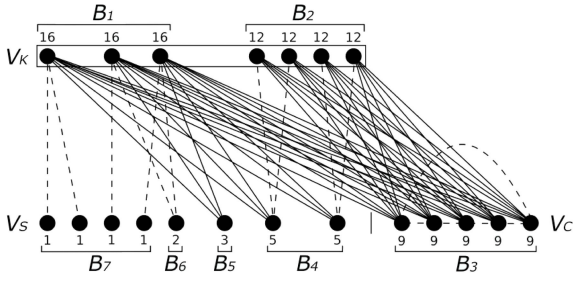


Fig. 6. A unigraph, where sets V_K, V_S and V_C are highlighted

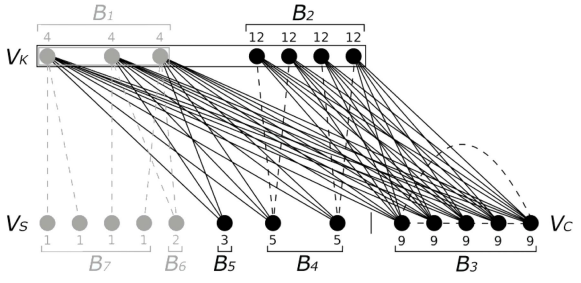


Fig. 7. The unigraph of Fig. 6 where component $S_3(1, 2; 1)$ is highlighted

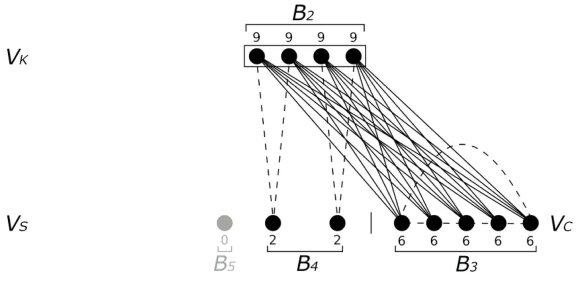


Fig. 8. The unigraph of Fig. 6 after pruning component $S_3(1, 2; 1)$

After the pruning operation, we have the new degree sequence is $9^4, 6^5, 2^2, 0^1$, representing the graph in Fig. 8.

For this degree sequence, COND. I=true, and therefore B_5 induces $K_1 \in V_S$ (see Fig. 8). After the remotion of this box (see Fig. 9), the pruned degree sequence is $9^4, 6^5, 2^2$ and COND. $\overline{S_2}$=true. Hence, we deduce that $B_2 \cup B_4$ induces $\overline{S_2(2, 2)}$, as shown in Fig. 9. The algorithm prunes the sequence obtaining sequence 2^5, that verifies COND. C_5, so B_3 induces C_5.

Also this sequence is pruned and the reduced graph is empty, hence the algorithm returns success.

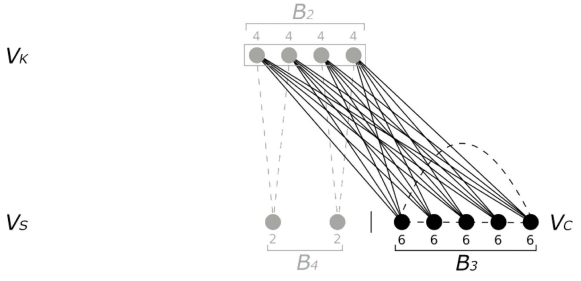


Fig. 9. The unigraph of Fig. 8 after pruning component $K_1 \in V_S$

6 Conclusions

In this paper we presented a linear time algorithm for recognizing unigraphs exploiting a new characterization for these graphs. It partitions the vertex set and the edge set into three and two disjoint sets, respectively, detecting two particular graphs whose superposition generates the given unigraph. As we observed in Section 4, the proof of the known characterization for matrogenic graphs is based on the heredity, that does not hold for unigraphicity; so, we have weakened our requirements proving that only some special subgraphs of a unigraph are still unigraphs. This is the picklock of the proof of our characterization.

Based on this characterization, we designed a linear time recognition algorithm.

We are convinced that the results presented in this paper can be helpful to solve some of those problems that are NP-hard in general, polynomially solved for subclasses of unigraphs and still unknown for unigraphs. This will be a future direction of our work.

References

1. Calamoneri, T., Petreschi, R.: λ-Coloring Matrogenic Graphs. Discrete Applied Mathematics 154(17), 2445–2457 (2006)
2. Chvatal, V., Hammer, P.: Aggregation of inequalities integer programming. Ann. Discrete Math. 1, 145–162 (1977)
3. Ding, G., Hammer, P.: Matroids arisen from matrogenic graphs. Discrete Mathematics 165-166(15), 211–217 (1997)
4. Foldes, S., Hammer, P.: Split graphs. In: Proc. 8th South-East Conference on Combinatorics, Graph Theory and Computing, pp. 311–315 (1977)
5. Foldes, S., Hammer, P.: On a class of matroid producing graphs. Colloq. Math. Soc. J. Bolyai (Combinatorics) 18, 331–352 (1978)
6. Hakimi, S.L.: On realizability of a set of integers and the degrees of the vertices of a linear graph. J. Appl. Math. 10/ 11, 496–506/ 165–147 (1962)
7. Johnson, R.H.: Simple separable graphs. Pacific Journal Math. 56(1), 143–158 (1975)
8. Kleiman, D., Li, S.-Y.: A Note on Unigraphic Sequences. Studies in Applied Mathematics, LIV(4), 283–287 (1975)

9. Koren, M.: Sequences with unique realization by simple graphs. J. of Combinatorial theory ser. B 21, 235–244 (1976)
10. Mahadev, N.V.R., Peled, U.N.: Threshold Graphs and Related Topics. Ann. Discrete Math., vol. 56. North-Holland, Amsterdam (1995)
11. Marchioro, P., Morgana, A., Petreschi, R., Simeone, B.: Degree sequences of matrogenic graphs. Discrete Math. 51, 47–61 (1984)
12. Nikolopoulos, S.D.: Recognizing Cographs And Threshold Graphs through a classification of their edges. Information Processing Letters 75, 129–139 (2000)
13. Orlin, J.B.: The minimal integral separator of a threshold graph. Ann. Discrete Math. 1, 415–419 (1977)
14. Suzdal, S., Tyshkevich, R.: (P,Q)-decomposition of graphs. Tech. Rep. Rostock: Univ. Fachbereich Informatik, II, 16 S (1999)
15. Tyshkevich, R.: Decomposition of graphical sequences and unigraphs. Discrete Math. 220, 201–238 (2000)
16. Tyshkevich, R.: Once more on matrogenic graphs. Discrete Math. 51, 91–100 (1984)

Random Generation and Enumeration of Proper Interval Graphs

Toshiki Saitoh[1], Katsuhisa Yamanaka[2], Masashi Kiyomi[1], and Ryuhei Uehara[1]

[1] School of Information Science, JAIST, Asahidai 1-1, Nomi, Ishikawa 923-1292, Japan
{toshikis,mkiyomi,uehara}@jaist.ac.jp
[2] Graduate School of Information Systems, The University of Electro-Communications, Chofugaoka 1-5-1, Chofu, Tokyo 182-8585, Japan
yamanaka@is.uec.ac.jp

Abstract. We investigate connected proper interval graphs without vertex labels. We first give the number of connected proper interval graphs of n vertices. Using it, a simple algorithm that generates a connected proper interval graph uniformly at random up to isomorphism is presented. Finally an enumeration algorithm of connected proper interval graphs is proposed. The algorithm is based on the reverse search, and it outputs each connected proper interval graph in O(1) time.

Keywords: Counting, enumeration, proper interval graphs, random generation, unit interval graphs.

1 Introduction

Recently we have to process huge amounts of data in the areas of data mining, bioinformatics, etc. In order to find and classify knowledge automatically from the data we sometimes assume that the data is structured, and such structures can be observed implicitly or explicitly. We assume that the data have a certain structure, and we enumerate them, and we test if the assumption is appropriate.

We have to attain three efficiencies to deal with the complex structures: the structure has to be represented efficiently; essentially different instances have to be enumerated efficiently; and the properties of the structure have to be checked efficiently. In the area of graph drawing, there are several researches [5,15,18,24]. From the viewpoint of graph classes, the previously studied structures are relatively primitive, and there are many unsolved problems for more complex structures: Trees are widely investigated as a model of such structured data [10,22,27,28,30], and recently, distance-hereditary graphs are studied [17].

A variety of graph classes have been proposed and studied [6]. Among them, interval graphs have been widely investigated since they were introduced in the 1950s by a mathematician, Hajös, and by a molecular biologist, Benzer, independently [11, Chapter 8]. A graph is called an interval graph if it represents intersecting intervals on a line. In this paper, we aim at a subclass of interval graphs. An interval graph is called a unit interval graph if it has a unit length interval representation. An interval representation is called proper if no interval properly contains another one on the representation. An interval graph is called a proper interval graph if it has a proper interval representation.

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 177–189, 2009.

Interestingly, unit interval graphs coincide with proper interval graphs; those notions define the same class [4,33].

In addition to the fact that proper interval graphs form a basic graph class, they have been of interest from a viewpoint of graph algorithms. There are many problems that can be solved efficiently on proper interval graphs [3,8,9,13,14,31]. It is also known that proper interval graphs are strongly related to the classic NP-hard problem, bandwidth problem [19]. The bandwidth problem is finding a layout of vertices; the objective is to minimize the maximum difference of two adjacent vertices on the layout. The bandwidth problem is NP-hard even on trees [26,32]. The bandwidth problem has been studied since the 1950s; it has many applications including sparse matrix computations (see [7,23] for survey). For any given graph $G = (V, E)$, finding a best layout of vertices is equivalent to finding a proper interval graph $G' = (V, E')$ with $E \subseteq E'$ whose maximum clique size is the minimum among all such proper interval graphs [19]. The proper interval completion problem is also motivated by molecular biology, and hence it attracts much attention (see, e.g., [20]).

In this paper, we investigate counting, random generation, and enumeration of proper interval graphs. More precisely, we aim to count, generate, and enumerate unlabeled connected proper interval graphs. We note that the graphs we deal with are unlabeled. This is reasonable to avoid redundancy from a practical point of view.

Unlabeled proper interval graphs can be naturally represented by a language over an alphabet $\Sigma = \{\text{`['}, \text{`]'}\}$. The number of strings representing proper interval graphs is strongly related to a well known notion called Dyck path, which is a staircase walk from $(0, 0)$ to (n, n) that lies strictly below (but may touch) the diagonal $x = y$. The number of Dyck paths of length n is equal to Catalan number $C(n)$. Thus, our results for counting and random generation of proper interval graphs with n vertices are strongly related to $C(n)$. The main difference between those notions is that we have to mention about isomorphism and symmetricity in the case of proper interval graphs. For example, to generate an unlabeled connected proper interval graph uniformly at random, we have to consider the number of valid representations of each graph since it depends on the symmetricity of the graph. We show in Section 3 that the number of connected proper interval graphs of $n + 1$ vertices is $\frac{1}{2}(C(n) + \binom{n}{\lfloor n/2 \rfloor})$. Extending the result, we give a linear time and space algorithm that generates a connected proper interval graph with n vertices uniformly at random.

Our enumeration algorithm is based on the reverse search developed by Avis and Fukuda [2]. We design a good parent-child relation among the string representations of the proper interval graphs in order to perform the reverse search efficiently. The relation admits us to perform each step of the reverse search in O(1) time, and hence we have an efficient algorithm that enumerates every unlabeled connected proper interval graph with n vertices in O(1) time and O(n) space. (Each graph G is output in the form of the difference between G and the previous one so that the algorithm can output it in O(1) time.) Here we notice that there are some similar known algorithms that enumerate every string of "[" and "]" in constant time [22]. However it is not always possible to obtain a constant delay algorithm for enumerating proper interval graphs from such constant delay string enumeration algorithms. For example think a string "[[...[]]...]" of length $2n$. This represents a complete graph of size n, and the number of edges of

it is $n(n-1)/2$. If we swap the 3rd "[" and the first "]", the number of edges of the represented graph becomes $(n-1)(n-2)/2+1$.

We note that all the results above can be extended from "n vertices" to "at most n vertices". This will be discussed in the concluding remarks.

2 Preliminaries

A graph (V, E) with $V = \{v_1, v_2, \ldots, v_n\}$ is an *interval graph* if there is a finite set of closed intervals $\mathcal{I} = \{I_{v_1}, I_{v_2}, \ldots, I_{v_n}\}$ on the real line such that $\{v_i, v_j\} \in E$ iff $I_{v_i} \cap I_{v_j} \neq \emptyset$ for each i and j with $0 < i, j \leq n$. We call the interval set $\mathcal{I}$ an *interval representation* of the graph. For each interval I, we denote by $L(I)$ and $R(I)$ the left and right endpoints of the interval, respectively (hence we have $L(I) \leq R(I)$ and $I = [L(I), R(I)]$). For two intervals I and J, we write $I \prec J$ if $L(I) \leq L(J)$ and $R(I) \leq R(J)$.

An interval representation is *proper* if no two distinct intervals I and J exist such that I properly contains J or vice versa. That is, either $I \prec J$ or $J \prec I$ holds for every pair of intervals I and J. An interval graph is *proper* if it has a proper interval representation. If an interval graph G has an interval representation $\mathcal{I}$ such that every interval in $\mathcal{I}$ has the same length, G is said to be a *unit interval graph*. Such interval representation is called a *unit interval representation*. It is well known that proper interval graphs coincide with unit interval graphs [33]. That is, given a proper interval representation, we can transform it to a unit interval representation. A simple constructive way of the transformation can be found in [4]. With perturbations if necessary, we can assume without loss of generality that $L(I) \neq L(J)$ (and hence $R(I) \neq R(J)$), and $R(I) \neq L(J)$ for any two distinct intervals I and J in a unit interval representation $\mathcal{I}$.

We denote an alphabet {'[', ']'} by Σ throughout the paper. We encode a unit interval representation $\mathcal{I}$ of a unit interval graph G by a string $s(\mathcal{I})$ in Σ^* as follows; we sweep the interval representation from left to right, and encode $L(I)$ by '[' and encode $R(I)$ by ']' for each $I \in \mathcal{I}$. We call the encoded string *a string representation* of G. We say that string x in Σ^* is *balanced* if the number of '['s in x is equal to that of ']'s. Clearly $s(\mathcal{I})$ is a balanced string of $2n$ letters. Using the construction in [4], $s(\mathcal{I})$ can be constructed from a proper interval representation $\mathcal{I}$ in $O(n)$ time and vice versa since the ith '[' and the ith ']' give the left and right endpoints of the ith interval, respectively.

We define $\overline{[}$ = ']' and $\overline{]}$ = '[' respectively. For two strings $x = x_1x_2\cdots x_n$ and $y = y_1y_2\cdots y_m$ in Σ^*, we say that x is *smaller* than y if (1) $n < m$, or (2) $n = m$ and there exists an index $i \in \{1, \ldots, n\}$ such that $x_{i'} = y_{i'}$ for all $i' < i$ and $x_i =$ '[' and $y_i =$ ']'. If x is smaller than y, we denote $x < y$. We note that the balanced string $x = [[\cdots[]]\cdots]$ is the smallest among those of the same length. For a string $x = x_1x_2\cdots x_n$ we define the *reverse* $\bar{x}$ of x by $\bar{x} = \bar{x}_n\bar{x}_{n-1}\cdots\bar{x}_1$. A string x is *reversible* if $x = \bar{x}$. Here we have the following lemma:

Lemma 1 (See, e.g., [9, Corollary 2.5]). *Let G be a connected proper interval graph, and $\mathcal{I}$ and $\mathcal{I}'$ be any two unit interval representations of G. Then either $s(\mathcal{I}) = s(\mathcal{I}')$ or $s(\mathcal{I}) = \overline{s(\mathcal{I}')}$ holds. That is, the unit interval representation and hence the string representation of a proper interval graph is determined uniquely up to isomorphism.*

A connected proper interval graph G is said to be *reversible* if its string representation is reversible. Note that G is supposed to be connected in Lemma 1. If G is disconnected, we can obtain several distinct string representations by arranging the connected components.

It is easier for our purpose (counting, random generation, and enumeration of unlabeled proper interval graphs) to deal with the encoded strings in Σ^* than to use interval representations. Given an interval representation $\mathcal{I}$ of a proper interval graph G, the smaller of the two string representations $s(\mathcal{I})$ and $\overline{s(\mathcal{I})}$ is called *canonical*. If $s(\mathcal{I})$ is reversible, $s(\mathcal{I})$ is the canonical string representation. Hereafter we sometimes identify a connected proper interval graph G with its canonical string representation.

For a string $x = x_1 x_2 \cdots x_n \in \Sigma^n$ of length n, we define the *height* $h_x(i)$ ($i \in \{0, \ldots, n\}$) as follows; (0) $h_x(0) = 0$, and (1) $h_x(i) = h_x(i-1)+1$ if $x_i =$ '[', and $h_x(i) = h_x(i-1)-1$ otherwise. We say that a string x is *nonnegative* if $\min_i\{h_x(i)\}$ is equal to 0 (we do not have $\min_i\{h_x(i)\} > 0$ since $h_x(0) = 0$). The following observation is immediate:

Observation 1. *Let $x = x_1 x_2 \cdots x_{2n}$ be a string in Σ^{2n}. (1) x is a string representation of a (not necessarily connected) proper interval graph if and only if x is balanced and nonnegative. (2) x is a string representation of a connected proper interval graph if and only if $x_1 =$ '[' and $x_{2n} =$ ']', and the string $x_2 \cdots x_{2n-1}$ is balanced and nonnegative.*

A balanced nonnegative string of length $2n$ corresponds to a well-known notion called *Dyck path*, which is a staircase walk from $(0, 0)$ to (n, n) that lies strictly below (but may touch) the diagonal $x = y$. The number of Dyck paths of length n is equal to *Catalan number* $C(n) = \frac{1}{n+1}\binom{2n}{n}$; see [34, Corollary 6.2.3] for further details. We note that Observation 1 does not care about isomorphism up to reversal. We have to avoid the duplications of isomorphic graphs for counting the number of mutually nonisomorphic graphs, and for uniform random generation of them.

We use one of the generalized notions of Catalan number: The number of nonnegative strings $x = x_1 x_2 \cdots x_n$ of length n with $h_x(n) = i > 0$ equals $C(n, i) = \frac{i}{2n+i}\binom{2n+i}{n}$. This can be obtained by a generalized Raney's lemma about m-Raney sequences with letting $m = 2$; see [12, Equation (7.69), p. 349] for further details.

Time complexity is measured by the number of arithmetic operations in this paper. Especially we assume that each binomial coefficient, each Catalan number, and its generalization can be computed in O(1) time. Moreover we assume that the basic arithmetic operations of these numbers can be done in O(1) time. Binomial coefficients and Catalan number $C(n, i)$ require O(n) bits, so this assumption is clearly out of the standard RAM model. We have to multiply the time complexity of calculation of these numbers to the time complexities we show, in the standard RAM model case. We employ the assumption only in Section 3 to simplify the discussion. It is worth stating that the enumeration algorithm in Section 4 does not require the assumption, and all the results in Section 4 are valid on the standard RAM model.

3 Counting and Random Generation

In this section, we count the number of mutually nonisomorphic proper interval graphs. And we propose an algorithm that efficiently generates a proper interval graph uniformly at random. First we show the following theorem:

Theorem 1 (Karttunen 2002). *For any positive integer n, the number of connected proper interval graphs of $n+1$ vertices is $\frac{1}{2}(C(n) + \binom{n}{\lfloor n/2 \rfloor})$.*

Note that Theorem 1 has been already mentioned informally by Karttunen in 2002 [21]. We here give an explicit proof so that we use some notions for random generation.

Proof. We define three sets $R(n), S(n)$, and $T(n)$ of strings in Σ^{2n} of length $2n$ by

$R(n) = \{x \mid x$ is balanced, nonnegative, $|x| = 2n$, and x is reversible$\}$,
$S(n) = \{x \mid x$ is balanced, nonnegative, $|x| = 2n$, and x is not reversible$\}$, and
$T(n) = \{x \mid x$ is balanced, nonnegative, and $|x| = 2n\}$.

The number of connected proper interval graphs of $n+1$ vertices is equal to $|S(n)|/2 + |R(n)| = |T(n)|/2 + |R(n)|/2 = \frac{1}{2}(C(n) + |R(n)|)$, by Observation 1. The number of elements in $R(n)$ is equal to that of nonnegative strings x' of length n, since each reversible string x is obtained by the concatenation of strings x' and $\bar{x}'$.

Now the task is the evaluation of the number of nonnegative strings x of length n. Let $f(n,h)$ be the number of nonnegative strings $x = x_1 \cdots x_n$ of length n with $h_x(n) = h$. Clearly we have $f(n,h) = 0$ if $h > n$. The following equations hold for each integers i and k with $0 \le i \le k$. (1) $f(2k, 2i+1) = 0$, $f(2k+1, 2i) = 0$, (2) $f(2k, 0) = C(k)$, $f(k,k) = 1$, and (3) $f(k,i) = f(k-1, i-1) + f(k-1, i+1)$. We have the following equation for $0 \le h \le n$:

$$f(n,h) = \begin{cases} 0 & \text{if } n-h \text{ is odd, and} \\ \frac{h+1}{n+1}\binom{n+1}{(n-h)/2} & \text{if } n-h \text{ is even,} \end{cases}$$

by carefully applying the generalized Raney's lemma about m-Raney sequences with $m = 2$ to our representation [12, Section 7.5].

It is necessary to show $\sum_{i=0}^{n} f(n,i) = \binom{n}{\lfloor \frac{n}{2} \rfloor}$ to complete the proof. This equation can be obtained from Equation (5.18) in [12]. □

Next we consider the uniform random generation of a proper interval graph of n vertices.

Theorem 2. *For any given positive integer n, a connected proper interval graph with n vertices can be generated uniformly at random in $O(n)$ space. The time complexity to generate a string representation of proper interval graph uniformly is $O(n)$, while that to convert it to a graph representation is $O(n+m)$ where m is the number of edges of the created graph.*

Proof. (Outline) We denote by $y = y_1 \cdots y_{2n}$ the canonical string of a connected proper interval graph $G = (V, E)$ to be obtained. We fix $y_1 =$ '[' and $y_{2n} =$ ']' and generate $x = x_1 \cdots x_{2n'}$ with $y = [x]$, where $n' = n - 1$ and x is a balanced nonnegative string.

The idea is simple; just generate balanced nonnegative string x. However each non-reversible graph corresponds to two balanced nonnegative strings, while each reversible graph corresponds to one balanced nonnegative (reversible) string. In order to adjust the generation probabilities, the algorithm first selects which type of string to generate: (1) a balanced nonnegative string (that can be reversible) with probability $|T(n')|/(|T(n')| + |R(n')|)$ or (2) a balanced nonnegative reversible string with probability

$|R(n')|/(|T(n')| + |R(n')|)$. This probabilistic choice adjusts the generation probabilities between reversible graphs and non-reversible graphs, by the equations in Theorem 1. In each case, the algorithm generates each string uniformly at random using the function $f(n, h)$ introduced in the proof of Theorem 1 as follows:

Case 1: Generation of a balanced nonnegative string of length $2n'$ uniformly at random. There is a known algorithm for this purpose [1]. We simply generate sequence of '[' and ']' from left to right. Assume that the algorithm has already generated a nonnegative string $x_1 \cdots x_k$ of length k with $k < 2n'$. Choose '[' as the next letter with probability $\frac{(h_x(k)-r)(r+2)}{2h_x(k)(r+1)}$ and choose ']' with probability $\frac{r(h_x(k)+r+2)}{2h_x(k)(r+1)}$, where r is equal to $2n' - k$. Then we have a balanced nonnegative string of length $2n'$ uniformly at random.

Case 2: Generation of a balanced nonnegative reversible string of length $2n'$ uniformly at random. The desired balanced nonnegative reversible string x can be represented as $x = x_1x_2 \cdots x_{n'-1}x_{n'}\bar{x}_{n'}\bar{x}_{n'-1} \cdots \bar{x}_2\bar{x}_1$, where $x_1x_2 \cdots x_{n'}$ is a nonnegative string of length n'. We thus generate a nonnegative string $x' := x_1x_2 \cdots x_{n'}$ of length n' uniformly at random.

Unfortunately, similar approach to Case 1 does not work; given a positive prefix $x_1x_2 \cdots x_i$, it seems to be hard to generate $x_{i+1} \cdots x_{n'}$ that ends at some $h_x(n')$ uniformly, since the string may pass below both of $h_x(i)$ and $h_x(n')$.

The key idea is to generate the desired string backwardly. This step consists of two phases. The algorithm first chooses the height $h_x(n')$ of the last letter $x_{n'}$ randomly. The number of nonnegative strings ending at height h is $f(n', h)$, and $\sum_{i=0}^{n'} f(n', i) = \binom{n'}{\lfloor n'/2 \rfloor}$, by the proof of Theorem 1. Thus the algorithm first sets, for each $h \in \{0, \ldots, n'\}$, $h_x(n') = h$ with probability $f(n', h)/\binom{n'}{\lfloor n'/2 \rfloor}$. Then the algorithm randomly chooses the height $h_x(i)$ from $h_x(i+1)$ for each $i = n' - 1, n' - 2, \ldots, 1$. The height $h_x(i)$ is $h_x(i) = h_x(i+1)+1$ with probability $p/(p+q)$ and $h_x(i) = h_x(i+1)-1$ with probability $q/(p+q)$, where $p = \frac{h_x(i+1)+1}{2i+h_x(i+1)+1}\binom{2i+h_x(i+1)+1}{i}$, and $q = \frac{h_x(i+1)-1}{2i+h_x(i+1)-1}\binom{2i+h_x(i+1)-1}{i}$. The algorithm finally obtains $h_x(1) = 1$ and $h_x(0) = 0$ with probability 1 after repeating this process. The string $x' = x_1x_2 \cdots x_{n'-1}x_{n'}$ can be obtained by traversing the sequence of the heights backwards, and hence we can obtain $x = x'\bar{x}'$.

By the assumption (a binomial coefficient, a Catalan number $C(n)$, and its generalization $C(n, i)$ can be computed in O(1) time), it is easy to see that the algorithm runs in O(n) time and space. □

Note that the only part that requires O(n) space is the generation of $x' = x_1x_2 \cdots x_{n'}$ from the sequence of their heights $h_x(n'), h_x(n'-1), \ldots, h_x(1)$ in Case 2 in the proof of Theorem 2. Therefore, if we admit to output the reversible string $x\bar{x}$ by just $\bar{x}$, the algorithm in Theorem 2 requires space of only O(n) bit.

4 Enumeration

We here enumerate all connected proper interval graphs with n vertices. It is sufficient for our enumeration to enumerate each string representation of connected proper interval graphs, by Lemma 1. Let S_n be the set of balanced and canonical strings

$x = x_1x_2\cdots x_{2n}$ in Σ^{2n} such that $x_1 =$ '[', $x_{2n} =$ ']', and the string $x_2\cdots x_{2n-1}$ is nonnegative. We define a tree structure, called *family tree*, in which each vertex corresponds to each string in S_n. We enumerate all the strings in S_n by traversing the family tree. Since S_n is trivial when $n = 1, 2$, we assume $n > 2$.

We start with some definitions. Let $x = x_1x_2\cdots x_{2n}$ be a string in Σ^{2n}. If $x_ix_{i+1} = [\,]$, i is called a *front index* of x. Contrary, if $x_ix_{i+1} =][$, i is called a *reverse index* of x. For example, a string [[[[ ][ ][ ]][ ]][ ][ ]] has 6 front indices 4, 6, 8, 11, 14, and 16, and has 5 reverse indices 5, 7, 10, 13, and 15. The string $[^n]^n$ in S_n is called *root* and denoted by r_n. Let $x = x_1x_2\cdots x_{2n}$ be a string in Σ^{2n}. We denote the string $x_1x_2\cdots x_{i-1}\bar{x}_i\bar{x}_{i+1}x_{i+2}\cdots x_{2n}$ by $x[i]$ for $i = 1, 2, \ldots, 2n-1$. We define $P(x)$ by $x[j]$ for $x \in \Sigma^{2n} \setminus \{r_n\}$, where j is the minimum reverse index of x. For example, for $x =$[[[[ ][ ]][ ]][ ][ ]], we have $P(x) =$[[[[ ][ ]][ ]][ ][ ]] (the flipped pair is enclosed by the grey box and the minimum reverse indices are underlined).

Lemma 2. *For every $x \in S_n \setminus \{r_n\}$, we have $P(x) \in S_n$.*

Proof. (Outline) For any $x \in S_n \setminus \{r_n\}$, it is easy to see that $P(x) = x'_1x'_2\cdots x'_{2n}$ satisfies that $x'_1 =$ '[', $x'_{2n} =$ ']', $x'_2\cdots x'_{2n-1}$ is balanced and nonnegative. Thus we show that $P(x)$ is canonical.

We first assume that x is reversible. Then the minimum reverse index j of x satisfies $j \le n$. If $j = n$, $P(x)$ is still reversible and $P(x)$ is thus canonical. When $j < n$, we have $P(x) < \overline{P(x)}$ and $P(x)$ is thus canonical again.

Next we consider the case that x is not reversible. There must be an index i such that $x_i = x_{2n-i+1} =$ '[', and $x_{i'} = \bar{x}_{2n-i'+1}$ for all $1 \le i' < i$, since x is canonical. Moreover we have $1 \le i < n$ since x is balanced. Let ℓ be the minimum reverse index of x. We first observe that $\ell \ne i$ since $x_\ell =$ ']'. We also see that $\ell < 2n - i + 1$ since $x_{i'} = \bar{x}_{2n-i'+1}$ for all $1 \le i' < i$. Now we have three cases (1) $\ell < i - 1$, (2) $i < \ell < 2n - i + 1$, and (3) $\ell = i - 1$. We can easily check $P(x) < \overline{P(x)}$ in any case. □

Next we define the family tree among strings in S_n. We call $P(x)$ *the parent* of x, and x is a *child* of $P(x)$ for each $x \in S_n \setminus \{r_n\}$. Note that $x \in S_n$ may have multiple or no children while each string $x \in S_n \setminus \{r_n\}$ has the unique parent $P(x) \in S_n$. Given a string x in $S_n \setminus \{r_n\}$, we have the unique sequence $x, P(x), P(P(x)), \ldots$ of strings in S_n by repeatedly finding the parent. We call it *the parent sequence* of x. For example, for $x =$[[[ ][ ]][ ]], we have $P(x) =$[[[[ ]][ ]], $P(P(x))=$[[[[ ][ ]]], $P(P(P(x))) =$[[[[ ][ ]]]], and $P(P(P(P(x)))) = r_5$. The next lemma ensures that the root r_n is the common ancestor of all the strings in S_n.

Lemma 3. *The parent sequence of x in S_n eventually ends up with r_n.*

Proof. (Outline) For a string $x = x_1x_2\cdots x_{2n}$ in S_n, we define a potential function $p(x) = \sum_{i=1}^{n} 2^{n-i}b(x_i) + \sum_{i=1}^{n} 2^{i-1}(1 - b(x_{n+i}))$, where $b($'['$) = 0$ and $b($']'$) = 1$. For any $x \in S_n$, $p(x)$ is a non-negative integer, and $p(x) = 0$ if and only if $x = r_n$. Additionaly it is easy to see that $p(P(x)) < p(x)$ for any $x \in S_n \setminus \{r_n\}$. Thus, by repeatedly finding the parent from a string in S_n, we eventually obtain the root r_n. □

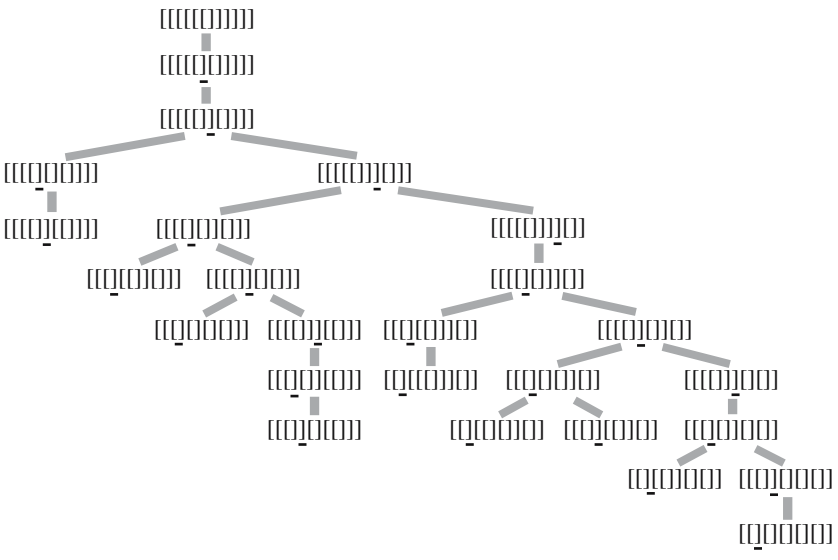


Fig. 1. The family tree T_6

We have *the family tree* T_n of S_n by merging all the parent sequences. Each vertex in the family tree T_n corresponds to each string in S_n, and each edge corresponds to each parent-child relation. See Fig. 1 for example.

Now we give an algorithm that enumerates all the strings in S_n. The algorithm traverses the family tree by reversing the procedure of finding the parent as follows. Given a string x in S_n, we enumerate all the children of x. Every child of x is in the form $x[i]$ where i is a front index of x. We consider the following cases to find every i such that $x[i]$ is a child of x.

Case 1: String x is the root r_n. The string x has exactly one front index n. Since $P(x[n]) = x$, $x[n]$ is a child of x. Since $x[i]$ is not a child of x when i is not a front index, x has exactly one child.

Case 2: String x is not the root. In this case, x has at least two front indices. Let i be any front index, and j be the minimum reverse index. If $j + 1 < i$ then $x \neq P(x[i])$, since i is not the minimum reverse index of $x[i]$. If $i \leq j + 1$, $x[i]$ may be a child of x. We call such i a *candidate index* of x. For a candidate index i, if $x[i]$ is in S_n (i.e. $x[i]$ is canonical), i is called a *flippable index* and $x[i]$ is a child of x. Since there must exist a reverse index between any two front indices, x has at most two candidate indices. Thus x has at most two children. For example, x = [[[]]][] has two candidate indices 3 and 6, one reverse index 5, and one child $x[3]$ = [[][][]].

Given a string x in S_n, we can enumerate all the children of x by the case analysis above. We can traverse T_n by repeating this process from the root recursively. Thus we can enumerate all the strings in S_n.

Now we have the following algorithm and lemma.

Procedure find-all-children($x = x_1x_2 \cdots x_{2n}$) // x is the current string.
begin
01 Output x // Output the difference from the previous string.
02 **for each** flippable index i
03 **find-all-children**($x[i]$) // Case 2
end

Algorithm find-all-strings(n)
begin
01 Output the root $x = r_n$
02 **find-all-children**($x[n]$) // Case 1
end

Lemma 4. *Algorithm find-all-strings*(n) *enumerates all the strings in* S_n.

By Lemma 4 we can enumerate all the strings in S_n. We need two more lemmas to generate each string in O(1) time. First we show an efficient construction of the candidate indices list.

Lemma 5. *Given a string x in S_n and its flippable indices, we can construct the candidate indices list of each child of x in $O(1)$ time.*

Proof. (Outline) Let $x[i]$ be a child of x. Each string x in S_n has at most two flippable indices (see the proof of Lemma 3). Let a and b be two flippable indices of x, and let a' and b' be two flippable indices of $x[i]$ (if they exist), respectively. Without loss of generality, we assume $a < b$ and $a' < b'$. With careful case analysis, we obtain either (1) $a' = a - 1$, $b' = a + 1$, (2) $a' = a - 1$ and b' does not exist, (3) $a' = a$, $b' = b + 1$, (4) $a' = a$, and b' does not exist, or (5) $x[i]$ has no child.

In each case the candidate indices list can be updated in O(1) time. □

Since the number of candidate indices of x is at most two, our family tree is a binary tree. We note that a candidate index of x can become "non-candidate" In the case, such index does not become a candidate index again.

Next lemma shows that there is a method of determining whether a candidate index is flippable.

Lemma 6. *One can determine whether or not a candidate index is flippable in $O(1)$ time.*

Proof. (Outline) Let $x = x_1x_2 \cdots x_{2n}$ be a string in S_n and a be a candidate index of x. We denote $x[a] = y = y_1y_2 \cdots y_{2n}$. A candidate index a is flippable if and only if $x[a]$ is canonical and $y_2y_3 \cdots y_{2n-1}$ is nonnegative.

We first check whether or not a string $y_2y_3 \cdots y_{2n-1}$ is nonnegative. We have $h_y(a) = h_x(a) - 2$ and $h_y(i) = h_x(i)$ for each $1 \le i < a$ and $a < i \le 2n$, since $a > 1$, $y_ay_{a+1} =$ '][', $x_ax_{a+1} =$ '[]', and $x_i = y_i$ for each $1 \le i < a$ and $a+1 < i \le 2n$. Thus $y_2y_3 \cdots y_{2n-1}$ is nonnegative if and only if $h_x(a) > 2$. Therefore we can check the negativity of $y_2y_3 \cdots y_{2n-1}$ in O(1) time using an array of size n to maintain the sequence of heights of the string. Updates of the array also can be done in O(1) time.

We next check whether or not a string is canonical. We call $x^{\mathrm{L}} = x_1 x_2 \cdots x_n$ the *left string* of x, and $x^{\mathrm{R}} = \overline{x}_{2n}\overline{x}_{2n-1}\cdots\overline{x}_{n+1}$ the *right string* of x. Then x is canonical if and only if $x^{\mathrm{L}}_j \leq x^{\mathrm{R}}_j$. We maintain a doubly linked list L in order to check it in O(1) time. The list L maintains the indices of different characters in x^{L} and x^{R}. First L is initialized by an empty since $x^{\mathrm{L}} = x^{\mathrm{R}}$ for $x = r_n$. In general L is empty if and only if x is reversible. We can check whether $x^{\mathrm{L}} < x^{\mathrm{R}}$ by comparing $x^{\mathrm{L}}_{L[1]}$ and $x^{\mathrm{R}}_{L[1]}$. When x is updated by $x[a]$, x^{L} and x^{R} changes in at most two consecutive indices. To find such indices, we also maintain two pointers associated to a that point $L[i]$ and $L[i+1]$ with $L[i] \leq a \leq L[i+1]$. The update of L can be done in O(1) time with these pointers. □

Lemmas 5 and 6 show that we can maintain the list of flippable indices of each string in O(1) time, during the traversal of the family tree. Thus we have the following lemma.

Lemma 7. *Our enumeration algorithm uses $O(n)$ space and runs in $O(|S_n|)$ time.*

By the lemma above, our algorithm generates each string in S_n in O(1) time "on average". However it may have to return from the deep recursive calls without outputting any string after generating a string corresponding to the leaf of a large subtree in the family tree. This takes much time. Therefore each string cannot be generated in O(1) time in the worst case.

This delay can be canceled by outputting the strings in the "prepostorder" manner in which strings are outputted in the preorder (and postorder) manner at the vertices of odd (and even, respectively) depth of the family tree. See [29] for further details of this method; in [29] the method was not explicitly named, and the name "prepostorder" was given by Knuth [16]. Now we have the main theorem in this section.

Theorem 3. *After outputting the root in $O(n)$ time, the algorithm enumerates every string in S_n in $O(1)$ time.*

Let G and $G[i]$ be two proper interval graphs corresponding to a string x and its child $x[i]$, respectively. We note that $G[i]$ can be obtained from G by removing the one edge which represents a intersection between (1) the interval with the right endpoint corresponding to x_{i+1} and (2) one with the left endpoint corresponding to x_i. And the root string represents a complete graph. Therefore our algorithm can be modified to deal with the graphs themselves without loss of efficiency. Note that it is not true that every constant delay enumeration algorithm for parentheses applys to that for proper interval graphs since the sizes of differences may not equal among string representations and graph representations.

Theorem 4. *After outputting the n-vertex complete graph in $O(n^2)$ time, the algorithm enumerates every connected proper interval graph of n vertices in $O(1)$ time.*

5 Conclusion

We concentrate to deal with the graphs of n vertices in this paper. For the counting and random generation, it is straightforward to extend the results to those of graphs with at most n vertices. For the enumeration, a naive way of enumerating i-vertex proper

interval graphs for each i is not sufficient, since the difference between the roots of proper interval graphs of i vertices and those of $i + 1$ vertices is not constant. However we can extend our results with suitable parent-child relation. Precisely we extend the relation and define the parent of the root r_i by $[^i]^{i-1}[] \in S_{i+1}$. From the viewpoint of the graphs, we add a pendant vertex to a complete graph of i vertices. From the algorithmic point of view, when the algorithm outputs the graph of $i + 1$ vertices, it recursively calls itself with the root r_i as a child of the graph, and enumerates all the smaller graphs. Thus we have the following corollary.

Corollary 1. *For any given positive integer n, (1) the number of connected proper interval graphs of at most n vertices can be computed in $O(n)$ time and space, (2) a connected proper interval graph of at most n vertices can be generated uniformly at random, and (3) there exists an algorithm that enumerates every connected proper interval graph of at most n vertices in $O(1)$ time and $O(n)$ space.*

We investigate unlabeled connected proper interval graphs. In some cases labeled graphs may be required. Modifying our algorithms to deal with labeled graphs are straightforward. In Observation 1, it is shown that any (not necessarily connected) proper interval graph can be represented by a balanced and nonnegative string. However, in the case, we have to deal with two or more connected components. Our algorithm for enumeration can be extended to disconnected case straightforwardly, however, counting and random generation cannot be.

To deal with unlabeled graphs, it is important to determine whether or not two unlabeled graphs are isomorphic. In this sense, counting/random generation/enumeration on a graph class seems to be intractable if the isomorphism problem for the class is as hard as that for general graphs (See [35] for further details of this topic). It is known that the graph isomorphism problem can be solved in linear time for interval graphs [25]. Hence the future work would be the extensions of our algorithms to general unlabeled interval graphs.

References

1. Arnold, D.B., Sleep, M.R.: Uniform Random Generation of Balanced Parenthesis Strings. ACM Transaction Programming Languages and Systems 2(1), 122–128 (1980)
2. Avis, D., Fukuda, K.: Reverse Search for Enumeration. Discrete Applied Mathematics 65, 21–46 (1996)
3. Bertossi, A.A.: Finding Hamiltonian Circuits in Proper Interval Graphs. Information Processing Letters 17(2), 97–101 (1983)
4. Bogart, K.P., West, D.B.: A short proof that 'proper=unit'. Discrete Mathematics 201, 21–23 (1999)
5. Bonichon, N.: A bijection between realizers of maximal plane graphs and pairs of non-crossing Dyck paths. Discrete Mathematics 298, 104–114 (2005)
6. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. SIAM, Philadelphia (1999)
7. Chinn, P.Z., Chvátalová, J., Dewdney, A.K., Gibbs, N.E.: The bandwidth problem for graphs and matrices — A survey. Journal of Graph Theory 6, 223–254 (1982)
8. de Figueiredo, C.M.H., Meidanis, J., de Mello, C.P.: A lexBFS Algorithm for Proper Interval Graph Recognition. IC DCC-04/93, Instituto de Computação, Universidade Estadual de Campinas (1993)

9. Deng, X., Hell, P., Huang, J.: Linear-time Representation Algorithms for Proper Circular-arc Graphs and Proper Interval Graphs. SIAM Journal on Computing 25(2), 390–403 (1996)
10. Geary, R., Rahman, N., Raman, R., Raman, V.: A Simple Optimal Representation for Balanced Parentheses. In: Sahinalp, S.C., Muthukrishnan, S.M., Dogrusoz, U. (eds.) CPM 2004. LNCS, vol. 3109, pp. 159–172. Springer, Heidelberg (2004)
11. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. In: Annals of Discrete Mathematics, 2nd edn., vol. 57. Elsevier, Amsterdam (2004)
12. Graham, R.L., Knuth, D.E., Patashnik, O.: Concrete Mathematics. Addison-Wesley Publishing Company, Reading (1989)
13. Hell, P., Huang, J.: Certifying LexBFS recognition algorithms for proper interval graphs and proper interval bigraphs. SIAM J. Discrete Math. 18, 554–570 (2005)
14. Hell, P., Shamir, R., Sharan, R.: A fully dynamic algorithm for recognizing and representing proper interval graphs. SIAM Journal on Computing 31, 289–305 (2001)
15. Nakano, S.-i.: Enumerating Floorplans with n Rooms. IEICE Transactions on Fundamentals E85-A(7), 1746–1750 (2002)
16. Nakano, S.-i.: Personal communication (2008)
17. Nakano, S.-i., Uehara, R., Uno, T.: A New Approach to Graph Recognition and Applications to Distance-Hereditary Graphs. In: Cai, J.-Y., Cooper, S.B., Zhu, H. (eds.) TAMC 2007. LNCS, vol. 4484, pp. 115–127. Springer, Heidelberg (2007)
18. Kaneko, Y., Nakano, S.-i.: Random Generation of Plane Graphs and Its Application. IEICE Transactions on Fundamentals J85-A(9), 976–983 (2002)
19. Kaplan, H., Shamir, R.: Pathwidth, Bandwidth, and Completion Problems to Proper Interval Graphs with Small Cliques. SIAM Journal on Computing 25(3), 540–561 (1996)
20. Kaplan, H., Shamir, R., Tarjan, R.E.: Tractability of Parameterized Completion Problems on Chordal, Strongly Chordal, and Proper Interval Graphs. SIAM Journal on Computing 28(5), 1906–1922 (1999)
21. Karttunen, A.: Personal communication (2008)
22. Knuth, D.E.: Generating All Trees, 4th edn. The Art of Computer Programming, vol. 4. Addison-Wesley, Reading (2005)
23. Lai, Y.L., Williams, K.: A survey of solved problems and applications on bandwidth, edgesum, and profile of graphs. Journal of Graph Theory 31(2), 75–94 (1999)
24. Li, Z., Nakano, S.-i.: Efficient generation of plane triangulations without repetitions. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 433–443. Springer, Heidelberg (2001)
25. Lueker, G.S., Booth, K.S.: A Linear Time Algorithm for Deciding Interval Graph Isomorphism. Journal of the ACM 26(2), 183–195 (1979)
26. Monien, B.: The Bandwidth Minimization Problem for Caterpillars with Hair Length 3 is NP-Complete. SIAM J. Alg. Disc. Meth. 7(4), 505–512 (1986)
27. Munro, J.I., Raman, V.: Succinct Representation of Balanced Parentheses, Static Trees and Planar graphs. In: Proc. 38th ACM Symp. on the Theory of Computing, pp. 118–126. ACM Press, New York (1997)
28. Munro, J.I., Raman, V.: Succinct Representation of Balanced Parentheses and Static Trees. SIAM Journal on Computing 31, 762–776 (2001)
29. Nakano, S.-i., Uno, T.: Constant time generation of trees with specified diameter. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 33–45. Springer, Heidelberg (2004)
30. Nakano, S.-i.: Efficient Generation of Plane Trees. Information Processing Letters 84(3), 167–172 (2002)

31. Panda, B.S., Das, S.K.: A Linear Time Recognition Algorithm for Proper Interval Graphs. Information Processing Letters 87, 153–161 (2003)
32. Papadimitriou, C.H.: The NP-completeness of the bandwidth minimization problem. Computing 16, 263–270 (1976)
33. Roberts, F.S.: Indifference graphs. In: Harary, F. (ed.) Proof Techniques in Graph Theory, pp. 139–146. Academic Press, London (1969)
34. Stanley, R.P.: Enumerative Combinatorics, vol. 2. Cambridge (1997)
35. Uehara, R., Toda, S., Nagoya, T.: Graph Isomorphism Completeness for Chordal Bipartite Graphs and Strongly Chordal Graphs. Discrete Applied Mathematics 145(3), 479–482 (2004)

A Fully Dynamic Graph Algorithm for Recognizing Proper Interval Graphs

Louis Ibarra

School of Computing, DePaul University, Chicago, IL, U.S.A.,
ibarra@cs.depaul.edu

Abstract. We present a fully dynamic graph algorithm to recognize proper interval graphs that runs in $O(\log n)$ worst case time per edge update, where n is the number of vertices in the graph. The algorithm also maintains the connected components and supports connectivity queries in $O(\log n)$ time.

1 Introduction

1.1 Dynamic Graph Algorithms

A dynamic graph algorithm maintains a solution to a graph problem as the graph undergoes a series of small changes, such as single edge deletions or insertions. For every change, the algorithm updates the solution faster than recomputing the solution from scratch, i.e., with no previously computed information. Typically, a dynamic graph algorithm has a preprocessing step to compute a solution for the initial graph, along with some auxiliary information.

We consider dynamic graph algorithms that support the following two operations: a *query* is a question about the solution being maintained, e.g., "Are vertices u, v connected?" or "Is the graph planar?", and an *update* is an edge deletion or edge insertion. A *fully* dynamic graph algorithm supports both deletions and insertions. Fully dynamic algorithms have been developed for numerous problems on undirected graphs, including connectivity, biconnectivity, 2-edge connectivity, bipartiteness, minimum spanning trees, and planarity [7,8]. There are also algorithms that support vertex insertions and vertex deletions, e.g., [10].

1.2 Previous Work

Chordal graphs can be recognized in $O(m + n)$ time using a graph search such as Lex-BFS or Maximum Cardinality Search [21,22]. Several well-known NP-complete problems can be solved on chordal graphs in $O(m+n)$ time [18]. There is a fully dynamic algorithm that maintains a clique tree of a chordal graph in $O(n)$ time per update [15]. This algorithm supports updates that yield a chordal graph and queries that ask whether a particular update is supported. (All the running times in this paper are worst-case.)

Interval graphs can be recognized in $O(m+n)$ time using PQ-trees [2], MPQ-trees [19], a substitution decomposition computed with Lex-BFS [12], or a vertex

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 190–201, 2009.

ordering computed with multiple passes of Lex-BFS [5]. Many well-known NP-complete graph problems can be solved on interval graphs in polynomial time [18]. There is an incremental algorithm for interval graphs that runs in $O(\log n)$ time per vertex insertion and $O(m + n \log n)$ total time [11]. This algorithm supports vertex insertions that yield an interval graph and queries that ask whether a particular insertion is supported. There is a fully dynamic algorithm for recognizing interval graphs that runs in $O(n \log n)$ time per update [14]. This algorithm supports updates that yield an interval graph and queries that ask whether a particular update is supported.

Proper interval graphs can also be recognized in $O(m + n)$ time [4,6]. There is a fully dynamic algorithm for recognizing a proper interval graph in $O(\log n)$ time per edge update and $O(d + \log n)$ time per vertex update, where d is the degree of the vertex [10]. This algorithm supports updates that yield a proper interval graph and queries that ask whether a particular update is supported. The algorithm supports connectivity queries in $O(\log n)$ time.

The clique-separator graph of a chordal graph G is a graph $\mathcal{G}$ whose nodes are the maximal cliques and minimal vertex separators of G and whose (directed) arcs and (undirected) edges represent the containment relations between the sets corresponding to the nodes [13]. The clique-separator graph reflects the structure of G and it has various structural properties when G is an interval graph, proper interval graph, or split graph. The clique-separator graph can be constructed in $O(n^3)$ time if G is a chordal graph, in $O(n^2)$ time if G is an interval graph, and in $O(m + n)$ time if G is a proper interval graph [13].

1.3 Our Results

Our algorithm for recognizing proper interval graphs runs in $O(\log n)$ time per edge update. It also maintains the connected components and supports connectivity queries in $O(\log n)$ time. This matches the running time of the algorithm in [10] for edge updates and connectivity queries, but it is somewhat simpler; the only data structure it uses is balanced binary trees. Also, the algorithm in [10] maintains a straight enumeration of a proper interval graph; our algorithm maintains the clique-separator graph of a proper interval graph G, which is a path $\mathcal{P}$ closely related to the clique path P of G. The clique-separator graph is the basis of a very simple algorithm to find a Hamiltonian cycle in a proper interval graph [16] and the clique path has many applications related to chordal graphs [1]. In our data structure, each vertex v is stored in the leftmost and rightmost nodes of $\mathcal{P}$ that contain v; these nodes specify an interval for v. Each node of $\mathcal{P}$ has a balanced binary tree of the vertices that it stores. When $\mathcal{P}$ is updated, a constant number of trees is moved to the left or right on $\mathcal{P}$, which extends or reduces the intervals for the corresponding vertices. (In the algorithm for chordal graphs in [15], each vertex v is stored in every node of the clique tree that contains v. In the algorithm in this paper, each vertex v is stored only in the leftmost and rightmost nodes of $\mathcal{P}$ that contain v.) Computing the data structure for a proper interval graph G requires $O(m + n)$ time [17]. Given the data structure, computing an interval representation of G requires $O(n \log n)$ time [17].

Since proper interval graphs $\subset$ interval graphs $\subset$ chordal graphs, and we will use results for these graph classes, we review them and the clique-separator graph in Section 2. We present the data structure used by the algorithm in Section 3. We present the insert operation in Section 4 and the delete operation in Section 5. We present conclusions in Section 6.

2 Preliminaries

Throughout this paper, let $G = (V, E) = (V(G), E(G))$ be a simple, undirected graph and let $n = |V|$ and $m = |E|$. For a set $S \subseteq V$, the subgraph of G *induced* by S is $G[S] = (S, E(S))$, where $E(S) = \{\{u, v\} \in E \mid u, v \in S\}$. For a set $S \subset V$, $G-S$ denotes $G[V-S]$. A *clique* of G is a set of pairwise adjacent vertices of G. A *maximal clique* of G is a clique of G that is not properly contained in any clique of G. Figure 1a shows a graph with maximal cliques $\{x, y, z\}, \{y, z, w\}, \{u, z\}$, and $\{v, z\}$.

Let $S \subset V$. S is a *separator* of G if there exist two vertices that are connected in G and not connected in $G-S$. S is a *minimal separator* of G if S is a separator of G that does not properly contain any separator of G. For $u, v \in V$, S is a *uv-separator* of G if u, v are connected in G and not connected in $G - S$. S is a *minimal vertex separator* of G if for some $u, v \in V$, S is a uv-separator of G that does not properly contain any uv-separator of G. Every minimal separator is a minimal vertex separator, but not vice versa. Figure 1a shows a graph with minimal separator $\{z\}$ and minimal vertex separators $\{z\}$ and $\{y, z\}$. For $u, v \in V$, S *separates* u and v if S is a uv-separator.

A graph is *chordal* (or *triangulated*) if every cycle of length greater than 3 has a *chord*, which is an edge joining two nonconsecutive vertices of the cycle. A graph G is chordal if and only if G has a *clique tree*, which is a tree T on the maximal cliques of G with the *clique intersection property*: for any two maximal cliques K and K', the set $K \cap K'$ is contained in every maximal clique on the K–K' path in T [1,9]. The clique tree is not necessarily unique. Blair and Peyton [1] discuss various properties of clique trees, including the following correspondence between the edges of T and the minimal vertex separators of G. (In [1], G is a connected chordal graph and then the clique intersection property implies that $K \cap K' \neq \emptyset$ for every $\{K, K'\} \in E(T)$.)

Theorem 1 ([1]). *Let G be a chordal graph with clique tree T. Let $S \neq \emptyset$ be a set of vertices of G. Then S is a minimal vertex separator of G if and only if $S = K \cap K'$ for some $\{K, K'\} \in E(T)$.*

It follows that the nodes and edges of T correspond to the maximal cliques and minimal vertex separators of G, respectively. A maximal clique corresponds to exactly one node of T and a minimal vertex separator may correspond to more than one edge of T. Furthermore, since a chordal graph G has at most n maximal cliques [1,9], G has at most $n - 1$ minimal vertex separators. A clique tree can be computed in $O(m + n)$ time [1].

An *interval graph* is a graph where each vertex can be assigned an interval on the real line so that two vertices are adjacent if and only if their assigned intervals intersect; such an assignment is an *interval representation*. A graph is an interval graph if and only if it has a clique tree that is a path, which is a *clique path* [9,20]. Figure 1a shows an interval graph with clique path $(\{x, y, z\}, \{y, z, w\}, \{u, z\}, \{v, z\})$. Interval graphs are a proper subclass of chordal graphs. Since interval graphs model many problems involving linear arrangements, interval graphs have applications in many areas, including archeology, computational biology, file organization, partially ordered sets, and psychology [9,20].

A *proper interval graph* (or *unit interval graph*) is an interval graph with an interval representation where no interval is properly contained in another. Proper interval graphs are a proper subclass of interval graphs. For example, $K_{1,3}$ (the graph consisting of one vertex adjacent to three pairwise nonadjacent vertices) is an interval graph but not a proper interval graph. In fact, an interval graph is a proper interval graph if and only if it has no induced $K_{1,3}$ [9,20]. $K_{1,3}$ is also called the *claw*. The interval graph in Figure 1a is not a proper interval graph because it has the induced claw $\{u, v, y, z\}$. Proper interval graphs have applications in physical mapping of DNA and in other areas [10,20].

2.1 The Clique-Separator Graph

Let G be a chordal graph. The *clique-separator graph* $\mathcal{G}$ of G has the following nodes, (directed) arcs, and (undirected) edges.

- $\mathcal{G}$ has a *clique node* K for each maximal clique K of G.
- $\mathcal{G}$ has a *separator node* S for each minimal vertex separator S of G.
- Each arc is from a separator node to a separator node. $\mathcal{G}$ has arc (S, S'') if $S \subset S''$ and there is no separator node S' such that $S \subset S' \subset S''$.
- Each edge is between a separator node and a clique node. $\mathcal{G}$ has edge $\{S, K\}$ if $S \subset K$ and there is no separator node S' such that $S \subset S' \subset K$.

Throughout this paper, we refer to the *vertices* of G and the *nodes* of $\mathcal{G}$ and we use lowercase variables for vertices and uppercase variables for nodes. We identify "maximal clique" and "clique node" and identify "minimal vertex separator" and "separator node". Figure 1 shows a chordal graph G and its clique-

Fig. 1. A chordal graph and its clique-separator graph

separator graph $\mathcal{G}$, which has clique nodes $K_1 = \{x, y, z\}, K_2 = \{y, z, w\}, K_3 = \{u, z\}, K_4 = \{v, z\}$ and separator nodes $S_1 = \{y, z\}$ and $S_2 = \{z\}$.

The clique-separator graph $\mathcal{G}$ is unique. $\mathcal{G}$ is *connected* if the underlying undirected graph of $\mathcal{G}$ (obtained by replacing every arc with an edge) is connected. $\mathcal{G}$ is connected if and only if G is connected. If $\{S, K\}$ is an edge of $\mathcal{G}$, then S is a *neighbor* of K and S is *adjacent* to K and vice versa.

A *box* of $\mathcal{G}$ is a connected component of the undirected graph obtained by deleting the arcs of $\mathcal{G}$. A box of $\mathcal{G}$ contains exactly one clique node K and no separator nodes if and only if K is a complete connected component of G. Every box is a tree with the clique intersection property [13]. (We extend the definition of the clique intersection property to a tree T on a set of nodes in the natural way: for any two nodes N and N' of T, the set $N \cap N'$ is contained in every node on the N–N' path in T.) For any separator node S with neighbors K and K', $S = K \cap K'$ [13].

If G is a chordal graph or an interval graph, $\mathcal{G}$ may have many boxes, arcs, and nodes with high degree. If G is a proper interval graph, however, $\mathcal{G}$ has a much simpler structure, as the following theorem shows. If the graph is not connected, the theorem can be applied to each of its connected components.

Theorem 2 ([13]). *Let G be a connected chordal graph with clique-separator graph $\mathcal{G}$. Then G is a proper interval graph if and only if $\mathcal{G}$ has exactly one box and this box is a path $\mathcal{P}$ such that there do not exist vertices $u \in K_1 \cap K_3$ and $v \in K_2 - (K_1 \cup K_3)$ where (K_1, K_2, K_3) is a subsequence of $\mathcal{P}$.*

3 The Data Structure for the Algorithm

The algorithm maintains the clique-separator graph $\mathcal{G}$ for the proper interval graph G. Each connected component of $\mathcal{G}$ is a path $\mathcal{P}$. Intuitively, each vertex v is stored in the leftmost and rightmost nodes of $\mathcal{P}$ that contain v; by the clique intersection property, every node between these two nodes contains v. Each node N of $\mathcal{P}$ has two balanced binary trees of the vertices that it stores: one tree stores the vertices whose leftmost node is N and the other tree stores the vertices whose rightmost node is N. If v is contained in exactly one node N, then v is stored in both trees of N. When G is updated, $\mathcal{P}$ is updated by adding or deleting a constant number of nodes on $\mathcal{P}$ and moving a constant number of trees to the left or right on $\mathcal{P}$.

A vertex is *simplicial* if the set of its neighbors is a clique. A vertex is simplicial if and only if it is contained in exactly one clique node [1,20]. A vertex is *complex* if it is contained in one or more separator nodes. By the clique intersection property, every vertex is either simplicial or complex.

Let G be a proper interval graph with clique-separator graph $\mathcal{G}$. We will store sequences of vertices of G and sequences of nodes of $\mathcal{G}$ in balanced binary trees. In a balanced binary tree such as a B-tree or red-black tree [3], the height is $O(\log n)$ and an insert, delete, split, or join operation runs in $O(\log n)$ time, where n is the number of leaves in the tree. We will not use the find operation. Since G has n

vertices and $\mathcal{G}$ has at most n clique nodes and $n-1$ separator nodes, every balanced binary tree will have at most $2n-1$ leaves and $O(\log n)$ height.

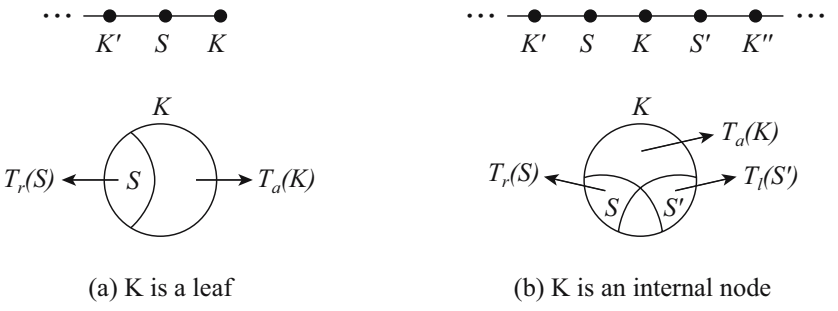
Each connected component of $\mathcal{G}$ is a path $\mathcal{P}$ and its nodes are stored in a balanced binary tree $T_{\mathcal{P}}$, which gives a left to right ordering of the nodes of $\mathcal{P}$. For each complex vertex v, $leftmost(v)$ and $rightmost(v)$ are the leftmost and rightmost separator nodes of $\mathcal{P}$ that contain v, respectively. Each clique node K has two balanced binary trees: $T_a(K)$ and $T_b(K)$ are identical trees and each contains the simplicial vertices in K. Each separator node S has two balanced binary trees: $T_r(S)$ contains every complex vertex u with $rightmost(u) = S$ and $T_l(S)$ contains every complex vertex u with $leftmost(u) = S$. Thus, every vertex v is contained in exactly two trees: if v is simplicial, then v is in $T_a(K)$ and $T_b(K)$ for some K, and if v is complex, then v is in $T_l(S)$ and $T_r(S')$ for some S and S', where $S = S'$ exactly when v is contained in only one separator node.

The vertices in $T_r(S)$ are sorted by the positions on $\mathcal{P}$ of their $leftmost$ nodes and the vertices in $T_l(S)$ are sorted by the positions on $\mathcal{P}$ of their $rightmost$ nodes. For example, suppose the separator nodes of $\mathcal{P}$ in left to right order are S_1, S_2, S, S_3, S_4. Then $T_r(S)$ contains the vertices whose $leftmost$ node is S_1, followed by the vertices whose $leftmost$ node is S_2, followed by the vertices whose $leftmost$ node is S, and $T_l(S)$ contains the vertices whose $rightmost$ node is S, followed by the vertices whose $rightmost$ node is S_3, followed by the vertices whose $rightmost$ node is S_4. A *block* is a maximal contiguous sequence of vertices in a T_l or T_r tree such that every vertex in B has the same $rightmost$ or $leftmost$ value, respectively. The first vertex in every block is marked. Each internal node of a T_l or T_r tree is labeled to indicate whether it is the ancestor of a marked vertex.

Every clique node K has a value $count(K)$, which is the number of vertices v such that the $leftmost(v)$–$rightmost(v)$ subpath of $\mathcal{P}$ contains K, or equivalently, K is between $leftmost(v)$ and $rightmost(v)$ on $\mathcal{P}$. Figure 2 shows the set of vertices of K when K is a leaf or an internal node of $\mathcal{P}$. The figure indicates the trees that contain the simplicial vertices and complex vertices of K. (If K is a leaf, then $count(K) = 0$. If K is an internal node with neighbors S and S', then $count(K) = |S \cap S'|$. By Theorem 2, if $count(K) > 0$, then K contains no simplicial vertices.)

Figure 3 shows a proper interval graph G and its clique-separator graph $\mathcal{P}$, where $S_1 = \{u, v\}$, $S_2 = \{w, v\}$, and $S_3 = \{x\}$. The first row under $\mathcal{P}$ shows the vertices in the T_a and T_l trees; the second row under $\mathcal{P}$ shows the vertices in the T_b and T_r trees. The vertices are listed in the order in which they appear in each tree, with commas between the blocks. We have $count(K_2) = 1$ and the other $count$ values are 0.

For each connected component $\mathcal{P}$ of $\mathcal{G}$, the algorithm maintains the data structure for $\mathcal{P}$ and the data structure for its reversal $\mathcal{P}^R$. We will present the algorithm for updating $\mathcal{P}$ and omit the algorithm for $\mathcal{P}^R$ since it is symmetric. (Similarly, the algorithm in [10] maintains the straight enumeration and its reversal for each connected component of G.) We will need $\mathcal{P}$ and $\mathcal{P}^R$ when we insert an edge between vertices in different connected components of G.



(a) K is a leaf (b) K is an internal node

Fig. 2. A clique node K on path $\mathcal{P}$

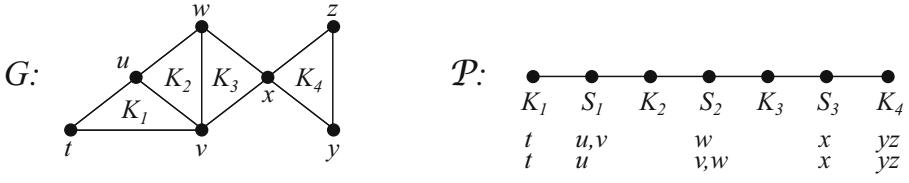


Fig. 3. A proper interval graph G and its clique-separator graph $\mathcal{P}$

For nodes N and N', $N <_{\mathcal{P}} N'$ denotes that N is to the left of N' on $\mathcal{P}$. For any vertex v, we decide whether v is simplicial or complex, and we find $leftmost(v)$ and $rightmost(v)$ if v is complex, by following parent pointers from v in the two trees containing v. For any complex vertices u and v, we decide whether $leftmost(u) <_{\mathcal{P}} leftmost(v)$ by finding $leftmost(u)$ and $leftmost(v)$ and then following parent pointers from $leftmost(u)$ and $leftmost(v)$ in $T_{\mathcal{P}}$. We decide whether $rightmost(u) <_{\mathcal{P}} rightmost(v)$ similarly. For any vertices x and y, we decide whether x and y are connected in G by following parent pointers from x and y in the trees containing x and y and then following parent pointers in the $T_{\mathcal{P}}$ trees. Each of these computations runs in $O(\log n)$ time.

For any vertex u in $T_l(S)$, we find the first and last vertices in the block B containing u by following parent pointers from u in $T_l(S)$ and examining the internal node labels. We can move u into the first or last position in B and then adjust the marks in the appropriate way. We can use the split operation to obtain a tree containing the vertices v of $T_l(S)$ with $rightmost(v) = S$ and a tree containing the vertices v of $T_l(S)$ with $rightmost(v) \neq S$. We denote these trees by $self(T_l(S))$ and $nonself(T_l(S))$, respectively. Similarly, we can process $T_r(S)$ to obtain the trees $self(T_r(S))$ and $nonself(T_r(S))$. Note that $self(T_l(S))$ and $self(T_r(S))$ each contain the vertices v with $leftmost(v) = rightmost(v) = S$. Each of these computations runs in $O(\log n)$ time.

We determine the number of vertices contained in a node in $O(1)$ time, as follows. Let $|T|$ denote the number of vertices stored in tree T. If K is a leaf clique node with left neighbor S (Figure 2a), then $|K| = |T_a(K)| + |T_r(S)|$ and $|S| = |T_r(S)|$. If K is an internal clique node with left neighbor S and right

neighbor S' (Figure 2b), then $|K| = |T_a(K)| + |T_r(S)| + |T_l(S')| + count(K)$, $|S| = |T_r(S)| + count(K)$, and $|S'| = |T_l(S')| + count(K)$. These equations are straightforward to prove. For example, suppose K has left neighbor S and right neighbor S'. Then every vertex $v \in K$ is a simplicial vertex in K (counted in $|T_a(K)|$), or a complex vertex in $S - S'$ (counted in $|T_r(S)|$), or a complex vertex in $S' - S$ (counted in $|T_l(S')|$), or a complex vertex in $S \cap S'$ (counted in $count(K)$). (The vertices in $T_l(S)$ or $T_r(S')$ are in $S \cap S'$.)

When we refer to both $T_a(K)$ and $T_b(K)$, or perform the same operation on both $T_a(K)$ and $T_b(K)$, for brevity we will write $T_{a+b}(K)$. Similarly, we will write $T_{l+r}(S)$. We denote the balanced binary tree operations as follows: $\leftarrow$ is assignment, $\cup$ is join, $-$ is deletion, and $\emptyset$ is an empty tree. In a join $T \cup T'$, the vertices in T are placed before the vertices in T'. In a deletion $T - v$, the marks are adjusted in the appropriate way if v is a marked vertex.

4 The Insert Operation

Let G be a chordal graph (not necessarily a proper interval graph). Throughout this section, $\{v_1, v_2\} \notin E$. We use $G + \{v_1, v_2\}$ to denote $(V, E \cup \{\{v_1, v_2\}\})$.

Theorem 3 ([15]). *Let G be a chordal graph and let $\{v_1, v_2\} \notin E(G)$. Then $G + \{v_1, v_2\}$ is chordal if and only if G has a clique tree T and maximal cliques K_1 and K_2 such that $v_1 \in K_1$ and $v_2 \in K_2$ and $\{K_1, K_2\} \in E(T)$.*

Suppose $v_1 \in K_1$ and $v_2 \in K_2$ and $\{K_1, K_2\} \in E(T)$. Let $S = K_1 \cap K_2$. Then $v_1 \in K_1 - S$ and $v_2 \in K_2 - S$ (Figure 4). Then $K = S \cup \{v_1, v_2\}$, which is not a maximal clique in G, is a maximal clique in $G + \{v_1, v_2\}$. Now K_1 and K_2 may or may not be maximal cliques in $G + \{v_1, v_2\}$; each K_i is a maximal clique in $G + \{v_1, v_2\}$ if and only if $K_i \supset S \cup \{v_i\}$, or equivalently, $|K_i - S| > 1$. Furthermore, K_1 and K_2 are the only maximal cliques of G that can be destroyed by inserting $\{v_1, v_2\}$ [15]. For example, consider the chordal graph G in Figure 1. Inserting edge $\{u, x\}$ yields a chordal graph because there is a clique path of G where the maximal cliques $\{u, z\}$ and $\{x, y, z\}$ are adjacent. Moreover, $\{u, x, z\}$ and $\{x, y, z\}$ are maximal cliques in $G + \{v_1, v_2\}$, but $\{u, z\}$ is not a maximal clique in $G + \{v_1, v_2\}$.

Let G be a proper interval graph with clique-separator graph $\mathcal{G}$. Let $\{v_1, v_2\}$ be the edge to be inserted into G. Since $\{v_1, v_2\} \notin E$, no clique node or separator node contains both v_1 and v_2. We have two main cases, depending on whether or not v_1 and v_2 are in the same connected component of G.

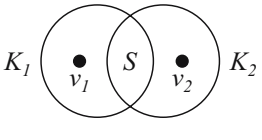


Fig. 4. Inserting edge $\{v_1, v_2\}$

Suppose v_1 and v_2 are in different connected components of G. Then there is always a clique tree T of G such that $v_1 \in K_1$ and $v_2 \in K_2$ and $\{K_1, K_2\} \in E(T)$, where $K_1 \cap K_2 = \emptyset$. Then $G + \{v_1, v_2\}$ is a chordal graph but possibly not a proper interval graph. Let v_1 and v_2 be vertices in connected components G_1 and G_2 of G with clique-separator graphs $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. We will later show that $G + \{v_1, v_2\}$ is a proper interval graph if and only if v_1 and v_2 are simplicial vertices of some clique nodes K_1 and K_2 that are endpoints of $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. If K_1 and K_2 do not exist, the operation rejects the insertion. Otherwise, the operation merges $\mathcal{P}_1$ and $\mathcal{P}_2$ by linking them with path (S_1, K, S_2) where $S_1 = \{v_1\}$ and $K = \{v_1, v_2\}$ and $S_2 = \{v_2\}$. But if v_i is the only vertex in G_i, then $K_i = \{v_i\}$ is not a maximal clique of $G + \{v_1, v_2\}$ and so the operation deletes K_i and S_i. (We present the operation this way for brevity. The operation can be readily rewritten so that it creates only the nodes that are not deleted.)

Suppose v_1 and v_2 are in the same connected component of G. Let $\mathcal{P}$ be the clique-separator graph of this connected component. By Theorem 3, $G + \{v_1, v_2\}$ is chordal if and only if $\mathcal{P}$ has a separator node S with left neighbor K_1 and right neighbor K_2 (implying $S = K_1 \cap K_2$) such that $v_1 \in K_1 - S$ and $v_2 \in K_2 - S$. If $G + \{v_1, v_2\}$ is not a proper interval graph, the operation rejects the insertion. Otherwise, the operation replaces S with path (S_1, K, S_2) where $S_1 = S \cup \{v_1\}$ and $K = S \cup \{v_1, v_2\}$ and $S_2 = S \cup \{v_2\}$. But if K_i is not a maximal clique of $G + \{v_1, v_2\}$, the operation deletes K_i and does not add S_i.

When the operation creates a node, its two trees are initially empty and its *count* value (if it is a clique node) is initially 0.

$Insert(v_1, v_2)$:

v_1 and v_2 are in different connected components of G:
Let v_1 and v_2 be vertices in connected components G_1 and G_2 of G with clique-separator graphs $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. If v_1 and v_2 are simplicial vertices of some clique nodes K_1 and K_2 that are endpoints of $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively, then continue, and otherwise, reject. Assume $\mathcal{P}_1$ has right endpoint K_1 and $\mathcal{P}_2$ has left endpoint K_2 (Figure 5a). We will merge $\mathcal{P}_1 + \mathcal{P}_2$ and merge $\mathcal{P}_2^R + \mathcal{P}_1^R$; we describe only the former since the latter is symmetric. The other cases, e.g. $\mathcal{P}_1$ has left endpoint K_1 and $\mathcal{P}_2$ has left endpoint K_2, are similar.

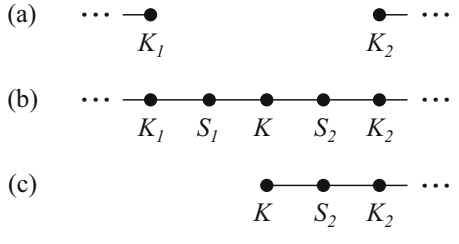


Fig. 5. v_1 and v_2 are in different connected components of G

Add path (S_1, K, S_2) and edges $\{K_1, S_1\}$ and $\{S_2, K_2\}$, where $S_1 = \{v_1\}$ and $K = \{v_1, v_2\}$ and $S_2 = \{v_2\}$ (Figure 5b). If v_1 is not the only vertex in G_1, move v_1 from $T_{a+b}(K_1)$ into $T_{l+r}(S_1)$, and otherwise, move v_1 from $T_{a+b}(K_1)$ into $T_{a+b}(K)$ and delete K_1 and S_1 (Figure 5c). If v_2 is not the only vertex in G_2, move v_2 from $T_{a+b}(K_2)$ into $T_{l+r}(S_2)$, and otherwise, move v_2 from $T_{a+b}(K_2)$ into $T_{a+b}(K)$ and delete K_2 and S_2.

v_1 and v_2 are in the same connected component of G:

Let v_1 and v_2 be vertices in a connected component of G with clique-separator graph $\mathcal{P}$. If there is a separator node S with left neighbor K_1 and right neighbor K_2 such that $v_1 \in K_1$ and $v_2 \in K_2$ (Figure 6a), then continue, and otherwise, reject. Note v_1 is in $T_{a+b}(K_1)$ or $T_r(S_1')$, where S_1' is the left neighbor of K_1 (if it exists), and v_2 is in $T_{a+b}(K_2)$ or $T_l(S_2')$, where S_2' is the right neighbor of K_2 (if it exists). If v_1 is in $T_r(S_1')$ and either $|T_{a+b}(K_1)| > 0$ or $leftmost(v_1) <_{\mathcal{P}} leftmost(u)$ for the last vertex u in $T_r(S_1')$, then reject. If the symmetric condition holds with v_2, then reject. Let $K = S \cup \{v_1, v_2\}$ and $S_1 = S \cup \{v_1\}$ and $S_2 = S \cup \{v_2\}$.

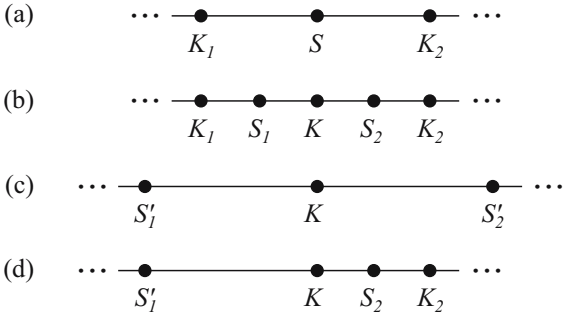


Fig. 6. v_1 and v_2 are in the same connected component of G

If $|K_1 - S| > 1$ and $|K_2 - S| > 1$, then do the following.

1. Replace node S with path (S_1, K, S_2) (Figure 6b). Do $T_l(S_1) \leftarrow T_l(S)$ and $T_r(S_2) \leftarrow T_r(S)$. Do $count(K) \leftarrow count(K_1) + |T_l(S_1)|$.
2. If v_1 is in $T_{a+b}(K_1)$, move v_1 to $T_{l+r}(S_1)$ (make it the first vertex in $T_l(S_1)$). If v_1 is in $T_r(S_1')$, move v_1 to $T_r(S_1)$ and increment $count(K_1)$.
3. If v_2 is in $T_{a+b}(K_2)$, move v_2 to $T_{l+r}(S_2)$ (make it the last vertex in $T_r(S_2)$). If v_2 is in $T_l(S_2')$, move v_2 to $T_l(S_2)$ and increment $count(K_2)$.

If $|K_1 - S| = 1$ and $|K_2 - S| = 1$, then do the following.

1. Replace path (K_1, S, K_2) with node K (Figure 6c). Do $T_{a+b}(K) \leftarrow self(T_{l+r}(S))$ and $count(K) \leftarrow count(K_1) - |nonself(T_r(S))|$.
2. If K_1 was an endpoint of $\mathcal{P}$, add v_1 to $T_{a+b}(K)$. Otherwise, do $T_r(S_1') \leftarrow T_r(S_1') \cup nonself(T_r(S))$.
3. If K_2 was an endpoint of $\mathcal{P}$, add v_2 to $T_{a+b}(K)$. Otherwise, do $T_l(S_2') \leftarrow nonself(T_l(S)) \cup T_l(S_2')$.

If $|K_1 - S| = 1$ and $|K_2 - S| > 1$, then do the following. Rename K_1 to K and rename S to S_2 (Figure 6d). If v_2 is in $T_{a+b}(K_2)$, move v_2 to $T_{l+r}(S_2)$ (make it the first vertex in $T_l(S_2)$ and the last vertex in $T_r(S_2)$). If v_2 is in $T_l(S_2')$, move v_2 to $T_l(S_2)$ (make it the last vertex), and increment $count(K_2)$.
If $|K_1 - S| > 1$ and $|K_2 - S| = 1$, then this is symmetric to the previous case.

Theorem 4. *Let G be a proper interval graph with clique-separator graph $\mathcal{G}$. If $G^+ = G + \{v_1, v_2\}$ is a proper interval graph, the Insert operation computes its clique-separator graph $\mathcal{G}^+$, and otherwise, the operation rejects. The operation runs in $O(\log n)$ time, where n is the number of vertices of G.*

The full paper [17] has the proof of the theorem and examples of the insert operation.

5 The Delete Operation

Let G be a proper interval graph with clique-separator graph $\mathcal{G}$. Let $\{v_1, v_2\}$ be the edge to be deleted from G. The operation decides whether v_1 and v_2 are simplicial or complex vertices and then determines which of a number of cases applies. If $G - \{v_1, v_2\}$ is not a proper interval graph, then the operation rejects the deletion, and otherwise, the operation computes the clique-separator graph of $G - \{v_1, v_2\}$. The operation is omitted here because of lack of space; it can be found in the full paper [17].

6 Conclusions

It would be interesting to extend the algorithm and data structure to handle interval graphs. The clique path of an interval graph is not unique, however, so it would probably be necessary to use a more complicated data structure such as the PQ-tree [2] or the train tree [14]. The algorithm in [14] maintains the clique-separator graph and the train tree of an interval graph in $O(n \log n)$ per edge insertion or deletion. It may be possible to speed up that algorithm by maintaining the intervals of the vertices as done by the algorithm in this paper.

References

1. Blair, J.R.S., Peyton, B.: An introduction to chordal graphs and clique trees. In: Graph Theory and Sparse Matrix Computation, vol. 56, pp. 1–29. Springer-Verlag, New York (1993)
2. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. J. Computer and System Sciences 13, 335–379 (1976)
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press and McGraw-Hill Book Co., Cambridge (2001)

4. Corneil, D.G., Kim, H., Nataranjan, S., Olariu, S., Sprague, A.P.: Simple linear time recognition of unit interval graphs. Information Processing Letters 55, 99–104 (1995)
5. Corneil, D.G., Olariu, S., Stewart, L.: The ultimate interval graph recognition algorithm? In: Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 175–180 (1998)
6. Deng, X., Hell, P., Huang, J.: Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs. SIAM J. Computing 25(2), 390–403 (1996)
7. Eppstein, D., Galil, Z., Italiano, G.F.: Dynamic graph algorithms. In: Atallah, M.J. (ed.) Algorithms and Theory of Computation Handbook, ch. 8. CRC Press, Boca Raton (1998)
8. Feigenbaum, J., Kannan, S.: Dynamic graph algorithms. In: Rosen (ed.) Handbook of Discrete and Combinatorial Mathematics, ch. 17. CRC Press, Boca Raton (1999)
9. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. In: Annals of Discrete Mathematics, vol. 57. Elsevier B.V., Amsterdam (2004)
10. Hell, P., Shamir, R., Sharan, R.: A fully dynamic algorithm for recognizing and representing proper interval graphs. SIAM J. Computing 31, 289–305 (2001)
11. Hsu, W.L.: On-line recognition of interval graphs. In: Deza, M., Manoussakis, I., Euler, R. (eds.) CCS 1995. LNCS, vol. 1120, pp. 27–38. Springer, Heidelberg (1996)
12. Hsu, W.L., Ma, T.H.: Fast and simple algorithms for recognizing chordal comparability graphs and interval graphs. SIAM J. Computing 28(3), 1004–1020 (1999)
13. Ibarra, L.: The clique-separator graph for chordal graphs (submitted, 2006), `http://facweb.cs.depaul.edu/ibarra/research.htm`
14. Ibarra, L.: A fully dynamic graph algorithm for recognizing interval graphs (submitted, 2007), `http://facweb.cs.depaul.edu/ibarra/research.htm`
15. Ibarra, L.: Fully dynamic algorithms for chordal graphs and split graphs. ACM Transactions on Algorithms 4(4), 1–20 (2008), `http://facweb.cs.depaul.edu/ibarra/research.htm`
16. Ibarra, L.: A simple algorithm to find Hamiltonian cycles in proper interval graphs (submitted, 2008), `http://facweb.cs.depaul.edu/ibarra/research.htm`
17. Ibarra, L.: A simple fully dynamic graph algorithm for recognizing proper interval graphs (submitted, 2008), `http://facweb.cs.depaul.edu/ibarra/research.htm`
18. Johnson, D.S.: The NP-completeness column: an ongoing guide. J. Algorithms 6, 434–451 (1985)
19. Korte, N., Möhring, R.H.: An incremental linear-time algorithm for recognizing interval graphs. SIAM J. Computing 18(1), 68–81 (1989)
20. McKee, T.A., McMorris, F.R.: Topics in Intersection Graph Theory, Philadelphia. Society for Industrial and Applied Mathematics (1999) (Monograph)
21. Rose, D.J., Tarjan, R.E., Lueker, G.S.: Algorithmic aspects of vertex elimination on graphs. SIAM J. Computing 5(2), 266–283 (1976)
22. Tarjan, R.E., Yannakakis, M.: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM J. Computing 13(3), 566–579 (1984)

Minmax Tree Cover in the Euclidean Space

Seigo Karakawa, Ehab Morsy, and Hiroshi Nagamochi

Department of Applied Mathematics and Physics
Graduate School of Informatics
Kyoto University
Yoshida Honmachi, Sakyo, Kyoto 606-8501, Japan
{seigo,ehab,nag}@amp.i.kyoto-u.ac.jp

Abstract. Let $G = (V, E)$ be an edge-weighted graph, and let $w(H)$ denote the sum of the weights of the edges in a subgraph H of G. Given a positive integer k, the balanced tree partitioning problem requires to cover all vertices in V by a set $\mathcal{T}$ of k trees of the graph so that the ratio α of $\max_{T \in \mathcal{T}} w(T)$ to $w(T^*)/k$ is minimized, where T^* denotes a minimum spanning tree of G. The problem has been used as a core analysis in designing approximation algorithms for several types of graph partitioning problems over metric spaces, and the performance guarantees depend on the ratio α of the corresponding balanced tree partitioning problems. It is known that the best possible value of α is 2 for the general metric space. In this paper, we study the problem in the d-dimensional Euclidean space $\mathbb{R}^d$, and break the bound 2 on α, showing that $\alpha < 2\sqrt{3} - 3/2 \fallingdotseq 1.964$ for $d \geq 3$ and $\alpha < (13 + \sqrt{109})/12 \fallingdotseq 1.953$ for $d = 2$. These new results enable us to directly improve the performance guarantees of several existing approximation algorithms for graph partitioning problems if the metric space is an Euclidean space.

Keywords: Minmax Tree Cover, Balanced Partition, Tree Cover, Approximation Algorithms, Graph Algorithms.

1 Introduction

Let $G = (V, E)$ be a simple undirected graph with vertex set V and edge set E such that edges are weighted by nonnegative reals, and let p be a specified positive integer. Then the minmax subtree cover problem requires to find a set of p trees covering all the vertices such that the maximum weight of a tree in the set is minimized. This problem arises in various types of practical applications, such as multi-vehicle scheduling problem [7,9,10,11], task sequencing problem [5], and political districting [4,17]. The minmax subtree cover problem on graphs can be described formally as follows.

Minmax Subtree Cover Problem (MSC)
Input: An undirected graph $G = (V, E)$, an edge weight function $w : E \to \mathbb{R}^+$, and a positive integer p.
Feasible solution: A partition $\mathcal{S} = \{S_1, S_2, \ldots, S_p\}$ of V and a set $\mathcal{T} =$

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 202–213, 2009.

$\{T_1, T_2, \ldots, T_p\}$ of p trees of G such that $S_i \subseteq V(T_i)$, $i = 1, 2, \ldots, p$.
Goal: Minimize $\max_{1 \le i \le p} w(T_i)$, where $w(T_i) = \sum_{e \in E(T_i)} w(e)$.

A *tree cover* $\mathcal{T}$ of a graph $G = (V, E)$ is defined by a set of trees of G such that the union of vertices of the trees in $\mathcal{T}$ contains V. That is, the MSC requires to find a tree cover with p trees that minimizes the maximum weight of a tree in the tree cover, where the weight of a tree is the sum of the weights of all edges in the tree. Trees in a tree cover are not necessarily vertex-disjoint or edge-disjoint.

The MSC is known to be NP-hard even if $p = 2$ and G is a tree, or if G can be embedded in the Euclidean space [1,6], and thus several approximation algorithms for the problem have been proposed in the literatures. Andersson et al. [1] provided a $(2 + \varepsilon)$-approximation algorithm for the MSC when G can be embedded in the Euclidean space, and Nagamochi and Kawada [13] presented a $(4 - 4/(p + 1))$-approximation algorithm for the problem when the given graph is a cactus. Even et al. [7] presented a 4-approximation algorithm for the MSC with an arbitrary graph G. The MSC on a tree-like structure graphs has also been studied extensively. Averbakh and Berman [3] presented a $(2 - 2/(p + 1))$-approximation algorithm with time complexity $O(p^{p-1}n^{p-1})$, and afterward Nagamochi and Okada [14] gave a polynomial time approximation algorithm with the same approximation factor which runs in $O(p^2 n)$ time, where n is the number of vertices in the tree. In an extension of the problem, a set of vertices to be covered is given as a subset $S \subseteq V$ of vertices and nonnegative weights to handle vertices in S are also introduced in the tree weight. For this problem, Nagamochi and Okada [15] proposed a $(2 - 2/(p + 1))$-approximation algorithm that runs in $O((p - 1)!n)$.

For a rooted version of the MSC, a graph $G = (V, E)$ with a set R of prescribed vertices is given and a tree cover is required to consist of trees each of which is rooted at a some vertex in R. In [7], a 4-approximation algorithm is proposed for an arbitrary graph G under an additional condition that all trees $\mathcal{T}$ are required to be rooted at distinct vertices in R. Nagamochi [12] proposed a $(3 - 2/(p + 1))$-approximation algorithm to the problem with $|R| = 1$. For the rooted version of the MSC on a tree, Averbakh and Berman [2] presented a linear time 4/3-approximation algorithm for the problem with $p = 2$, and Nagamochi and Okada [14,16] gave an $O(n \log \log_{1+\varepsilon/2} 3)$ time $(2 + \varepsilon)$-approximation algorithms for arbitrary p, where $\varepsilon > 0$ is a prescribed constant.

Some of the above approximation algorithms for the MSC include a procedure for partitioning a minimum spanning tree T^* of a given graph into k trees of the graph as uniformly as possible, and their performance analysis relies on the fact that $w(T^*)/k$ is a lower bound on the maximum weight of a tree in any such partition. Thus the ratio α of the maximum weight of a tree to $w(T^*)/k$ appears as a factor of their performance guarantees. Motivated by the importance of the analysis on tree covers, we consider *the balanced tree partitioning problem* in this paper. For a given a positive integer k, the problem requires to find a tree cover $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ with k trees of the graph that minimizes the ratio of $\max_{1 \le i \le k} w(T_i)$ to $w(T^*)/k$, where T^* denotes a minimum spanning tree of G.

Note that a tree T_i in a tree cover is allowed to contain an edge not in T^*. In particular, we analyze an upper bound on the factor α such that

$$\max_{1 \le i \le k} w(T_i) \le \alpha \cdot w(T^*)/k.$$

For any metric, it is known that $\alpha \le 2$ holds [2,3]. We here remark that $\alpha \le 2$ is best possible for the general metric. Consider a metric graph $G = (V, E)$ consisting of a star S on $k+1$ vertices $V = \{s, v_1, \ldots, v_{k+1}\}$ such that $E(S) = \{(s, v_i) \mid i = 1, 2, \ldots, k+1\}$, where each edge in S is weighted by 1 and each of the remaining edges in G is weighted by 2. Clearly, S is the minimum spanning tree of G with weight $w(S) = k+1$. On the other hand, the maximum weight of any k trees of G is at least 2, since one of the trees must contain two leaves of S. Hence α is at least $2/(1+1/k)$, which is nearly 2 when k is sufficiently large.

In this paper, we prove that there are better upper bounds on α in the Euclidean space. The following two results are the main contribution of the paper.

Theorem 1. *For a set V of n points in the Euclidean space $\mathbb{R}^2$ and a positive integer k, there exists a tree cover $T_1, T_2, \ldots, T_k$ of V such that*

$$\frac{\max_{1 \le i \le k} w(T_i)}{w(T^*)/k} \le (13 + \sqrt{109})/12 \fallingdotseq 1.953,$$

where T^* is a minimum weight tree spanning V. Furthermore, such a tree cover can be obtained in polynomial time.

Theorem 2. *For a set V of n points in the Euclidean space $\mathbb{R}^d$ $(d \ge 3)$ and a positive integer k, there exists a tree cover $T_1, T_2, \ldots, T_k$ of V such that*

$$\frac{\max_{1 \le i \le k} w(T_i)}{w(T^*)/k} \le 2\sqrt{3} - 3/2 \fallingdotseq 1.964,$$

where T^* is a minimum weight tree spanning V. Furthermore, such a tree cover can be obtained in polynomial time.

These new results enable us to directly improve the performance guarantees of several existing approximation algorithms for graph partitioning problems if the metric space is a Euclidean space $\mathbb{R}^d$. For example, the performance guarantee on the MSC due to Andersson et al. [1] is given as $\alpha + \varepsilon$, which is at most $1.964 + \varepsilon$ for $d \ge 3$ and $1.953 + \varepsilon$ for $d = 2$. Also the performance guarantees on the rooted and unrooted versions of the MSC due to Even et al. [7] are $2 + \alpha$ and 2α, respectively. Thus, our results imply that the unrooted versions of the MSC is 3.928-approximable in $\mathbb{R}^d$ with $d \ge 3$ and 3.906-approximable in $\mathbb{R}^2$.

The paper is organized as follows. Section 2 introduces terminologies and definitions on graphs. Section 3 gives a framework of an approximation algorithm to the balanced tree partitioning problem, from which Theorems 1 and 2 follow directly. Sections 4 and 5 give detailed proofs of some tree cover results based on which we build our algorithm in Section 3. Section 6 makes some concluding remarks.

2 Preliminaries

Throughout this paper a graph stands for a simple undirected graph unless otherwise stated. A singleton set $\{x\}$ may be denoted by x. Let G be a graph. The vertex and edge sets of G are denoted by $V(G)$ and $E(G)$, respectively. A graph G with a vertex set V and an edge set E is denoted by (V, E). Let G_1 and G_2 be subgraphs of G. We let G_1+G_2 denote the graph $(V(G_1)\cup V(G_2), E(G_1)\cup E(G_2))$. For a vertex $u \in V(G)$ and an edge $e = (u, v) \in E(G)$, we denote by $G_1 + u$ (resp., $G_1 + e$) the subgraph $G_1 + (\{u\}, \emptyset)$ (resp., $G_1 + (\{u, v\}, \{e\})$ of G. For a subset $X \subseteq V(G)$, let $G[X]$ denote the subgraph induced from G by X, and $G - X$ denote the subgraph obtained from G by removing the vertices in X together with all edges incident to a vertex in X. The *degree* of u, i.e., the number of edges incident to vertex $u \in V(G)$, is denoted by $\deg(u)$.

For a graph G and an edge weight $w : E(G) \to \mathbb{R}^+$, where $\mathbb{R}^+$ denotes the set of nonnegative reals, the weight $w(G_1)$ of a subgraph G_1 of G is defined by $\sum_{e\in E(G_1)} w(e)$. For a set $\mathcal{S}$ of subgraphs of G, let $w(\mathcal{S})$ denote $\sum_{G'\in\mathcal{S}} w(G')$. The weight $w(e)$ of edge $e = (u, v)$ may be denoted by $w(u, v)$. An edge-weighted graph (G, w) is a *metric* if it satisfies the triangle inequality, $w(x, y) + w(y, z) \geq w(x, z)$, $x, y, z \in V$. An edge-weighted graph (G, w) in the Euclidean space $\mathbb{R}^d$ is a complete graph whose vertex set is defined by a set V of points in the space, where the edge weight $w(u, v)$, $u, v \in V$ is defined by the Euclidean distance between the two points u and v. The line segment between points u and v in the Euclidean space $\mathbb{R}^d$ is denoted by uv and its weight by $\overline{uv}$.

Let T be a rooted tree. The set of children of a vertex v is denoted by $Ch(v)$, and the set of descendants of a vertex v is denoted by $D(v)$, where $D(v)$ includes v. Let $D(S) = \cup_{v\in S} D(v)$ for a subset $S \subseteq V(T)$. A non-root vertex of degree 1 is called a *leaf* in T. The *subtree T_v rooted at a vertex $v \in V(T)$* is the subtree of T induced by the descendants of v. An edge $e = (u, v) \in E(T)$, where $v \in Ch(u)$, is called the *parent edge* of v and a *child edge* of u, and the *subtree T_e rooted at e* is defined by $T_v + e$. The parent edge of a vertex v is denoted by $e(v)$. A *branch* of a vertex u is defined as the subtree T_e rooted at a child edge e of u, and $\mathcal{B}(u)$ denotes the set of all branches of u. We may mean by a subset $\mathcal{S} \subseteq \mathcal{B}(u)$ the subtree that consists of all branches in $\mathcal{S}$ and denote by $V(\mathcal{S})$ and $E(\mathcal{S})$ the vertex and edge sets of the subtree $\mathcal{S}$, respectively, unless confusion arises.

For a real $\beta > 0$, we define a *β-boundary vertex* of T as a non-root vertex v such that

$$w(T_{e(v)}) \geq \beta \text{ for its parent edge } e(v),$$

and no proper descendant of v has this property, i.e.,

$$w(T_{e'}) < \beta \text{ for every child edge } e' \text{ of } v \text{ (if any)}.$$

For a subtree T' of T, let $V_\beta(T')$ denote the set of all β-boundary vertices in T', where the root r' of T' is chosen as the vertex closest to the root r of T. Note that, for two distinct β-boundary vertices $u, v \in V_\beta(T')$, none of u and v is a descendent or an ancestor of the other in T'. Also, $r' \notin V_\beta(T')$ by definition. We observe the next property.

Lemma 1. *Let T_e be a branch of a vertex u and β be a positive real.*

(i) *$V_\beta(T_e) \neq \emptyset$ if and only if $w(T_e) \geq \beta$.*
(ii) *If $V_\beta(T_e) \neq \emptyset$, then $V_\gamma(T_e) \neq \emptyset$ for any $\gamma \in (0, \beta]$.*

Proof. (i) The only-if part is obvious. We show the if part. Choose a vertex $v \in V(T_e) - u$ closest to u such that v is a leaf or $w(S) < \beta$ holds for all $S \in \mathcal{B}(v)$. For the parent edge $e(v)$ of v, we have $w(T_{e(v)}) \geq \beta$. Then v is a β-boundary vertex in T_e, and $V_\beta(T_e) \neq \emptyset$ holds.

(ii) For any real $\gamma \in (0, \beta]$, $w(T_e) \geq \beta \geq \gamma > 0$ holds. Then $V_\gamma(T_e) \neq \emptyset$ holds by (i).

For a specified real $\alpha > 0$ and a 1-boundary vertex $u \in V_1(T)$ in a rooted tree T, we define a *canonical partition* $\{\mathcal{L}(u), \mathcal{M}(u), \mathcal{H}(u)\}$ of $\mathcal{B}(u)$ by

$$\mathcal{H}(u) = \{S \in \mathcal{B}(u) \mid \alpha - 1 < w(S) < 1\},$$
$$\mathcal{M}(u) = \{S \in \mathcal{B}(u) \mid 2 - \alpha \leq w(S) \leq \alpha - 1\},$$
$$\mathcal{L}(u) = \{S \in \mathcal{B}(u) \mid 0 < w(S) < 2 - \alpha\}.$$

Due to space limitation, we omit some of the proofs of the results presented in the paper (see to the full version of the paper [8] for the detail).

3 Approximation Algorithm

To prove Theorems 1 and 2, we assume without loss of generality that $w(T^*) = k$ throughout the paper for a minimum spanning tree T^*. We design an approximation algorithm to the balanced tree partitioning problem to derive the upper bounds Theorems 1 and 2. In this section, we first give a framework of the approximation algorithm. The algorithm computes a set of at most k trees of G that cover the vertex set of T^* by repeatedly applying three main theorems on tree partition described below.

We first introduce the following basic definition.

Definition 1. *Let T be a tree in a graph G, and let $\alpha \in [1, 2]$ be a real number. A family of h subsets $S_1, S_2, \ldots, S_h \subseteq V(T)$ is* admissible (*or* h-admissible) *in T if there are trees $T_{S_1}, T_{S_2}, \ldots, T_{S_h}$ of G (which are not necessarily subtrees of T) such that*

(i) *For each S_i, $S_i \subseteq V(T_{S_i})$ and $w(T_{S_i}) \leq \alpha$.*
(ii) *If $V(T) - \bigcup_{1 \leq i \leq h} S_i \neq \emptyset$, then $T' = T - \bigcup_{1 \leq i \leq h} S_i$ remains connected and $w(T) - w(T') \geq h$ holds.*
(iii) *If $V(T) = \bigcup_{1 \leq i \leq h} S_i$, then $\lceil w(T) \rceil \geq h$ holds.*

We call a set of such subtrees T_{S_i}, $i = 1, 2, \ldots, h$, an h-admissible forest.

To prove Theorems 1 and 2, where the weight of a minimum spanning tree T^* of G is assumed to be k, it suffices to show that a minimum spanning tree $T = T^*$ has a k-admissible family of subsets $S_1, S_2, \ldots, S_k \subseteq V(G) = V(T)$ for $\alpha = (13 + \sqrt{109})/12$ and $\alpha = 2\sqrt{3} - 3/2$, respectively.

Lemma 2. *Let T be a tree and $\alpha \in [1, 2]$ be a specified real number.*

(i) *For a p-admissible family $\{S_1, S_2, \ldots, S_p\}$ in T and a q-admissible family $\{S'_1, S'_2, \ldots, S'_q\}$ in $T' = T - \bigcup_{1 \le i \le p} S_i$, their union $\{S_1, S_2, \ldots, S_p\} \cup \{S'_1, S'_2, \ldots, S'_q\}$ is $(p + q)$-admissible in T.*
(ii) *For a non-root vertex $v \in V(T)$ with $w(T_v) \le \alpha$ and $w(T_{e(v)}) \ge 1$, $D(v)$ is 1-admissible.*
(iii) *For a vertex $u \in V(T)$ and a subset $C_1 \subseteq Ch(u)$ with $1 \le \sum_{v \in C_1} w(T_{e(v)}) \le \alpha$, $S_1 = D(C_1)$ is 1-admissible.*
(iv) *If $w(T) \le 3\alpha/2$, then there is an h-admissible family $\{S_i \mid i = 1, \ldots, h\}$ $(h \le 2)$ such that $V(T) = \cup_{1 \le i \le h} S_i$.*

We consider whether a 1-boundary vertex u in a tree satisfies the following condition.

Definition 2. *For a real number $\alpha \in [1, 2]$, a 1-boundary vertex $u \in V_1(T)$ in a rooted tree T is called* inactive *if it satisfies the following condition A, and is called* active *otherwise.*

Condition A
(i) $w(\mathcal{B}(u)) > \alpha$, (ii) $|\mathcal{H}(u)| \ge 2$, (iii) $\mathcal{M}(u) = \emptyset$, *and* (iv) $w(\mathcal{L}(u)) < 2 - \alpha$.

We use the following results, which will be verified in Sections 4 and 5.

Theorem 3. *Let T be a tree rooted at a vertex r with $deg(r) = 1$, and let $\alpha \in [5/3, 2]$ be a real number. Assume that $w(T) > 3\alpha/2$ and there is an active 1-boundary vertex in $V_1(T)$. Then there is an admissible family $\mathcal{S}$ in T such that the tree $T' = T - \bigcup_{S \in \mathcal{S}} S$ has no active 1-boundary vertices in $V_1(T')$.*

Theorem 4. *Let T be a tree in the Euclidean space $\mathbb{R}^d$ $(d \ge 3)$, and let $\alpha = 2\sqrt{3} - 3/2$. Assume that T is rooted at a vertex r with $deg(r) = 1$. If $w(T) > 3\alpha/2$ and every 1-boundary vertex $u \in V_1(T)$ is inactive, then there is an admissible family $\mathcal{S}$ in T.*

Theorem 5. *Let T be a tree in the Euclidean space $\mathbb{R}^d$ $(d = 2)$, and let $\alpha = (13 + \sqrt{109})/12$. Assume that T is rooted at a vertex r with $deg(r) = 1$. If $w(T) > 3\alpha/2$ and every 1-boundary vertex $u \in V_1(T)$ is inactive, then there is an admissible family $\mathcal{S}$ in T.*

Now we are ready to present the approximation algorithm. We are given a spanning tree T of a graph G in the Euclidean space and a positive integer k. The algorithm repeats the following procedure on the current tree T as long as $w(T) > 3\alpha/2$ holds. Depending on the properties of the current tree T, we

first apply one of the above theorems to find an admissible family $\mathcal{S}$ in T and then remove $\mathcal{S}$ from the current tree T. Finally, we find an h-admissible family $\{S_i \mid i = 1, \ldots, h\}$ $(h \leq 2)$ in the current T such that $V(T) = \cup_{1 \leq i \leq h} S_i$ by Lemma 2. The algorithm is described as follows.

Algorithm TreeCover
Input: A spanning tree T^* of a graph G in the Euclidean space $\mathbb{R}^d$ $(d \geq 2)$, where $w(T^*) = k$ for an integer $k \geq 1$.
Output: A k-admissible family $\mathcal{S}^*$ for $\alpha = 1/2 + \sqrt{2}$ $(d = 2)$ and $\alpha = 2\sqrt{3} - 3/2$ $(d \geq 3)$.
1 Let $\mathcal{S}^* := \emptyset$; Let $T := T^*$, regarding T as a tree rooted at a vertex r with $\deg(r) = 1$;
2 **while** $w(T) > 3\alpha/2$ **do**
3 **if** T has an active 1-boundary vertex **then**
4 Find an admissible family $\mathcal{S}$ in T by Theorem 3;
5 Let $\mathcal{S}^* := \mathcal{S}^* \cup \mathcal{S}$; $T := T - \bigcup_{S \in \mathcal{S}} S$
6 **end**; /* if */
/* the current tree T has no active 1-boundary vertex */
7 Find an admissible family $\mathcal{S}$ in T by Theorem 4 (if $d \geq 3$) and Theorem 5 (if $d = 2$);
8 $\mathcal{S}^* := \mathcal{S}^* \cup \mathcal{S}$; $T := T - \bigcup_{S \in \mathcal{S}} S$
9 **end**; /* while */
/* $w(T) \leq 3\alpha/2$ holds */
10 Find an admissible family $\mathcal{S} = \{S_i \mid i = 1, \ldots, h\}$ $(h \leq 2)$ by Lemma 2(iv);
11 $\mathcal{S}^* := \mathcal{S}^* \cup \mathcal{S}$.

Theorem 6. *Algorithm* TreeCover *delivers a k-admissible family S^* in T^* such that $\bigcup_{S \in S^*} S = V(T^*)$.*

Proof. Note that, in each iteration of the while-loop, at least one of Theorems 3, 4, and 5 is applied to find an admissible family in the current tree T. After the last iteration of the while-loop, we have $w(T) \leq 3\alpha/2$ and hence Lemma 2(iv) can be applied to find an admissible family for the remaining tree T. By Lemma 2(i), the union of all such admissible families gives a desired admissible family in T^*. Moreover, by the definition of admissible family, we conclude that $|\mathcal{S}^*| \leq k$ since $w(T^*) = k$. This completes the proof. □

4 Proof of Theorem 3

This section is devoted to present a proof of Theorem 3 which holds for any $\alpha \in [5/3, 2]$.

For a non-root vertex u in a rooted tree T, a partition $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_p, \mathcal{L}, \mathcal{M}, \mathcal{H}\}$ of $\mathcal{B}(u)$ is called *valid* if $1 \leq w(\mathcal{S}_i) \leq \alpha$, $i = 1, 2, \ldots, p$, $\mathcal{L} \subseteq \mathcal{L}(u)$, $\mathcal{M} \subseteq \mathcal{M}(u)$, and $\mathcal{H} \subseteq \mathcal{H}(u)$. Note that, by Lemma 2(iii), $S_i = V(\mathcal{S}_i) - \{u\}$ in a valid partition forms an admissible family with a 1-admissible tree $\mathcal{S}_i$.

Lemma 3. *Let $u \in V_1(T)$ be a 1-boundary vertex in a rooted tree T. Then there exists a valid partition $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_p, \mathcal{L}, \mathcal{M}, \mathcal{H}\}$ of $\mathcal{B}(u)$ that satisfies the following* (i)-(ii)*, where $\mathcal{B} = \mathcal{L} \cup \mathcal{M} \cup \mathcal{H}$.*

(i) *If $\mathcal{H} \neq \emptyset$, then $\mathcal{M} = \emptyset$ and $w(\mathcal{L}) < 2 - \alpha$ hold.*
(ii) *$w(\mathcal{B}) < 1$ holds if and only if $|\mathcal{H}| \leq 1$ holds.*

Given a rooted tree T, the following algorithm converts T into another tree T' in which all 1-boundary vertices are inactive. Such a tree T' is constructed by applying a procedure that first chooses an active vertex $u \in V_1(T)$ in the current tree T, computes a valid partition $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_p, \mathcal{L}, \mathcal{M}, \mathcal{H}\}$ of $\mathcal{B}(u)$ by Lemma 3, and then removes the admissible family $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_p\}$ (if any) from T. We observe that if u is a 1-boundary vertex in the resulting tree T' (i.e., $w(T'_{e(u)}) \geq 1$), then either (i) u is inactive if $w(T'_u) = w(\mathcal{B}) > \alpha$, or (ii) $w(T'_u) = w(\mathcal{B}) \leq \alpha$ and $w(T'_{e(u)}) \geq 1$ otherwise, where $\mathcal{B} = \mathcal{L} \cup \mathcal{M} \cup \mathcal{H}$. In the latter case, $D_{T'}(u)$ is an admissible set by Lemma 2(ii). In other words, the above procedure makes an active 1-boundary vertex inactive or creates an admissible set.

Algorithm RemoveActive
Input: A rooted tree T.
Output: An admissible family $\mathcal{S}$ in T such that $T' = T - \bigcup_{S \in \mathcal{S}} S$ has no active 1-boundary vertices in $V_1(T')$.

$\mathcal{S} := \emptyset$; Let V_A be the set of all inactive 1-boundary vertices in $V_1(T)$;
while $V_1(T) - V_A \neq \emptyset$ **do**
  Choose a 1-boundary vertex $u \in V_1(T) - V_A$;
  Find a valid partition $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_p, \mathcal{L}, \mathcal{M}, \mathcal{H}\}$ of $\mathcal{B}(u)$ in Lemma 3;
  Let $\mathcal{S} := \mathcal{S} \cup \{V(\mathcal{S}_i) - \{u\} \mid i = 1, 2, \ldots, p\}$; $T := T - (V(\mathcal{S}) - \{u\})$;
    /* $T_u = \mathcal{B} = \mathcal{L} \cup \mathcal{M} \cup \mathcal{H}$ holds */
  if $w(T_{e(u)}) \geq 1$ (i.e., $u \in V_1(T)$) **then**
    if $w(T_u) = w(\mathcal{B}) > \alpha$ **then**
      /* $|\mathcal{H}| \geq 2$, $\mathcal{M} = \emptyset$ and $w(\mathcal{L}) < 2 - \alpha$ by Lemma 3 */
      $V_A := V_A \cup \{u\}$ /* u is inactive in the current T */
    else /* $w(T_u) = w(\mathcal{B}) \leq \alpha$ and $w(T_{e(u)}) \geq 1$ */
      $\mathcal{S} := \mathcal{S} \cup \{V(T_u)\}$; $T := T - V(T_u)$
    end /* if */
  end /* if */
end; /* while */
/* $V_1(T) = V_A$ holds */
Return $\mathcal{S}$ and $T' := T$.

Lemma 4. *Given a rooted tree T,* RemoveActive *outputs an admissible family $\mathcal{S}$ and a tree T' such that all vertices in $V_1(T')$ are inactive.*

Theorem 3 follows from Lemma 4.

5 Proof of Theorem 4

This section gives a proof of Theorem 4. For this, we assume that $\alpha = 2\sqrt{3} - 3/2$ throughout this section.

For a rooted tree T, consider a branch T_e of a vertex u in T such that $2\alpha - 3 \leq w(T_e) < 1$ and $T_{e(u)} \geq 1$. We denote by $\mathcal{H}^*(T)$ the set of such branches T_e in T.

Lemma 5. *Let T be a rooted tree which has no active 1-boundary vertices in $V_1(T)$. Then every 2-boundary vertex $z \in V_2(T)$ satisfies the following condition.*

Condition B
(i) $|\mathcal{H}^*(T_z)| \geq 2$, (ii) $|\mathcal{H}(u)| = 2$ *for all* $u \in V_1(T_z)$, (iii) $w(z,u) < 2 - \alpha$ *for all* $u \in V_1(T_z)$.

Note that, for any $T' \in \mathcal{H}^*(T_z)$ with a root r', the root r' must be z or a 1-boundary vertex in $V_1(T_z)$ since $w(T_{e(r')}) \geq 1$ must hold by the maximality of T'.

We distinguish two cases; (i) $|\mathcal{H}^*(T_z)| = 2$ and (ii) $|\mathcal{H}^*(T_z)| \geq 3$. In case (i), we find a 2-admissible family in T_z. In case (ii), we find a 1-admissible family $\{S\}$, where a corresponding 1-admissible tree T_S will be constructed by introducing an edge which is not in tree T.

5.1 Case of $|\mathcal{H}^*(T_z)| = 2$

We first consider the case where $|\mathcal{H}^*(T_z)| = 2$.

Lemma 6. *Let T be a rooted tree which has no active 1-boundary vertex, and let $z \in V_2(T)$ satisfy $|\mathcal{H}^*(T_z)| = 2$. Then $|V_1(T) \cap V(T_z)| = 1$. Furthermore, for $\mathcal{H}^*(T_z) = \{T_1, T_2\}$, one of the following holds:*

(i) *If $V_1(T) \cap V(T_z) = \{z\}$ holds, then $S_1 := V(T_1)$ and $S_2 := V(T_z) - S_1$ give a 2-admissible family in T.*
(ii) *If $V_1(T) \cap V(T_z) = \{u\}$, $u \neq z$, and $w(T_z) < 2$ hold, then $S_1 := V(T_1)$ and $S_2 := V(T_z) - S_1$ give a 2-admissible family in T.*
(iii) *If $V_1(T) \cap V(T_z) = \{u\}$, $u \neq z$, and $w(T_z) \geq 2$ hold, then for any minimal subset $\mathcal{S} \subseteq \mathcal{B}(z) - \{T_e\}$ satisfying $w(T_e) + w(\mathcal{S}) \geq 2$, $S_1 := V(T_1)$ and $S_2 := V(T_e) \cup V(\mathcal{S}) - (S_1 \cup \{z\})$ give a 2-admissible family in T, where $T_e \in \mathcal{B}(z)$ satisfies $u \in V(T_e)$.*

5.2 Case of $|\mathcal{H}^*(T_z)| \geq 3$

We next consider the case where $|\mathcal{H}^*(T_z)| \geq 3$. In this case, we construct a 1-admissible tree by introducing an edge not in the current tree T.

The next lemma describes a property of the angle formed by two of three line segments in the Euclidean space $\mathbb{R}^d$.

Lemma 7. *For vertices z, v_1, v_2, and v_3, the minimum angle θ formed by line segments $\overline{zv_i}$ and $\overline{zv_j}$, $i \neq j$, $i, j = 1, 2, 3$, is no more than $120°$.*

For a descendent v of a vertex u in a rooted tree T, the $(1/2)$-*distant point* of v with u is defined by the point p_{uv} on the line segment uv such that $w(u, p_{uv}) = \min\{1/2, w(u, v)\}$, i.e., $p_{uv} = v$ if $w(u, v) \leq 1/2$, and p_{uv} is the point on uv satisfying $w(u, p_{uv}) = 1/2$ otherwise.

Let z be a 2-boundary vertex in a rooted tree T which has no active 1-boundary vertices. By definition, the root u_i of any branch $T_i \in \mathcal{H}^*(T_z)$ is a 1-boundary vertex in T. Therefore, the intersection of any two branches $T_i, T_j \in \mathcal{H}^*(T_z)$ contains a vertex if and only if they have the same root, i.e., $u_i = u_j$. We also note that a 1/2-boundary vertex $v_i \in V_{1/2}(T_i)$ is unique since $w(T_i) < 1$.

The next lemma claims that if $|\mathcal{H}^*(T_z)| \geq 3$, then an admissible set $\mathcal{S}$ can be found or $\mathcal{H}^*(T_z)$ contains a pair of branches T_i and T_j that satisfies a certain condition.

Lemma 8. *Let T be a rooted tree which has no active 1-boundary vertices in $V_1(T)$. Let z be a 2-boundary vertex in $V_2(T)$ such that there are three subtrees $T_1, T_2, T_3 \in \mathcal{H}^*(T_z)$, where u_i denotes the root of T_i ($u_i \in V_1(T) \cap V(T_z)$), $i = 1, 2, 3$. For each $i = 1, 2, 3$, let v_i be a 1/2-boundary vertex in $V_{1/2}(T_i)$. Then:*

(a) *For each $T_i \in \{T_1, T_2, T_3\}$, there exists a branch $\tilde{T}_i \in \mathcal{H}(\tilde{u}_i) \cap \mathcal{H}^*(T_z) - T_i$ such that the root $\tilde{u}_i \in V_1(T) \cap V(T_z)$ of $\tilde{T}_i$ satisfies $w(u_i, \tilde{u}_i) < 2 - \alpha$.*

(b) *Assume that there is $i \in \{1, 2, 3\}$ such that $w(T_{v_i}) \geq 1/2$ holds. If there exists a minimal subtree $\mathcal{S} \subseteq \mathcal{B}(v_i)$ such that $w(\mathcal{S}) + w(\tilde{T}_i) \geq 1$ and*

$$w(\mathcal{S}) + w(\tilde{T}_i) + w(v_i, \tilde{u}_i) \leq \alpha, \tag{1}$$

then $S := (V(\tilde{T}_i) - \{\tilde{u}_i\}) \cup (V(\mathcal{S}) - \{v_i\})$ is admissible. Testing whether such a subtree exists or not can be done in $O(|\mathcal{B}(v_i)|)$ time.

(c) *If (b) does not hold, then there exists $\{T_i, T_j\} \subseteq \{T_1, T_2, T_3\}$ $(i \neq j)$ such that the tuple $(T, z, T_i, T_j, u_i, u_j, p_{u_i v_i}, p_{u_j v_j})$ satisfies the following condition; $i = 1$ and $j = 2$ are assumed without loss of generality.*

Condition C

(i) *For each $i = 1, 2$, $1/2 \leq w(T_i) < 1$ holds.*

(ii) *$\theta := \angle p_{u_1 v_1} z p_{u_2 v_2} \leq 120°$.*

(iii) *For each $i = 1, 2$, $w(z, u_i) < 2 - \alpha$ holds.*

(iv) *For each $i = 1, 2$, if $w(T_{v_i}) \geq 1/2$ holds, then $\bar{\mathcal{B}}_i := \mathcal{B}(v_i) - \{B_i\}$ satisfies $w(\bar{\mathcal{B}}_i) < 2 - \alpha$, where $B_i \in \mathcal{B}(v_i)$ is the heaviest branch of v_i.*

Let $(T, z, T_1, T_2, u_1, u_2, p_{u_1 v_1}, p_{u_2 v_2})$ denote a tuple for a 2-boundary vertex $z \in V_2(T)$ in a rooted tree T, two branches $T_1 \in \mathcal{B}(u_1), T_2 \in \mathcal{B}(u_2)$ rooted at $u_1, u_2 \in V(T_z)$ (possibly $u_1 = z$, $u_2 = z$ or $u_1 = u_2$), respectively, 1/2-boundary vertices $v_1 \in V_{1/2}(T_1), v_2 \in V_{1/2}(T_2)$, and $(1/2)$-distant points $p_{u_1 v_1}, p_{u_2 v_2}$ of v_1, v_2 with u_1, u_2, respectively. Assume that such a tuple $(T, z, T_1, T_2, u_1, u_2, p_{u_1 v_1}, p_{u_2 v_2})$ satisfies Condition C. The following procedure, called COMBINE returns an 1-admissible set S, where a corresponding 1-admissible tree will be generated by introducing an edge which is not in the tree T.

COMBINE
Input: A tuple $(T, z, T_1, T_2, u_1, u_2, p_{u_1v_1}, p_{u_2v_2})$ satisfying Condition C.
Output: An admissible set S in T.
1 **If** $w(T_{v_1}) < 1/2$ or $w(T_{v_2}) < 1/2$ hold **then**
2 Return $S := D(v_1) \cup D(v_2)$
3 **else** /* $w(T_{v_1}) \geq 1/2$ and $w(T_{v_2}) \geq 1/2$ hold */
4 Choose a subset $\mathcal{S} \subset \mathcal{B}(v_1) \cup \mathcal{B}(v_2)$ such that $1 \leq w(\mathcal{S}) < 3 - \alpha$ holds;
5 Return $S := \mathcal{S} - \{v_1, v_2\}$
6 **end**. /* if */

The correctness of COMBINE is shown by using the following two lemmas.

Lemma 9. *For a tuple* $(T, z, T_1, T_2, u_1, u_2, p_{u_1v_1}, p_{u_2v_2})$ *satisfying Condition C,*

$$\overline{v_1v_2} < 2\,(5/2 - \alpha)\sin(\theta/2) + \overline{p_{u_1v_1}v_1} + \overline{p_{u_2v_2}v_2}.$$

Lemma 10. *For a tuple* $(T, z, T_1, T_2, u_1, u_2, p_{u_1v_1}, p_{u_2v_2})$ *satisfying Condition C, the following* (i), (ii) *and* (iii) *hold:*

(i) *For each* $i \in \{1, 2\}$ *satisfying* $w(T_{v_i}) < 1/2$, $w(T_{v_i}) + \overline{p_{u_iv_i}v_i} < 1/2$ *holds.*
(ii) *For each* $i \in \{1, 2\}$ *satisfying* $w(T_{v_i}) \geq 1/2$, $w(T_{v_i}) < 5/2 - \alpha$ *and* $\overline{p_{u_iv_i}v_i} = 0$ *hold.*
(iii) *If* $w(T_{v_1}) \geq 1/2$ *and* $w(T_{v_2}) \geq 1/2$ *hold, then there exists a subset* $\mathcal{S} \subset \mathcal{B}(v_1) \cup \mathcal{B}(v_2)$ *such that* $1 \leq w(\mathcal{S}) < 3 - \alpha$ *holds.*

Lemma 11. *Any set* S *output by* COMBINE *is an admissible set.*

The proof of Theorem 5 is similar to that of Theorem 4 described above.

6 Concluding Remarks

In this paper, we have studied the balanced tree partitioning problem in the Euclidean space $\mathbb{R}^d$, and have shown that the ratio α of the maximum tree weight to $w(T^*)/k$ for the weight $w(T^*)$ of a minimum spanning tree is at most 1.964 for $d \geq 3$ and at most 1.953 for $d = 2$, by designing polynomial time algorithms for finding a tree cover that attains such α. To derive these results, we allowed for trees in a tree cover to use edges not in a minimum spanning tree T^*. We remark that if a tree cover is required to consist of subtrees of T^*, then the best possible α is at least 2 (see [8] for a tight example). It would be interesting and challenging to improve the current upper bounds on the ratio α by developing further insightful geometric arguments over the Euclidean space.

References

1. Andersson, M., Gudmundsson, J., Levcopoulos, C., Narasimhan, G.: Balanced partition of minimum spanning trees. International Journal of Computational Geometry and Applications 13, 303–316 (2003)
2. Averbakh, I., Berman, O.: A heuristic with worst-case analysis for minmax routing of two traveling salesman on a tree. Discrete Applied Mathematics 68, 17–32 (1996)
3. Averbakh, I., Berman, O.: $(p-1)/(p+1)$-approximate algorithm for p-traveling salesman problems on a tree with minmax objective. Discrete Applied Mathematics 75, 201–216 (1997)
4. Bozkaya, B., Erkut, E., Laporte, G.: A tabu search heuristics and adaptive memory procedure for political districting. European J. Operation Research 144, 12–26 (2003)
5. Bruno, J., Downey, P.: Complexity of task sequencing with deadlines, set-up times and changeover costs. SIAM J. Comput. 7, 393–581 (1978)
6. Cordone, R., Maffioli, F.: On the complexity of graph tree partition problems. Discrete Applied Mathematics 134, 51–65 (2004)
7. Even, G., Garg, N., Könemann, J., Ravi, R., Sinha, A.: Min-max tree covers of graphs. Operations Research Letters 32, 309–315 (2004)
8. Karakawa, S., Morsy, E., Nagamochi, H.: Minmax Tree Cover in the Euclidean Space. Technical Report, 2008-013, Discrete Mathematics Lab., Graduate School of Informatics, Kyoto University, `http://www.amp.i.kyoto-u.ac.jp/tecrep/TR2008.html`
9. Karuno, Y., Nagamochi, H.: 2-approximation algorithms for the multi-vehicle scheduling on a path with release and handling times. Discrete Applied Mathematics 129, 433–447 (2003)
10. Karuno, Y., Nagamochi, H.: Scheduling vehicles on trees. Pacific J. Optim. 1, 527–543 (2005)
11. Karuno, Y., Nagamochi, H.: Vehicle scheduling problems in graphs. In: Gonzalez, T.F. (ed.) Handbook of Approximation Algorithms and Metaheuristics, ch. 46. Chapman & Hall/CRC, Boca Raton (2007)
12. Nagamochi, H.: Approximating the minmax rooted-subtree cover problem. IEICE Trans. Fundamentals E88-A(5), 1335–1338 (2005)
13. Nagamochi, H., Kawada, T.: Minmax subtree cover problem on cacti. Discrete Applied Mathematics 154(8), 1254–1263 (2006)
14. Nagamochi, H., Okada, K.: Polynomial time 2-approximation algorithms for the minmax subtree cover problem. In: Ibaraki, T., Katoh, N., Ono, H. (eds.) ISAAC 2003. LNCS, vol. 2906, pp. 138–147. Springer, Heidelberg (2003)
15. Nagamochi, H., Okada, K.: A faster 2-approximation algorithm for the minmax p-traveling salesman problem on a tree. Discrete Applied Mathematics 140, 103–114 (2004)
16. Nagamochi, H., Okada, K.: Approximating the minmax rooted-tree cover in a tree. Information Processing Letters 104, 173–1178 (2007)
17. Williams Jr., J.C.: Political districting: a review. Papers in Regional Science 74, 12–40 (1985)

Network Design with Weighted Degree Constraints*

Takuro Fukunaga and Hiroshi Nagamochi

Department of Applied Mathematics and Physics,
Graduate School of Informatics, Kyoto University, Japan
{takuro,nag}@amp.i.kyoto-u.ac.jp

Abstract. In an undirected graph $G = (V, E)$ with a weight function $w : E \times V \to \mathbb{Q}_+$, the weighted degree $d_w(v; E)$ of a vertex v is defined as $\sum\{w(e, v) \mid e \in E \text{ incident with } v\}$. In this paper, we consider a network design problem which has upper-bounds on weighted degrees of vertices as its constraints while the objective is to compute a minimum cost graph with a prescribed connectivity. We propose bi-criteria approximation algorithms based on the iterative rounding, which has been successfully applied to the degree-bounded network design problem. A problem minimizing the maximum weighted degree of vertices is also discussed.

1 Introduction

Let $G = (V, E)$ be an undirected graph. A weight function $w : E \times V \to \mathbb{Q}_+$ is defined on pairs of edges and their end vertices, where $\mathbb{Q}_+$ is the set of non-negative rational numbers. Let $\delta(v; E)$ denote the set of edges in E incident with $v \in V$. We define the *weighted degree* of a vertex $v \in V$ in G as $\sum_{e \in \delta(v;E)} w(e, v)$, and denote it by $d_w(v; E)$. The weighted degree of G is defined as $\max_{v \in V} d_w(v; E)$.

The weighted degree of a vertex measures load on the vertex in applications. For constructing a network with balanced load, it is important to consider weighted degree of networks. Take a communication network for example, and suppose that $w(e, v)$ represents the load (e.g., communication traffic, communication charge) for the communication device on a node v to use a link e incident with v. Then the weighted degree of v indicates the total load of v for using the network.

In this paper, we consider a network design problem which has upper-bounds on weighted degrees of vertices as its constraints while the objective is to compute a minimum cost graph with a prescribed connectivity. In the above example of the communication network, this corresponds to the case in which each node has an upper-limit on the load that can be handled on the node.

The problem introduces two types of edges. This is useful for modeling various ways to allocating loads. For an edge $e = uv$ of the first type, weights of e on

* This work was partially supported by Grant-in-Aid for Scientific Research from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 214–225, 2009.

u and v are given as inputs. For an edge $e = uv$ of the second type, the sum of weights of e on u and v is given, and we can decide how much is allocated to end vertices u and v.

For stating our problems formally, let us define several notations related to connectivity of graphs. For a subset U of V and a subset F of E, $\delta(U;F)$ denotes the set of edges in F which join vertices in U with those in $V-U$, and $F(U)$ denotes the set of edges in F whose both end vertices are in U. Let $\mathbb{N}$ be the set of natural numbers. For a given set function $f : 2^V \to \mathbb{N}$ on V, a graph $G' = (V, F)$ is called *f-connected* when $|\delta(U;F)| \geq f(U)$ holds for every non-empty $U \subset V$. If $f(X)+f(Y) \leq f(X \cap Y)+f(X \cup Y)$ or $f(X)+f(Y) \leq f(X-Y)+f(Y-X)$ holds for any $X, Y \subseteq V$, then f is called *skew supermodular*. With a skew supermodular set function, f-connectivity represents a wide variety of connectivity of graphs such as the local edge-connectivity.

Now we formulate our problem.

Weighted Degree Bounded Survivable Network Problem (WDBoundedNetwork**):** Let $G = (V, E)$ be an undirected graph where E is the union of disjoint sets E_1 and E_2. For those edge sets, weights $w_1 : E_1 \times V \to \mathbb{Q}_+$ and $\mu : E_2 \to \mathbb{Q}_+$ are respectively defined. As inputs, we are given the graph $G = (V, E = E_1 \cup E_2)$ with the weights w_1 and μ, an edge-cost $c : E \to \mathbb{Q}$ ($\mathbb{Q}$ is the set of rational numbers), a skew supermodular set function $f : 2^V \to \mathbb{N}$, and a degree-bound $b : V \to \mathbb{Q}_+$. A solution consists of $F \subseteq E$, weights $w_2(e, u) \in \mathbb{Q}_+$ and $w_2(e, v) \in \mathbb{Q}_+$ for each $e = uv \in F_2$, where F_i denotes $F \cap E_i$. We call w_2 *allocation* of μ when $w_2(e, u) + w_2(e, v) = \mu(e)$ for $e = uv \in F_2$. Throughout this paper, we let $w : F \times V \to \mathbb{Q}_+$ refer to the function that returns $w_i(e, v)$ for $e \in F_i$ and $v \in V$. The solution is defined to be feasible if $G' = (V, F)$ is f-connected, w_2 is an allocation of μ, and degree constraint $d_w(v;F) \leq b(v)$ for each $v \in V$ is satisfied. The goal of this problem is to find a feasible solution that minimizes its cost $\sum_{e \in F} c(e)$.

If $f(U) = 1$ for all non-empty $U \subset V$, then the minimal solutions are spanning trees. We particularly call such instances *weighted degree bounded spanning tree problem* (WDBoundedTree).

Feasible solutions of WDBoundedTree are Hamiltonian paths when $E_2 = \emptyset$, $w_1(e, u) = w_1(e, v) = 1$ for all $e = uv \in E_1$, and $b(v) = 2$ for all $v \in V$. This means that it is NP-hard to test whether an instance of WDBoundedTree (and hence WDBoundedNetwork) is feasible or not. By this reason, it is natural to relax the degree constraints and consider bi-criteria approximation algorithms. We say that, for an instance of WDBoundedNetwork and some $\alpha, \beta \geq 1$, a solution consisting of $F \subseteq E$ and an allocation w_2 of μ is an *(α, β)-approximate solution* if it satisfies

- $\sum_{e \in F} c(e) \leq \alpha \min\{\sum_{e \in F'} c(e) \mid F' \subseteq E \text{ is in a feasible solution}\}$, and
- $d_w(v;F) \leq \beta b(v)$ for all $v \in V$.

Define θ as $\max\{b(u)/b(v), b(v)/b(u) \mid uv \in E_2\}$ if $E_2 \neq \emptyset$, and 0 otherwise. For problems WDBoundedTree and WDBoundedNetwork, we propose algorithms which achieve approximation ratios $(1, 4+3\theta)$ and $(2, 7+5\theta)$ respectively.

Our algorithms take the approach successfully applied to the bounded degree spanning tree problem by Singh and Lau [18] and to the bounded-degree survivable network design problem by Lau et al. [12], which correspond to instances with uniform w_1 and $E_2 = \emptyset$ in our problems. Their approach is based on the iterative rounding originally used for the generalized Steiner network problem by Jain [8]. Roughly illustrating, they iterate rounding fractional variables in basic optimal solutions or removing constraints of a linear programming relaxation. The key for guaranteeing the correctness of the algorithm is an analysis of the structure of tight constraints which determine the basic optimal solutions. In this paper, we show that this approach remains useful even if the weighted degree is introduced.

In addition, we also discuss the following variation of the above problem.

Minimum weighted degree survivable network problem (MinimumWDNetwork**):** An undirected graph $G = (V, E)$ with $E = E_1 \cup E_2$, weights $w_1 : E_1 \times V \to \mathbb{Q}_+$ and $\mu : E_2 \to \mathbb{Q}_+$, and a skew supermodular set function $f : 2^V \to \mathbb{N}$ are given. A feasible solution consists of a f-connected subgraph $G' = (V, F)$ of G and an allocation $w_2 : F_2 \times V \to \mathbb{Q}_+$ of μ. The objective is to minimize the weighted degree $\max_{v \in V} d_w(v; F)$ of G'.

Similarly for problem WDBoundedNetwork, we call instances with $f(U) = 1$ for all non-empty $U \subset V$ *minimum weighted degree spanning tree problem* (MinimumWDTree).

For problems MinimumWDTree and MinimumWDNetwork, our algorithms achieve approximation ratios $7 + \epsilon$ and $12 + \epsilon$ in polynomial time of $\log(1/\epsilon)$ and input size for an arbitrary $\epsilon > 0$. If $E_2 = \emptyset$, we can remove ϵ from the ratios while the algorithms run in polynomial time of only input size.

Previous Works

The bounded degree spanning tree problem has been studied extensively in the last two decades [2,3,10,11,16,17]. For the uniform cost (i.e., $c(e) = 1$ for $e \in E$), an optimal result was given by Fürer and Raghavachari [4]. Their algorithm computes a spanning tree which violates degree upper-bounds by at most one. For general costs, Goemans [6] gave an algorithm to compute a spanning tree of the minimum cost although it violates degree upper-bounds by at most two. The algorithm obtains such a spanning tree by rounding a basic optimal solution of an LP relaxation with the matroid intersection algorithm. Afterwards an optimal result for general cost was presented by Singh and Lau [18]; Their algorithm computes a spanning tree of minimum cost which violates degree upper-bounds by at most one. As mentioned above, their result is achieved by extending the iterative rounding due to Jain [8], who applied it for designing a 2-approximation algorithm to the generalized Steiner network problem.

This approach is also applied to several problems with degree bounds. Lau et al. [12] considered the survivable network problem, and proposed an algorithm that outputs a network of cost at most twice the optimal and the degree of $v \in V$ is at most $2b(v) + 3$. This result was improved in Lau and

Singh [13]. Bansal et al. [1] considered the arborescence problem and survivable network problem with intersecting supermodular connectivity. Kiraly et al. [9] generalized bounded degree spanning tree to bounded degree matroid. They also considered degree bounded submodular flow problem.

There are also several works on the network design problem with weighted degree constraints. All of these correspond to the case with $E_2 = \emptyset$ and $w_1(e, u) = w_2(e, v)$ for $e = uv \in E_1$. Ravi [15] presented an $O(\log |V|, \log |V|)$-approximation algorithm to problem WDBoundedTree and an $O(\log |V|)$-approximation algorithm to problem MinimumWDTree. For problem MinimumWDTree, Ghodsi et al. [5] presented a 4.5-approximation algorithm under the assumption that G is a complete graph and c is a metric cost (i.e., triangle inequality holds) while they also showed that it is NP-hard to approximate it within a factor less than 2. Notice that our algorithm described in this paper achieves $(1, 4)$-approximation to problem WDBoundedTree and 4-approximation to problem MinimumWDTree when $E_2 = \emptyset$. Hence it improves these previous works. Nutov [14] considers problem WDBoundedNetwork for digraphs.

Organization

The rest of this paper is organized as follows. Section 2 presents our algorithms to problems WDBoundedTree and MinimumWDTree. The algorithms are derived from a good property of polytopes that give a linear programming relaxation of the problems. Section 2 also shows that our analysis on the property is tight. Section 3 gives our algorithms to problems WDBoundedNetwork and MinimumWDNetwork, and shows that our analysis on the property of polytopes is tight.

2 Spanning Trees with Weighted Degree Constraints

In this section, we generalize problem WDBoundedTree by defining b as a function $A \to \mathbb{Q}_+$ for some $A \subseteq V$. This means that the degree upper-bound is defined on only the vertices in A. This is is necessary for our algorithm to work inductively.

Let $\mathcal{I}$ stand for the instence of problem WDBoundedTree consisting of an undirected graph $G = (V, E)$ with $E = E_1 \cup E_2$, weights $w_1 : E_1 \times V \to \mathbb{Q}_+$ and $\mu : E_2 \to \mathbb{Q}_+$, a subset A of V, and $b : A \to \mathbb{Q}_+$. We denote by $\mathrm{P_T}(\mathcal{I})$ the polytope that consists of vectors $x \in \mathbb{Q}^E$ and $y \in \mathbb{Q}^{E_2 \times V}$ that satisfy

$$0 \le x(e) \quad \text{for all } e \in E, \tag{1}$$

$$0 \le y(e, u),\ y(e, v) \quad \text{for all } e = uv \in E_2, \tag{2}$$

$$y(e, u) + y(e, v) = x(e) \quad \text{for all } e = uv \in E_2, \tag{3}$$

$$x(E) = |V| - 1, \tag{4}$$

$$x(E(U)) \le |U| - 1 \quad \text{for all } U \subset V \text{ with } 2 \le |U|, \tag{5}$$

and

$$\sum_{e\in\delta(v;E_1)} w_1(e,v)x(e) + \sum_{e\in\delta(v;E_2)} \mu(e)y(e,v) \le b(v) \quad \text{for all } v \in A, \tag{6}$$

where $x(F)$ denotes $\sum_{e\in F} x(e)$ for $F \subseteq E$. Remark that (5) with $U = \{u, v\}$, $uv \in E$ implies

$$x(e) \le 1 \quad \text{for all } \ e \in E. \tag{7}$$

Also constraints (4) and (5) with $U = V - v$ imply

$$x(\delta(v; E)) \ge 1 \text{ for all } v \in V, \tag{8}$$

since $x(\delta(v; E)) = x(E) - x(E(V - v)) \ge (|V| - 1) - (|V - v| - 1) = 1$.

Observe that $\min\{c^T x \mid (x, y) \in \mathrm{P_T}(\mathcal{I})\}$ with $A = V$ is a linear programming relaxation of problem WDBoundedTree. (Variable $x(e)$ decides whether edge e is chosen, and variable $y(e, v)$ decides how much of $\mu(e)$ is allocated to an end vertex v of e.) Although (5) has an exponential number of constraints, the linear program is solvable in polynomial time by using the ellipsoid method [2] or by transforming it to an equivalent formulation of polynomial-size [7].

For a vector $x \in \mathbb{Q}_+^E$, let E_x denote $\{e \in E \mid x(e) > 0\}$. We say that polytope $\mathrm{P_T}(\mathcal{I})$ is $(1, \beta)$-*bounded* for some $\beta \ge 1$ if every extreme point (x^*, y^*) of the polytope satisfies at least one of the following:

- There exists a vertex $v \in V$ such that $|\delta(v; E_{x^*})| = 1$;
- There exists a vertex $v \in A$ such that $|\delta(v; E_{x^*})| \le \beta$.

If $|\delta(v; E_{x^*})| = 1$, then $x^*(e) = 1$ holds for the edge $e \in \delta(v; E_{x^*})$ by the equalities $x(\delta(v; E_{x^*})) = x(\delta(v; E)) \ge 1$ and $x(e) \le 1$.

In what follows, we see that the iterative rounding can be applied to problem WDBoundedTree when $\mathrm{P_T}(\mathcal{I})$ is $(1, \beta)$-bounded. By this and the fact that $\mathrm{P_T}(\mathcal{I})$ is $(1, 3)$-bounded (Theorem 3), we can obtain an approximation algorithm for problem WDBoundedTree.

Now let us describe the algorithm which works under the assumption that $\mathrm{P_T}(\mathcal{I})$ is $(1, \beta)$-bounded. Roughly illustrating, if the former condition of the $(1, \beta)$-boundedness holds, then the algorithm rounds the variable corresponding to $e \in \delta(v; E_{x^*})$, and otherwise, the algorithm removes the upper-bound on the weighted degree of v satisfying $|\delta(v; E_{x^*})| \le \beta$.

Algorithm for problem WDBoundedTree

Input: An undirected graph $G = (V, E)$ with $E = E_1 \cup E_2$, weights $w_1 : E_1 \times V \to \mathbb{Q}_+$ and $\mu : E_2 \to \mathbb{Q}_+$, an edge-cost $c : E \to Q$, and a degree-bound $b : V \to \mathbb{Q}_+$.

Output: A solution consisting of a spanning tree $T \subseteq E$ of G and an allocation $w_2 : T_2 \times V \to \mathbb{Q}_+$ of μ, or message "INFEASIBLE".

Step 1: Set $A := V$ and $T := \emptyset$.

- Delete $e = uv \in E_1$ from G if $w_1(e, u) > b(u)$ or if $w_1(e, v) > b(v)$.
- Delete $e = uv \in E_2$ from G if $\mu(e) > b(u) + b(v)$.

If $\mathrm{P_T}(\mathcal{I}) = \emptyset$, then output "INFEASIBLE", and terminate;

Step 2: Compute a basic solution (x^*, y^*) that minimizes $\sum_{e \in E} c(e)x^*(e)$ over $(x^*, y^*) \in \mathrm{P_T}(\mathcal{I})$.

Step 3: Remove edges in $E - E_{x^*}$ from E;

Step 4: If there exists a vertex $v \in V$ such that $|\delta(v; E_{x^*})| = 1$ (i.e., the edge $e = uv \in \delta(v; E_{x^*})$ satisfies $x^*(e) = 1$), then add e to T and delete v from G. Moreover, execute one of the following operations according to the class of e:

Case of $e \in E_1$**:** If $u \in A$, then set $b(u) := b(u) - w_1(e, u)$;

Case of $e \in E_2$**:** Set $w_2(e, u) := \mu(e)y^*(e, u)$ and $w_2(e, v) := \mu(e)y^*(e, v)$. If $u \in A$, then set $b(u) := b(u) - w_2(e, u)$.

Step 5: If there exists a vertex $v \in A$ such that $|\delta(v; E_{x^*})| \leq \beta$, then remove v from A;

Step 6: If $|V| = 1$, then output (T, w_2) as a solution, and terminate. Otherwise, return to Step 2.

Define $\theta = \max\{b(u)/b(v), b(v)/b(u) \mid uv \in E_2\}$ if $E_2 \neq \emptyset$, and $\theta = 0$ otherwise.

Theorem 1. *If each* $\mathrm{P_T}(\mathcal{I})$ *constructed in Step 2 of the algorithm is* $(1, \beta)$*-bounded, then problem* WDBoundedTree *is* $(1, 1 + \beta(1 + \theta))$*-approximable in polynomial time.*

Proof. It is clear that the algorithm described above runs in polynomial time. In what follows, we see that the algorithm computes a $(1, 1+\beta(1+\theta))$-approximate solution.

Observe that the linear program over $\mathrm{P_T}(\mathcal{I})$ is still a relaxation of the given instance after Step 1. Hence the original instance has no feasible solutions when the algorithm outputs "INFEASIBLE". Each edge $e = uv \in E$ satisfies the following properties after Step 1:

- If $e = uv \in E_1$, then $w_1(e, u) \leq b(u)$ and $w_1(e, v) \leq b(v)$;
- If $e = uv \in E_2$, then $\mu(e) \leq b(u)+b(v) \leq (1+\theta)b(u)$ and $\mu(e) \leq b(u)+b(v) \leq (1 + \theta)b(v)$;

Now suppose that $\mathrm{P_T}(\mathcal{I}) \neq \emptyset$ after Step 1. We then prove that $\mathrm{P_T}(\mathcal{I}) \neq \emptyset$ also throughout the subsequent iterations and that the spanning tree T outputted by the algorithm satisfies $c(T) \leq \min\{c^T x \mid (x, y) \in \mathrm{P_T}(\mathcal{I})\}$ and $d_w(v; T) \leq (1 + \beta(1 + \theta))b(v)$ for all $v \in V$.

Let $e_i = u_i v_i$ denote the i-th edge added to T, $\mathcal{I}_i = (G_i = (V_i, E^i), w_1, \mu, A_i, b_i)$ denote $\mathcal{I}$ at the beginning of the iteration in which e_i is added to T, and (x_i^*, y_i^*) denote the basic solution computed in Step 2 of that iteration. We also let $\mathcal{I}_0$ stand for $\mathcal{I}$ immediately after Step 1 of the algorithm. Assume that e_i is chosen by $|\delta(v_i; E_{x_i^*})| = 1$ in Step 4 (i.e., $V_{i+1} - V_i = \{v_i\}$).

By Steps 4 and 5, $A_{i+1} \subseteq A_i$ holds, and

$$b_{i+1}(v) = \begin{cases} b_i(v) - w_1(e_i, v) & \text{if } v = u_i \in A \text{ and } e_i \in E_1, \\ b_i(v) - \mu(e_i)y_i^*(e_i, v) & \text{if } v = u_i \in A \text{ and } e_i \in E_2, \\ b_i(v) & \text{otherwise.} \end{cases} \tag{9}$$

also holds for $i \geq 1$. Moreover, each edge in $E_i - (E_{i+1} \cup \{e_i\})$ is the one such that the corresponding variable of x^* becomes 0 in some iteration before e_{i+1} is chosen in Step 4. These facts indicate that the projection of (x_i^*, y_i^*) satisfies all constraints in $\mathrm{P_T}(\mathcal{I}_{i+1})$. Hence we have the following:

$$\text{If } \mathrm{P_T}(\mathcal{I}_i) \neq \emptyset, \text{ then } \mathrm{P_T}(\mathcal{I}_{i+1}) \neq \emptyset \text{ for } i \geq 0; \tag{10}$$

$$c^T x_i^* \geq c(e_i) + \min\{c^T x \mid (x, y) \in \mathrm{P_T}(\mathcal{I}_{i+1})\} = c(e_i) + c^T x_{i+1}^* \quad \text{for } i \geq 1. \tag{11}$$

(i) We first see that the algorithm outputs a solution. Recall that we are assuming that $\mathrm{P_T}(\mathcal{I}_0) \neq \emptyset$. By this and (10), $\mathrm{P_T}(\mathcal{I}_i) \neq \emptyset$ for all $i \geq 1$. The algorithm then terminates with outputting a spanning tree $T = \{e_1, \ldots, e_{|V|-1}\}$ and an allocation $w_2 : T_2 \times V \to \mathbb{Q}_+$ of μ by the way of the construction.

(ii) Next we see the optimality of $c(T)$. By applying (11) repeatedly, we obtain

$$c^T x_1^* \geq c(e_1) + c^T x_2^* \geq \cdots \geq \sum_{i=1}^{|V|-2} c(e_i) + c^T x_{|V|-1}^*.$$

Since $|V_{|V|-1}| = 2$, $x_{|V|-1}^*(e_{|V|-1}) = 1$ and $x_{|V|-1}^*(e) = 0$ for $e \in E_{|V|-1} - \{e_{|V|-1}\}$ obviously hold, and hence $\sum_{i=1}^{|V|-2} c(e_i) + c^T x_{|V|-1}^* = \sum_{i=1}^{|V|-1} c(e_i) = c(T)$. Notice that the algorithm constructs $\mathcal{I}_1$ from $\mathcal{I}_0$ by relaxing the degree constraints (i.e., $A_1 \subseteq A_0$). Hence $\min\{c^T x \mid (x, y) \in \mathrm{P_T}(\mathcal{I}_0)\} \geq c^T x_1^*$ holds. Hence we have $\min\{c^T x \mid (x, y) \in \mathrm{P_T}(\mathcal{I}_0)\} \geq c(T)$, as required.

(iii) Fix v as an arbitrary vertex. Now we prove that $d_w(v; T) \leq (1 + \beta(1 + \theta))b(v)$ holds.

Consider Step 4 of the iterations during $v \in A$. Let T' be the set of edges that are added to T during those iterations. By applying (9) repeatedly, we obtain

$$b(v) \geq \sum_{e_i \in \delta(v; T_1')} w_1(e_i, v) + \sum_{e_i \in \delta(v; T_2')} \mu(e_i) y_i^*(e_i, v).$$

If $e_i \in \delta(v; T_2)$, then the algorithm set $w_2(e_i, v)$ to $\mu(e_i) y_i^*(e_i, v)$. Therefore,

$$\sum_{e_i \in \delta(v; T_1')} w_1(e_i, v) + \sum_{e_i \in \delta(v; T_2')} \mu(e_i) y_i^*(e_i, v) = d_{w_1}(v; T_1') + d_{w_2}(v; T_2').$$

It implies that $d_w(v; T') \leq b(v)$ holds.

Consider the iterations after v is removed from A. Let T'' denote the set of edges that are added to T during those iterations. When v is removed from A in Step 5, the number of remaining edges incident with v is at most β by the condition in Step 5. Hence $|\delta(v; T'')| \leq \beta$ holds. We have already seen that, after Step 1, each $e = uv \in E_1$ satisfies $w_1(e, v) \leq b(v)$ and each $e = uv \in E_2$ satisfies $w_2(e, v) \leq \mu(e) \leq (1 + \theta)b(v)$. So $d_w(v; T'') \leq \beta(1 + \theta)b(v)$. Because $d_w(v; T) = d_w(v; T') + d_w(v; T'')$, we have $d_w(v; T) \leq (1 + \beta(1 + \theta))b(v)$.

The following theorem shows that the algorithm to problem WDBoundedTree gives an algorithm to problem MinimumWDTree.

Theorem 2. *Suppose that problem* WDBoundedTree *with uniform b is (α', β')-approximable for some α' and β'. For an arbitrary $\epsilon > 0$, problem* MinimumWDTree *is $(\beta' + \epsilon)$-approximable in polynomial time of $\log(1/\epsilon)$ and input size. If $E_2 = \emptyset$, then it is β'-approximable in polynomial time of only input size.*

Proof. For an $r \in \mathbb{Q}$, define G_r as the subgraph obtained from G by deleting each edge $e = uv \in E_1$ such that $\max\{w_1(e,u), w_1(e,v)\} > r$ and each edge $e \in E_2$ such that $\mu(e) > 2r$. Let $b_r : V \to \mathbb{Q}_+$ be the function such that $b_r(v) = r$ for all $v \in V$, and $\mathcal{I}_r = (G_r, w_1, \mu, A = V, b_r)$.

We denote $\min\{r \in \mathbb{Q}_+ \mid \mathrm{P_T}(\mathcal{I}_r) \neq \emptyset\}$ by R, and the minimum weighted degree of the given instance by OPT. Let ω and W stand for the minimum and maximum entry of w_1 and μ, respectively. For given ϵ, define $\epsilon' = \epsilon\omega/(2\beta')$. Since $\omega/2 \leq \mathrm{OPT}$, we have $\epsilon' \leq \epsilon\mathrm{OPT}/\beta'$. Since the characteristic vector of an optimal solution to the given instance of problem MinimumWDTree satisfies all constraints of $\mathrm{P_T}(\mathcal{I}_{OPT})$, we have $R \leq \mathrm{OPT}$. It is possible to compute a value R' such that $R \leq R' \leq R + \epsilon'$ by the binary search on interval $[0, W]$, which needs to solve the linear program over $\mathrm{P_T}(\mathcal{I}_r)$ $\log(W/\epsilon')$ times.

Let T be an (α', β')-approximate solution to the instance of problem WDBoundedTree consisting of $\mathcal{I}_{R'}$ and an arbitrary edge-cost c. We then have $d_w(v;T) \leq \beta' b_{R'}(v) \leq \beta'(R + \epsilon') \leq (\beta' + \epsilon)\mathrm{OPT}$ for any $v \in V$. This implies that T is a $(\beta' + \epsilon)$-approximate solution to problem MinimumWDTree.

When $E_2 = \emptyset$, set ϵ so that $1/(\psi + 1) \leq \epsilon' < 1/\psi$ holds, where ψ is the maximum deliminator of all enteries in w_1 and μ. In this case, if R' satisfies $R \leq R' \leq R + \epsilon'$, then $R' \leq \mathrm{OPT}$. Such R' can be computed by solving the linear program over $\log(W/\epsilon') \leq \log(W(\psi+1))$ times. Then we have $d_w(v;T) \leq \beta' b_{R'}(v) \leq \beta'\mathrm{OPT}$ for any $v \in V$, which implies that T is a β'-approximate solution. □

Now we see that $\mathrm{P_T}(\mathcal{I})$ is $(1,3)$-bounded. First let us observe that the key property of tight constraints observed in [18] also holds in our setting.

Lemma 1. *For any extreme point (x^*, y^*) of $\mathrm{P_T}(\mathcal{I})$, there exists a laminar family $\mathcal{L} \subseteq \{U \subseteq V \mid |U| \geq 2\}$ (i.e., any $U, U' \in \mathcal{L}$ satisfy either $U \subseteq U'$, $U' \subseteq U$, or $U \cap U' \neq \emptyset$) and a subset X of A such that $|E_{x^*}| \leq |\mathcal{L}| + |X|$.*

Proof. By the definitions of x^* and y^*, the number of variables is equal to the dimension of the vector space spanned by the coefficients vectors of tight constraints in $\mathrm{P_T}(\mathcal{I})$. If $x^*(e) = 0$ (resp., $y^*(e,v)$), then fix the variable $x(e)$ (resp., $y(e,v)$) to 0 and remove the corresponding tight constraint of (1) (resp., (2)). We can also remove tight constraints of (3) by fixing $y(e,u)$ to $x(e) - y(e,v)$. Then the number of remaining variables, which is at least $|E_{x^*}|$, is equal to the dimension of the vector space spanned by the tight constraints of (4), (5) and (6).

Let $\mathcal{F} = \{U \subseteq V \mid |U| \geq 2, x^*(E(U)) = |U| - 1\}$ (i.e., family of vertex subsets defining tight constraints of (4) and (5)) and $X = \{v \in A \mid$

$\sum_{e\in\delta(v;E_1)} w_1(e,v)x^*(e) + \sum_{e\in\delta(v;E_2)} \mu(e)y^*(e,v) = b(v)\}$ (i.e., set of vertices defining tight constraints of (6)).

For a subfamily $\mathcal{F}'$ of $\mathcal{F}$, we denote by $span(\mathcal{F}' \cup X)$ the vector space spanned by the coefficient vectors of constraints corresponding to $\mathcal{F}'$ and X. (Notice that coefficient vectors corresponding to X are changed from the original by the above operations.) In [18], it is proven that a maximal laminar subfamily $\mathcal{L}$ of $\mathcal{F}$ satisfies $span(\mathcal{L} \cup X) = span(\mathcal{F} \cup X)$. Since the dimension of $span(\mathcal{L} \cup X)$ is at most $|\mathcal{L}| + |X|$, we have $|E_{x^*}| \leq |\mathcal{L}| + |X|$, as required. □

Theorem 3. *Polytope* $\mathrm{P_T}(\mathcal{I})$ *is* $(1,3)$*-bounded for any* $\mathcal{I}$*.*

Proof. It can be derived from Lemma 1 although we omit the detail here.

Corollary 1. *Problem* WDBoundedTree *is* $(1, 4+3\theta)$*-approximable in polynomial time. Problem* MinimumWDTree *is* 4*-approximable in polynomial time if* $E_2 = \emptyset$*, and is* $(7+\epsilon)$*-approximable in polynomial time of* $\log(1/\epsilon)$ *and input size for any* $\epsilon > 0$ *otherwise.*

Proof. Immediate from Theorems 1, 2 and 3. □

It is a natural question to ask whether the $(1,3)$-boundedness of $\mathrm{P_T}(\mathcal{I})$ can be improved to $(1,2)$-boundedness. Let us discuss this assuming that $E_2 = \emptyset$. Unfortunately $(1,2)$-boundedness does not hold even if $w_1(e,u) = w_1(e,v) = 1$ for all $e = uv \in E_1$ as mentioned in [18]. Singh and Lau [18] weakened the $(1,2)$-boundedness by replacing its first condition with "There exists an edge $e \in E$ such that $x^*(e) = 1$." They then designed their algorithm by observing that the property holds for more general polytopes than $\mathrm{P_T}(\mathcal{I})$. This approach does not work in our setting because there exists a counterexample for the weakened $(1,2)$-boundedness, which we will give in the rest of this section.

Let G be the graph in Figure 1. We let $w_1(e,u) = w_1(e,v)$ for all $e = uv \in E_1$ and integers beside edges in the figure represent their weights. Rational numbers beside vertices represent the values of b for them. Let $A = V$, and the set of $|E| = 6$ tight constraints consist of constraints (4), (5) for the set of white vertices and for the set of black vertices, and (6) for all vertices. Then these tight constraints determine an extreme point x^* of $\mathrm{P_T}(\mathcal{I})$ such that

$$x^*(e) = \begin{cases} 2/3 & \text{for edges represented by solid lines,} \\ 1/3 & \text{for edges represented by dotted lines.} \end{cases}$$

Clearly, $x^*(e) < 1$ for any edge $e \in E$ and $\min_{v\in A=V} |\delta(v; E_{x^*})| = 3$.

3 Survivable Network with Weighted Degree Constraints

Similarly for the previous section, we introduce a general form of problem WD-BoundedNetwork by defining $b : A \to \mathbb{Q}_+$ on a subset A of V. Moreover

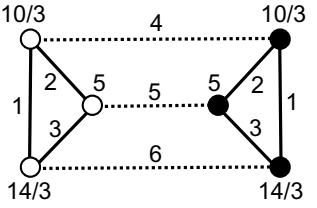


Fig. 1. A counterexample for $(1, 2)$-boundedness of $\mathrm{P_T}(\mathcal{I})$

we let $\mathcal{I}$ stand for an instence of the problem consisting of an undirected graph $G = (V, E)$ with $E = E_1 \cup E_2$, weights $w_1 : E_1 \times V \to \mathbb{Q}_+$ and $\mu : E_2 \to \mathbb{Q}_+$ a skew supermodular set function $f : 2^V \to \mathbb{N}$, a subset A of V, and $b : A \to \mathbb{Q}_+$. We denote by $\mathrm{P_N}(\mathcal{I})$ the polytope that consists of vectors $x \in \mathbb{Q}^E$ and $y \in \mathbb{Q}^{E_2 \times V}$ that satisfy

$$0 \le x(e) \le 1 \quad \text{for all } e \in E, \tag{12}$$

$$0 \le y(e, u),\ y(e, v) \quad \text{for all } e = uv \in E_2, \tag{13}$$

$$y(e, u) + y(e, v) = x(e) \quad \text{for all } e = uv \in E_2, \tag{14}$$

$$x(\delta(U)) \ge f(U) \quad \text{for all non-empty } U \subset V, \tag{15}$$

and

$$\sum_{e \in \delta(v; E_1)} w_1(e, v)x(e) + \sum_{e \in \delta(v; E_2)} \mu(e)y(e, v) \le b(v) \quad \text{for all } v \in A. \tag{16}$$

Observe that $\min\{c^T x \mid (x, y) \in \mathrm{P_N}(\mathcal{I})\}$ with $A = V$ is a linear programming relaxation of problem WDBoundedNetwork.

We say that $\mathrm{P_N}(\mathcal{I})$ is (α, β)-*bounded* for some $\alpha, \beta \ge 1$ if every extreme point (x^*, y^*) of the polytope satisfies at least one of the following:

- There exists an edge $e \in E_{x^*}$ such that $x^*(e) \ge 1/\alpha$;
- There exists a vertex $v \in A$ such that $|\delta(v; E_{x^*})| \le \beta$.

Notice that $(1, \beta)$-boundedness of $\mathrm{P_N}(\mathcal{I})$ is weaker than that of $\mathrm{P_T}(\mathcal{I})$.

In this section, we show that polytope $\mathrm{P_N}(\mathcal{I})$ is $(2, 5)$-bounded. From this result, we can derive approximation algorithms to problems WDBoundedNetwork and MinimumWDNetwork as in Section 2. Here we only mention the result due to the space limitation.

Theorem 4. *Let* θ *denote* $\max\{b(u)/b(v), b(v)/b(u) \mid uv \in E_2\}$ *if* $E_2 \neq \emptyset$, *and* 0 *otherwise. Problem* WDBoundedNetwork *is* $(2, 7+5\theta)$*-approximable in polynomial time. Problem* MinimumWDNetwork *is* 7*-approximable in polynomial time if* $E_2 = \emptyset$, *and is* $(12 + \epsilon)$*-approximable in polynomial time of* $\log(1/\epsilon)$ *and input size for any* $\epsilon > 0$ *otherwise.* □

For proving the $(2, 5)$-boundedness of $\mathrm{P_N}(\mathcal{I})$, let us see that the key property of tight constraints observed in [8] also holds in our setting.

Lemma 2. *Let (x^*, y^*) be any extreme point of $\mathrm{P_N}(\mathcal{I})$ and suppose that $x^*(e) < 1$ for all $e \in E$. There exists a laminar family $\mathcal{L} \subseteq 2^V$ and a subset X of A such that characteristic vectors of $\delta(U; E_{x^*})$ for $U \in \mathcal{L}$ are linearly independent and $|E_{x^*}| \leq |\mathcal{L}| + |X|$.*

Proof. Omitted due to the space limitation. □

Theorem 5. *Polytope $\mathrm{P_N}(\mathcal{I})$ is $(2, 5)$-bounded for any $\mathcal{I}$.*

Proof. Suppose the contrary, i.e., all edges $e \in E_{x^*}$ satisfy $x^*(e) < 1/2$, and all vertices $v \in A$ satisfy $|\delta(v; E_{x^*})| \geq 6$.

Let $\mathcal{L}$ and X be those in Lemma 2. We define a child-parent relationship between all elements in $\mathcal{L}$ and X as follows: For $U \in \mathcal{L}$ or $v \in X$, define its parent as the inclusion-wise minimal element in $\mathcal{L}$ that contains it if any. Note that when $v \in X$ and $\{v\} \in \mathcal{L}$, $\{v\}$ is the parent of v.

We assign one token to each end vertex of edges in E_{x^*}. Define the *co-requirement* of $U \in \mathcal{L}$ as $|\delta(U; E_{x^*})|/2 - f(U)$. Following the approach in [8], we observe that it is possible to distribute these tokens to all elements in $\mathcal{L}$ and in X so that

- each element having the parent owns two tokens,
- each element having no parent owns at least three tokens,
- and it owns exactly three only if its co-requirement equals to $1/2$.

First two of these mean that the number of all tokens is more than $2(|\mathcal{L}| + |X|)$. Since the number of tokens is exactly $2|E_{x^*}|$, this indicates that $|E_{x^*}| > |\mathcal{L}|+|X|$, which contradicts $|E_{x^*}| \leq |\mathcal{L}| + |X|$.

We prove the claim inductively. The base case is when the elements have no child. An element $v \in X$ owns at least six tokens by $|\delta(v; E_{x^*})| \geq 6$. An element $U \in \mathcal{L}$ with no child owns at least three tokens because $|\delta(U; E_{x^*})| \geq 3$ by $x^*(e) < 1/2$ for each $e \in \delta(U; E_{x^*})$ and $f(U) \geq 1$. It owns exactly three tokens if and only if $|\delta(U; E_{x^*})| = 3$. Since $|\delta(U; E_{x^*})| = 3$ indicates that $f(U) = 1$, it means the co-requirement $|\delta(U; E_{x^*})|/2 - f(U)$ equals to $1/2$.

Let us consider the case in which an element $U \in \{\mathcal{L}\}$ has some children. If U has children from X, then it is possible to redistribute tokens so that U owns at least four tokens, and each child owns two tokens. If the children of U are all from $\mathcal{L}$, then the argument is proven in [8]. □

Lau et. al. [12] designed their algorithm for $w_1(e, u) = w_1(e, v) = 1$, $e = uv \in E_1$ and $E_2 = \emptyset$ by observing that $\mathrm{P_N}(\mathcal{I})$ is $(2, 4)$-bounded with such instances. However, an example indicates that this does not hold in our problem even if $w_1(e, u) = w_1(e, v)$ for all $e = uv \in E_1$ and $E_2 = \emptyset$ although we do not present it here due to the space limitation.

References

1. Bansal, N., Khandekar, R., Nagarajan, V.: Additive guarantees for degree bounded directed network design. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, pp. 769–778 (2008)

2. Chaudhuri, K., Rao, S., Riesenfeld, S., Talwar, K.: What would Edmonds do? augmenting paths and witnesses for degree-bounded MSTs. In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX 2005 and RANDOM 2005. LNCS, vol. 3624, pp. 26–39. Springer, Heidelberg (2005)
3. Chaudhuri, K., Rao, S., Riesenfeld, S., Talwar, K.: A push-relabel algorithm for approximating degree bounded MSTs. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 191–201. Springer, Heidelberg (2006)
4. Fürer, M., Raghavachari, B.: Approximating the minimum-degree Steiner tree to within one of optimal. Journal of Algorithms 17(3), 409–423 (1994)
5. Ghodsi, M., Mahini, H., Mirjalali, K., Gharan, S.O., Sayedi, A.S., Zadimoghaddam, R.M.: Spanning trees with minimum weighted degrees. Information Processing Letters 104, 113–116 (2007)
6. Goemans, M.X.: Minimum bounded-degree spanning trees. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, pp. 273–282 (2006)
7. Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization. Springer, Heidelberg (1988)
8. Jain, K.: A factor 2 approximation algorithm for the generalized Steiner network problem. Combinatorica 21, 39–60 (2001)
9. Kiraly, T., Lau, L.C., Singh, M.: Degree bounded matroids and submodular flows. In: Lodi, A., Panconesi, A., Rinaldi, G. (eds.) IPCO 2008. LNCS, vol. 5035, pp. 259–272. Springer, Heidelberg (2008)
10. Könemann, J., Ravi, R.: A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees. SIAM Journal on Computing 31(6), 1783–1793 (2002)
11. Könemann, J., Ravi, R.: Primal-dual meets local search: approximating MST's with nonuniform degree bounds. SIAM Journal on Computing 34, 763–773 (2005)
12. Lau, L.C., Naor, J.S., Singh, M., Salavatipour, M.R.: Survivable network design with degree or order constraints. In: Proceedings of the 39th ACM Symposium on Theory of Computing (STOC), pp. 651–660 (2007)
13. Lau, L.C., Singh, M.: Additive approximation for bounded degree survivable network design. In: Proceedings of the 40th ACM Symposium on Theory of Computing (STOC), pp. 759–768 (2008)
14. Nutov, Z.: Approximating directed weighted-degree constrained networks. In: Goel, A., Jansen, K., Rolim, J.D.P., Rubinfeld, R. (eds.) APPROX and RANDOM 2008. LNCS, vol. 5171, pp. 219–232. Springer, Heidelberg (2008)
15. Ravi, R.: Steiner Trees and Beyond: Approximation Algorithms for Network Design. PhD thesis, Department of Computer Science, Brown University (1993)
16. Ravi, R., Marathe, M.V., Ravi, S.S., Rosenkrantz, D.J., Hunt III, H.B.: Many birds with one stone: Multi-objective approximation algorithms. In: Proceedings of the 25th ACM Symposium on Theory of Computing, pp. 438–447 (1993)
17. Ravi, R., Singh, M.: Delegate and conquer: An LP-based approximation algorithm for minimum degree MSTs. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 169–180. Springer, Heidelberg (2006)
18. Singh, M., Lau, L.C.: Approximating minimum bounded degree spanning trees to within one of optimal. In: Proceedings of the 39th ACM Symposium on Theory of Computing (STOC), pp. 661–670 (2007)

Minimum Cuts of Simple Graphs in Almost Always Linear Time

Michael Brinkmeier[1]

Institut für Theoretische Informatik
Technische Universität Ilmenau, Germany
mbrinkme@tu-ilmenau.de

Abstract. In this paper we prove, that the minimum cut algorithm presented previously by the author (Brinkmeier 07), requires only linear time with high probability, if the input graph if chosen randomly from the graphs with constant expected degree. In fact a more general lower bound for the probability of a low runtime depending on several parameters is given.

1 Introduction

The problem of finding a minimum cut of a graph appears in many applications, for example, in network reliability, clustering, information retrieval and chip design. More precisely, a minimum cut of an undirected graph with edge weights, is a set of edges with minimum sum of weights, such that its removal would cause the graph to become unconnected. The total weight of edges in a minimum cut of G is denoted by λ_G and called *(edge-)connectivity* of G.

In the literature many algorithms can be found that apply various methods. One group of algorithms is based on the well-known result of Ford and Fulkerson [4] regarding the duality of maximum s-t-flows and minimum s-t-cuts for arbitrary vertices s and t. In [7] Gomory and Hu presented an algorithm which calculated $n-1$ maximum s-t-flows from a given source s to all other vertices t. Hao and Orlin adapted the maximum flow algorithm of Goldberg and Tarjan [6] and were able to construct a minimum cut of a directed graph with nonnegative real numbers as edge weights in time $\mathcal{O}(nm \log(n^2/m))$. Two other algorithms are related to an approach of Gabow [5] based on matroids.

Nagamochi and Ibaraki [13,17,12] described an algorithm without using maximum flows. Instead they constructed spanning forests and iteratively contracted edges with high weights. This leads to an asymptotic runtime of $\mathcal{O}(nm+n^2 \log n)$ on undirected graphs with nonnegative real edge weights. On undirected, unweighted multigraphs they obtained a runtime of $\mathcal{O}(n(n+m))$. Translated to integer weighted graphs without parallel edges, this corresponds to a runtime of $\mathcal{O}(n(n+M))$ where M is the sum of all edge weights. Using a 'searching' technique, they improved this to $\mathcal{O}(M+\lambda_G n^2)$.

Karger and Stein [10] used the contraction technique for a randomized algorithm calculating the minimum cut of undirected graphs in $\mathcal{O}(n^2 \log^3 n)$ expected

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 226–237, 2009.

time. Later Karger [8,9] presented two related algorithms using expected time $\mathcal{O}(m \log^3 n)$ and $\mathcal{O}(n^2 \log n)$.

Nagamochi and Ibaraki's approach was refined in [19] by Stoer and Wagner. They replaced the construction of spanning forests with the construction of *Maximum Adjacency Orders (MA-order)* but the asymptotic runtime stayed unchanged. In [2], the author introduced *Lax Adjacency Orders (LA-order)*, a relaxed version of MA-orders. These orderings can be constructed faster and lead to an asymptotic worst case runtime $\mathcal{O}(\delta_G n^2)$ for graphs with non-negative integer edge weights, with δ_G is the minimum degree of G. But as simple experiments already indicated in [2], the average runtime on random graphs is even lower.

In this paper, we prove, that for random graphs of constant expected degree the algorithm presented in [2] requires time $\mathcal{O}(n + m)$ with high probability. To be slightly more precise, we obtain an lower bound for the probability that the algorithm requires only a certain time, if the input is chosen randomly from $\mathcal{G}(n, p_n)$, ie. the graphs with n vertices and edge probability p_n. This bound is then used to prove, that for a constant $\omega \geq 2$ and $p_n = \frac{\omega}{n-1}$ the probability, that the algorithm requires linear time, converges to 1 as $n \to \infty$. Furthermore, we prove that if $p_n \in o(\sqrt{n})$ and $p_n \cdot (n-1) \to \infty$, the time required by the algorithm is subquadratic in n. The precise formulation is given in Theorem 3.

The relevance of this result does not rely on the fact, that the minimum cut on random graphs can be solved fast, since this is quite easy. With high probability, the vertex of minimum degree is a minimum cut, resulting in a very simple, linear time approximation algorithm, giving almost always the correct result. But the analysis strengthens the impression, that the worst case bound in fact is much too conservative. Furthermore, it is the first analysis of the 'expected' runtime of a minimum cut algorithm – at least as far as the author knows.

2 Definitions

In the following let $G = (V, E)$ be an unweighted, undirected multi-graph, ie. it may contain parallel edges, but no loops. Let $n = |V|$ be its number of vertices and $m = |E|$ its number of edges. For an edge $e \in E$ let denote ends(e) the set of both ends of e. The *degree* $\deg(v)$ of a vertex v is the number of edges incident to it. The *minimum degree* over all vertices of G is denoted by δ_G. Obviously we have $m \geq \frac{1}{2} \cdot \delta_G \cdot n$.

A *cut* of G is an (unordered) partition $(C, V \setminus C)$ of the vertex set into two parts. For shorter notation the cut will often be written as C. The *weight* $w(C)$ of the cut C is the number of "cut" edges. For two vertices u and v a *u-v-cut* is a cut C such that $u \in C$ and $v \notin C$ or vice versa, ie. C *separates u and v*. A *minimum u-v-cut* is a u-v-cut, whose weight is minimal amongst all u-v-cuts of G. The weight of a minimum u-v-cut is denoted by $\lambda_G(u, v)$. A *minimum cut* is a cut C with minimal weight among all cuts of G. Its weight λ_G is called the *edge-connectivity* of G.

For a subset $U \subseteq V$ of vertices $G[U]$ denotes the induced subgraph of G, ie. the graph consisting of the vertices in U and all edges of G between them, ie. $E(G[U]) = \{e \in E \mid \text{ends}(e) \subseteq U\}$. If $U = \{v_1, \ldots, v_m\}$ we also write $G[v_1, \ldots, v_m]$.

Let u and v be two vertices of $G = (V, E)$. The graph $G/u \sim v$ is obtained from G by identifying u and v, this means that u and v are replaced by a new vertex $[u] = [v]$ and that edges incident to u and/or v are now incident to $[u]$. Edges which become loops are removed.

This construction can be generalized to an arbitrary equivalence relation $\sim$ on the vertices. Let $[u]$ be the equivalence class of u according to v. Then $G/\sim = (V', E')$ is given by $V' = \{[u] \mid u \in V\}$ and the edges are adapted apropriately. The graph $G/\sim$ is called a *contraction* of G. Obviously the contracted graph $G/u \sim v$ is a special case of the more general quotient.

3 The Algorithm

The main tool for our algorithm are *Lax Adjacency Orderings*, which are a generalization of the Maximum Adjacency Ordering as introduced by Stoer and Wagner in [19].

Definition 1 (Lax Adjacency Order; [2], Definition 2). *Let $G = (V, E)$ be an undirected, unweighted multi-graph and $\tau \geq 0$. An order $v_1, v_2, \ldots, v_n$ on the vertices of G is a* lax adjacency order *or* LA-order for threshold τ, *if*

$$\min(\tau, w(v_1, \ldots, v_{i-1}; v_i)) \geq \min(\tau, w(v_1, \ldots, v_{i-1}; v_k))$$

for all $k \geq i$, where $w(v_1, \ldots, v_{i-1}; v_i)$ is the number of edges adjacent to v_i and the set $\{v_1, \ldots, v_{i-1}$. These values are called adjacencies.

Assume that $v_1, \ldots, v_n$ is an arbitrary LA-order with threshold τ. Let G^τ be the contraction of G by the equivalence relation $\sim$ induced by

$$v_{i-1} \sim v_i \quad :\Leftrightarrow \quad w(v_1, \ldots, v_{i-1}; v_i) \geq \tau.$$

In other words we contract sequences of vertices with adjacencies $\geq \tau$ into a single vertices. The usefulness of this construction in the context of connectivity is caused by the following result.

Theorem 1 ([2], Theorem 3). *For an LA-order $v_1, \ldots, v_n$ with threshold $\tau > 0$ on $G = (V, E)$ the following equation holds for $1 \leq i \leq n$*

$$\min(\lambda_G, \tau) = \min(\delta_G, \lambda_{G^\tau}, \tau).$$

The contraction G^τ has an important property, which is a consequence of Lemma 3 and Lemma 4 of [2].

Lemma 1. *G^τ has at most $(\tau - 1) \cdot n$ edges, which are not loops (i.e. starting and ending in the same vertex).*

Using a *Priority Queue with Threshold*, as described in [2], we can construct a Lax Adjacency Ordering with threshold τ in time $\mathcal{O}(\tau \cdot n + m)$.

Theorem 2 ([2]). *A Lax-Adjacency order with threshold τ on $G = (V, E)$ with edge weights from $\mathbb{N}$ can be computed in $\mathcal{O}(\tau \cdot n + m)$ time.*

Theorem 1 provides us with a simple algorithm for the determination of a minimum cut. Algorithm 1 is a description of the algorithm given in [2] at a quite high-level of abstraction.

The notation $[v]$ has two meanings. In line 3, $[v]$ is a specific vertex in G_i, which in turn represents a set of original vertices in G, which is also denoted by $[v]$ in all other situations.

Algorithm 1. The Minimum Cut Algorithm

Input: An undirected graph $G = (V, E)$
Output: A minimum cut C

```
1  Set G_0 := G, i := 0, τ := ∞ and C := ∅
2  while G_i contains two or more vertices do
3      Determine a vertex [v] of G_i with w([v]) = δ_{G_i}
4      if δ_{G_i} < τ then
5          Set C := [v]
6          τ := min(τ, δ_{G_i})
7      G_{i+1} := G_i^τ for some LA-order with threshold τ.
8      if v_i, i > 1, had adjacency 0 during the construction of the LA-order then
9          C := {v_1, ..., v_{i-1}}
10         τ := 0
11         return
12     i := i + 1
13 The cut is given by the vertices in [v] and has weight τ.
```

In addition, the occurence of adjacencies of value 0 is treated differently. It is easily checked, that G is unconnected if and only if the first LA-ordering contains at least one vertex with adjacency 0. If v_i is one of these vertices, then it is clear, that the set $\{v_1, \ldots, v_{i-1}\}$ is a union of components of G. Hence, we may terminate after the first round, if G is unconnected. This is done in lines 9-13.

As proven in [2], the construction of G^τ reduces the number of vertices by at least one, since the last vertex has an adjacecy $\geq \delta_G \geq \tau$. Hence the algorithm requres at most $(n-1)$ executions of the while-loop. G_i is the graph constructed at the end of the i-th execution of the while-loop. n_i is its number of vertices and m_i its number of edges. Furthermore, let τ_i be the value of τ used for the construction of G_{i+1}.

The detection of a class $[v]$ of minimum incident weight in G_i requires $\mathcal{O}(n_i + m_i)$ time, as does the construction of the contraction $G_{i+1} = G_i^\tau$. Hence, the i-th

execution of the body of the loop requires at most time $\mathcal{O}(\delta_{G_{i-1}} \cdot n_{i-1} + m_{i-1})$, since $\tau_{i-1} \leq \delta_G$. Because $2 \cdot \delta_{G_{i-1}} \cdot n_{i-1} \leq m_{i-1}$, this implies, that the i-th execution of the loop requires at most $\mathcal{O}(n_{i-1} + m_{i-1})$ time.

Lemma 2. *Algorithm 1 requires at most $\mathcal{O}(n_{i-1} + m_{i-1})$ time for the execution of the i-th round, $i \geq 1$. Furthermore, observe that if G is unconnected, Algorithm 1 only requires one round.*

4 The Runtime

The proofs of the worst case runtimes in Theorem 1 and Lemma 2 assume, that during each round at most one contraction is conducted. But as experiments [1] and several examples (eg. trees) indicate, the average number of contractions per round is higher and hence the runtime lower.

We assume, that the graph $\mathcal{G}$ is constructed by a classical random process. Let $\mathcal{G}(n, p)$ be the set of all (unweighted) undirected graphs with n vertices and edge probability p, $0 \leq p \leq 1$. Ie. for every pair (u, v) of vertices the edge (u, v) is added to $\mathcal{G}$ with probability p.

Let $\mathcal{G}_i$ be the graph obtained after the i-th round, starting with $\mathcal{G} = \mathcal{G}_0$. Let N_i be the number of vertices/classes in $\mathcal{G}_i$ and M_i the number of edges of $\mathcal{G}$, that 'survived' the contractions (i.e. those edges with ends in two different classes of the contraction). Let δ_i be the minimum degree of $\mathcal{G}_i$ and $\overline{D}_i$ the average degree of $\mathcal{G}_i$, ie

$$\delta_i \leq \overline{D}_i = \frac{2 \cdot M_i}{N_i}. \tag{1}$$

τ_{i-1} is the threshold during the i-th round, ie. we have $\tau_0 = \delta_0$ and $\tau_i \leq \delta_j$ for $j \leq i$. Due to Lemma 1 we have $M_{i+1} \leq \tau_i \cdot N_i$ for $i \geq 0$.

Let T_i be the time required by the i-th round of Algorithm 1, $i \geq 1$, and $T = \sum_{i=1}^{n-1} T_i$ the total time. Following Theorem 1, there exist constants $\tilde{C} > 0$ and $C = 2\tilde{C}$, such that

$$T_{i+1} \leq \tilde{C} \cdot (M_i + (\delta_i + 1)N_i) \leq \tilde{C} \cdot \left(M_i + N_i + 2\frac{M_i}{N_i} N_i \right) = C \cdot (M_i + N_i).$$

We assume $T_i = 0$ if the algorithm requires less than i rounds.

If $\mathcal{G}$ is unconnected, Algorithm 1 terminates after one round and its runtime satisfies $T \leq C_u \cdot (M_0 + n)$ for some constant $C_u > 0$. If $\mathcal{G}$ is connected, we always have $M_i \geq N_i - 1$ and hence $T_{i+1} \leq C_c \cdot M_i$ for some constant $C_c > 0$. For simplicity we assume $C = C_u = C_c$ and assume $\mathcal{G}$ to be connected in the following.

Since the number of classes decreases every round, $N_i \leq \sqrt{n}$ holds from some point on. Let L be the first round after which at most $\sqrt{n}$ vertices remain, ie. $L = \min\{i \mid N_i \leq \sqrt{n}\}$ and hence $N_L \leq \sqrt{n}$ and $N_{L-1} > \sqrt{n}$. For $i \geq L$, we obtain $M_{i+1} \leq \tau_i \cdot N_i \leq \delta_0 \cdot \sqrt{n}$. If $\mathcal{G}$ is connected the rounds following the L-th round, require time

$$T_{>L} = \sum_{i=L+1}^{n-1} T_i \leq C \cdot \sqrt{n} \cdot M_L \leq C \cdot \sqrt{n} \cdot \delta_0 \cdot \sqrt{n} = C \cdot \delta_0 \cdot n,$$

where we used, that at most $\sqrt{n}$ rounds can follow the L-th round, and that the sequences M_i and N_i are monotone decreasing. Since δ_o is at most the average degree of $\mathcal{G}$, we obtain

$$T_{>L} \leq C \cdot n \cdot \frac{2 \cdot M_0}{n} = 2 \cdot C \cdot M_0, \tag{2}$$

if $\mathcal{G}$ is connected.

Now we examine the time $T_{\leq L}$ required for the first L rounds. Assume, that during these rounds in k rounds the inequality

$$M_i \leq (1 - \varepsilon) \cdot M_{i-1} \tag{3}$$

is satisfied. Let $1 \leq j_1 < \cdots < j_k \leq L$ be the indices of these rounds. Then $M_i \leq (1 - \varepsilon)^h \cdot M_0$ for $i \geq j_h$, and hence the total time required for the first L rounds is bounded by

$$\begin{aligned} T_{\leq L} &= \sum_{i=1}^{L} T_i \leq C \cdot \sum_{i=1}^{L} M_{i-1} \\ &\leq C \cdot M_0 \cdot \sum_{i=1}^{k} (1 - \varepsilon)^{i-1} + (L - k) \cdot C \cdot M_0 \leq C \cdot M_0 \cdot \left(\frac{1}{\varepsilon} + (L - k) \right). \end{aligned} \tag{4}$$

Since in a connected graph $M_i < \sqrt{n}$ implies $N_i < \sqrt{n}$, and since

$$M_0 \cdot (1 - \varepsilon)^i < \sqrt{n} - 1 \quad \Leftrightarrow \quad i < -\frac{\log(M_0) - \frac{1}{2}\log(n)}{\log(1 - \varepsilon)}$$

at most k_n rounds with

$$\left\lfloor -\frac{\log(M_0) - \frac{1}{2}\log(n)}{\log(1 - \varepsilon)} \right\rfloor \leq \left\lfloor -\frac{\frac{3}{2}\log(n)}{\log(1 - \varepsilon)} \right\rfloor =: k_n,$$

have to satisfy condition (3), to reach the second phase. Let K_l be the number of rounds among the l first, satisfying either (3) or $N_i < \sqrt{n}$. Then $K_l \geq k_n$ implies $L \leq l$ and hence

$$P\left(T \leq \left(2 + \frac{1}{\varepsilon} + (l - k_n) \right) \cdot C \cdot (M_0 + n) \right) \geq P(K_l \geq k_n), \tag{5}$$

because at least k_n of the first l rounds require at most $\frac{C \cdot (M_0 + n)}{\varepsilon}$ time, while the remaining $l - k_n$ of these rounds require at most $C \cdot (M_0 + n)$ each. Every round following the l-th, satisfies $N_i < \sqrt{n}$ and hence, $T_{>l} \leq 2 \cdot C \cdot (M_0 + n)$ time.

To obtain estimates for $P(K_l \geq k_n)$, we consider conditions, which imply (3). Assume that for a constant $0 < \varepsilon < 1$, the inequality

$$\tau_0 = \delta_0 \leq (1-\varepsilon) \cdot \frac{M_i}{N_i} \tag{6}$$

holds. Then Lemma 1 implies $M_{i+1} \leq \tau_i \cdot N_i \leq \delta_0 \cdot N_i \leq (1-\varepsilon) \cdot M_i$. One way to achieve (6) is to require

$$\delta_0 \leq \gamma \cdot p \cdot (n-1) \qquad \text{and} \qquad \frac{\gamma \cdot p \cdot (n-1)}{1-\varepsilon} \leq \frac{M_i}{N_i} \tag{7}$$

for a chosen constant $0 < \gamma < 1$. For $i \geq 0$ and $\tau := \gamma \cdot p \cdot (n-1)$ the following events are defined:

$$A : \delta_0 \leq \tau \quad \text{and} \quad B_i : \frac{\tau}{1-\varepsilon} \leq \frac{M_i}{N_i} \quad \text{and} \quad C_i : N_i < \sqrt{n}. \tag{8}$$

Furthermore, let D_i^1 be the union of B_i and C_i and D_i^0 its complement. Additionally, let $\Sigma^l := \{0,1\}^l$ be the set of all binary strings of length l, and $\Sigma^{l,k} \subseteq \Sigma^l$ the subset of all those strings with at least k 'ones', ie.

$$\Sigma^{l,k} := \left\{ (x_1, \ldots, x_l) \in \{0,1\}^l \mid \sum_{i=1}^{l} x_i \geq k \right\}.$$

For $\mathbf{x} \in \Sigma^{l,k}$, let $\mathbf{x}_i$ denote the i-th component of $\mathbf{x}$. With $D(\mathbf{x}) := \bigcap_{i=1}^{l} D_i^{\mathbf{x}_i}$ we get

$$P\left(T \leq \left(2 + \frac{1}{\varepsilon} + (l - k_n)\right) \cdot C \cdot (M_0 + n)\right) \geq \sum_{\mathbf{x} \in \Sigma^{l,k}} P\left(A \cap D(\mathbf{x})\right) \tag{9}$$

4.1 The Minimum Degree

In this subsection we only consider the input graph $\mathcal{G} = \mathcal{G}_0$. Hence we may omit the index 0. For two distinct vertices u and v, let $E_{u,v}$ be the random variable indicating, whether the edge (u,v) exists, and let M be the number of edges:

$$E_{u,v} = \begin{cases} 1 & \text{if } (u,v) \in E(\mathcal{G}) \\ 0 & \text{otherwise.} \end{cases} \qquad \text{and} \qquad M = \sum_{v \neq u} E_{v,u}. \tag{10}$$

For two distinct vertices $v, u \in \{1, \ldots, n\}$, let D_v be the random variable designating the degree of v, and $D_{v,u}$ the degree of v, without counting the edge (u,v), ie.

$$D_v = \sum_{u \neq v} E_{v,u} \qquad \text{and} \qquad D_{v,u} = D_v - E_{v,u}. \tag{11}$$

Furthermore, let $\overline{D}$ be the average degree of $\mathcal{G}$ and δ the minimum degree, ie.

$$\overline{D} = \frac{1}{n} \sum_{v} D_v = 2\frac{M}{n} \qquad \text{and} \qquad \delta = \min\left\{D_v \mid v \in V(\mathcal{G})\right\}. \tag{12}$$

Obviously, D_v is binomially distributed with success probability p and n tries. Ie. we have

$$P(D_v = k) = b(k; n-1, p) := \binom{n-1}{k} p^k (1-p)^{n-1-k} \tag{13}$$

$$P(D_v \leq k) = B(k; n-1, p) := \sum_{i=0}^{k} b(i; n-1, p). \tag{14}$$

Application of the Cauchy-Schwarz Inequality leads to the following result.

Lemma 3. *In $\mathcal{G}(n,p)$ with $0 < p < 1$ and $0 \leq \tau \leq n-1$ we have*

$$\frac{1}{P(\delta \leq \tau)} \leq \frac{1}{nB(\lfloor \tau \rfloor; n-1, p)} + \left(1 - \frac{1}{n}\right) \cdot \frac{B(\lfloor \tau \rfloor; n-2, p)^2}{B(\lfloor \tau \rfloor; n-1, p)^2} =: \varphi(n, p, \tau).$$

In other words, for the event A, as defined in (8), we have

$$P(A) \geq \varphi(n, p, \gamma \cdot p \cdot (n-1))^{-1}, \tag{15}$$

4.2 The Average Degree of a Contraction

Let $\mathcal{U}$ be an arbitrary partition of the vertices of $\mathcal{G}$ into $N > \sqrt{n}$ classes, and let $H = \mathcal{G}/\mathcal{U}$ be the contraction of the random graph $\mathcal{G}$ by this partition. Furthermore, let M be the number of edges in H. With $0 < \gamma < 1$, the second condition of (7) is equivalent to

$$\frac{\gamma}{1-\varepsilon} \cdot p \cdot (n-1) \cdot N \leq M.$$

Let $R_1, \ldots, R_N$ be the sizes of the classes of H and set

$$S := \sum_{j=1}^{N} R_j \cdot (n - R_j) = n^2 - \sum_{j=1}^{N} (R_j)^2,$$

ie. S is the number of possible edges in H. Consequently, we we have $E(M) = S \cdot p$. To obtain useful bounds for S, we use the following result.

Lemma 4. *Let $r_1, \ldots r_k$ be given, such that $\sum_{i=1}^{k} r_i = n$ and $1 \leq r_i$ for $i = 1, \ldots, k$. Then*

$$\frac{n^2}{k} \leq \sum_{i=1}^{k} r_i^2 \leq (k-1) + (n-k+1)^2$$

Due to this lemma, we have $(N-1) \cdot (2n-N) \leq S \leq n^2 \left(1 - \frac{1}{N}\right)$ and since $n - N \geq 0$, we obtain $N \cdot (n-1) \leq S \leq n^2 \left(1 - \frac{1}{N}\right)$ if $N \geq 2$ (which is the case for $n > 4$). Hence, $M \geq \frac{\gamma}{1-\varepsilon} \cdot p \cdot S$ implies $M \geq \frac{\gamma}{1-\varepsilon} \cdot p \cdot (n-1) \cdot N$, and thus, for a given partition $\mathcal{U}$, we have

$$P\left(\frac{\gamma}{1-\varepsilon}\cdot p\cdot(n-1)\cdot N\leq M\mid\mathcal{U}\right)$$
$$\geq P\left(\frac{\gamma}{1-\varepsilon}\cdot p\cdot S\leq M\mid\mathcal{U}\right)=P\left(\frac{\gamma}{1-\varepsilon}\cdot E(M)\leq M\mid\mathcal{U}\right),\quad(16)$$

leading to the follwoing result.

Lemma 5. *Let $\mathcal{G}$ be chosen randomly from $\mathcal{G}(n,p)$ with $0<p<1$ and let $\mathcal{U}$ be a partition of the vertices of $\mathcal{G}$ chosen with probability distribution $P(\mathcal{U})$ from all partitions with more than $\sqrt{n}$ classes. Furthermore, let ε and γ be two constants with $0<\gamma,\varepsilon$ and $\gamma+\varepsilon<1$. Then*

$$P\left(\tfrac{\gamma}{1-\varepsilon}\cdot p\cdot(n-1)\leq\tfrac{M}{N}\mid N>\sqrt{n}\right)$$
$$\geq\sum_{\mathcal{U}\mid|\mathcal{U}|>\sqrt{n}}P\left(\frac{\gamma}{1-\varepsilon}\cdot E(M)\leq M\mid\mathcal{U}\right)\cdot P(\mathcal{U}),$$

where M is the number of edges in $\mathcal{G}/\mathcal{U}$ and N the number of classes. In addition,

$$P\left(\frac{\gamma}{1-\varepsilon}\cdot p\cdot(n-1)\leq\frac{M}{N}\mid N>\sqrt{n}\right)$$
$$\geq 1-\sum_{s=\sqrt{n}\cdot(n-1)}^{n^2-n}B\left(\left\lceil\frac{\gamma}{1-\varepsilon}\cdot s\cdot p\right\rceil-1;s,p\right)\cdot P(S=s),$$

where S is the number of possible edges in $\mathcal{G}/\mathcal{U}$.

A Bound Based on the Chernoff-Inequality. Another bound can be retrieved from the Chernoff-inequality

$$P((1-\kappa)\cdot E(X)\geq X)\leq e^{-\frac{\kappa^2}{2}E(X)}\quad(17)$$

for a binomial distributed random variable X and $0<\kappa<1$. In our situation it implies

$$P\left(\frac{\gamma}{1-\varepsilon}\cdot E(M)>M\mid\mathcal{U}\right)\leq P\left(\frac{\gamma}{1-\varepsilon}\cdot E(M)\geq M\mid\mathcal{U}\right)\leq e^{-\left(1-\frac{\gamma}{1-\varepsilon}\right)^2\cdot\frac{E(M)}{2}}$$

for a partition $\mathcal{U}$ with at least $\sqrt{n}$ classes. Since $E(M)=p\cdot S\geq p\cdot(n-1)\cdot N$,

$$P\left(\frac{\gamma}{1-\varepsilon}\cdot p\cdot(n-1)\leq\frac{M}{N}\mid N>\sqrt{n}\right)\quad\geq\quad 1\;-\;e^{-\left(1-\frac{\gamma}{1-\varepsilon}\right)^2\cdot\frac{p\cdot(n-1)\cdot\sqrt{n}}{2}}$$

if $0<\frac{\gamma}{1-\varepsilon}<1$, which is equivalent to $0<\gamma,\varepsilon$ and $\gamma+\varepsilon<1$.

Lemma 6. *Let $\mathcal{G}$ be chosen randomly from $\mathcal{G}(n,p)$ with $0 < p < 1$ and $\mathcal{U}$ a partition chosen with probability distribution $P(\mathcal{U})$ from the set of all partitions with more than $\sqrt{n}$ classes. Furthermore let ε and γ be two constants with $0 < \gamma, \varepsilon$ and $\gamma + \varepsilon < 1$, then*

$$P\left(\frac{\gamma}{1-\varepsilon} \cdot p \cdot (n-1) \leq \frac{M}{N} \mid N > \sqrt{n}\right) \geq 1 - e^{-\left(1-\frac{\gamma}{1-\varepsilon}\right)^2 \cdot \frac{p(n-1)\cdot\sqrt{n}}{2}} =: \psi(n,p,\varepsilon,\gamma),$$

where M is the number of edges in $\mathcal{G}/\mathcal{U}$ and N the number of classes.

Corollary 1. *Let $\mathcal{G}$ be chosen randomly from $\mathcal{G}(n,p)$ with $0 < p < 1$. Furthermore let ε and γ be two constants with $0 < \gamma, \varepsilon$ and $\gamma + \varepsilon < 1$, then*

$$P\left(D_i^1\right) \geq \psi(n,p,\varepsilon,\gamma).$$

The Number of 'Good' Rounds. Let the random variables X_i with $1 \leq i \leq l$ be given by

$$X_i := \begin{cases} 0 & \text{if } N_i \geq \sqrt{n} \text{ and } \frac{\gamma \cdot p \cdot (n-1)}{1-\varepsilon} > \frac{M_i}{N_i} \\ 1 & \text{if } N_i < \sqrt{n} \text{ or } \frac{\gamma \cdot p \cdot (n-1)}{1-\varepsilon} \leq \frac{M_i}{N_i} \end{cases}$$

for two constants $0 < \varepsilon, \gamma$ with $\varepsilon + \gamma < 1$. Obviously $X_i = 1$ if and only if the event D_i^1 occurs, and hence

$$E(X_i) = P(X_i = 1) = P(D_i^1).$$

The number $X^{(l)}$ of the first l rounds in which D_i^1 is satisfied is the sum of the X_i, ie.

$$X^{(l)} := \sum_{i=1}^{l} X_i \quad \text{and hence} \quad E(X^{(l)}) = \sum_{i=1}^{l} E(X_i) = \sum_{i=1}^{l} P(D_i^1).$$

This immediately leads to the following lower bounds for $E(X)$.

Lemma 7. *Let $\mathcal{G}$ be chosen randomly from $\mathcal{G}(n,p)$ with $0 < p < 1$. Furthermore let ε and γ be two constants with $0 < \gamma, \varepsilon$ and $\gamma + \varepsilon < 1$. Then*

$$E(X^{(l)}) \geq l \cdot \psi(n,p,\varepsilon,\gamma).$$

If $\frac{\gamma}{1-\varepsilon} \leq 1 - \frac{1}{p \cdot (n-1) \cdot \sqrt{n}}$, then

$$E(X^{(l)}) \geq \frac{1}{2} \cdot l.$$

4.3 Asymptotic Bounds for the Running Time

In this section, we consider the asymptotic probability, as the number n of vertices increases. We assume that every parameter, i.e. the constants ε and γ, the probability p, and the number l_n of rounds, varies with n. As seen above (9),

$$P\left(T \leq \left(2 + \frac{1}{\varepsilon} + (l_n - k_n)\right) \cdot C \cdot (M_0 + n)\right) \geq \sum_{\mathbf{x} \in \Sigma^{l_n, k_n}} P\left(A \cap D(\mathbf{x})\right)$$

with $k_n = \left\lfloor -\frac{\frac{3}{2}\log(n)}{\log(1-\varepsilon)} \right\rfloor$ and Σ^{l_n,k_n} the set of all $\{0,1\}$-tuples of length l_n containing 1 at least k_n times. Since for $\mathbf{x} \in \Sigma^{l_n,k_n}$ the event union of all events $D(\mathbf{x})$ is equivalent to $X^{(l_n)} \geq k_n$, we get

$$P\left(T \leq \left(2 + \frac{1}{\varepsilon_n} + (l_n - k_n)\right) \cdot C \cdot (M_0 + n)\right) \geq P\left(A \cap (X^{(l_n)} \geq k_n)\right). \quad (18)$$

Now our aim is to describe a situation, under which the probability on the right side approximates 1, resulting in an almost always satisfied bound for the running time of Algorithm 1. (18) provides two hooks to achieve this. The first is the number of "bad" rounds, namely $l_n - k_n$. If this difference is not too high (ie. significantly below n), the runtime might be subquadratic. Secondly, the "constant" ε has to behave properly with growing n, ie. it may not be of the type $\frac{1}{n}$, because then our time bound will be quadratic again. In the best case, ε really is constant, ie. does not depend on n. Hence, our main task is the choice of $\varepsilon = \varepsilon_n$ and l_n, such that in addition $P(A \cap (X^{(l_n)} \geq k_n))$ converges to 1.

Using the theorem of de Moivre-Laplace, we can obtain the following result about the time required by Algorithm 1. The proof is omitted due to space restrictions.

Lemma 8. *Let $\mathcal{G} = (V, E)$ be chosen randomly from $\mathcal{G}(n, p_n)$ with $n \in \mathbb{N}$, $p_n = \frac{\omega_n}{n-1}$ and $l_n > k_n$. If $P(A) \to 1$ and $\lim_{n\to\infty}\left(l_n - E\left(X^{(l_n)}\right)\right) = 0$, then*

$$P\left(T \leq (3 + \omega_n + (l_n - k_n)) \cdot C \cdot (M_0 + n)\right) \to 1.$$

Combining the preceeding results, we obtain two theorems giving two time bounds with high probability.

Theorem 3. *Let $\mathcal{G} = (V, E)$ be chosen randomly from $\mathcal{G}(n, p)$ with $n \in \mathbb{N}$ and $p = \frac{\omega_n}{n-1}$. And let $C > 0$ be a constant, such that a round of Algorithm 1 requires at most $C \cdot (|V| + |E|)$ time.*

1. *If $\omega_n = \omega$ for a given constant $\omega \geq 2$, then*

$$P\left(T \leq (2 + \omega) \cdot C \cdot (M_0 + n)\right) \to 1.$$

2. *If $\omega_n = o(\sqrt{n}$ and $\omega_n \to \infty$, then*

$$P\left(T \leq (3 + \omega_n) \cdot C \cdot (|E| + n)\right) \to 1.$$

References

1. Bauer, S.: RNA Sekundärstrukturvorhersage. Studienarbeit, TU Ilmenau (April 2004)
2. Brinkmeier, M.: A simple and fast min-cut algorithm. Theory of Computing Systems 41, 369–380 (2007)

3. Chekuri, C., Goldberg, A.V., Karger, D.R., Levine, M.S., Stein, C.: Experimental study of minimum cut algorithms. In: Symposium on Discrete Algorithms, pp. 324–333 (1997)
4. Ford, L.R., Fulkerson, D.R.: Maximal flow through a network. Can. J. Math. 8, 399–404 (1956)
5. Gabow, H.N.: A matroid approach to finding edge connectivity and packing arborescences. J. Comput. Syst. Sci. 50(2), 259–273 (1995)
6. Goldberg, A.V., Tarjan, R.E.: A new approach to the maximum flow problem. J. Assoc. Comput. Mach. 35, 921–940 (1988)
7. Gomory, R.E., Hu, T.C.: Multi-terminal network flows. J. SIAM 9, 551–570 (1961)
8. Karger, D.R.: Minimum cuts in near-linear time. In: STOC, pp. 56–63 (1996)
9. Karger, D.R.: Minimum cuts in near-linear time. CoRR, cs.DS/9812007 (1998)
10. Karger, D.R., Stein, C.: A new approach to the minimum cut problem. J. ACM 43(4), 601–640 (1996)
11. Matula, D.W.: A linear time 2+epsilon approximation algorithm for edge connectivity. In: SODA, pp. 500–504 (1993)
12. Nagamochi, H., Ishii, Ibaraki, T.: A simple proof of a minimum cut algorithm and its applications. TIEICE: IEICE Transactions on Communications/Electronics/Information and Systems (1999)
13. Nagamochi, H., Ibaraki, T.: Computing edge-connectivity in multigraphs and capacitated graphs. SIAM J. Disc. Math. 5(1), 54–66 (1992)
14. Nagamochi, H., Ibaraki, T.: A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. Algorithmica 7(5&6), 583–596 (1992)
15. Nagamochi, H., Ibaraki, T.: Graph connectivity and its augmentation: applications of MA orderings. Discrete Applied Mathematics 123(1-3), 447–472 (2002)
16. Nagamochi, H., Nishimura, K., Ibaraki, T.: Computing all small cuts in undirected networks. In: Du, D.-Z., Zhang, X.-S. (eds.) ISAAC 1994. LNCS, vol. 834, pp. 190–198. Springer, Heidelberg (1994)
17. Nagamochi, H., Ono, T., Ibaraki, T.: Implementing an efficient minimum capacity cut algorithm. Math. Program. 67, 325–341 (1994)
18. Padberg, M., Rinaldi, G.: An efficient algorithm for the minimum capacity cut problem. Math. Program. 47, 19–36 (1990)
19. Stoer, M., Wagner, F.: A simple min-cut algorithm. Journal of the ACM 44(4), 585–591 (1997)

The Generalized Stable Allocation Problem

Brian C. Dean and Namrata Swar

School of Computing, Clemson University
Clemson, SC, USA
{bcdean,nswar}@cs.clemson.edu

Abstract. We introduce and study the generalized stable allocation problem, an "ordinal" variant of the well-studied generalized assignment problem in which we seek a fractional assignment that is stable (in the same sense as in the classical stable marriage problem) with respect to a two-sided set of ranked preference lists. We develop an $O(m \log n)$ algorithm for solving this problem in a bipartite assignment graph with n nodes and m edges. When edge costs are present, we show that it is NP-hard to compute a stable assignment of minimum cost, a result that stands in stark contrast with most other stable matching problems (e.g., the stable marriage and stable allocation problems) for which we can efficiently optimize over the set of all stable assignments.

1 Introduction

The generalized assignment problem (GAP) is a well-studied classical optimization problem that commonly appears in areas of application such as scheduling and load balancing. Using scheduling notation for convenience, consider a set of I jobs indexed by $[I] := \{1, \ldots, n\}$ and J machines indexed by $[J] = \{1, \ldots, J\}$, and let us form a bipartite assignment graph $G = ([I] \cup [J], E)$ with $n = I + J$ nodes and $m = |E|$ edges, where each edge $(i, j) \in E$ has an associated cost $c(i, j) \geq 0$, capacity $u(i, j) > 0$, and multiplier $\mu(i, j) > 0$. We can then state the GAP as the following linear program:

$$\begin{aligned} \text{Minimize} \quad & \sum_{(i,j) \in E} c(i,j)x(i,j) & \\ \text{Subject to} \quad & \sum_{j \in [J]} x(i,j) = 1 & \forall i \in [I] \\ & \sum_{i \in [I]} x(i,j)\mu(i,j) \leq 1 & \forall j \in [J] \\ & 0 \leq x(i,j) \leq u(i,j) & \forall (i,j) \in E. \end{aligned}$$

The multiplier $\mu(i, j)$ tells us that one unit of job i uses up $\mu(i, j)$ units of capacity when assigned to machine j. It is these multipliers that differentiate a *generalized* assignment problem from a traditional linear assignment problem, and these give us quite a bit of flexibility in modeling a variety of different scenarios. Multipliers allow us to convert between different units of measurement or currency, and *lossy* multipliers (less than one) allow us to account for spoilage of goods or other forms of leakage or loss during transportation. In the scheduling context above, we can

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 238–249, 2009.

scale our multipliers appropriately to assign non-uniform processing times to jobs and non-uniform capacities to machines.

In this paper, we introduce and study the *generalized stable allocation problem* (GSAP), an ordinal analog of the GAP where we express the quality of an assignment in terms of ranked preference lists submitted by the jobs and machines, rather than by numeric costs.

1.1 Previous Work

The fundamental underlying problem in our domain of interest is the classical bipartite stable matching (or stable marriage) problem, where we seek to match n men with n women, and each (man, woman) submits a ranked preference list over all of the (women, men). Here, we seek a matching M between the men and women that is *stable*, in the sense that it there is no man-woman pair $(m, w) \notin M$ who both prefer each-other to their partners in M. Gale and Shapley showed in 1962 how to solve any instance of the stable matching problem with a simple $O(m)$ algorithm [6]. Building on this work, a series of "high multiplicity" variants of the stable matching problem have been considered in the past few decades:

- Roth [9] studied the many-to-one bipartite *stable admission* problem, where we wish to match unit-sized entities with non-unit-sized entities. The most prominent application of this problem is the *National Residency Matching Program* (NRMP), a centralized program in the USA used to assign medical school graduates to hospitals, where each hospital has a quota governing the number of graduates it can accept.
- Baiou and Balinski [1] introduced what we call the bipartite *stable b-matching problem*, where we are matching non-unit-sized elements with non-unit-sized elements, and each element i in our biparite assignment graph has a specified quota $b(i)$ governing the number of elements on the other side of the graph to which it should be matched.
- A more recent paper of Baiou and Balinski [2] introduced the bipartite *stable allocation problem*, which is similar to the bipartite stable b-matching problem except that the quotas $b(i)$ can now be arbitrary real numbers. This problem is also known as the *ordinal transportation problem*, since it is a direct analog of the classical cost-based transportation problem. The generalized stable allocation problem is an extension of the stable allocation problem to allow for edge multipliers.

In terms of algorithmic complexity, the stable admissions and stable b-matching problems are not much different from the simpler stable matching problem; they all can be solved using the Gale-Shapley algorithm in $O(m)$ time by first expanding each element i of non-unit size $b(i)$ into $b(i)$ unit elements, each with the same preference list as i. However, the stable allocation problem is somewhat harder: the Gale-Shapley algorithm solves the problem but with exponential worst-case time, an algorithm of Baiou and Balinski [2] runs in $O(mn)$ time, and a more recent algorithm of Dean and Munshi [5] unifies these techniques and employs sophisticated data structures to achieve an $O(m \log n)$ running time.

1.2 Our Results

In this paper we discuss the algorithmic ramifications of adding edge multipliers to the stable allocation problem. Our first result is a positive one: we show that the Dean-Munshi algorithm can be appropriately enhanced to solve the generalized stable allocation problem while maintaining its fast $O(m \log n)$ running time. Our second result, however, is negative: we show that the if edges are also given associated costs, then the problem of finding a generalized stable assignment of minimum cost is NP-hard. This highlights a strong distinction between the generalized and non-generalized stable allocation problems, since with the non-generalized stable allocation problem, it is possible to optimize over the set of all stable assignments in an efficient fashion.

2 Preliminaries

Let $N(i)$ denote the set of machines adjacent to job i in our bipartite assignment graph, and let $N(j)$ be all the jobs adjacent to machine j. Each job i submits a ranked preference list over machines in $N(i)$ and each machine j submits a ranked preference list over jobs in $N(j)$. If job i prefers machine $j \in N(i)$ to machine $j' \in N(i)$, then we write $j >_i j'$. If machine j prefers job i to job i' in $N(j)$, then we similarly write $i >_j i'$. Preference lists are strict, containing no ties. As with the GAP, each edge $(i, j) \in E$ has an associated multiplier $\mu(i, j) > 0$ and an upper capacity $u(i, j) > 0$. Later on, when we consider the "optimal" variant of the GSAP, we will also associate a cost $c(i, j)$ with each edge. Let $x(i, j)$ denote the amount of assignment from job i to machine j, and let $x \in \mathbf{R}^m$ denote the vector representing an entire assignment. We use set notation in the standard way to describe aggregate quantities; for example, $x(i, N(i)) = \sum_{j \in N(i)} x(i, j)$.

In order for an assignment x to be feasible, every job i must be fully assigned (i.e., $x(i, N(i)) = 1$). It is typically not necessary for every machine to be fully assigned. However, it will simplify our model considerably to make this assumption as well, which we can do by introducing a "dummy" job (which we assign index 1), such that the dummy assignment $x(1, j)$ to machine j indicates the amount of machine j that is in reality unassigned. We set $\mu(1, j) = J$ and $u(1, j) = 1/J$ for each $j \in [J]$, so that the dummy job can simultaneously provide any amount of "slack" assignment required at each machine j. To balance out the dummy job, we also introduce a dummy machine (also assigned index 1). The dummy job and machine appear last in the preference lists of the remaining jobs and machines, including the dummies themselves. The remaining entries in the preference lists of the dummy job and machine are arbitrarily ordered.

We now say that an assignment x is *feasible* if

$$\begin{aligned} \sum_{j \in [J]} x(i, j) &= 1 && \forall i \in [I] \\ \sum_{i \in [I]} x(i, j)\mu(i, j) &= 1 && \forall j \in [J] - \{1\} \\ 0 \leq x(i, j) &\leq u(i, j) && \forall (i, j) \in E. \end{aligned}$$

Note that the dummy machine is the only machine whose incoming assignment is unconstrained; all other machines are constrained to be fully assigned in any feasible assignment.

An assignment x is *stable* if it is feasible and contains no *blocking pair*, where a blocking pair is a pair $(i, j) \in E$ such that $x(i, j) < u(i, j)$, there exists a machine $j' < j$ for which $x(i, j') > 0$, and there exists a job $i' < i$ for which $x(i', j) > 0$. Intuitively, (i, j) is a blocking pair if i and j would both be happier switching some of their less-preferred assignments to the edge (i, j), and if there is room to increase $x(i, j)$. Note that traditionally, stable matching problems are defined with complete preference lists, with $E = [I] \times [J]$. In our case, we can conceive of extending every preference list so it is complete — for example, by appending the entries in $[J] - N(i)$ to the end of job i's preference list in arbitrary order. However, we note that this is unnecessary, since any assignment that utilizes one of these additional edges will be unstable due to a blocking pair formed with the dummy machine. Hence, we can think of our instance has having complete preference lists, but for all practical purposes we can ignore the extra edges not in E.

An assignment x is *job-optimal* if, for every job i, the vector describing i's assignments (ordered by i's preference list) is lexicographically maximal over all stable assignments. As we will see shortly, a stable, job-optimal assignment exists for every instance of the GSAP.

3 Algorithmic Results

In this section, we develop our $O(m \log n)$ algorithm for the GSAP. We begin with the much simpler Gale-Shapley (GS) algorithm, which solves the problem with worst-case exponential time, and whose analysis provides many of the underlying theoretical properties we need for our later analysis. We then describe the Dean-Munshi (DM) algorithm for the stable allocation problem, on which our approach is based.

3.1 The Gale-Shapley Algorithm

To generalize the classical Gale-Shapley (GS) algorithm so it solves the GSAP, we introduce a small amount of additional notation. Fixing an assignment x, we define r_j to be the job $i \in N(j)$ with $x(i, j) > 0$ that is least-preferred by j. We refer to r_j as the "rejection pointer" of j, since it points to the job that j would prefer to reject first, given a choice amongst its current assignments. We also define the "proposal pointer" of job i, q_i[1], to be the machine j most preferred by i such that $x(i, j) < u(i, j)$ and $i >_j r_j$. Informally, q_i points to the first machine on i's preference list that would be willing to receive additional assignment from i.

[1] We use q_i rather than p_i for i's proposal pointer to maintain consistency with the notation in [5].

The GS algorithm begins with an empty assignment. In our case, we start with all machines being assigned to the dummy job: $x(1, j) = 1$ for all $j \in [J]$. The algorithm will terminate when all jobs are fully assigned: $x(i, N(i)) = 1$ for all $i \in [I]$. In each iteration, we select a job i that is not fully assigned and have it propose all $b = 1 - x(i, N(i))$ units of unassigned load to machine $j = q_i$. Machine j accepts, thereby sending it over capacity, after which it proceeds to reject load from machine r_j until j's incoming assignment once again satisfies $\sum_i x(i, j)\mu(i, j) = 1$. Note that the rejection pointer r_j can move up j's preference list during this process, as edges become emptied out due to rejection, and that this in turn can cause proposal pointers to advance. More precisely, whenever $x(r_j, j)$ drops to zero due to rejection, we advance r_j up j's preference list until $x(r_j, j) > 0$. Any time a proposal pointer q_i becomes invalid due to either $x(i, j)$ reaching $u(i, j)$ or $i \leq_j r_{q_i}$, we advance q_i down i's preference list until it becomes valid. In total, since the proposal and rejection pointers move in a monotonic fashion, their maintenance requires only $O(m)$ time over the entire algorithm (note that this is exactly the same pointer management as used with the DM algorithm). Unfortunately, the GS algorithm can take exponential time in the worst case, as originally shown in [2].

Just as the GS algorithm itself generalizes readily to solve the GSAP, most of its well-known properties also generalize in a straightforward fashion. Proofs of the following lemmas are omitted in the interest of brevity, as they are straightforward to obtain by generalizing the corresponding results from the stable allocation problem (see [3,5]).

Lemma 1. *Irrespective of proposal order, the GS algorithm for the GSAP always terminates in finite time (even with irrational problem data), and it does so with a stable job-optimal assignment.*

Lemma 2. *For each edge (i, j), as the GS algorithm executes, $x(i, j)$ will never increase again after it experiences a decrease.*

Corollary 1. *During the execution of the GS algorithm, each edge (i, j) becomes saturated at most once, and it also becomes empty at most once.*

We also note that the "integral" variant of the GS algorithm developed in [4] for the *unsplittable* variant of the stable allocation problem also generalizes in a straightforward manner to the generalized case with edge multipliers.

3.2 The (Generalized) Dean-Munshi Algorithm

Baiou and Balinski were the first to propose a polynomial-time ($O(mn)$ time) algorithm for the stable allocation problem; one can think of their algorithm as an "end-to-end" variant of the GS algorithm, which makes a series of proposals and rejections along "augmenting paths". The Dean-Munshi (DM) algorithm is an enhancement of this approach that uses additional structural insight coupled with dynamic tree data structures to improve the running time for each augmentation from $O(n)$ to $O(\log n)$, thereby resulting in an overall running time

of $O(m \log n)$. We now describe how this algorithm should be modified in the context of the GSAP; the reader is referred to [5] for a more detailed discussion of the DM algorithm.

For any assignment x, we define $G(x)$ to be a bipartite graph on the same set of nodes as our original instance, having "proposal" edges (i, q_i) for all $i \in [I]$ and "rejection" edges (r_j, j) for all $j \in [J] - \{1\}$ (note that the dummy machine is the only node that lacks a proposal or rejection pointer). The graph $G(x)$ has a relatively simple structure: each of its connected components is either a tree or a tree plus one edge, forming a single cycle with paths branching off it. Moreover, there is only one "tree" component — the one containing the dummy machine; all remaining components are "cycle" components.

The DM algorithm maintains the structure of each component in $G(x)$, along with a list of the jobs in each component that are not yet fully assigned. In each iteration, a component C with some unassigned job i is selected, and an augmentation is performed within C. If C is the tree component, then the augmentation is straightforward: job i proposes ε units to machine $j = q_i$ (which arrive as $\varepsilon\mu(i,j)$ units at machine j), then machine j accordingly rejects $\varepsilon\mu(i,j)$ units from job $i' = r_j$ (which arrive as $\varepsilon\mu(i,j)/\mu(i',j)$ units at i'), after which i' proposes to machine $j' = q_{i'}$, and so on, until some job eventually proposes to the dummy machine, which accepts and does not issue a subsequent rejection. The quantities being proposed and rejected along this path vary in accordance with the initial amount being proposed, ε, and the aggregate product of the multipliers along the path (treating the multiplier of a rejection edge as the reciprocal of its forward multiplier). By traversing the path and accumulating this multiplier product, we can easily compute the maximum amount ε that job i can initially propose so that either a proposal edge along the path becomes saturated, some rejection edge becomes empty, or i becomes fully assigned. To be somewhat more precise, suppose C is the tree component of $G(x)$, and let $\pi(s) = \pi^+(s) \cup \pi^-(s)$ denote the unique augmenting path through C from source job s to machine 1, where $\pi^+(s)$ is the set of proposal edges along the path and $\pi^-(s)$ is the set of rejection edges. Let us define the *potential* of job s with respect to machine 1 as

$$\alpha(s) = \frac{\prod_{(i,j)\in\pi^+(s)} \mu(i,j)}{\prod_{(i,j)\in\pi^-(s)} \mu(i,j)}.$$

For any source machine $t \neq 1$ in C, we similarly define its potential as $\beta(t) = \alpha(r_t)/\mu(r_t, t)$, and we define $\beta(1) = 1$. Note that if we initially propose ε units from job $s \in C$, then $\varepsilon\alpha(s)$ units will eventually reach machine 1, assuming no edge on $\pi(i)$ becomes saturated or empty in the process. Similarly, a rejection of ε units from machine $t \in C$ will eventually arrive at machine 1 as $\varepsilon\beta(t)$ units.

Consider now some edge (i,j) along the augmenting path $\pi(s)$ in C, and suppose we initially propose $\varepsilon/\alpha(s)$ units from job s. If (i,j) is a proposal edge, then $\varepsilon/\alpha(i)$ units arrive at job i and are proposed along (i,j), and if (i,j) is a rejection edge, then $\varepsilon/\beta(j)$ units arrive at j and are then rejected along (i,j), so $\varepsilon/\alpha(i)$ rejected units arrive at i. We define the *scaled* assignments and capacities

of edge (i, j) as $x'(i, j) = x(i, j)\alpha(i)$ and $u'(i, j) = u(i, j)\alpha(i)$, and we define the *scaled residual capacity* of a proposal edge (i, q_i) in C as $r(i, q_i) = u'(i, q_i) - x'(i, q_i)$, and of a rejection edge (r_j, j) in C as $r(r_j, j) = x'(r_j, j)$. Finally, we define the residual capacity of the path $\pi(s)$ as $r(\pi(s)) = \min\{r(i, j) : (i, j) \in \pi(s)\}/\alpha(s)$. The quantity $r(\pi(s))$ tells us exactly how large we can set ε so that some edge along $\pi(s)$ becomes saturated or empty. We therefore augment $\min(1-x(s, N(s)), r(\pi(s)))$ units from s, since this is sufficient to either saturate or empty an edge, or to make s fully assigned. We call such an augmentation within the tree component a type I augmentation.

If our algorithm chooses to augment within a cycle component C, the details are slightly more involved. Let π_C denote the unique cycle in C, and let $\mu(\pi_C)$ denote the product of the multipliers on the proposal edges in π_C divided by the product of the multipliers on the rejection edges in π_C. As with other generalized assignment problems, we can classify π_C as being *gainy* if $\mu(\pi_C) > 1$, *lossy* if $\mu(\pi_C) < 1$ or *break-even* if $\mu(\pi_C) = 1$. Suppose some job $i \in \pi_C$ is not fully assigned, and that we augment around π_C by initially proposing ε units from job i, so $\varepsilon\mu(\pi_C)$ rejected units eventually arrive back at job i after propagating the augmentation all the way around π_C. If π_C is gainy, then job i will end up seeing more than ε units returned to it, so augmenting from job i around a gainy cycle results in a net *decrease* in i's total assignment. Similarly, augmenting around a lossy cycle results in a net *increase* in i's assignment. In either case, we augment, as before, until some edge along π_C becomes either saturated or empty, or until job i is fully assigned. We call this a type II augmentation.

The final case to consider is the type III augmentation — when we augment a cycle component C, but where all jobs along π_C are fully assigned, so in this case we are starting our augmentation from a source job $s \notin \pi_C$. Let s' be the first job in π_C we reach when we follow an augmenting path from s. Here, we augment along the path only from s to s', again proposing the maximum possible amount from s so that some edge on our path becomes either saturated or empty, or so that s becomes fully assigned. As a result of this augmentation, s' will now no longer be fully assigned, so this triggers a subsequent type II augmentation within π_C.

It is easy to justify termination and correctness of our algorithm. Since it is performing a series of proposals and rejections (in a batch fashion) that the GS algorithm could have performed, Lemma 1 asserts that we must terminate with a stable job-optimal assignment. Running time analysis is fairly straightforward. Without using any sophisticated machinery, we can perform each augmentation in $O(n)$ time (we show how to reduce this to $O(\log n)$ time in a moment). To bound the number of augmentations, note that each one either saturates an edge, empties out an edge, or fully assigns some job. Corollary 1 tells us that each edge can be saturated or emptied out at most once, so there are at most $O(m)$ augmentations that saturate or empty an edge. We must take slightly more care with the augmentations that end up fully assigning a job. For the simpler stable allocation problem, the DM algorithm ensures that a job, once fully assigned, remains fully assigned. In the generalized case, however, this is

no longer necessarily true. Note that we never perform type-II augmentations in a cycle whose jobs are all fully-assigned, so a type-II augmentation can never cause a fully-assigned job to become not fully assigned (and of course, a type-I augmentation cannot cause this to happen either). Only a type III augmentation can result in a fully-assigned job becoming not fully assigned. However, every type III augmentation that causes some job i to become not fully assigned is followed by a subsequent type II augmentation that either saturates/empties an edge or fully assigns job i. Hence a type III followed by type II augmentation either saturates/empties an edge, or results in a net increase in the number of fully-assigned jobs. We therefore have $O(m + n)$ total augmentations.

3.3 Fast Augmentation with Dynamic Trees

In order to augment in $O(\log n)$ time, we use dynamic trees [10,11]. Each component of $G(x)$ is stored in a single dynamic tree, and each cycle component is stored as a dynamic tree plus one additional edge, arbitrarily chosen. A dynamic tree data structure maintains an edge-weighted and/or node-weighted free tree and supports a host of useful operations all in $O(\log n)$ amortized time. The main operations we need are the following:

- *Split/join:* Split a tree into two trees by deleting an edge, or join two trees by adding an edge.
- *Global-update:* Modify the weight on every edge or node in a tree by a given additive or multiplicative factor.
- *Path-min:* Find the minimum edge or node weight along the unique tree path joining two specified nodes i and j.
- *Path-update:* Update the edge or node weights along the unique tree path from i to j by a given additive offset.

For simplicity, we describe augmentation in the context of the tree component. The same general approach also works for cycle components, except there we must also account for the extra edge separately (i.e., augmentation on a cycle is equivalent to augmentation on a path through a tree, plus augmentation on a single edge).

Along with every edge (i, j) in our dynamic trees we store two weights: the scaled assignment $x'(i, j)$ and the scaled residual capacity $r(i, j)$. Every job node i in one of our dynamic trees is weighted with its potential $\alpha(i)$, and each machine node j is weighted with its potential $\beta(j)$. For the tree component, recall that these potentials are defined relative to the dummy machine. For a cycle component, we define potentials relative to an arbitrarily-chosen "reference" node within the component. Note that we can change the reference node in only $O(\log n)$ time; for example, we can change from reference node s to reference node t by applying a *global-update* to multiply all node potentials and divide all edge weights in the component by the ratio of the potential of t to that of s.

Augmentation from some source job s uses the *path-min* operation to compute $r(\pi(s)) = \min\{r(i, j) : (i, j) \in \pi(i)\}\alpha(s)$, followed by the *path-update* operation to decrease the residual capacity of every edge in $\pi(s)$ by $\varepsilon = \min$

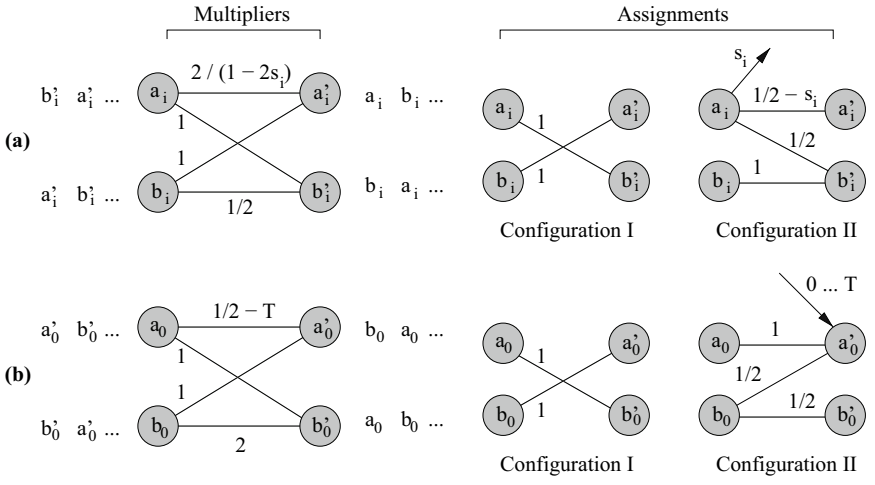


Fig. 1. Illustrations of (a) a "gainy" component C_i (left, with edge multipliers indicated) along with its two stable configurations (right, with edge assignments indicated), and (b) the "lossy" component C_0 along with its two stable configurations. In configuration II, the external edge incoming to a'_0 can carry any amount of assignment in the range $0 \ldots T$.

$(1 - x(s, N(s)), r(\pi(s)))/\alpha(s)$, and to adjust the scaled assignment $x'(i,j)$ of every edge $(i,j) \in \pi(s)$ by ε. Note that proposal edges and rejection edges must be treated slightly differently during the path update, since $x'(i,j)$ must increase by ε for a proposal edge and decrease by ε for a rejection edge; however, this is easy to accommodate by introducing some relatively standard extra machinery into the dynamic tree infrastructure. The fact that we use scaled residual capacity, where the weight of each edge (i,j) is scaled relative to the potential $\alpha(i)$, is absolutely crucial, since this "normalizes" all of the edge weights along a path so they are effectively in the same unit of measurement (relative to an endpoint of the path), allowing us to apply the *path-min* and *path-update* operations in a uniform fashion across a path.

Augmentations often cause some subset of edges to become saturated or empty. These edges leave $G(x)$, while other edges (due to advancement of proposal or rejection pointers) may enter, thereby changing the structure of the connected components of $G(x)$. Each leaving edge results in a *split* operation, and each entering edge result in a *join*. Since each edge leaves or enters at most a constant number of times, this accounts for at most $O(m \log n)$ work throughout the entire algorithm. Finally, whenever we perform a *split* or *join*, we must call *global-update* to update node potentials appropriately; for example, when a component with reference node s is split into two components by removing edge (i,j), we need to re-weight the potentials in the newly-created component not containing s relative to an arbitrarily-chosen reference node in that component.

4 The Optimal Generalized Stable Allocation Problem

If the edges in our bipartite assignment instance have associated costs, we can consider the *optimal* generalized stable assignment problem: among all stable assignments, find one having minimum total cost. In this section, we show that this problem is NP-hard. This result highlights a major distinction between the generalized stable assignment problem and its simpler variants, since the "optimal" variants of the both the stable matching and stable allocation problems can be solved in polynomial time [8,5]. For both of these simpler variants, there exists a convenient combinatorial description of the set of all stable assignments in terms of closed subsets of a carefully crafted "rotation DAG" (and for the stable matching problem, one can also write a linear program whose extreme points correspond to stable solutions). However, since the optimal generalized stable assignment problem is NP-hard, this strongly suggests that there is no convenient mathematical characterization of the set of all stable solutions.

Theorem 1. *The optimal generalized stable allocation problem is NP-hard.*

The proof of this theorem uses a reduction from the well-known NP-complete SUBSET-SUM problem [7], which takes as input a set of positive numbers $s_1 \ldots s_n$ and a target T, and asks whether or not there exists a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} s_i = T$. Given an instance I of SUBSET-SUM, we show how to construct an instance I' of the optimal generalized stable allocation problem such that I has a YES answer if and only if I' has a stable solution of cost at most $\sum_i s_i - T$. Without loss of generality, we assume by rescaling that $s_1 \ldots s_n$ and T all lie in the range $(0, 1/2)$.

To build I', we construct a bipartite assignment graph with $2n + 2$ elements on each side, comprised of n "gainy" components $C_1 \ldots C_n$ and one "lossy" component C_0. Each gainy components C_i is constructed as shown in Figure 1(a), with two jobs a_i and b_i and two machines a'_i and b'_i. Preference lists are as shown in the figure, with elements external to C_i placed last on each preference list in arbitrary order. Edge multipliers are given by $\mu(a_i, a'_i) = 2/(1 - 2s_i)$, $\mu(b_i, b'_i) = 1/2$, and $\mu(a_i, b'_i) = \mu(b_i, a'_i) = 1$. The lossy component is similarly constructed, as shown in Figure 1(b). All remaining edges crossing between components are assigned unit multipliers. For each gainy component $i = 1 \ldots n$, we assign the cost of edge (b_i, a'_i) to s_i. All other edges have zero cost. For simplicity, we do not explicitly include the dummy job and machine in our construction, since these were introduced primarily to simplify our algorithms; rather, we simply allow each machine to be only partially assigned (which is equivalent to its assignment to a dummy job).

For the gainy component C_i, we define two specific assignment *configurations*, also shown in Figure 1(a). Configuration I has $x(a_i, a'_i) = x(b_i, b'_i) = 0$ and $x(a_i, b'_i) = x(b_i, a'_i) = 1$, and configuration II has $x(a_i, a'_i) = 1/2 - s_i$, $x(a_i, b'_i) = 1/2$, $x(b_i, b'_i) = 1$, with a_i having s_i units assigned to some machine external to C_i. We similarly define two assignment configurations for the lossy component C_0, shown in Figure 1(b).

Lemma 3. *For each gainy component C_i, any assignment pattern involving the edges incident to C_i other than configuration I and II is unstable.*

Proof. Let us say that an element is *externally assigned* if it has some positive allocation to an element outside C_i; otherwise, we say it is *internally assigned*. We note that job b_i and machines a'_i and b'_i cannot be externally assigned in any stable assignment: if job b_i were externally assigned, then (b_i, b'_i) would be a blocking pair, if machine a'_i were externally assigned, (b_i, a'_i) would be a blocking pair, and if machine b'_i were externally assigned, then (a_i, b'_i) would be a blocking pair. Both a_i and b_i must be fully assigned, since this is a requirement of any feasible assignment. Although a'_i and b'_i are not explicitly required to be fully assigned, this must still be the case in any stable solution: if a'_i is not fully assigned, then (b_i, a'_i) is a blocking pair, and if b'_i is not fully assigned, then (a_i, b'_i) is a blocking pair.

Now consider the following two cases: (i) if job a_i is internally assigned, then all of the elements in C_i are assigned internally, and it is easy to show that as a consequence of our multipliers (and the fact that every element is fully assigned), the only possible stable feasible assignment is configuration I. On the other hand (ii) if job a_i is externally assigned, then (a_i, a'_i) will be a blocking pair unless a'_i is assigned only to a_i, so $x(a_i, a'_i) = 1/2 - s_i$. Moreover, since b_i can now only be assigned to b'_i, we have $x(b_i, b'_i) = 1$, and since b'_i needs exactly one unit of incoming assignment we must have $x(a_i, b'_i) = 1/2$. Therefore, a_i must have s_i of its units assigned externally in order for it to have exactly one unit of outgoing assignment. This is configuration II.

Lemma 4. *For the lossy component C_0, any assignment pattern involving the edges incident to C_0 other than configuration I and II shown in Figure 1(b) is unstable.*

Proof. The argument in this case is quite similar to the preceding proof. We begin by arguing that in any stable assignment, jobs a_0 and b_0 and machine b'_0 must be internally assigned: if job a_0 is externally assigned, then (a_0, b'_0) is a blocking pair, if job b_0 is externally assigned, then (b_0, a'_0) is a blocking pair, and if machine b'_0 is externally assigned, then (b_0, b'_0) is a blocking pair. Moreover, a_0 and b_0 are explicitly constrained to be fully assigned, and b'_0 must be fully assigned or else (b_0, b'_0) is a blocking pair. We now argue, just as before, that if machine a'_0 is internally assigned, then we must be in configuration I, otherwise we must be in configuration II.

Lemma 5. *Given an instance I of SUBSET-SUM, the corresponding instance I' of the optimal generalized stable allocation instance I' has a stable allocation of cost at most $\sum_i s_i - T$ if and only if I has a YES answer.*

Proof. If I admits a set S such that $\sum_{i \in S} s_i = T$, then we can obtain a stable solution for I' by placing C_0 and each C_i for $i \in S$ in configuration II, and the remaining components in configuration I. For each $i \in S$, the external assignment of s_i units from job a_i is directed to machine a'_0, thereby exactly matching the

T units of external assignment needed at a'_0. The total cost of this solution is $\sum_i s_i - T$. On the other hand, if I' has a stable solution of cost at most $\sum_i s_i - T$, then we must have in aggregate at least T units of external assignment from $a_1 \ldots a_n$, and a'_0 can accept at most T units. Hence, we must have exactly T units of external assignment, and these T units must come from a subset of gainy components (in configuration II) corresponding to a solution for the SUBSET-SUM instance I.

References

1. Baiou, M., Balinski, M.: Many-to-many matching: Stable polyandrous polygamy (or polygamous polyandry). Discrete Applied Mathematics 101, 1–12 (2000)
2. Baiou, M., Balinski, M.: Erratum: The stable allocation (or ordinal transportation) problem. Mathematics of Operations Research 27(4), 662–680 (2002)
3. Dean, B.C., Immorlica, N., Goemans, M.X.: Finite termination of "augmenting path" algorithms in the presence of irrational problem data. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 268–279. Springer, Heidelberg (2006)
4. Dean, B.C., Immorlica, N., Goemans, M.X.: The unsplittable stable marriage problem. In: Proceedings of the 4th IFIP International Conference on Theoretical Computer Science (2006)
5. Dean, B.C., Munshi, S.: Faster algorithms for stable allocation problems. In: Proceedings of the 1st MATCH-UP (Matching Under Preferences) Workshop, pp. 133–144 (2008)
6. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. American Mathematical Monthly 69(1), 9–14 (1962)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completenes. W.H. Freeman, New York (1979)
8. Irving, R.W., Leather, P., Gusfield, D.: An efficient algorithm for the "optimal" stable marriage. Journal of the ACM 34(3), 532–543 (1987)
9. Roth, A.E.: The evolution of the labor market for medical interns and residents: a case study in game theory. Journal of Political Economy 92, 991–1016 (1984)
10. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. Journal of Computer and System Sciences 26(3), 362–391 (1983)
11. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. Journal of the ACM 32(3), 652–686 (1985)

Crossing-Optimal Acyclic Hamiltonian Path Completion and Its Application to Upward Topological Book Embeddings

Tamara Mchedlidze and Antonios Symvonis

Dept. of Mathematics, National Technical University of Athens, Athens, Greece
{mchet,symvonis}@math.ntua.gr

Abstract. Given an embedded planar acyclic digraph G, we define the problem of *acyclic hamiltonian path completion with crossing minimization (Acyclic-HPCCM)* to be the problem of determining a *hamiltonian path completion set* of edges such that, when these edges are embedded on G, they create the smallest possible number of edge crossings and turn G to an acyclic hamiltonian digraph. Our results include:

1. We provide a characterization under which a triangulated st-digraph G is hamiltonian.
2. For the class of planar st-digraphs, we establish an equivalence between the Acyclic-HPCCM problem and the problem of determining an upward 2-page topological book embedding with minimum number of spine crossings. Based on this equivalence we infer for the class of outerplanar triangulated st-digraphs an upward topological 2-page book embedding with minimum number of spine crossings and at most one spine crossing per edge.

1 Introduction

In the *hamiltonian path completion problem* (for short, *HP-completion*) we are given a graph G (directed or undirected) and we are asked to identify a set of edges (refereed to as an *HP-completion set*) such that, when these edges are embedded on G they turn it to a hamiltonian graph, that is, a graph containing a hamiltonian path[1]. The resulting hamiltonian graph G' is referred to as the *HP-completed graph* of G. When we treat the HP-completion problem as an optimization problem, we are interested in an HP-completion set of minimum size.

When the input graph G is a planar embedded digraph, an HP-completion set for G must be naturally extended to include an embedding of its edges on the plane, yielding to an embedded HP-completed digraph G'. In general, G' is not planar, and thus, it is natural to attempt to minimize the number of edge crossings of the embedding of the HP-completed digraph G' instead of the size

[1] In the literature, a *hamiltonian graph* is traditionally referred to as a graph containing a hamiltonian cycle. In this paper, we refer to a hamiltonian graph as a graph containing a hamiltonian path.

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 250–261, 2009.

of the HP-completion set. We refer to this problem as the *HP-completion with crossing minimization problem* (for short, *HPCCM*). The HPCCM problem can be further refined by placing restrictions on the maximum number of permitted crossings per edge of G.

When the input digraph G is acyclic, we can insist on HP-completion sets which leave the HP-completed digraph G' also acyclic. We refer to this version of the problem as the *acyclic HP-completion problem.*

A *k-page book* is a structure consisting of a line, referred to as *spine*, and of k half-planes, referred to as *pages*, that have the spine as their common boundary. A *book embedding* of a graph G is a drawing of G on a book such that the vertices are aligned along the spine, each edge is entirely drawn on a single page, and edges do not cross each other. If we are interested only in two-dimensional structures we have to concentrate on 2-page book embeddings and to allow spine crossings. These embeddings are also referred to as 2-page *topological* book embeddings.

For acyclic digraphs, an upward book embedding can be considered to be a book embedding in which the spine is vertical and all edges are drawn monotonically increasing in the upward direction. As a consequence, in an upward book embedding of an acyclic digraph the vertices appear along the spine in topological order.

The results on topological book embedding that appear in the literature focus on the number of spine crossings per edge required to book-embed a graph on a 2-page book. However, approaching the topological book embedding problem as an optimization problem, it makes sense to also try to minimize the number of spine crossings.

In this paper, we introduce the problem of *acyclic hamiltonian path completion with crossing minimization* (for short, *Acyclic-HPCCM*) for planar embedded acyclic digraphs. To the best of our knowledge, this is the first time that edge-crossing minimization is studied in conjunction with the acyclic HP-completion problem. Then, we provide a characterization under which a triangulated st-digraph is hamiltonian. For the class of planar st-digraphs, we establish an equivalence between the acyclic-HPCCM problem and the problem of determining an upward 2-page topological book embedding with a minimal number of spine crossings. In [21] we describe a linear-time algorithm that solves the Acyclic-HPCCM problem for the class of triangulated outerplanar st-digrpahs with at most one crossing per edge of G. Based on the equivalence and this algorithm, we infer for the class of triangulated outerplanar st-digraphs an upward topological 2-page book embedding with minimum number of spine crossings and at most one spine crossing per edge.

1.1 Problem Definition

Let $G = (V, E)$ be a graph. Throughout the paper, when we use the term *"graph"* we refer to both directed and undirected graphs. We use the term "digraph" when we want to restict our attention to directed graphs. We assume familiarity with basic graph theory [14]. A *hamiltonian path* of G is a path that visits every

vertex of G exactly once. Determining whether a graph has a hamiltonian path or circuit is NP-complete [12]. The problem remains NP-complete for cubic planar graphs [12], for maximal planar graphs [31] and for planar digraphs [12]. It can be trivially solved in polynomial time for acyclic digraphs.

Given a graph $G = (V, E)$, a non-negative integer $k \leq |V|$ and two vertices $s,\ t \in V$, *the hamiltonian path completion (HPC)* problem asks whether there exists a superset E' containing E such that $|E' - E| \leq k$ and the graph $G' = (V, E')$ has a hamiltonian path from vertex s to vertex t. We refer to G' and to the set of edges $|E' - E|$ as the *HP-completed graph* and the *HP-completion set* of graph G, respectively. We assume that all edges of a HP-completion set are part of the Hamiltonian path of G', otherwise they can be removed. When G is a directed acyclic graph, we can insist on HP-completion sets which leave the HP-completed digraph also acyclic. We refer to this version of the problem as the *acyclic HP-completion problem*. The hamiltonian path completion problem is NP-complete [11]. For acyclic digraphs the HPC problem is solved in polynomial time [18].

Let $G = (V, E)$ be an embedded planar graph, E' be a superset of edges containing E, and $\Gamma(G')$ be a drawing of $G' = (V, E')$. When the deletion from $\Gamma(G')$ of the edges in $E' - E$ induces the embedded planar graph G, we say that $\Gamma(G')$ *preserves the embedded planar graph* G.

Definition 1. *Given an embedded planar graph $G = (V, E)$, directed or undirected, a non-negative integer c, and two vertices $s,\ t \in V$,* ***the hamiltonian path completion with edge crossing minimization (HPCCM) problem*** *asks whether there exists a superset E' containing E and a drawing $\Gamma(G')$ of graph $G' = (V, E')$ such that (i) G' has a hamiltonian path from vertex s to vertex t, (ii) $\Gamma(G')$ has at most c edge crossings, and (iii) $\Gamma(G')$ preserves the embedded planar graph G.*

Over the set of all HP-completion sets for a graph G, and over all of their different drawings that respect G, the one with a minimum number of edge-crossings is called a *crossing-optimal HP-completion set*. We refer to the version of the HPCCM problem where the input is an acyclic digraph and we insist on HP-completion sets which leave the HP-completed digraph also acyclic as the *Acyclic-HPCCM* problem.

Let $G = (V, E)$ be an embedded planar graph, let E_c be an HP-completion set of G and let $\Gamma(G')$ of $G' = (V, E')$ be a drawing with c crossings that preserves G. The graph G_c induced from drawing $\Gamma(G')$ by inserting a new vertex at each edge crossing and by splitting the edges involved in the edge-crossing is referred to as the *HP-extended graph of G w.r.t. $\Gamma(G')$*. (See Figure 5.a).

A *planar st-digraph* is an embedded planar acyclic digraph with exactly one source (i.e., a vertex with in-degree equal to 0) and exactly one sink (i.e., a vertex with out-degree equal to 0) both of which appear on the boundary of the external face. Traditionally, the source and the sink of an st-digraph are denoted by s and t, respectively. It is known that a planar st-digraph admits a planar upward drawing [7,19]. In the rest of the paper, all st-digraphs will be drawn

upward. A planar graph G is *outerplanar* if there exist a drawing of G such that all of G's vertices appear on the boundary of the same face (which is usually drawn as the external face). A *triangulated outerplanar* graph is an outerplanar graph with triangulated interior.

1.2 Related Work

For acyclic digraphs, the Acyclic-HPC problem has been studied in the literature in the context of partially ordered sets (posets) under the terms *linear extensions* and *jump number*. See [2] for detailed definitions. An optimal linear extension of a poset P (or its corresponding acyclic digraph G), is identical to an acyclic HP-completion set E_c for G of minimum size, and its jump number is equal to the size of E_c. This problem has been widely studied, in part due to its applications to scheduling. It has been shown to be NP-hard even for bipartite ordered sets [25]. Nevertheless, polynomial time algorithms are known for several classes of ordered sets, including cycle-free orders [6], orders of bounded width [5], bipartite orders of dimension two [28] and K-free orders [27]. Brightwell and Winkler [2] showed that counting the number of linear extensions is ♯P-complete. An algorithm that generates all of the linear extensions of a poset in constant amortized time was presented by Pruesse and Ruskey [24]. Later, Ono and Nakano [23] presented an algorithm which generates each linear extension in worst case constant time.

With respect to related work on book embeddings, Yannakakis[32] has shown that planar graphs have a book embedding on a 4-page book and that there exist planar graphs that require 4 pages for their book embedding. In the literature, the book embeddings where spine crossings are allowed are referred to as *topological book embeddings* [10]. Every planar graph admits a 2-page topological book embedding with only one spine crossing per edge [8].

For acyclic digraphs and posets, *upward book embeddings* have been also studied in the literature [1,15,16,17]. An upward book embedding can be considered to be a book embedding in which the spine is vertical and all edges are drawn monotonically increasing in the upward direction. The minimum number of pages required for an upward book embedding of a planar acyclic digraph is unbounded [15], while, the minimum number of pages required by an upward

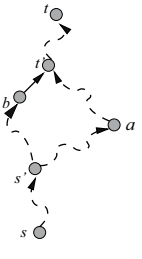


Fig. 1. Subgraph used in the proof of Lemma 2. Vertices a and b are not connected by a path in either direction.

planar digraph is not known [1,15]. Giordano et al. [13] showed that any embedded upward planar digraph has a topological book embedding with at most one spine crossing per edge.

We emphasize that the presented bibliography is in no way exhaustive. The topics of *hamiltonian paths*, *linear extensions* and *book embeddings* have been studied for a long time and an extensive body of literature has been accumulated.

2 Triangulated Hamiltonian *st*-Graphs

In this section, we develop the necessary and sufficient condition for a triangulated *st*-digraph to be hamiltonian. Based on the provided characterization we can determine whether the Acyclic-HPCCM problem can be solved for triangulated digraphs without crossings. The characterization is also used in the development of crossing-optimal HP-completion sets for outerplanar triangulated *st*-digraphs (see [21] for details).

It is well known[30] that for every vertex v of an *st*-digraph, its incoming (outgoing) incident edges appear consecutively around v. We denote by $Left(v)$ (resp. $Right(v)$) the face to the left (resp. right) of the leftmost (resp. rightmost) incoming and outgoing edges incident to v. The following lemma is a direct consequence from Lemma 7 by Tamassia and Preparata [29].

Lemma 1. *Let u and v be two vertices of a planar st-digraph such that there is no directed path between them in either direction. Then, in the dual G^* of G there is either a path from $Right(u)$ to $Left(v)$ or a path from $Right(v)$ to $Left(u)$.* □

The following lemma demonstrates a property of *st*-digraphs.

Lemma 2. *Let G be an st-digraph that does not have a hamiltonian path. Then, there exist two vertices in G that are not connected by a directed path in either direction.*

Proof. Let P be a longest path from s to t and let a be a vertex that does not belong in P. Since G does not have a hamiltonian path, such a vertex always exists. Let s' be the last vertex in P such that there exists a path $P_{s' \rightsquigarrow a}$ from s' to a with no vertices in P. Similarly, define t' to be the first vertex in P such that there exists a path $P_{a \rightsquigarrow t'}$ from a to t' with no vertices in P. Since G is acyclic, s' appears before t' in P (see Figure 1). Note that s' (resp. t') might be vertex s (resp. t). From the construction of s' and t' it follows that any vertex b, distinct from s' and t', that is located on path P between vertices s' and t', is not connected with vertex a in either direction. Thus, vertices a and b satisfy the property of the lemma.

Note that such a vertex b always exists. If this was not the case, then path P would contain edge (s', t'). Then, path P could be extended by replacing (s', t') by path $P_{s' \rightsquigarrow a}$ followed by path $P_{s' \rightsquigarrow a}$. This would lead to new path P' from s to t that is longer than P, a contradiction since P was assumed to be of maximum length. □

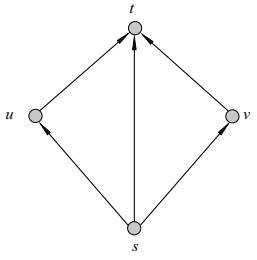


Fig. 2. The rhombus embedded digraph

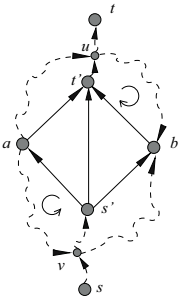


Fig. 3. The *st*-digraph used in the proof of Theorem 2

The embedded digraph in Figure 2 is called a *rhombus*. The central edge (s, t) of a rhombus is referred to as the *median of the rhombus* and is always drawn in the interior of its drawing.

The following theorem provides a characterization of triangulated planar *st*-digraphs that have a hamiltonian path.

Theorem 1. *Let G be a triangulated planar st-digraph. G has a hamiltonian path if and only if G does not contain any rhombus as a subgraph.*

Proof. ($\Rightarrow$) We assume that G has a hamiltonian path and we will show that it contains no rhombus as a subgraph. For the sake of contradiction, assume that G contains a rhombus composed from vertices s', t', a and b as a subgraph (see Figure 3). Then, vertices a and b of the rhombus are not connected by a directed path in either direction. To see this, assume wlog that there was a path connecting a to b. Then, this path has to intersect either the path from t' to t at a vertex u or the path from s to s' at a vertex v. In either case, there must exist a cycle in G, contradicting the fact that G is acyclic. So, we have shown that vertices a and b of the rhombus are not connected by a directed path in either direction, and thus there cannot exist any hamiltonian path in G, a clear contradiction.

($\Leftarrow$) We assume that G contains no rhombus as a subgraph and we will prove that G has a hamiltonian path. For the sake of contradiction, assume that G does not have a hamiltonian path. Then, from Lemma 2, if follows that there exist two vertices u and v of G that are not connected by a directed path in either direction. From Lemma 1, it then follows that there exists in the dual G^* of G a directed path from either $Right(u)$ to $Left(v)$, or from $Right(v)$ to $Left(u)$. Wlog, assume that the path in the dual G^* is from $Right(u)$ to $Left(v)$ (see Figure 4.a) and let $f_0,\ f_1,\ \ldots,\ f_k$ be the faces the path passes through, where $f_0 = Right(u)$ and $f_k = Left(v)$. We denote the path from $Right(u)$ to $Left(v)$ by $P_{u,v}$.

Note that path $P_{u,v}$ can exit face $f_0 = \triangle(u, w_0, w_1)$ only through edge (w_0, w_1) (see Figure 4.a). The path will enter a new face and, in the rest of the proof, we will construct the sequence of faces it goes through.

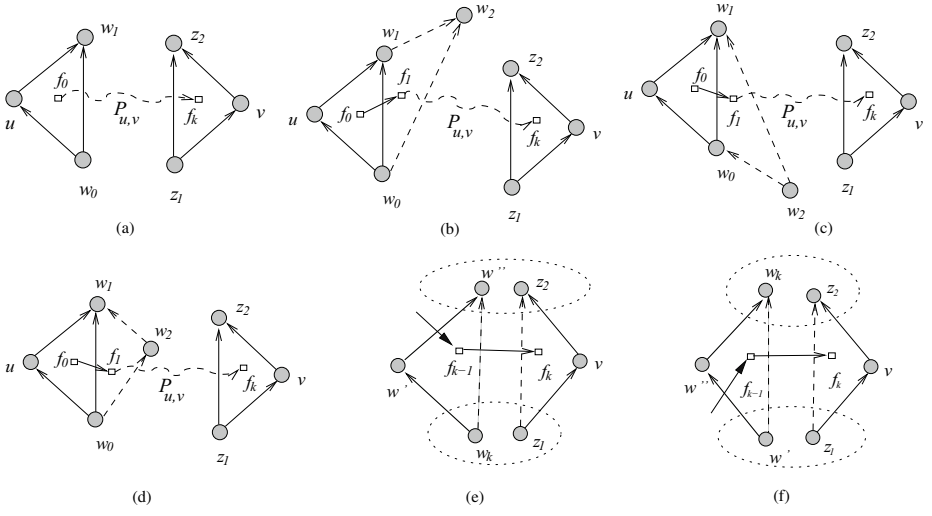


Fig. 4. The different cases occurring in the construction of path $P_{u,v}$ as described in the proof of Theorem 1

The next face f_1 of the path, consists of edge (w_0, w_1) which is connected to a vertex w_2. For face $f_1 = \triangle(w_0, w_1, w_2)$ there are 3 possible orientations (which do not violate acyclicity) for the direction of the two edges that connect w_2 with w_0 and w_1):

Case 1: Vertex w_2 has incoming edges from both w_0 and w_1 (see Figure 4.b). Observe that path $P_{u,v}$ can continue from f_1 to f_2 only by crossing edge (w_0, w_2). This is due to the fact that, in the dual G^*, the only outgoing edge of f_1 corresponds to the dual edge that crosses edge (w_0, w_2) of G.

Case 2: Vertex w_2 has outgoing edges to both w_0 and w_1 (see Figure 4.c). Observe that path $P_{u,v}$ can continue from f_1 to f_2 only by crossing edge (w_2, w_0). This is due to the fact that, in the dual G^*, the only outgoing edge of f_1 corresponds to the dual edge that crosses edge (w_2, w_0) of G.

Case 3: Vertex w_2 is connected to w_0 and w_1 by an incoming and an outgoing edge, respectively (see Figure 4.d). Note that in this case, f_0 and f_1 form a rhombus. Thus, this case cannot occur, since we assumed that G has no rhombus as a subgraph.

A common characteristic of either of the first two case that allow to further continue the identification of the faces path $P_{u,v}$ goes through is that there is *a single* edge that can be used to exit face f_1. Thus, we can continue identifying the faces path $P_{u,v}$ passes through, building in such a way sequence $f_0,\ f_1,\ \ldots,\ f_{k-1}$.

At the end, path $P_{u,v}$ has to leave face f_{k-1} by either edge (w_k, w'') (see Figure 4.e) or by edge (w', w_k) (see Figure 4.f). In either of these two cases, the outgoing boundary edge of face $f_{k-1} = \triangle(w', w'', w_k)$ has to be identified with the

single incoming edge of face $f_{k-1} = \Delta(z_1, z_2, v)$. Thus, the last two faces on the path $P_{u,v}$ will form a rhombus (see Figures 4.e and 4.f). Thus, in either case, in order to complete the path, graph G must contain a rhombus as a subgraph. This is a clear contradiction since we assumed that G does not contain any rhombus as a subgraph. □

3 Spine Crossing Minimization for Upward Topological 2-Page Book Embeddings

In this section, we establish for the class of *st*-digrpahs an equivalence (through a linear time transformation) between the Acyclic-HPCCM problem and the problem of obtaining an upward topological 2-page book embeddings with minimum number of spine crossings.

Theorem 2. *Let $G = (V, E)$ be an n node planar st-digraph. G has a crossing-optimal HP-completion set E_c with Hamiltonian path $P = (s = v_1, v_2, \ldots, v_n = t)$ such that the corresponding optimal drawing $\Gamma(G')$ of $G' = (V, E \cup E_c)$ has c crossings* ***if and only if*** *G has an optimal (wrt the number of spine crossings) upward topological 2-page book embedding with c spine crossings where the vertices appear on the spine in the order $\Pi = (s = v_1, v_2, \ldots, v_n = t)$.*

Proof. We show how to obtain from an HP-completion set with c edge crossings an upward topological 2-page book embedding with c spine crossings and vice versa. From this is follows that a crossing-optimal HP-completion set for G with c edge crossing corresponds to an optimal upward topological 2-page book embedding with the same number of spine crossings.

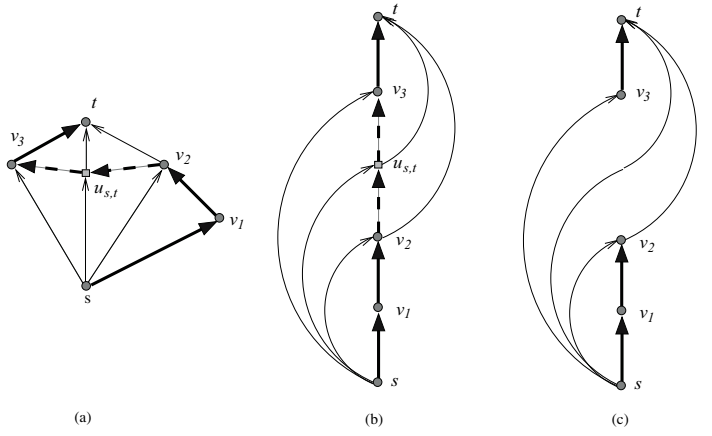


Fig. 5. (a) A drawing of an HP-extended digraph for an *st*-digraph G. The dotted segments correspond to the single edge (v_2, v_3) of the HP-completion set for G. (b)An upward topological 2-page book embedding of G_c with its vertices placed on the spine in the order they appear on a hamiltonian path of G_c. (c)An upward topological 2-page book embedding of G.

"⇒" We assume that we have an HP-completion set E_c that satisfies the conditions stated in the theorem. Let $\Gamma(G')$ of $G' = (V, E \cup E_c)$ be the corresponding drawing that has c crossings and let $G_c = (V \cup V_c, E' \cup E'_c)$ be the acyclic HP-extended digraph of G wrt $\Gamma(G')$. V_c is the set of new vertices placed at each edge crossing. E' and E'_c are the edge sets resulting from E and E_c, respectively, after splitting their edges involved in crossings and maintaining their orientation (see Figure 5(a)). Note that G_c is also an st-planar digraph.

Observe that in $\Gamma(G')$ we have no crossing involving two edges of G. If this was the case, then $\Gamma(G')$ would not preserve G. Similarly, in $\Gamma(G')$ we have no crossing involving two edges of the HP-completion set E_c. If this was the case, then G_c would contain a cycle.

The hamiltonian path P on G' induces a hamiltonian path P_c on the HP-extended digraph G_c. This is due to the facts that all edges of E_c are used in hamiltonian path P and all vertices of V_c correspond to crossings involving edges of E_c. We use the hamiltonian path P_c to construct an upward topological 2-page book embedding for graph G with exactly c spine crossings. We place the vertices of G_c on the spine in the order of hamiltonian path P_c, with vertex $s = v_1$ being the lowest. Since the HP-extended digraph G_c is a planar st-digraph with vertices s and t on the external face, each edge of G_c appears either to the left or to the right of the hamiltonian path P_c. We place the edges of G_c on the left (resp. right) page of the book embedding if they appear to the left (resp. right) of path P_c. The edges of P_c are drawn on the spine (see Figure 5(b)). Later on they can be moved to any of the two book pages.

Note that all edges of E_c appear on the spine. Consider any vertex $v_c \in V_c$. Since v_c corresponds to a crossing between an edge of E and an edge of E_c, and the edges of E'_c incident to it have been drawn on the spine, the two remaining edges of E' correspond to (better, they are parts of) an edge $e \in E$ and drawn on different pages of the book. By removing vertex v_c and merging its two incident edges of E' we create a crossing of edge e with the spine. Thus, the constructed book embedding has as many spine crossings as the number of edge crossings of HP-completed graph G' (see Figure 5(c)).

It remains to show that the constructed book embedding is upward. It is sufficient to show that the constructed book embedding of G_c is upward. For the sake of contradiction, assume that there exists a downward edge $(u, w) \in E'_c$. By the construction, the fact that w is drawn below u on the spine implies that there is a path in G_c from w to u. This path, together with edge (u, w) forms a cycle in G_c, a clear contradiction since G_c is acyclic.

"⇐" Assume that we have an upward 2-page topological book embedding of st-digraph G with c spine crossings where the vertices appear on the spine in the order $\Pi = (s = v_1, v_2, \ldots, v_n = t)$. Then, we construct an HP-completion set E_c for G that satisfies the condition of the theorem as follows: $E_c = \{(v_i, v_{i+1}) \mid 1 \leq i < n \text{ and } (v_i, v_{i+1}) \notin E\}$, that is, E_c contains an edge for each consecutive pair of vertices of the spine that (the edge) was not present in G. By adding/drawing these edges on the spine of the book embedding we get a drawing $\Gamma(G')$ of $G' = (V, E \cup E_c)$ that has c edge crossings. This is due to the fact that all spine

crossing of the book embedding are located, (i) at points of the spine above vertex s and below vertex t, and (ii) at points of the spine between consecutive vertices that are not connected by an edge. By inserting at each crossing of $\Gamma(G')$ a new vertex and by splitting the edges involved in the crossing while maintaining their orientation, we get an HP-extended digraph G_c. It remains to show that G_c is acyclic. For the sake of contradiction, assume that G_c contains a cycle. Then, since graph G is acyclic, each cycle of G_c must contain a segment resulting from the splitting of an edge in E_c. Given that in $\Gamma(G')$ all vertices appear on the spine and all edges of E_c are drawn upward, there must be a segment of an edge of G that is downward in order to close the cycle. Since, by construction, the book embedding of G is a sub-drawing of $\Gamma(G')$, one of its edges (or just a segment of it) is downward. This is a clear contradiction since we assume that the topological 2-page book embedding of G is upward. □

In [21] we describe a linear-time algorithm that solves the Acyclic-HPCCM problem for the class of triangulated outerplanar st-digrpahs with at most one crossing per edge of G. The following theorem is the consequence of this algorithm and its detailed proof can be found in [21].

Theorem 3. *[21] Given an n node triangulated outerplanar st-digrpah G, a crossing-optimal HP-completion set for G with at most one crossing per edge and the corresponding number of edge-crossings can be computed in $O(n)$ time.*

Theorem 4. *Given an n node triangulated outerplanar st-digrpah G, an upward 2-page topological book embedding for G with minimum number of spine crossings and at most one spine crossing per edge and the corresponding number of edge-crossings can be computed in $O(n)$ time.*

Proof. By Theorem 2 we know that by solving the Acyclic-HPCCM problem on G, we can deduce the wanted upward book embedding. By Theorem 3, the Acyclic-HPCCM problem can be solved in $O(n)$ time. □

4 Conclusions - Open Problems

We have studied the problem of Acyclic-HPCCM and we have presented a linear time algorithm that computes a crossing-optimal Acyclic HP-completion set for an OT st-digraphs G with at most one crossing per edge of G. Future research topics include the study of the Acyclic-HPCCM on the larger class of st-digraphs, as well as relaxing the requirement for G to be triangulated.

References

1. Alzohairi, M., Rival, I.: Series-parallel planar ordered sets have pagenumber two. In: North, S.C. (ed.) GD 1996. LNCS, vol. 1190, pp. 11–24. Springer, Heidelberg (1997)
2. Brightwell, G., Winkler, P.: Counting linear extensions. Order 8, 225–242 (1991)

3. Chein, M., Habib, M.: Jump number of dags having dilworth number 2. Discrete Applied Math. 7, 243–250 (1984)
4. Cogis, O., Habib, M.: Nombre de sauts et graphes series-paralleles. In: RAIRO Inf. Th., vol. 13, pp. 3–18 (1979)
5. Colbourn, C.J., Pulleyblank, W.R.: Minimizing setups in ordered sets of fixed width. Order 1, 225–229 (1985)
6. Duffus, D., Rival, I., Winkler, P.: Minimizing setups for cycle-free orders sets. Proc. of the American Math. Soc. 85, 509–513 (1982)
7. Di Battista, G., Tamassia, R.: Algorithms for plane representations of acyclic digraphs. Theor. Comput. Sci. 61(2-3), 175–198 (1988)
8. Di Giacomo, E., Didimo, W., Liotta, G., Wismath, S.K.: Curve-constrained drawings of planar graphs. Comput. Geom. Theory Appl. 30(1), 1–23 (2005)
9. Diestel, R.: Graph Theory, 3rd edn. Springer, Heidelberg (2005)
10. Enomoto, H., Miyauchi, M.S., Ota, K.: Lower bounds for the number of edge-crossings over the spine in a topological book embedding of a graph. Discrete Appl. Math. 92(2-3), 149–155 (1999)
11. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
12. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified np-complete problems. In: STOC 1974: Proceedings of the sixth annual ACM symposium on Theory of computing, pp. 47–63. ACM, New York (1974)
13. Giordano, F., Liotta, G., Mchedlidze, T., Symvonis, A.: Computing upward topological book embeddings of upward planar digraphs. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 172–183. Springer, Heidelberg (2007)
14. Harary, F.: Graph Theory. Addison-Wesley, Reading (1972)
15. Heath, L.S., Pemmaraju, S.V.: Stack and queue layouts of posets. SIAM J. Discrete Math. 10(4), 599–625 (1997)
16. Heath, L.S., Pemmaraju, S.V.: Stack and queue layouts of directed acyclic graphs: Part II. SIAM J. Comput. 28(5), 1588–1626 (1999)
17. Heath, L.S., Pemmaraju, S.V., Trenk, A.N.: Stack and queue layouts of directed acyclic graphs: Part I. SIAM J. Comput. 28(4), 1510–1539 (1999)
18. Karejan, Z.A., Mosesjan, K.M.: The hamiltonian completion number of a digraph (in Russian). Akad. Nauk Armyan. SSR Dokl. 70(2-3), 129–132 (1980)
19. Kelly, D.: Fundamentals of planar ordered sets. Discrete Math. 63(2-3), 197–216 (1987)
20. Mitas, J.: Tackling the jump number of interval orders. Order 8, 115–132 (1991)
21. Mchedlidze, T., Symvonis, A.: Optimal acyclic hamiltonian path completion for outerplanar triangulated st-digraphs (with application to upward topological book embeddings), arXiv:0807.2330, `http://arxiv.org/abs/0807.2330`
22. Nowakowski, R., Parker, A.: Ordered sets, pagenumbers and planarity. Order 6(3), 209–218 (1989)
23. Ono, A., Nakano, S.: Constant Time Generation of Linear Extensions. In: Liśkiewicz, M., Reischuk, R. (eds.) FCT 2005. LNCS, vol. 3623, pp. 445–453. Springer, Heidelberg (2005)
24. Pruesse, G., Ruskey, F.: Generating linear extensions fast. SIAM Journal on Computing 23(2), 373–386 (1994)
25. Pulleyblank, W.R.: On minimizing setups in precedence constrained scheduling. Report 81105-OR (1981)
26. Rival, I.: Optimal linear extensions by interchanging chains. vol. 89, pp. 387–394 (Novomber 1983)

27. Sharary, A., Zaguia, N.: On minimizing jumps for ordered sets. Order 7, 353–359 (1991)
28. Steiner, G., Stewart, L.K.: A linear algorithm to find the jump number of 2-dimensional bipartite partial orders. Order 3, 359–367 (1987)
29. Tamassia, R., Preparata, F.P.: Dynamic maintenance of planar digraphs, with applications. Algorithmica 5(4), 509–527 (1990)
30. Tamassia, R., Tollis, I.G.: A unified approach a visibility representation of planar graphs. Discrete & Computational Geometry 1, 321–341 (1986)
31. Wigderson, A.: The complexity of the hamiltonian circuit problem for maximal planar graphs. Technical Report TR-298, Princeton University, EECS Department (1982)
32. Yannakakis, M.: Embedding planar graphs in four pages. J. Comput. Syst. Sci. 38(1), 36–67 (1989)

Core and Conditional Core Path of Specified Length in Special Classes of Graphs

S. Balasubramanian, S. Harini, and C. Pandu Rangan

Department of Computer Science and Engineering,
IIT Madras, India 600036
{balu2901,harini.santhanam}@gmail.com, rangan@iitm.ernet.in

Abstract. A core path of a graph is a path P in G that minimizes $d(P) = \sum_{v \in V} d(v, P)w(v)$. In this paper, we study the location of core path of specified length in special classes of graphs. Further, we extend our study to the problem of locating a core path of specified length under the condition that some existing facilities are already located (known as conditional core path of a graph). We study both the problems stated above in vertex weighted bipartite permutation graphs, threshold graphs and proper interval graphs and give polynomial time algorithms for the core path and conditional core path problem in these classes. We also establish the NP-Completeness of the above problems in the same classes of graphs when arbitrary positive weights are assigned to edges.

Keywords: Core path, Conditional core path, Bipartite permutation graphs, Threshold graphs, Proper Interval graphs.

1 Introduction

The objective of any facility location algorithm in a network is to locate a site/facility that optimizes some criterion. The criteria that have been most generally employed are the *minimax* and *minisum* criteria. In the *minimax* criteria, the distance of the farthest vertex from the facility is minimized. In the *minisum* criteria, the sum of the distances of the vertices of the graph from the facility is minimized. Many practical facility location problems however, involve the location of several facilities rather than just one facility. In particular, the problem of locating a path-shaped or tree-shaped facility has received wide attention due to applications in metro rail routing, pipeline planning, laying irrigation ditches etc. When the path to be located is such that it satisfies the *minisum* criterion,(i.e the sum of the distances of the vertices of the graph from the path is minimized) then we call the path a core path which we will define formally in this section.

Often, it might happen that some facilities have already been located and the new facilities should be located such that the *minisum* criterion is optimized taking into account both the already existing facilities and the newly located facilities. For example, a district may already be equipped with a gas pipeline and

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 262–273, 2009.

we should plan the layout of a new pipeline. Any user can obtain a connection either from the old or the new pipeline depending on which one is closer to him. Where should we lay this new pipeline such that it minimizes the sum of the distances of the users in the district from the pipelines (old and new). This problem is called the conditional facility location problem. When the new facility to be established is a path, the problem is known as conditional-core problem (defined formally later). The network can be assigned vertex and edge weights. Vertex weights can denote the population of a locality and edge weights the distance between two localities. In this paper, we study the problem of locating the core path and conditional core path of a specified length in bipartite permutation graphs, threshold graphs and proper interval graphs. Specifically, we give polynomial time algorithms for both the problems in the above classes of graphs when vertices are assigned arbitrary positive weights and edges are assigned unit weights. When the edges are assigned arbitrary weights, we prove the NP-Completeness of both the problems in all the three classes of graphs.

Let $G = (V, E)$ be a simple, undirected, connected, weighted (positive vertex and edge weights) graph. The length of a path P is defined as sum of weights of edges in P. Let $d(u, v)$ be the shortest distance between two vertices u and v. The set of vertices of a path P is denoted by $V(P)$ and the set of vertices of $V(P) \cap X$ for any set $X \subseteq V$, is denoted by $V_X(P)$. We extend the notion of distance between a pair of vertices in a natural way to the notion of distance between a vertex and a path. The distance between a vertex v and path P is $d(v, P) = \underset{u \in V(P)}{Min}\, d(v, u)$. If $v \in V(P)$, then $d(v, P)$ is zero. The **cost of a path** P denoted by $d(P)$ is $\sum_{v \in V} d(v, P)w(v)$, where $w(v)$ is the weight of the vertex v.

Definition 1. *[9] The **Core path or Median path** of a graph G is a path P that minimizes $d(P)$.* □

Definition 2. *[8] Let $\mathcal{P}_l$ be the set of all paths of length l in G. The **Core path of length l** of a graph G is a path $P \in \mathcal{P}_l$ where $d(P) \leq d(P')$ for any path $P' \in \mathcal{P}_l$.* □

Let S denote the set of vertices in which facilities have already been deployed. The **conditional cost of a path** P denoted by
$d^c(P) = \sum_{v \in V} min(d(v, P), d(v, S))w(v)$, where $d(v, S) = \underset{u \in S}{Min}\, d(v, u)$.

Definition 3. *Let $\mathcal{P}_l^S$ be the set of all paths of length l in G such that $V(P) \cap S = \emptyset$. The **Conditional Core path of length l** of a graph G is a path $P \in \mathcal{P}_l^S$ where $d^c(P) \leq d^c(P')$ for any path $P' \in \mathcal{P}_l^S$.* □

Previous work: So far, the core-path problem has been analyzed only in trees and recently in grid graphs. In [5] Hakimi, Schmeichel, and Labb'e have given 64 variations of the above problem and have also proved that finding the core path is NP-Hard on arbitrary graphs. In [9] Morgan and Slater give a linear time algorithm for finding core path of a tree with arbitrary edge weights. In [7], [8] Minieka et al. consider the problem of finding the core path with a constraint on

the length of the path. Their work considers locating path or tree shaped facilities of specified length in a tree and they present an $O(n^3)$ algorithm. In [10] Peng and Lo extend their work by giving a $O(nlogn)$ sequential and $O(log^2(n))$ parallel algorithm using $O(n)$ processors for finding the core path of a tree(unweighted) with a specified length. In [3] Becker et al. give an $O(nl)$ algorithm for the unweighted case and a $O(nlog^2n)$ for the weighted case for trees. [2] presents a study of the core path in grid graphs. In [1] Alstrup et al. give an $O(n\ min\{\log n\ \alpha(n,n), l\})$ algorithm for finding the core path of a tree. The conditional location of path and tree shaped facilities have also been extensively studied on trees. In [11], Tamir et. al. prove that the continuous conditional median subtree problem is NP-hard and they develop a fully polynomial time approximation scheme for the same. They also provide an $O(n \log^2 n)$ algorithm for the discrete conditional core path problem with a length constraint and an $O(n^2)$ algorithm for the continuous conditional core path problem with a length constraint. They also study the conditional location center paths on trees. Further, in [13], Wang et al improve the algorithms in [11] for both discrete and continuous conditional core path problem with a length constraint. For the discrete case, they presented an $O(n \log n)$ algorithm and for the continuous case they presented an $O(n \log n\ \alpha(n,n))$ algorithm.

The rest of the paper is organized as follows. In section 2, we give a polynomial time algorithm to solve the core path problem in vertex weighted bipartite permutation graphs. In section 3, we solve the conditional core problem in vertex weighted bipartite permutation graphs. We prove the NP-Completeness of the core path problem in bipartite permutation graphs with arbitrary edge weights in *Appendix*. We also briefly present our solution for core and conditional core path problems in unit edge weighted threshold graphs and proper interval graphs in *Appendix*.

2 Core Path of a Bipartite Permutation Graph with Vertex Weights

2.1 Preliminaries

Let $\pi = (\pi_1, \pi_2, \ldots \pi_n)$ be a permutation of the numbers $1, 2 \ldots n$. The graph $G(\pi) = (V, E)$ is defined as follows: $V = \{1, 2 \ldots n\}$ and $(i, j) \in E \iff (i - j)(\pi_i^{-1} - \pi_j^{-1}) < 0$ where π_i^{-1} is the position of number i in the sequence π. An undirected graph G is called a permutation graph iff there exists a permutation π such that G is isomorphic to $G(\pi)$. A graph is a bipartite permutation graph, if it is both a bipartite graph and a permutation graph. We assume that the bipartite permutation graph $G = (X, Y, E)$ has unit edge weights and arbitrary vertex weights. We present a polynomial time algorithm for solving core path problem on G. The next definition is taken from [6].

Definition 4. *A strong ordering of the vertices of a bipartite graph $G = (X, Y, E)$ consists of an ordering $<_x$ of X and an ordering $<_y$ of Y such that for all (x, y), $(x', y') \in E$, where x, $x' \in X$ and y, $y' \in Y$, $x <_x x'$ and $y' <_y y$ imply (x, y') and $(x', y) \in E$.* □

From now on, we drop the subscripts for $<_x$ and $<_y$ and denote them by $<$, interpreting the meaning from the context. For any two vertices a and b that are in the same partition of the bipartite graph, if $a < b$, we say a is *above* b and b is *below* a. Given an edge (x_i, y_j), we call the union of all the vertices *above* x_i and *above* y_j as *above* the edge (x_i, y_j).

[6] A bipartite graph B is a bipartite permutation graph iff *it admits a strong ordering.*

Ordered Paths: Any path $P = v_1v_2v_3v_4v_5v_6\ldots$ of a bipartite permutation graph G is said to be an *ordered* path *iff* $v_1 < v_3 < v_5 < \ldots$ and $v_2 < v_4 < v_6 < \ldots$. The following lemma relates every path with an ordered path of same vertex set.

Lemma 1. *In a bipartite permutation graph, for every path P, there exists an ordered path Q, such that $V(P)$=$V(Q)$.*

Proof in Appendix.

In the above lemma, since $V(P)$=$V(Q)$, it follows that $d(P)$=$d(Q)$ and $d^c(P) = d^c(Q)$. So, for every path there is an ordered path with the same cost and the same set of vertices. Therefore, we consider only ordered paths henceforth.

2.2 Algorithm

Brief overview of the algorithm: The naive technique searches the set of all paths in the graph to find the core path. Since we have proved that for every path there exists an ordered path with the same set of vertices, we can cut down on the search space for paths to ordered paths alone. The set of vertices adjacent to a vertex u is called the neighborhood of u, and is written as $N(u)$. Let $L(u)$ and $R(u)$ be the vertices with smallest and largest index in $N(u)$ according to the strong ordering. Let $P = v_1v_2v_3\ldots v_k$ be an ordered path in G. Then the edge (v_1, v_2) is called the *first edge* of P and the edge (v_{k-1}, v_k) is called the *last edge* of P. For every ordered path P, let $\alpha_r(P)$ denote the path obtained by taking the *first* r edges of the ordered path P. In case, the path does not contain r edges, then $\alpha_r(P) = \perp$. We define $d(\perp) = \infty$. Also, $d(\alpha_r(P)v) = \infty$, when $\alpha_r(P) = \perp$, where $\alpha_r(P)v$ denotes the concatenation of $\alpha_r(P)$ and vertex v.

Remark 1. For every edge $(x, y) \in E$ let $path^l_{xy}$ denote the path with (x, y) as its *last edge* such that it is the minimum cost ordered path of length l with (x, y) as its *last edge*. The $\underset{(x,y)\in E}{Min}\ d(path^l_{xy})$ will yield us the cost of core path of length l of the graph G.

Remark 2. Let G_{ij} be a graph induced by the vertices $\{x_1, x_2, \ldots, x_i\}$ and $\{y_1, y_2, \ldots, y_j\}$. The ordered path with (x_i, y_j) as the *last edge* cannot contain a vertex v such that $x_i < v$ or $y_j < v$. Hence for every edge (x_i, y_j), it is sufficient to consider the ordered paths in G_{ij} and not the entire graph G.

Finding the cost of any ordered path: Here we give a method to compute the cost of any ordered path efficiently. For all edges, $(x_i, y_j) \in E$ we define, $U(x_i, y_j) = \{v \in V | (v < x_i \text{ or } v < y_j)\}$, $W_U(x_i, y_j) = \sum_{v \in U(x_i, y_j)} w(v)$, $UAdj(x_i, y_j) = \{v \in V | (v < x_i \text{ and } (v, y_j) \in E) \text{ or } (v < y_j \text{ and } (v, x_i) \in E)\}$, $W_{UAdj}(x_i, y_j) = \sum_{v \in UAdj(x_i, y_j)} w(v)$,
$USUM(x_i, y_j) = \sum_{v \in U(x_i, y_j)} min(d(v, x_i), d(v, y_j))w(v)$.
Intuitively, $U(x_i, y_j)$ denotes the set of all vertices that are *above* (x_i, y_j) as per the strong ordering. $W_U(x_i, y_j)$ is the sum of the weight of vertices in the set $U(x_i, y_j)$. The set $UAdj(x_i, y_j)$ is the set of all vertices that are *above* (x_i, y_j) and are adjacent to either x_i or y_j. $USUM(x_i, y_j)$ denotes the sum of the costs incurred by all the vertices *above* (x_i, y_j) in either reaching x_i or y_j(whichever is closer). When all terms given above are preprocessed and stored for all the edges, the cost of any path P can be computed in $O(1)$ time. We know that $W_U(x_1, y_1) = 0$. We can compute $W_U(x_i, y_j)$ and $W_{UAdj}(x_i, y_j)$ for all $(x_i, y_j) \in E$ in O($|E|$) time using the equations:

$$W_U(x_i, y_j) = \begin{cases} W_U(x_{i-1}, y_j) + w(x_{i-1}) & \text{if } (x_{i-1}, y_j) \in E \\ W_U(x_i, y_{j-1}) + w(y_{j-1}) & \text{if } (x_{i-1}, y_j) \notin E \text{ but } (x_i, y_{j-1}) \in E \end{cases}$$

$W_{UAdj}(x_i, y_j) = W_U(x_i, y_j) - W_U(L(y_j), L(x_i))$.

Lemma 2. *The following equation can be used to compute the value of* $USUM(x_i, y_j)$ *iteratively*

Let $x' = L(y_j)$ *and* $y' = L(x_i)$
$USUM(x_i, y_j) = USUM(x', y') + W_U(x', y') + W_{UAdj}(x_i, y_j)$.

Proof. We will first give an intuitive sketch of the proof. $USUM(x_i, y_j)$ is the cost incurred by all the vertices *above* (x_i, y_j) (i.e. vertices in $U(x_i, y_j)$) in reaching the nearer of x_i or y_j. This cost can be viewed as a sum of two terms : the cost due to vertices adjacent to and above (x_i, y_j) (i.e. in $UAdj(x_i, y_j)$) and the cost due to vertices *above* (x', y') (i.e. in $U(x', y')$). All the vertices in $UAdj(x_i, y_j)$ are at a distance of one from x_i or y_j and hence the cost incurred by them is $W_{UAdj}(x_i, y_j)$. All the vertices in $U(x', y')$ have to reach one of x' or y' to reach x_i or y_j. The cost incurred to reach x' or y' is $USUM(x', y')$. From (x', y'), all the vertices have to travel a distance of one to reach x_i or y_j and hence the cost incurred is $W_U(x', y')$. This completes the proof. We give an inductive proof below.

By definition, we know that $USUM(x_1, y_1) = 0$. We can verify by inspection that the expression for $USUM$ is true for $(x_1, y) \in E \; \forall y$ satisfying $L(x_1) \leq y \leq R(x_1)$. By induction hypothesis we assume that $USUM$ is correctly computed for all (x_i, y), $1 \leq i \leq r-1$ and $y = L(x_i)$ to $R(x_i)$. We prove the claim for the edge (x_r, y_j) where $y_j = L(x_r)$ and the proof follows similarly for the case where $L(x_r) < y_j$. By definition,

$USUM(x_r, y_j) = \sum_{v \in U(x_r, y_j)} min(d(v, x_r), d(v, y_j))w(v)$.

$USUM(x_r, y_j) = \sum_{v \in U(x',y')} min(d(v,x_r), d(v,y_j))w(v) + W_{UAdj}(x_r, y_j)$.
where $x' = L(y_j)$ and $y' = L(x_r)$.

For every $v \in U(x', y')$, $min(d(v,x'), d(v,y')) \leq min(d(v,x''), d(v,y''))$ where $x' < x''$ and $y' < y''$. This statement implies each vertex $v \in U(x', y')$ can reach one of (x_r or y_j) only through one of (x' or y'). Therefore we have that,

$$\sum_{v \in U(x',y')} min(d(v,x_r), d(v,y_j))w(v) = USUM(x', y') + W_U(x', y').$$

The cost $W_U(x', y')$ is attributed to the extra distance of length one per vertex in $U(x', y')$ to reach x_r or y_j. □

For all edges $(x_i, y_j) \in E$ we define,$B(x_i, y_j) = \{v \in V | (x_i < v \text{ or } y_j < v)\}$, $W_B(x_i, y_j) = \sum_{v \in B(x_i,y_j)} w(v)$, $BAdj(x_i, y_j) = \{v \in V | (x_i < v \text{ and } (v, y_j) \in E)$ or $(y_j < v \text{ and } (v, x_i) \in E)\}$, $W_{BAdj}(x_i, y_j) = \sum_{v \in BAdj(x_i,y_j)} w(v)$,
$BSUM(x_i, y_j) = \sum_{v \in B(x_i,y_j)} min(d(v,x_i), d(v,y_j))w(v)$.

Similar to $USUM$, we can compute $BSUM$ value for every $(x, y) \in E$ in $O(|E|)$ time. We define,

$W = \sum_{v \in V} w(v)$ and $W(P) = \sum_{v \in V(P)} w(v)$ for any path P.

W can be computed in $O(|V|)$ time and we give a method to calculate $W(P)$ in *Lemma* 4.

The following *Lemma* gives a method to compute cost of any ordered path.

Lemma 3. *For any ordered path P, the cost $d(P)$ can be computed in $O(1)$ time after $O(|E|)$ preprocessing.*

Proof. Let x_a and x_b be the vertices of $V_X(P)$ such that they have respectively the smallest and largest index in the strong ordering. Similarly, let y_a and y_b be the vertices of $V_Y(P)$ such that they have respectively the smallest and largest index in the strong ordering. We claim that

$$d(P) = USUM(x_a, y_a) + BSUM(x_b, y_b) + W - W(P) - W_B(x_b, y_b) - W_U(x_a, y_a)$$

Using the above formula, $d(P)$ can be computed in $O(1)$ time after $O(|E|)$ preprocessing. For $v \in \{X \cup Y\}$ - $V(P)$ - $B(x_b, y_b)$ - $U(x_a, y_a)$, we know that $d(v, P) = 1$. For all of these vertices, the total cost incurred is W - $W(P)$ - $W_B(x_b, y_b)$ - $W_U(x_a, y_a)$. Value $USUM(x_a, y_a)$ accounts $\forall v \in U(x_a, y_a)$ and $BSUM(x_b, y_b)$ accounts $\forall v \in B(x_b, y_b)$. Note that we still have not calculated $W(P)$ which is necessary for calculating $d(P)$. We specify the method to compute this in *Lemma* 4. □

Finding the core path: As noted by *Remark* 1, we have to find $path^l_{xy}$ $\forall xy \in E$. Let $P^l_{x_i y_j}$ denote an ordered path of length l and of minimum cost among all ordered paths of length l in G_{ij} with (x_i, y_j) as the *last edge* and y_j being the

vertex of degree one in the path P. Let $P^l_{y_jx_i}$ denote an ordered path of length l and of minimum cost among all ordered paths of length l in G_{ij} with (x_i, y_j) as the *last edge* and x_i being the vertex of degree one in the path P. From $P^l_{x_iy_j}$ and $P^l_{y_jx_i}$, we can compute $path^l_{x_iy_j}$ which is defined such that $d(path^l_{x_iy_j}) = Min\{d(P^l_{x_iy_j}), d(P^l_{y_jx_i})\}$. The path corresponding to the cost $\underset{xy\in E}{Min}\, d(path^l_{xy})$ will yield a core path of length l of graph G.

$P^1_{x_iy_j} = x_iy_j$, $P^1_{y_jx_i} = y_jx_i$ and their costs can be computed using *Lemma* 3. We now give, equations to compute the path of least cost of length r from the knowledge of the same for length $r-1$. Initially we give the equations that follow from definition and later we give a dynamic programming equation to compute the same efficiently.

Lemma 4. *The following equations can be used to find the costs of* $P^r_{x_iy_j}$, $P^r_{y_jx_i}$ $\forall r \geq 2$. *In graph* G_{ij}, $\forall (x_i, y_j) \in E(G_{ij})$ *we have that,*

$$d(P^r_{x_iy_j}) = \begin{cases} \underset{\forall (y_k,x_i)\in E, k<j}{Min}\, d(P^{r-1}_{y_kx_i}y_j) & \text{if such } y_k\text{'s exist} \\ \infty & \text{otherwise} \end{cases} \tag{1}$$

$$= \begin{cases} Min\{d(P^{r-1}_{y_{j-1}x_i}y_j),\ d(\alpha_{r-1}(P^r_{x_iy_{j-1}})y_j)\} & \text{if } j > 1\ ,\ (x_i, y_{j-1}) \in E \\ \infty & \text{otherwise} \end{cases} \tag{2}$$

$$d(P^r_{y_jx_i}) = \begin{cases} \underset{\forall (x_k,y_j)\in E, k<i}{Min}\, d(P^{r-1}_{x_ky_j}x_i) & \text{if such } x_k\text{'s exist} \\ \infty & \text{otherwise} \end{cases} \tag{3}$$

$$= \begin{cases} Min\{d(P^{r-1}_{x_{i-1}y_j}x_i),\ d(\alpha_{r-1}(P^r_{y_jx_{i-1}})x_i)\} & \text{if } i > 1\ ,\ (x_{i-1}, y_j) \in E \\ \infty & \text{otherwise} \end{cases} \tag{4}$$

$$d(path^r_{x_iy_j}) = Min\{d(P^r_{x_iy_j}),\ d(P^r_{y_jx_i})\} \tag{5}$$

Also, for calculating the cost $d(P)$ *of a path* P, *we need* $W(P)$. *This can be calculated while we construct the path using the following equations*

$$W(P^{r-1}_{y_kx_i}y_j) = W(P^{r-1}_{y_kx_i}) + w(y_j),\ W(P^{r-1}_{x_ky_j}x_i) = W(P^{r-1}_{x_ky_j}) + w(x_i)$$

Proof. We will prove equations (1) and (2). The proofs for (3) and (4) follow analogously.

Proof for (1)
(1) states that we can the compute minimum cost path of length r having (x_i, y_j) as the last edge, by considering minimum cost paths of length $r-1$ having (y_k, x_i) as last edge $\forall k < j$. We choose the minimum cost path among them, and append it with (x_i, y_j) to get the required path. Note that y_k's are chosen such

that $k < j$ because $\forall k > j$, the path is not ordered and we have proved that it is enough to search the set of all ordered paths to find the core path. The number of operations in (1) to compute $d(P^r_{x_i y_j})$ can be as high as $degree(x_i)$

Proof for (2)
For computing $d(P^r_{x_i y_j})$, we note that it is just enough to consider the path of least cost of length $r-1$ having (y_{j-1}, x_i) as the last edge and the path of least cost of length $r-1$ having (y_k, x_i) as the last edge $\forall\ k < j-1$. The latter term is $d(\alpha_{r-1}(P^r_{x_i y_{j-1}})y_j)$. Note that this value would have already been computed while computing $d((P^r_{x_i y_{j-1}})y_j)$ and hence no special effort is required now.

Claim. $d(\alpha_{r-1}(P^r_{x_i y_{j-1}})y_j) = \underset{\forall (y_k, x_i) \in E | k < j-1}{Min} d(P^{r-1}_{y_k x_i} y_j)$

It is clear that if the above claim is established, then equations (2) and (1) will become equivalent and (2) is proven.

Let $\alpha_{r-1}(P^r_{x_i y_{j-1}})$ be such that it has (y_t, x_i) as the *last edge.*

$$\Rightarrow d(P^{r-1}_{y_t x_i} y_{j-1}) = \underset{\forall (y_{t'}, x_i) \in E | t' < j-1}{Min} d(P^{r-1}_{y_{t'} x_i} y_{j-1})$$

$$\Rightarrow d(P^{r-1}_{y_t x_i}) = \underset{\forall (y_{t'}, x_i) \in E | t' < j-1}{Min} d(P^{r-1}_{y_{t'} x_i})$$

$$\Rightarrow d(P^{r-1}_{y_t x_i} y_j) = \underset{\forall (y_{t'}, x_i) \in E | t' < j-1}{Min} d(P^{r-1}_{y_{t'} x_i} y_j)$$

which is precisely the statement of our *claim.* Thus the number of operations if (2) is used to compute $d(P^r_{x_i y_j})$ is just two. □

Theorem 1. *The Algorithm computes the core path of a bipartite permutation graph in $O(l|E|)$ time.*

Proof. The algorithm computes the values of $d(path^l_{xy})$ for each edge $(x, y) \in E$ using *Lemma* 4 and then computes the minimum cost path by finding $\underset{(x,y) \in E}{Min}$ $d(path^l_{xy})$ and hence the correctness follows.

Time complexity: For a given length and a given edge (x, y) or $(y, x) \in E$, the algorithm takes $O(1)$ time to compute the value of $d(P^{length}_{xy})$ or $d(P^{length}_{yx})$. Therefore to compute the value of $d(P^l_{xy})$ and $d(P^l_{yx})$ for all edges (x, y) and $(y, x) \in E$, it takes $O(l|E|)$ time. To compute the value of $Mincost$ from these values, the algorithm takes utmost $O(|E|)$ time. □

3 Conditional Core Path of a Bipartite Permutation Graph with Vertex Weights

In this section, we give a polynomial time algorithm for the problem of finding conditional core path of a bipartite permutation graph $G = (X, Y, E)$(G has arbitrary vertex weights and unit edge weights). Let S ($\subset V$) denote

the subset of vertices of V where the facilities have already been located. As already defined, the **conditional cost of a path** P denoted by $d^c(P) = \sum_{v \in V} Min(d(v,P), d(v,S))w(v)$, where $d(v,P) = \underset{u \in P}{Min}\, d(v,u)$ and $d(v,S) = \underset{v' \in S}{Min}\, d(v,v')\ \forall v \in V$.

Computing d(v,S): Let $N(S) - S = \{v \in V - S | (v,u) \in E, u \in S\}$. We give here an efficient method to compute $d(v,S)\ \forall\ v \in V$.

$\forall\ v \in S$, initialize $d(v,S) = 0$.
$\forall\ v \in N(S) - S$ initialize $d(v,S) = 1$.
For all the remaining vertices, initialize $d(v,S) = \infty$.

We first give a method to find $d(x_i,S)\ \forall x_i \in X$. A similar procedure can be used to find $d(y_j,S)\ \forall y_j \in Y$.

1. For every vertex x_i, we define two vertices x_i^a, $x_i^b \in X \cap S$ such that $x_i^a < x_i < x_i^b$. Also
   - $d(x_i,x_i^a) \leq d(x_i,x')\ \forall x' \in X \cap S$ and $x' < x_i$
   - $d(x_i,x_i^b) \leq d(x_i,x')\ \forall x' \in X \cap S$ and $x' > x_i$
2. For every vertex x_i, we define two vertices x_i^c, $x_i^d \in Y \cap S$ such that $x_i^c < L(x_i) \leq R(x_i) < x_i^d$. Also
   - $d(x_i,x_i^c) \leq d(x_i,y')\ \forall y' \in Y \cap S$ and $y' < L(x_i)$
   - $d(x_i,x_i^d) \leq d(x_i,y')\ \forall y' \in Y \cap S$ and $y' > R(x_i)$

Clearly, if $d(x_i,S) \neq 0$ or 1, then $d(x_i,S) = \text{Min}\ \{d(x_i,x_i^a), d(x_i,x_i^b), d(x_i,x_i^c), d(x_i,x_i^d)\}$. If we find x_i^a, x_i^b, x_i^c and x_i^d, we can evaluate the above Min function and find $d(x_i,S)$. We now state two lemmas from [4] as the following *Remark*.

Remark 3

1. Suppose $i < j < k$. Then $d(x_i,x_j) \leq d(x_i,x_k)$.
2. Suppose x_i is not adjacent to y_j or y_k, $x_i < L(y_j)$, and $j < k$. Then $d(x_i,y_j) \leq d(x_i,y_k)$.

The above *Remark* characterizes x_i^a to be the maximum indexed vertex in $X \cap S$, that lie above x_i. Similarly, x_i^b is the minimum indexed vertex in $X \cap S$, that lie below x_i. Also, x_i^c is the maximum indexed vertex in $Y \cap S$, that lie above $L(x_i)$ and x_i^d is the minimum indexed vertex in $Y \cap S$, that lie below $R(x_i)$. We use the above property to find $d(x_i,x_i^a)$ efficiently for all the vertices. We can find x_i^a, $\forall x_i \in X$, using the following code.

1. $s_{up} = NIL$.
2. For $i = 1$ to $|X|$ If $x_i \in S$ then $s_{up} = x_i$. Else $x_i^a = s_{up}$.

From [4] we know that, for a bipartite permutation graph, after $O(n^2)$ preprocessing, the value of the shortest distance between any given pair of vertices can be computed in $O(1)$ time. Hence we can compute $d(x_i,x_i^a)\ \forall x_i \in X$, in $O(|X|)$ time. Since $x_i^c = (L(x_i))^a$, we note that $d(x_i,x_i^c) = 1 + d(L(x_i),(L(x_i))^a)$. Similarly we calculate all the required values and $d(v,S)\ \forall v \in V$ in $O(|V| + |E|)$ time.

Finding the cost of any ordered path: We first set $w(v) = 0, \forall v \in S$ because for any path P, cost due to these vertices is zero and setting $w(v) = 0$ simplifies some calculations that follow. We now show that for a path P, the conditional cost $d^c(P)$ can be calculated using *Lemma* 3 with a modification to the definition of $USUM$ and $BSUM$ to comply with the definition of conditional cost of a path. For all edges, $(x_i, y_j) \in E(G)$ we define,

$$USUM(x_i, y_j) = \sum_{v \in U(x_i, y_j)} min(d(v, x_i), d(v, y_j), d(v, S))w(v)$$

In order to calculate the value of this newly defined $USUM(x_i, y_j)$ efficiently, we define the following terms

$$\tau(x_i, y_j) = \{v : v \in U(x_i, y_j)\ ,\ d(v, (x_i, y_j)) < d(v, S)\}$$
$$TOADD(x_i, y_j) = \sum_{v \in \tau(x_i, y_j)} w(v)$$

Lemma 5. *The following equation can be used to compute $USUM(x_i, y_j)$ iteratively:*

$USUM(x_1, y_1) = 0$. Let $x' = L(y_j)$ and $y' = L(x_i)$.
$USUM(x_i, y_j) = USUM(x', y') + TOADD(x', y') + W_{UAdj}(x_i, y_j)$.

Proof in Appendix

In order to calculate $TOADD(x_i, y_j)$ we need some more definitions which we present below.

$Q(x_i, y_j) = \{v \in U(x_i, y_j) :\ d(v, x_i) = d(v, S)$ and $d(v, y_j) = d(v, S) + 1\}$
Also, $W(Q(x_i, y_j)) = \sum_{v \in Q(x_i, y_j)} w(v)$.

Algorithm 1. Algorithm to compute $W(Q(x_i, y_j))$

```
for all v ∈ V do
  for i = 1 to |X| do
    for j = 1 to |Y| do
      if d(v, x_i) = d(v, S) and d(v, y_j) = d(v, S)+1 then
        W(Q(x_i, y_j)) = W(Q(x_i, y_j)) + w(v)
      else if d(v, y_j) = d(v, S) and d(v, x_i) = d(v, S)+1 then
        W(Q(y_j, x_i)) = W(Q(y_j, x_i)) + w(v)
      end if
    end for
  end for
end for
```

Lemma 6. *Algorithm 1 computes $W(Q(x_i, y_j))$ in $O(|V|.|E|)$ time.*

Proof. From [4] we know that, for a bipartite permutation graph, after $O(n^2)$ preprocessing, the value of the distance between any given pair of vertices can

be computed in $O(1)$ time. From this, it immediately follows that the time complexity of Algorithm 1 is $O(|V|.|E|)$ □

Lemma 7. *The following equation can be used to compute* $TOADD(x_i, y_j)$ *iteratively.*

$$TOADD(x_i, y_j) = \begin{cases} 0 & \text{if } i, j = 1 \\ TOADD(x_{i-1}, y_j) + w(x_{i-1}) - W(Q(y_j, x_i)) + W(Q(y_j, x_{i-1})) & \text{if } i \neq 1,\ j = index(L(x_i)) \\ TOADD(x_i, y_{j-1}) + w(y_{j-1}) - W(Q(x_i, y_j)) + W(Q(x_i, y_{j-1})) & \text{otherwise} \end{cases}$$

Proof in Appendix
Note that we compute $TOADD(x_i, y_j)$ along-side while computing $USUM$. We similarly calculate $BSUM$ where

$$BSUM(x_i, y_j) = \sum_{v \in B(x_i, y_j)} min(d(v, x_i), d(v, y_j), d(v, S))w(v).$$

Lemma 8. *For any ordered path* P*, the conditional cost can be computed in* $O(1)$ *time after* $O(|V||E|)$ *preprocessing.*

Proof. The conditional cost for a path P is given by $d(P) = USUM(x_a, y_a) + BSUM(x_b, y_b) + W$ - $W(P)$ - $W_B(x_b, y_b)$ - $W_U(x_a, y_a)$. Here $USUM$ and $BSUM$ is as defined in this section. All other definitions is same as given in *Lemma* 3 and the proof also follows in exactly same fashion. □

Finding the Conditional Core Path: The conditional core path should be vertex disjoint from S by definition. Let H be the graph induced by $V(G)$ - S. We ought to search for the conditional core path in H. However, while calculating the conditional cost of the path, we must use the vertices in the entire graph G. Note that, we can modify *Remark* 1 to use conditional cost as follows.

Remark 4. For every edge $(x, y) \in E(H)$ let $path^l_{xy}$ denote the path with (x, y) as its *last edge* such that it is the minimum conditional cost ordered path of length l with (x, y) as its *last edge*.

Now, we can use the dynamic programming equations given in *Lemma* 4 on the graph H (instead of G) to find the conditional core path due to the validity of *Remark* 4. But to calculate the conditional cost, we use the definitions stated in this section and *Lemma* 8.

Theorem 2. *The conditional core path of a bipartite permutation graph can be computed in* $O(|V||E|)$ *time.* □

4 Conclusion

In this paper, we have presented an $O(l|E|)$ time algorithm for finding the core path of specific length l in vertex weighted bipartite permutation graphs, threshold graphs and proper interval graphs. We have extended our study of core path problem to the conditional core path problem on the same graph classes. For

the conditional core path problem of specified length, we have presented $O(l|E|)$ time algorithms for threshold and proper interval graphs and $O(|V||E|)$ time algorithm for bipartite permutation graphs. In all the three classes of graphs, due to their inherent property of vertex ordering we were able to conceptualize the notion of ordered paths. However, such a notion of ordered paths (i.e. for every path there exists an ordered path of the same vertex set) is not valid on interval or permutation graphs. Also, the complexity of the longest path problem is still unresolved in interval graph [12] and thus even the existence of a path of length l in interval graphs is still unresolved. Therefore, the techniques used in this paper cannot be directly applied to interval or permutation graphs and hence the problem of finding core path in them is an interesting open problem in this direction.

References

1. Alstrup, S., Lauridsen, P.W., Sommerlund, P., Thorup, M.: Finding cores of limited length. In: Rau-Chaplin, A., Dehne, F., Sack, J.-R., Tamassia, R. (eds.) WADS 1997. LNCS, vol. 1272, pp. 45–54. Springer, Heidelberg (1997)
2. Becker, R., Lari, I., Scozzari, A., Storchi, G.: The location of median paths on grid graphs. Annals of Operations Research 150(1), 65–78 (2007)
3. Becker, R.I., Chang, Y.I., Lari, I., Scozzari, A., Storchi, G.: Finding the l-core of a tree. Discrete Appl. Math. 118(1-2), 25–42 (2002)
4. Chen, L.: Solving the shortest-paths problem on bipartite permutation graphs efficiently. Inf. Process. Lett. 55(5), 259–264 (1995)
5. Hakimi, S.L., Schmeichel, E.F., Labb'e, M.: On locating path- or tree-shaped facilities on networks. networks 23(6), 543–555 (1993)
6. Brandstadt, A., Spinrad, J., Stewart, L.: Bipartite Permutation Graphs. Discrete Applied Mathematics 18, 279–292 (1987)
7. Minieka, E.: The optimal location of a path or a tree in a tree network. Networks 15, 309–321 (1985)
8. Minieka, E., Patel, N.H.: On finding the core of a tree with a specified length. J. Algorithms 4(4), 345–352 (1983)
9. Morgan, C.A., Slater, P.J.: A linear algorithm for a core of a tree. J. Algorithms 1(3), 247–258 (1980)
10. Peng, S., Lo, W.-T.: Efficient algorithms for finding a core of a tree with a specified length. J. Algorithms 20(3), 445–458 (1996)
11. Tamir, A., Puerto, J., Mesa, J.A., Rodríguez-Chía, A.M.: Conditional location of path and tree shaped facilities on trees. J. Algorithms 56(1), 50–75 (2005)
12. Uehara, R., Uno, Y.: On computing longest paths in small graph classes. Int. J. Found. Comput. Sci. 18(5), 911–930 (2007)
13. Wang, B.-F., Ku, S.-C., Hsieh, Y.-H.: The conditional location of a median path. In: Ibarra, O.H., Zhang, L. (eds.) COCOON 2002. LNCS, vol. 2387, pp. 494–503. Springer, Heidelberg (2002)

The Planar k-Means Problem is NP-Hard*

Meena Mahajan[1], Prajakta Nimbhorkar[1], and Kasturi Varadarajan[2]

[1] The Institute of Mathematical Sciences, Chennai 600 113, India
{meena,prajakta}@imsc.res.in
[2] The University of Iowa, Iowa City, IA 52242-1419 USA
kvaradar@cs.uiowa.edu

Abstract. In the k-means problem, we are given a finite set S of points in $\Re^m$, and integer $k \geq 1$, and we want to find k points (centers) so as to minimize the sum of the square of the Euclidean distance of each point in S to its nearest center. We show that this well-known problem is NP-hard even for instances in the plane, answering an open question posed by Dasgupta [6].

1 Introduction

In the k-means problem, we are given a finite set S of points in $\Re^m$, and integer $k \geq 1$, and we want to find k points (centers) so as to minimize the sum of the square of the Euclidean distance of each point in S to its nearest center. This is a well-known and popular clustering problem that has also received a lot of attention in the algorithms community.

Lloyd [16] proposed a very simple and elegant local search algorithm that computes a certain local (and not necessarily global) optimum for this problem. Har-Peled and Sadri [10] and Arthur and Vassilvitskii [3, 4] examine the question of how quickly this algorithm and its variants converge to a local optimum. Lloyd's algorithm also does not provide any significant guarantee about how well the solution that it computes approximates the optimal solution. Ostrovsky et al. [18] and Arthur and Vassilvitskii [5] show that randomized variants of Lloyd's algorithm can provide reasonable approximation guarantees.

The k-means problem has also been studied directly from the point of view of approximation algorithms. There are polynomial time algorithms that compute a constant factor approximation to the optimal solution; see for instance the local search algorithm analyzed by Kanungo et al. [12]. If k, the number of centers, is a fixed constant, then the problem admits polynomial-time approximation schemes [7, 13]. If both k and the dimension m of the input are fixed, the problem can be solved exactly in polynomial time [11].

Drineas et al. [8], Aloise et al. [1], and Dasgupta [6] show that the k-means problem is NP-hard when the dimension m is part of the input even for $k = 2$.

* Part of the work by the third author was done when visiting The Institute of Mathematical Sciences, Chennai. He was also supported by NSF CAREER award CCR 0237431.

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 274–285, 2009.

However, to the best of our knowledge, there is no known NP-hardness result when the dimension m is fixed and k, the number of clusters, is part of the input. Dasgupta [6] raises the question of whether k-means is hard in the plane.

In this paper, we establish the NP-hardness of the k-means problem in the plane. Our proof uses a reduction from the planar 3SAT problem [15] and is inspired by a construction used by Megiddo and Supowit [17] in the context of showing the NP-hardness of the k-center and the k-median problem.

While clustering problems generally tend to be NP-hard even in the plane, there are surprising exceptions – the problem of covering a set of points by k balls so as to minimize the sum of the radii of the balls admits a polynomial time algorithm if we use L_1 balls, and a $(1+\varepsilon)$-approximation algorithm that runs in time polynomial in the input size and $\log \frac{1}{\varepsilon}$ for the usual Euclidean balls [9].

The rest of this article is organized as follows. In Section 2, we define the problem formally and state some useful properties of the optimal clustering. In Section 3, we describe our reduction from planar 3SAT to the k-means in the plane.

2 Preliminaries

The problem of k-means clustering is defined as follows:

Definition 1. *Given a set of n points $S = \{p_1, \ldots, p_n\}$ in m dimensions, find a set of k points $B = \{b_1, b_2, \ldots, b_k\}$ such that*

$$\sum_{i=1}^{n} [d(p_i, B)]^2$$

is minimized. This minimum value is denoted $Opt(S, k)$.

Here $d(p_i, B)$ is the Euclidean distance from p_i to the nearest point in B; $d(p_i, B) = \min_{1 \le j \le k} d(p_i, b_j)$.

We consider the problem for $m = 2$, and refer to it as *planar k-means*.

Choosing the k centers B fixes a clustering $\mathcal{C}$ of the points in S, with each point going to its nearest center (breaking ties arbitrarily). On the other hand, if a set $C \subseteq S$ is known to form a cluster, then the center of the cluster is uniquely determined as the centroid of the points in C. Thus we can talk of the cost of a cluster $C = \{p_1, \ldots, p_m\}$ and a clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$:

$$\mathrm{Opt}(C, 1) = \mathrm{Cost}(C) = \min_b \sum_{i=1}^{m} [d(p_i, b)]^2$$

$$\mathrm{Cost}(\mathcal{C}) = \sum_{j=1}^{k} \mathrm{Cost}(C_j)$$

Thus $\mathrm{Opt}(S, k)$ is the minimum, over all clusterings $\mathcal{C}$ of S into k clusters, of $\mathrm{Cost}(\mathcal{C})$.

We show the NP-hardness of planar k-means by a reduction from planar 3-SAT [15].

Definition 2. *([15]) Let F be a 3-CNF formula with variables $\{v_1, \ldots, v_n\}$ and clauses $\{c_1, \ldots, c_m\}$. We call $G(F)$=(V, E) the graph of F, where*

$$\begin{aligned} V &= \{v_i | 1 \leq i \leq n\} \cup \{c_j | 1 \leq j \leq m\} \\ E &= E_1 \cup E_2 \text{ where} \\ E_1 &= \{(v_i, c_j) | v_i \in c_j \text{ or } \bar{v_i} \in c_j\} \\ E_2 &= \{(v_j, v_{j+1}) | 1 \leq j < n\} \cup \{\{v_n, v_1\}\} \end{aligned}$$

If $G(F)$ is a planar graph, F is called a planar 3-CNF formula. The planar 3-SAT problem is to determine whether a given planar 3-CNF formula F is satisfiable.

We note that our reduction, in fact, requires only the graph (V, E_1) to be planar. (Some of the literature in fact refers to this sub-graph as the graph of F, but we follow the convention from [15].)

Henceforth throughout this note, we use the term distance to mean square of Euclidean distance. That is, $\text{dist}(p, q) = [d(p, q)]^2$. We will be explicit when deviating from this convention.

We use the following well known or easily verifiable facts about the k-means problem [11, 8].

Proposition 1

1. *The cost of a cluster of points is half the average sum of distances from a point to the other points in the cluster:*

$$Cost(S) = \frac{1}{2|S|} \sum_{p \in S} \sum_{q \in S; q \neq p} dist(p, q)$$

2. *If, in an instance of the k-means problem, the given points form a multiset, then we say that a clustering is multiset-respecting if it puts all points at the same location into the same cluster.*
   Every instance of the k-means problem has a multiset-respecting optimal clustering.
3. *Let S be a multiset instance of the k-means problem, and let S' be the instance obtained by adding a point p to S. Then $\forall k$, $Opt(S, k) \leq Opt(S', k)$.*
4. *In particular, adding a point to a cluster cannot decrease the cost of that cluster; $Cost(S) \leq Cost(S + p)$.*
5. *If clustering $\mathcal{C}'$ refines clustering $\mathcal{C}$ (that is, every cluster in $\mathcal{C}$ is the union of some clusters in $\mathcal{C}'$), then $Cost(\mathcal{C}') \leq Cost(\mathcal{C})$.*

3 Reduction from Planar 3-SAT to k-Means

Let F be the given planar 3SAT instance with n variables and m clauses. The corresponding k-means instance I we construct will satisfy the following:

Properties of the Layout

1. Corresponding to each variable x_i, there is a simple circuit s_i in the plane, with an even number of vertices marked on it. At each vertex on such a circuit, M copies of a point are placed. The circuits for different variables do not intersect.

   For each circuit, its vertices can be partitioned into pairs of adjacent vertices in two ways. We associate one of them (chosen arbitrarily) with the assignment $x_i = 1$ and the other with $x_i = 0$. We call the first pairing the 'true matching' and the other pairing the 'false matching'.
2. Let u, v be any two distinct vertices taken from any of the circuits (not necessarily the same circuit). If u and v are adjacent on some circuit, then the distance between them is β. Otherwise, the distance between them is at least 2β.
3. There is a point p_j corresponding to every clause C_j. If $x_i \in C_j$ ($\bar{x_i} \in C_j$) then there is a unique nearest edge (u, v) on the true (respectively false) matching of the circuit s_i such that p_j is equi-distant from u and v. It is at distance α from the midpoint of uv, and hence at a distance $\alpha + \frac{\beta}{4}$ from u and v. All vertices other than the endpoints of these nearest edges (two per literal in the clause, so at most six) are at a distance at least $\alpha + \frac{5\beta}{4}$ from p_j.

   Clause points p_j and p_l, for $l \neq j$, are at distance at least θ from each other.
4. The instance I consists of all the clause points, and M copies of a point at each vertex on each circuit s_i. The parameters satisfy

$$M \geq \frac{6\alpha m}{\beta} \qquad\qquad \theta \geq 2(M+1)\alpha m$$

5. The value of k is given by

$$k = \sum_{i=1}^{n} \frac{|s_i|}{2}$$

We ensure that the optimal k-means clustering puts the points in each circuit s_i into $\frac{|s_i|}{2}$ clusters by dividing them into either true pairs or false pairs. (Thus these clusters contain $2M$ points.) Every clause point p_j has at most three pairs of point locations at distance α from itself. It is clustered with one of these pairs if that pair forms a cluster in the circuit s_i. Otherwise, the optimal clustering puts p_j along with some pair of point locations that forms a cluster in the circuit it appears in. In particular, if x_i is assigned a value 1, then in the corresponding k-means clustering, points of s_i are clustered according to the true matching, otherwise they are clustered according to the false matching. Similarly, if the assignment to a variable x_i satisfies a clause c_j, then the clause point p_j is at distance $\alpha + \frac{\beta}{4}$ from the vertices of a cluster in s_i, otherwise it is at distance strictly greater than $\alpha + \frac{\beta}{4}$ from at least one vertex in every cluster in s_i.

What is left now is to show that

1. A layout satisfying the above properties gives a correct reduction from planar 3-SAT to planar k-means.
2. The layout is indeed possible for some choice of α, β, θ, M, and can be obtained in polynomial time.

Consider clustering of only circuit points into k non-empty clusters.

Lemma 1

1. *Clustering the circuit points into consecutive pairs (i.e. into the true or the false matching for each variable) has cost* $\frac{kM\beta}{2}$.
2. *Any other multiset-respecting clustering of circuit points has cost at least* $\frac{kM\beta}{2} + \frac{M\beta}{3}$.

Proof. Let A be any matching-based k-means clustering of the circuit points. Then using Proposition 1(1) we can see that $\text{Cost}(A) = \frac{kM\beta}{2}$.

Let B be some multiset-respecting clustering that does not correspond to a matching on the circuits. By the size of a cluster, we mean the number of distinct vertices (and hence all M points at that vertex) in it.

If the largest cluster in B has 2 vertices, then every cluster is a pair, and at least one pair is not consecutive on any circuit. Hence

$$\text{Cost}(B) \geq \frac{(k-1)M\beta}{2} + \frac{M^2(2\beta)}{2M} = \frac{kM\beta}{2} + \frac{M\beta}{2}$$

satisfying the claimed bound.

So now assume that B has some larger clusters too. Let B contain p clusters of sizes $l_1, \ldots, l_p$ more than 3 each, q clusters of size 3 each, r clusters of size 2 each, and s clusters of size 1 each. Then we have the following:

$$p + q + r + s = k \tag{1}$$

$$\sum_{i=1}^{p} l_i + 3q + 2r + s = 2k \tag{2}$$

Subtracting twice the first equation from the second, and using $p = \sum_{i=1}^{p} 1$, we get

$$s = \sum_{i=1}^{p} (l_i - 2) + q \tag{3}$$

For a cluster C of size $l \geq 4$, the best possible situation is that l of the pairs are edges on some circuit. Thus the cost of such a cluster is at least

$$\text{Cost}(C) \geq \frac{1}{lM}\left[lM^2\beta + \left(\binom{l}{2} - l\right) M^2 2\beta\right] = (l-2)M\beta$$

Similarly, in a cluster of size 3, at most two pairs can be edges on a circuit (the circuits are of even length), so the cost is at least $4M\beta/3$.

The cost of the (p, q, r, s) clustering B thus satisfies:

$$\begin{aligned}
\text{Cost}(B) &\geq M\beta\left[\sum_{i=1}^{p}(l_i - 2) + \frac{4q}{3} + \frac{r}{2}\right] \\
&= \frac{M\beta}{2}\left[s + r + q + \frac{2q}{3} + \sum_{i=1}^{p}(l_i - 2)\right] \qquad \text{from } (\textit{Equation } 3) \\
&\geq \frac{M\beta}{2}\left[s + r + q + p + \frac{2q}{3} + p\right] \qquad \because l_i - 2 \geq 2 \\
&= \frac{M\beta}{2}\left[k + p + \frac{2q}{3}\right] \qquad \text{from } (\textit{Equation } 1) \\
&\geq \frac{kM\beta}{2} + \frac{(p+q)M\beta}{3} \\
&\geq \frac{kM\beta}{2} + \frac{M\beta}{3} \qquad \because p + q \geq 1
\end{aligned}$$

Thus any multiset-respecting clustering of circuit points that is not a matching based clustering has a cost larger than the matching based clusterings, and the difference is at least $\frac{M\beta}{3}$. □

Lemma 2. *The formula is satisfiable if and only if there is a clustering of value at most* $\frac{kM\beta}{2} + \frac{2M}{2M+1}\alpha m$.

Proof. ($\Rightarrow$:) Consider one of the satisfying assignments of the formula. A clustering can be constructed from it as follows:

If $x_i = 1$ (respectively $x_i = 0$), cluster the points of s_i according to the true (false) matching. As every clause C_j is satisfied, fix one of the variables x_i that satisfies it. Put the clause point p_j with the nearest pair of s_i. If $x_i = 1$, then points of s_i are clustered into true matching pairs. Further, x_i appears in C_j in non-negated form, and so, by our construction, p_j is at a distance α from the midpoint of one of the true matching pairs. Thus p_j can be clustered with this pair. The cost of this cluster is

$$\begin{aligned}
\text{Cost(cluster)} &= \frac{1}{2M+1}\left(M^2\beta + 2M\left(\alpha + \frac{\beta}{4}\right)\right) \\
&= \frac{2M}{2M+1}\alpha + \frac{M\beta}{2}
\end{aligned}$$

The case $x_i = 0$ is analogous. Clustering all clause points in this way gives a clustering where m clusters contribute $\frac{M\beta}{2} + \frac{2M}{2M+1}\alpha$ each, and the remaining contribute $\frac{M\beta}{2}$ each, giving an overall value of $\frac{kM\beta}{2} + \frac{2M}{2M+1}\alpha m$.

($\Leftarrow$:) Suppose there is a clustering of value at most $\frac{kM\beta}{2} + \frac{2M}{2M+1}\alpha m$. By Proposition 1(2), we can assume that there is a multiset-respecting clustering $\mathcal{C}$ with this value. Let $\mathcal{C}'$ denote the restriction of $\mathcal{C}$ to circuit points. By Proposition 1(4), adding the clause points cannot decrease the cost of the clustering; thus $\text{Cost}(\mathcal{C}') \leq \text{Cost}(\mathcal{C})$. Now we prove a series of claims:

1. The restriction of $\mathcal{C}$ to circuit points, $\mathcal{C}'$, has exactly k non-empty clusters. If $\mathcal{C}'$ has fewer clusters, then there is a cluster with more than 2 points. Refine the clustering by removing one point from such a cluster and putting it in a cluster by itself. Repeat until there are exactly k clusters, to get clustering $\mathcal{C}''$ of circuit points. By Proposition 1(5), $\text{Cost}(\mathcal{C}'') \leq \text{Cost}(\mathcal{C}')$. $\mathcal{C}''$ is not matching-based (since we created singleton clusters), so by Lemma 1, it contributes a value of at least $\frac{kM\beta}{2} + \frac{M\beta}{3}$. Since

$$\frac{kM\beta}{2} + \frac{M\beta}{3} \leq \text{Cost}(\mathcal{C}'') \leq \text{Cost}(\mathcal{C}') \leq \text{Cost}(\mathcal{C}),$$

we have

$$\text{Cost}(\mathcal{C}) - \left(\frac{kM\beta}{2} + \frac{2M}{2M+1}\alpha m\right) \geq \frac{M\beta}{3} - \frac{2M}{2M+1}\alpha m \geq \frac{M\beta}{6} > 0,$$

where the second inequality follows by our choice of M. We have reached a contradiction to our assumption about $\text{Cost}(\mathcal{C})$.

2. $\mathcal{C}'$ is a matching-based clustering. That is, in $\mathcal{C}$, all circuit points are clustered into a matching based clustering.
If not, then by the argument used above for $\mathcal{C}''$, we obtain a contradiction to our assumption about $\text{Cost}(\mathcal{C})$.

3. No cluster in $\mathcal{C}$ has more than one clause point.
If some cluster C has two or more clause points, then let u, v be the vertices of the matching in C, and let p, q be two distinct clause points in it. By Proposition 1(4), the cost of the cluster C is at least the cost of the cluster containing just u, v, p, q. Thus using Proposition 1(1), we have

$$\text{Cost}(C) \geq \frac{1}{2M+2}\left[M^2\beta + 4M\left(\alpha + \frac{\beta}{4}\right) + \theta\right] = \frac{M\beta}{2} + \frac{4M\alpha + \theta}{2(M+1)}$$

All other clusters have a cost of at least $M\beta/2$ each, so the overall cost is at least

$$\begin{aligned}
\text{Cost}(\mathcal{C}) &\geq (k-1)\frac{M\beta}{2} + \frac{M\beta}{2} + \frac{4M\alpha + \theta}{2(M+1)} \\
&\geq \frac{kM\beta}{2} + \frac{4M\alpha + 2(M+1)\alpha m}{2(M+1)} \qquad \text{by our choice of } \theta \\
&> \frac{kM\beta}{2} + \frac{2M}{2M+1}\alpha m \qquad \text{a contradiction.}
\end{aligned}$$

4. Each clause point is clustered with the nearest pair of circuit points, which should also be a matching pair in the matching based clustering.
Every cluster containing a clause point has cost $\frac{M\beta}{2} + \frac{2M\alpha}{2M+1}$ if the circuit edge in the cluster is nearest the clause point, and has cost at least $\frac{M\beta}{2} + \frac{2M}{2M+1}\alpha + \frac{M\beta}{2M+1}$ otherwise.

Thus a satisfying assignment can be constructed from this clustering. □

We now describe the layout obtained from the planar 3SAT formula F that gives us the desired instance I of the k-means problem. Let $G = (V, E)$ be the associated planar clause-variable incidence matrix. (From Definition 2, $G = (V, E_1)$. Since $G(F)$ is planar, so is G.) Note that the vertex set V of G can be partitioned into two sets: X corresponding to variable vertices, and Y corresponding to clause vertices, and G is bipartite with $E \subset X \times Y$. All vertices in Y have degree at most 3, and all vertices in X have degree at most m.

1. Let $\mathcal{E}$ be a planar combinatorial embedding of G; such an embedding can be obtained in polynomial time, and even in log space. $\mathcal{E}$ corresponds to some plane drawing of G and specifies, for each vertex v, the cyclic ordering of the edges incident on v in this drawing.
2. Construct a related bounded-degree planar graph H and an embedding $\mathcal{E}'$ as follows: replace each vertex $v \in X$ by a cycle C_v on m vertices, $v_1, v_2, \ldots, v_m$. Reroute the $d(v)$ edges incident on v in G to the first $d(v)$ of these vertices, in the same order as dictated by $\mathcal{E}$. It is straightforward to see that H is planar, and its embedding $\mathcal{E}'$ can be easily obtained from $\mathcal{E}$. The maximum degree of any vertex in H is 3. The vertex set of H is the disjoint union of X' and Y, where $X' = X \times [m]$.
3. Consider a plane drawing of H where vertices are embedded at points on an integer grid, and edges are embedded as rectilinear paths. Such a drawing can be obtained in polynomial time [14, 19], and even in logarithmic space [2].
4. Inflate the grid by a factor of $b \geq 14$.
   This ensures, in particular, that every vertex or bend point u is at the centre of a big box B_u of size $b \times b$, and a small box S_u of size 6×6. The big boxes for different grid points have disjoint interiors, and thus contain no other vertex or bend point even on their boundaries.

   Consider an edge connecting vertex $[x, k] \in X'$ with vertex $y \in Y$. Replace it by a pair of parallel rectilinear paths separated by two grid squares. At the y end, join up these paths along the boundary of S_y. At the $[x, k]$ end, splice them along with the edges to $[x, k-1]$ and $[x, k+1]$ to form a continuous path. See Figure 1

   For each vertex $x \in X$ (and hence for each variable in F), this process distorts the cycle C_x in H into a circuit t. Let t_i denote the circuit corresponding to variable x_i. Since t_i is a rectilinear circuit on a grid, it is of even length.

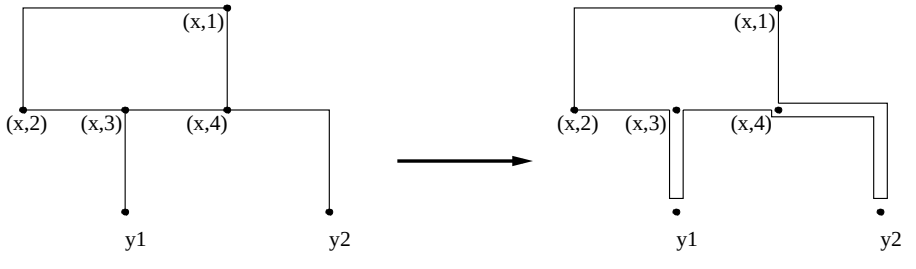


Fig. 1. Creating circuits for variables

5. Each clause point y_j is now moved to the center of one of the grid squares touching it, the one that is to the North-West. Extend the three circuits "incident" to the clause point, if necessary, so that all incident circuits are at a Euclidean distance of precisely $\frac{5}{2}$ times the grid length from the moved clause point. See the layout and the modification in Figure 2

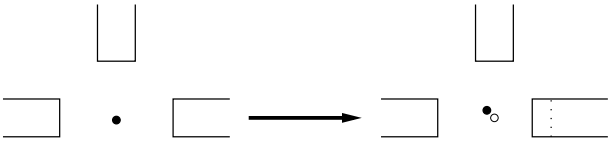


Fig. 2. Repositioning clause points

6. For each circuit t_i, arbitrarily fix one of its perfect matchings as the true matching, and the other as the false matching.

   Let clause C_j contain variable x_i positively (negatively, respectively). If in the layout so far, y_j is nearest a true (false, resp.) edge of t_i, then nothing needs to be done. If, however, the edge of t_i nearest y_j is a false (true, resp.) edge, further deform t_i in the area within B_{y_j} but outside S_{y_j}. Replace a sub-path of length two (on each parallel path) by a path of length three, with the vertices laid out on a regular semi-hexagon and hence at distance one from their neighbours on the circuit. Change the true/false matchings within B_{y_j} to be consistent with the labelling outside. This makes the edge nearest y_j a true edge if it was false earlier, and vice versa. The overall length of the circuit remains even. See Figure 3.

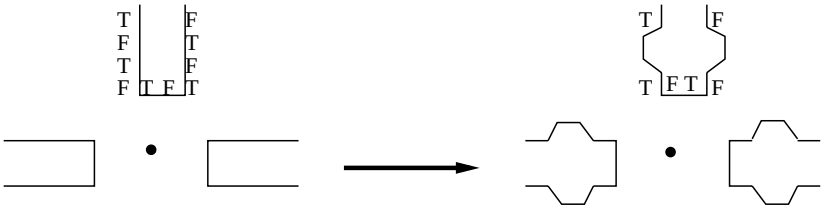


Fig. 3. Adjusting the parity of circuits relative to clause points

   Since the grid was inflated sufficiently, these distortions do not affect other vertices / bends.

   After doing this distortion wherever needed, the resulting circuit for a variable is the required circuit s_i.

Figure 4 gives the complete layout for a small planar 3SAT instance.

Let the squared unit length of the grid be β. Then $\alpha = (\frac{5}{2})^2\beta = 6.25\beta$. Any two clause points are separated either vertically or horizontally by b grid lengths, so the distance between them is at least $\theta = b^2\beta$.

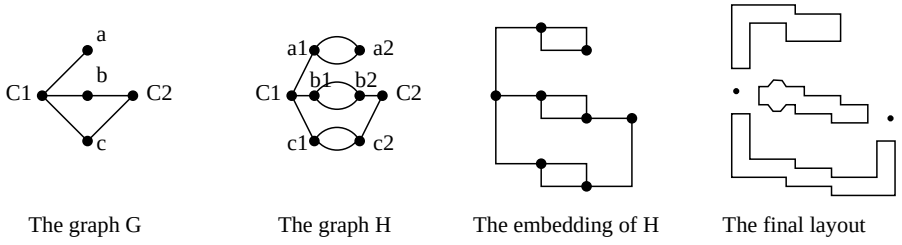


Fig. 4. The layout for $F = (a \vee b \vee c) \wedge (\overline{b} \vee c)$

Figure 5 shows the box B_u for a clause point u, and within this box the smallest distances are demonstrated. The nearest vertices are A_3 and A_4 (and the corresponding vertices on the other two circuits as well). The distances satisfy the following:

point pair	distance	point pair	distance
		u, A_2	$\alpha + 6\beta + \beta/4$
A_i, A_{i+1}	β	u, A_3	$\alpha + \beta/4$
A_1, A_3	3β	u, A	α
A_2, A_4	2β	u, A_4	$\alpha + \beta/4$
A_3, A_5	4β	u, A_5	$\alpha + 2\beta + \beta/4$

It is straightforward to see that with $M = 38m$, $b = 28m$, all the required conditions on the parameters are satisfied.

Dealing with the irrational coordinates: In the last step of the layout, where we replace certain sub-paths of length 2 by sub-paths of length 3, the numerators of the point coordinates become irrational. Essentially, we introduce multiples of $\sqrt{3}$ in the numerator. However, there is a gap in Lemma 2 between the k-means clustering costs corresponding to satisfiable and unsatisfiable instances. So we may "round" these irrational points to sufficiently close rational points.

Lemma 2 shows that the optimal k-means clustering cost is at most $\mu = \frac{kM\beta}{2} + \frac{2M}{2M+1}\alpha m$ in the case where the original planar 3-CNF formula is satisfiable; a quick glance at the proof shows that if the original formula is not satisfiable, the optimal k-means clustering cost is at least $\mu + \lambda$, where

$$\lambda = \min\left(\frac{M\beta}{6}, \frac{2M\alpha}{M+1}, \frac{M\beta}{2M+1}\right).$$

Let $\varepsilon = \frac{\lambda/2}{\mu+\lambda} < \frac{\lambda/2}{\mu}$.

Let d_{min} denote the smallest Euclidean distance between two different locations in the layout. We consider a simplistic rounding: for a location with coordinates (x, y) where, say, x is irrational (only one coordinate is irrational in the construction so far), move the location to (x', y) so that x' is rational, and

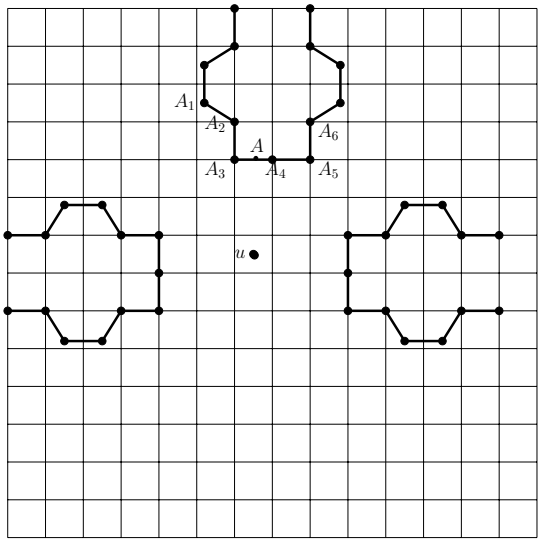


Fig. 5. The distances α, β shown inside B_u for a clause point u

$|x' - x| < \frac{\varepsilon d_{min}}{8}$. Observe that it is possible to find such an x' in polynomial time. Now if p and q denote two points in the original layout, and p' and q' denote the corresponding points in the rounded layout, then

$$d(p', q') < d(p, q) + \frac{\varepsilon d_{min}}{4} \leq d(p, q) \left[1 + \frac{\varepsilon}{4}\right]$$

(recall, $d(u, v)$ is the Euclidean distance between u and v) and so

$$\text{dist}(p', q') < \text{dist}(p, q) \left[1 + \frac{\varepsilon}{2} + \frac{\varepsilon^2}{8}\right] \leq \text{dist}(p, q) \left[1 + \varepsilon\right]$$

Similarly, we can see that $\text{dist}(p', q') > \text{dist}(p, q) \left[1 - \varepsilon\right]$. Thus,

$$(1 - \varepsilon)\text{dist}(p, q) < \text{dist}(p', q') < (1 + \varepsilon)\text{dist}(p, q).$$

Now, Proposition 1 (1) implies that for any clustering of the points, the ratio of the cost in the rounded layout to the cost in the original layout is strictly greater than $(1 - \varepsilon)$ and strictly less than $(1 + \varepsilon)$.

Thus, the optimal k-means clustering cost in the rounded layout is strictly less than $(1 + \varepsilon)\mu \leq \mu + \lambda/2$ when the input formula is satisfiable, and is strictly greater than $(1 - \varepsilon)(\mu + \lambda) \geq \mu + \lambda/2$ when the input formula is not satisfiable.

References

[1] Aloise, D., Deshpande, A., Hansen, P., Popat, P.: NP-Hardness of Euclidean Sum-of-Squares Clustering. Technical Report G-2008-33, Les Cahiers du GERAD (to appear in Machine Learning) (April 2008)

[2] Allender, E., Datta, S., Roy, S.: The directed planar reachability problem. In: Ramanujam, R., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 238–249. Springer, Heidelberg (2005)

[3] Arthur, D., Vassilvitskii, S.: Worst-case and smoothed analysis of the ICP algorithm, with an application to the k-means method. In: Proc. IEEE Symp. Foundations of Computer Science (2006)

[4] Arthur, D., Vassilvitskii, S.: How slow is the k-means method? In: Proc. Symp. on Comput. Geom. (2006)

[5] Arthur, D., Vassilvitskii, S.: k-means++: The advantages of careful seeding. In: Proc. ACM-SIAM Symp. Discrete Algorithms (2007)

[6] Dasgupta, S.: The hardness of k-means clustering. Technical Report CS2007-0890, University of California, San Diego (2007)

[7] de la Vega, F., Karpinski, M., Kenyon, C.: Approximation schemes for clustering problems. In: Proc. ACM Symp. Theory of Computing, pp. 50–58 (2003)

[8] Drineas, P., Friexe, A., Kannan, R., Vempala, S., Vinay, V.: Clustering large graphs via the singular value decomposition. Machine Learning 56, 9–33 (2004)

[9] Gibson, M., Kanade, G., Krohn, E., Pirwani, I., Varadarajan, K.: On clustering to minimize the sum of radii. In: Proc. ACM-SIAM Symp. Discrete Algorithms (2008)

[10] Har-Peled, S., Sadri, B.: How fast is the k-means method? In: Proc. ACM-SIAM Symp. Discrete Algorithms, pp. 877–885 (2005)

[11] Inaba, M., Katoh, N., Imai, H.: Applications of weighted Voronoi diagrams and randomization to variance-based clustering. In: Proc. Annual Symp. on Comput. Geom., pp. 332–339 (1994)

[12] Kanungo, T., Mount, D., Netanyahu, N., Piatko, C., Silverman, R., Wu, A.: A local search approximation algorithm for k-means clustering. Comput. Geom. 28, 89–112 (2004)

[13] Kumar, A., Sabharwal, Y., Sen, S.: A simple linear time $(1 + \varepsilon)$ approximation algorithm for k-means clustering in any dimensions. In: Proc. IEEE Symp. Foundations of Computer Science, pp. 454–462 (2004)

[14] Leiserson, C.E.: Area-efficient graph layouts (for VLSI). In: Proc. 21st Ann. IEEE Symp. Foundations of Computer Science, pp. 270–281 (1980)

[15] Lichtenstein, D.: Planar formulae and their uses. SIAM J. Comput. 11, 329–343 (1982)

[16] Lloyd, S.: Least squares quantization in PCM. IEEE Transactions on Information Theory 28, 129–136 (1982)

[17] Megiddo, N., Supowit, K.: On the complexity of some common geometric location problems. SIAM J. Comput. 13, 182–196 (1984)

[18] Ostrovsky, R., Rabani, Y., Schulman, L., Swamy, C.: The effectiveness of Lloyd-type methods for the k-means problem. In: Proc. IEEE Symp. Foundations of Computer Science (2006)

[19] Valiant, L.G.: Universality considerations in VLSI circuits. IEEE Transactions on Computers 30, 135–140 (1981)

On the Computational Complexity of Monotone Constraint Satisfaction Problems

Miki Hermann and Florian Richoux

LIX (CNRS, UMR 7161), École Polytechnique, 91128 Palaiseau, France
{hermann,richoux}@lix.polytechnique.fr

Abstract. Constraint Satisfaction Problems (CSP) constitute a convenient way to capture many combinatorial problems. The general CSP is known to be NP-complete, but its complexity depends on a parameter, usually a set of relations, upon which they are constructed. Following the parameter, there exist tractable and intractable instances of CSPs. In this paper we show a dichotomy theorem for every finite domain of CSP including also disjunctions. This dichotomy condition is based on a simple condition, allowing us to classify monotone CSPs as tractable or NP-complete. We also prove that the meta-problem, verifying the tractability condition for monotone constraint satisfaction problems, is fixed-parameter tractable. Moreover, we present a polynomial-time algorithm to answer this question for monotone CSPs over ternary domains.

1 Introduction

Constraint Satisfaction Problems (CSP) constitute a common formalism to describe many algorithmic problems originating from combinatorics and graph theory, artificial intelligence, computational molecular biology, etc. These problems require a quantitative analysis studying their computational complexity. The goal to study the complexity of constraint satisfaction problems is the recognition of conditions allowing us to distinguish between tractable and intractable instances of the considered problem, as well as the understanding of the complexity classes to which these instances belong. The study of computational complexity of constraints satisfaction problems was started by Schaefer in his landmark paper [9], where he completely characterized the complexity of Boolean CSP, distinguishing between polynomial and NP-complete instances. Feder and Vardi [4] extended this study to constraint satisfaction problems over finite domains, for which they conjectured the existence of a Dichotomy Theorem. So far, this claim was proved only for the ternary domain by Bulatov [1], exhibiting an involved Dichotomy Theorem, whereas the claim remains open for higher cardinality domains.

The main difficulty to resolve this conjecture resides in the nonexistence of a finite or countable number of function sets, called clones, under which the relations building the CSPs are closed. The closure under the functions of a given clone provides us with the required characterization of (in)tractability. There is a countably infinite number of clones for the Boolean domain and the cases discriminating between tractable and NP-complete instances appear in a finite part of the lattice. The characterization for each clone can always be expressed by a finite set of function, called a basis. These facts were

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 286–297, 2009.

largely exploited by Schaefer in [9]. However, the corresponding lattice for domains of cardinality 3 and more is uncountable, as it was proved by Yanov and Muchnik [10].

One possibility to circumvent this difficulty is to introduce more structure into the constraint satisfaction problems, hoping to find only a finite number of cases to be analyzed. To introduce more structure, we choose to extend the constraint satisfaction problems with disjunction. Usual CSPs consider a set of constraints logically interpreted as a conjunction. Introducing a disjunction connective among constraints might seem to destroy this correspondence between the logic and set-theoretic approaches. If we consider each set of relations including the disjunction, we can easily recover the aforementioned correspondence. Our approach explores that particular part of constraint satisfaction problems that always include the disjunction relation. Monotone CSPs have been also considered from a different viewpoint by Cohen, Jeavons, Jonsson and Koubarakis in [2]. A moments reflection might consider monotone constraint satisfaction problems as too restrictive. However, this is incorrect since they represent the CSPs over weak Krasner algebras, a large and well-studied structure in universal algebra [6, 7]. The advantage of weak Krasner algebras is that they are closed only under endomorphisms. Since there is only a finite number of endomorphisms on a given finite domain, we are ensured to obtain a finite complexity characterization.

In this paper we study the complexity of monotone constraint satisfaction problems, also considered to be the CSPs over weak Krasner algebras. We derive for them a complete characterization of complexity by means of a Dichotomy Theorem for each finite domain. Once the tractability conditions derived, we study also the complexity of the meta-problem, i.e., the complexity of the decision problem whether a given monotone CSP satisfies the tractability condition. This analysis is first performed for the general case of any domain cardinality. Since the ternary domain presents a particular behavior, we perform a new complexity analysis for its meta-problem, deriving a sharper result than for the general case.

2 Basic Notions

All along this paper, a function f is an unary function $f\colon D \to D$ with D being a fixed size domain. The *range* of f, denoted by $\operatorname{ran} f$, is the set $\{f(x) \mid x \in D\}$. We write $f(A)$ for a subset $A \subseteq D$ to denote the set $\{f(a) \mid a \in A\}$.

The study of constraint satisfaction problems often uses the notion of *relations*, *clones* and *co-clones*. A n-ary **relation** R on domain D is a subset of D^n. We denote by S a set of relations $R \subseteq D^k$ where k is the arity of R. A **clone**, or a **functional clone**, is a set of functions containing the identity and closed under composition. The smallest clone containing the functions F is denoted by $[F]$. In our scope, a **co-clone**, or a **relational clone**, is a set of relations containing the equality $eq = \{(d, d) \mid d \in D\}$ and closed under conjunction, *disjunction*, variable identification and existential quantification. The smallest co-clone containing the relations S is denoted by $\langle S \rangle$. Note that we authorize the closure under disjunction in addition to the usual definition of closure.

Universal algebra often uses the concept of *kernel*. We define this notion with the help of equivalence classes. Let $f\colon D \to D$ be a function with range $\operatorname{ran} f$. A *d-equivalence class*, for $d \in \operatorname{ran} f$, is the set of elements $[d]_f = \{x \in D \mid f(x) = d\}$.

The **kernel** $\ker f$ of a function f is the set of all equivalence classes with cardinality strictly greater than 1 for all $d \in \operatorname{ran} f$, *i.e.* $\ker f = \{[d]_f \mid d \in \operatorname{ran} f \text{ and } |[d]_f| \geq 2\}$.

We need the notion of function depth to describe the computational complexity of a clone $[F]$. The *depth* of a function f *with respect to* a set F, denoted by $\mathrm{Depth}(F, f)$, is the length of the shortest composition of functions $f_i \in F$ required to obtain f. The *depth* of a function f, denoted by $\mathrm{Depth}(f)$, is defined by the identity $\mathrm{Depth}(f) = \max(\mathrm{Depth}(F, f))$ for all sets F such that $f \in [F]$. Thus, if we want to obtain a function f from a set F, it is useless to make compositions of functions longer than $\mathrm{Depth}(f)$. If all combinations with length less or equal than $\mathrm{Depth}(f)$ do not allow us to get a function f, it means that f does not belong to the clone generated by F.

3 Monotone Constraint Satisfaction Problems

Constraint satisfaction problems are usually presented as conjunctions of constraints. The parametric problem $\textsc{csp}(S)$, with S being a set of relations, is a constraint satisfaction problem where constraints are built upon relations from S. Constraint satisfaction problems are known to be NP-complete in general, but the complexity of parametric problems $\textsc{csp}(S)$ depends on S, ranging from polynomial to NP-complete. Schaefer [9] gave a dichotomy theorem on Boolean constraints. If S verifies one of the six conditions presented by Schaefer, then $\textsc{csp}(S)$ is polynomial. Otherwise it is NP-complete.

It is difficult to obtain such a dichotomy for finite domains of higher cardinality. The only exception is a result from Bulatov [1] showing a dichotomy for ternary domains. A dichotomy theorem for other finite domains is still an open question. We will focus our interest on the complexity of parametric problems $\textsc{mcsp}(S)$ where MCSP is a generalization of CSP defined as follows: atomic constraints play the role of literals and constraints are built by means of conjunctions and disjunctions. In this paper, we present a dichotomy with simple conditions for the algorithmic problem $\textsc{mcsp}(S)$ with S being a set of relations over a finite domain D.

Let $R \subseteq D^k$ be a relation on D and V be a set of variables. A *literal* is a predicate $R(x_1, \ldots, x_k)$ formed from the relation R and variables $x_1, \ldots, x_k$ in V, where $\operatorname{ar}(R) = k$. A *formula* is defined inductively in the following way: (1) TRUE and FALSE, respectively denoted by $\top$ and $\bot$, are formulas; (2) a literal l is a formula; (3) if φ_1 and φ_2 are formulas, then $\varphi_1 \vee \varphi_2$ and $\varphi_1 \wedge \varphi_2$ are formulas; (4) finally if φ is a formula containing a free variable x then $\exists x\, \varphi$ is a formula. We do not consider negation in our formulas because it brings us to an obvious generalization of the problem $\textsc{csp}(S)$, which is NP-complete over every finite domain of cardinality greater than or equals to three, for each S. An *interpretation* $I \colon V \to D$ satisfies the literal $R(x_1, \ldots, x_k)$, denoted by $I \vDash R(x_1, \ldots, x_k)$, if $(I(x_1), \ldots, I(x_k)) \in R$. It is extended to formulas in the following way: (a) $I \vDash R_1(\boldsymbol{x}) \vee R_2(\boldsymbol{x})$ if $I \vDash R_1(\boldsymbol{x})$ or $I \vDash R_2(\boldsymbol{x})$, (b) $I \vDash R_1(\boldsymbol{x}) \wedge R_2(\boldsymbol{x})$ if $I \vDash R_1(\boldsymbol{x})$ and $I \vDash R_2(\boldsymbol{x})$. We note that for all formulas φ, we have $\top \vDash \varphi$ and $\bot \nvDash \varphi$.

We define the **monotone constraint satisfaction problem** as follows.

Problem: $\textsc{mcsp}(S)$
Input: A formula $\varphi(\boldsymbol{x})$.
Question: Is the formula $\varphi(\boldsymbol{x})$ satisfiable?

Our study of monotone constraint satisfaction problems is made easier by the existence of a Galois connection among clones and co-clones. Before defining what a Galois connection is, we need the following notions. Let R be a relation of arity $\mathrm{ar}(R) = k$ and f be a function of arity $\mathrm{ar}(f) = m$. We say that f is a polymorphism of R if the membership condition

$$(f(r_1[1], \ldots, r_m[1]), \ldots, f(r_1[k], \ldots, r_m[k])) \in R \tag{1}$$

is verified, with $r_i = (r_i[1], \ldots, r_i[k]) \in R$ for all $i \in \{1, \ldots, m\}$. The set of polymorphisms of a relation R is denoted by $\mathrm{Pol}\, R$. The set of polymorphisms of a relations set S is also denoted by $\mathrm{Pol}\, S$, defined by $\mathrm{Pol}\, S = \bigcap_{R \in S} \mathrm{Pol}\, R$. In a similar way, a relation R is an invariant of a function f if the condition (1) holds; we also say that R is closed under f. We denote by $\mathrm{Inv}\, f$ the invariants of f and $\mathrm{Inv}\, F$ the invariants of a set of functions F.

Pöschel proved in [7] that applying disjunction on constraints implies that relations satisfying an instance of a MCSP problem are invariant only under unary functions. Polymorphisms of these relations are thus endomorphisms and we denote them by End instead of Pol. Let us for a moment denote respectively by A and B sets of relations and functions. Pöschel [7] specifies that the mappings $\mathrm{End}\colon A \to B$ and $\mathrm{Inv}\colon B \to A$ present a Galois connection between the ordered structures $(A, \subseteq)$ and $(B, \subseteq)$. Moreover, for all sets of relations S and functions F, the identities $\mathrm{Inv}\,\mathrm{End}\, S = \langle S \rangle$ and $\mathrm{End}\,\mathrm{Inv}\, F = [F]$ hold. Pöschel points out that the sets $\langle S \rangle$ and $[F]$ are respectively a **weak Krasner algebra** and an **endomorphisms monoid**. In this scope we can consider the monotone constraint satisfaction problems as CSP defined upon weak Krasner algebras. The existence of the aforementioned Galois connection allows us to easily decide the complexity of monotone constraint satisfaction problems. The complexity analysis is based on the following result.

Proposition 1. *Let S_1 and S_2 be two sets of relations over the domain D, such that $eq \in S_1$. The inclusion* $\mathrm{End}\, S_1 \subseteq \mathrm{End}\, S_2$ *implies* $\textsc{mcsp}(S_2) \leq_m^{\mathrm{P}} \textsc{mcsp}(S_1)$.

Proof. From $\mathrm{End}\, S_1 \subseteq \mathrm{End}\, S_2$ follows $\langle S_1 \rangle = \mathrm{Inv}\,\mathrm{End}\, S_1 \supseteq \mathrm{Inv}\,\mathrm{End}\, S_2 = \langle S_2 \rangle$. Since every co-clone (equivalent to a weak Krasner algebra) contains the equality relation eq and is closed under conjunction, disjunction, and existential quantification, we immediately derive the reduction $\textsc{mcsp}(S_2) \leq_m^{\mathrm{P}} \textsc{mcsp}(S_2 \cup \{eq\}) \leq_m^{\mathrm{P}} \textsc{mcsp}(S_1 \cup \{eq\})$. Since $eq \in S_1$, we have $\textsc{mcsp}(S_1 \cup \{eq\}) = \textsc{mcsp}(S_1)$ and the result follows. □

According to Proposition 1, a complexity result proved for a set of relations S immediately extends to all relations in the co-clone $\langle S \rangle$, provided that S contains the equality relation eq. Therefore from now on we always assume that the **equality relation eq is included in every set S**. Moreover, the presence of the quaternary relation $J = \{(x, y, z, w) \in D^4 \mid (x = y) \vee (z = w)\}$ in a classical constraint satisfaction problem reduces it to a monotone constraint satisfaction problem.

Proposition 2. *Let S be a set of relations on the domain D and consider the relation $J = \{(x, y, z, w) \in D^4 \mid (x = y) \vee (z = w)\}$. If $J \in S$ then* $\mathrm{Pol}\, S = \mathrm{End}\, S$.

Proof. Suppose that there exists a binary function $g \in \operatorname{Pol} S$ depending on both arguments, i.e., there exist values $a_0, a_1, a_2, b_0, b_1, b_2 \in D$ such that $a_0 \neq a_1$, $b_1 \neq b_2$, $g(a_0, a_2) \neq g(a_1, a_2)$ and $g(b_0, b_1) \neq g(b_0, b_2)$. Clearly, the vectors $j_1 = (a_0, a_1, b_0, b_0)$ and $j_2 = (a_2, a_2, b_1, b_2)$ belong to J, but the vector $(g(a_0, a_2),\ g(a_1, a_2),\ g(b_0, b_1),\ g(b_0, b_2))$ is absent from J. Therefore the function g cannot be a polymorphism of a set of relations containing J. From any function f of arity $\operatorname{ar}(f) > 2$ we can always produce a binary function by variable identification. □

Since the number of endomorphisms over a finite domain is always finite, we are ensured to obtain a finite complexity characterization for MCSP.

4 Complexity of MCSP(S)

We will exhibit a dichotomy of MCSP(S) complexity characterized by a simple criterion. It is easier to prove this dichotomy for MCSP$(f(S))$ where f is a permutation keeping invariant the set S. We need the following proposition adapted from Jeavons [5] showing that the problems MCSP(S) and MCSP$(f(S))$ are logspace-equivalent.

Proposition 3 (Jeavons [5]). *Let S be a set of relations on D and f be an unary function on D. Let $f(S) = \{f(R) \mid R \in S\}$. If each relation $R \in S$ is closed under f then* MCSP$(f(S))$ *is logspace-equivalent to* MCSP(S).

Proof. Suppose that each relation $R \in S$ is closed under f. Let $\varphi(\boldsymbol{x})$ be an instance of MCSP$(f(S))$. We have a formula $\varphi(\boldsymbol{x})$ where literals $R(x_1, \ldots, x_k)$ are constructed from relations $R \in f(S)$. According to the hypothesis, all relations $R \in S$ are closed under f, *i.e.* the inclusion $f(R) \subseteq R$ holds for all $R \in S$. We deduce that $\varphi(\boldsymbol{x})$ is also an instance of MCSP(S).

Take a formula $\varphi(\boldsymbol{x})$ being an instance of MCSP(S). It can be transformed by a logspace reduction to an instance $\varphi'(\boldsymbol{x})$ of MCSP$(f(S))$ by replacing each literal constructed from a relation R by a literal constructed from $f(R)$. Moreover, because R is closed under f, we have $f(R) \subseteq R$ and we derive that all solutions of $\varphi'(\boldsymbol{x})$ are also solutions of $\varphi(\boldsymbol{x})$. Conversely, if h is a solution of $\varphi(\boldsymbol{x})$ then $f(h)$ is a solution of $\varphi'(\boldsymbol{x})$. Thus we have a logspace-equivalence among MCSP$(f(S))$ and MCSP(S). □

We will study monotone constraint satisfaction problems MCSP(S) through sets of functions F satisfying $\operatorname{Inv} F = S$. The following propositions prove that MCSP(S) is polynomial if $[F]$ contains a constant function, and it is NP-complete otherwise.

Proposition 4. *Let F be a set of unary functions such that $\operatorname{End} S = [F]$ for a set of relations S. If $[F]$ contains a constant function then* MCSP(S) *is polynomial.*

Proof. If the endomorphisms of S contain a constant function $f_d(x) = d$ for all $x \in D$, then for the set of relations $\langle S \rangle$ invariant under $[F]$, each relation $R \in \langle S \rangle$ contains a d-vector, *i.e.* a mapping of each variable to the value d. Therefore each instance of MCSP(S) is satisfiable by a d-vector. □

Lemma 5. *Let $[F]$ contain no constant functions. Let $f \in [F]$ be a function with the smallest cardinality of $\operatorname{ran} f$. Then $\operatorname{End} f(S)$ contains only permutations.*

Proof. Suppose there is a function $g \in \operatorname{End} f(S)$ not being a permutation. Necessarily g is not injective, *i.e.* the inclusion $\operatorname{ran} g \subsetneq \operatorname{ran} f$ holds. This is a contradiction with the fact that the cardinality of $\operatorname{ran} f$ is the smallest among all functions in $[F]$. □

Proposition 6. *Let F be a set of unary functions such that the clone $[F]$, equivalent to $\operatorname{End} S$ for a set of relations S, does not contain constant functions. Then* MCSP(S) *is* NP*-complete.*

Proof. We adapt the proof of Proposition 5.6 from [5]. Let $[F]$ be without constant functions. Let $f \in [F]$ be a function with minimal cardinality of its range $\operatorname{ran} f$. By Lemma 5, we know that $\operatorname{End} f(S)$ contains only permutations. Since $[F]$ does not contain constant functions, we also know that cardinality of $\operatorname{ran} f$ satisfies the condition $|\operatorname{ran} f| \geq 2$. We have to separate two cases.

If $|\operatorname{ran} f| = 2$, then we assume without loss of generality that $\operatorname{ran} f = \{0, 1\}$. The set $\operatorname{End} f(S)$ contains only permutations on $\{0, 1\}$. Let R_{NAE} be the relation $\{0, 1\}^3 \setminus \{000, 111\}$. It is clear that relation R_{NAE} is closed under every permutations on $\{0, 1\}$. Hence we have the inclusion $\operatorname{End} f(S) \subseteq \operatorname{End} R_{NAE}$. The relation R_{NAE} gives rise to the NOT-ALL-EQUAL-3SAT problem, known to be NP-complete. We conclude that MCSP$(f(S))$ is NP-complete.

Let now $|\operatorname{ran} f| \geq 3$. The set of relations $\operatorname{Inv} \operatorname{End}(f(S))$ is closed under permutations on $\operatorname{ran} f$. In particular, it contains the set of relations $Q \subseteq D^2$ where $Q = \{a_1, a_2, \ldots, a_k\}^2 \setminus \{(a_1, a_1), (a_2, a_2), \ldots, (a_k, a_k)\}$, such that the elements a_1, $\ldots$, a_k present in the relation satisfy the identity $|\{a_1, a_2, \ldots, a_k\}| = |\operatorname{ran} f|$. Relations in Q are the valid valuations for all instances of the $|\operatorname{ran} f|$-coloring problem. This problem is known to be NP-complete since $|\operatorname{ran} f| \geq 3$. We conclude that MCSP$(f(S))$ is also NP-complete in this case.

We have seen in Proposition 3 that the problem MCSP$(f(S))$ is logspace-equivalent to MCSP(S). We conclude that MCSP(S) is NP-complete. □

From Propositions 4 and 6 we derive the main theorem of this section.

Theorem 7. *The monotone constraint satisfaction problem* MCSP(S) *is polynomial if the set $\operatorname{End} S$ contains a constant function. Otherwise, it is* NP*-complete.*

We have presented a very simple dichotomy condition for the problem MCSP(S) on a finite domain D. To decide this condition, it is sufficient to check if the endomorphisms set $\operatorname{End} S$ contains a constant function. The endomorphism set $\operatorname{End} S$ is always equal to the clone $[F]$ for some set of functions F.

We can also consider the monotone CSPs starting from a set of functions F. Given a set of unary functions, we consider the problem MCSP$(\operatorname{Inv} F)$. An interesting question from a complexity point of view is to ask now, given a set of unary functions F, if the clone $[F]$ contains a constant function. This meta-problem is treated in the next section.

5 Complexity of Clones

To determine if a composition of functions from the set F constructs a constant function, one has to compute at least a part of the clone $[F]$. Salomaa [8] calls *class membership problem* the question to decide, given a set of functions F and a function class C, whether

the clone $[F]$ contains a function belonging to C. Salomaa proved this problem to be NP-hard. We will show that it is even NP-complete if C is the class of constant functions. However, it is interesting to note that we are working on constraint satisfaction problem where the domain size is fixed. We will see that this allows us to prove the *class membership problem* to be fixed-parameter tractable (FPT).

We need first a result by Salomaa [8] allowing us to bind the combination depth of functions belonging to a set F to obtain a constant function. This bound will be useful for the NP-membership proof in the sequel.

Proposition 8 (Salomaa [8]). *Let F be a set of functions without constant functions. Let D be the functions domain and n the domain size. Each constant function f_c on D verifies the condition* $\mathrm{Depth}(f_c) \leq n^3/2 - 3n^2/2 + 2n$.

Following Salomaa [8], it is not necessary to go beyond this polynomial bound in order to find a constant function in $[F]$. Once arrived at this bound without finding a constant function, we know that there are no constant functions in $[F]$.

Proposition 9. *The class membership problem is* NP*-complete if C is the class of constant functions.*

Proof. We know from Salomaa [8] that this problem is NP-hard. We just have to show that it is also in NP. By Proposition 8, we know that the depth of every constant function is limited by $n^3/2 - 3n^2/2 + 2n$, with n being the domain size. A certificate f_c cannot be longer than this bound. We have to check two conditions. First, that each function composing the sequence f_c is in F, and second that f_c is a constant function. The first step is obviously in $O(n^3.|F|)$, and the second step asking to compute f_c is in $O(n^4)$ because we have to compute n values n^3 times. Notice that $|F|$ is the number k of functions in F times the size of a function, which is n. Hence, the certificate f_c has a size depending of n, and it can be decided in $O(k.n^4)$ whether f_c is a constant function. So the class membership problem is in NP if C is the class of constant function, and thus this problem is NP-complete. □

We now focus on the complexity of the problem to determine if a constant function can be obtained from a function set F, *i.e.* the *class membership problem* where C is the class of constant functions. If we assume that D has a fixed size, we can consider the following *parametric class membership problem* version:

Problem: CLASS MEMBERSHIP PROBLEM
Input: A set of functions F.
Parameter: The domain size n.
Question: Does the clone $[F]$ contain a constant function?

We will show that the complexity of this problem is *fixed-parameter tractable*. We begin to introduce this complexity class.

Definition 10 (Downey & Fellows [3]). *A parametric problem P is* fixed-parameter tractable*, or* FPT*, if there exists an algorithm taking inputs (I, k), where I is the principal part of input and k the parameter, and deciding if the membership $(I, k) \in P$ holds in time $f(k).|I|^c$, with f being an arbitrary function and c a constant.*

Algorithm 1. Class Membership Problem

```
1: Q ← {f_i | f_i ∈ F}
2: S ← ∅
3: while Q ≠ ∅ do
4:     f ← dequeue(Q)
5:     S ← S ∪ {f}
6:     for all f_i ∈ F do
7:         g = f_i ∘ f
8:         if g is a constant function then
9:             return "YES"
10:        end if
11:        if g ∉ S then
12:            enqueue(Q, g)
13:        end if
14:    end for
15: end while
16: return "NO"
```

Theorem 11. *The parametric class membership problem for a set of unary functions F is fixed-parameter tractable and it can be decide in time $O(n^n.|F|)$, where n is the domain size parameter.*

Proof. It is sufficient to exhibit an algorithm deciding the *class membership problem*, taking as parameter the domain size n, and terminating in $O(f(n).|F|^c)$ for some constant c. Thus the proof is relative to complexity of Algorithm 1.

First, we prove that Algorithm 1 is sound and terminating. Notice that Q is the set of functions we have to treat (represented by a queue) and S the set of already produced functions. At the beginning Q is instantiated to F. For each element in Q, the algorithm composes them with each function in F, and puts these combinations into Q if they are new (that is, not in S). The following property is the loop invariant: Q never contains twice the same function. Every possible function originating from a combination of F will be explored by the algorithm and tested whether it is a constant function, showing the algorithm soundness.

Since the number of unary functions on a finite domain of size n cannot be greater that n^n, the size of Q cannot pass this limit. From Line 11 follows that the queue Q cannot contain twice the same function and Line 4 shows that an element is removed from Q at each pass through the while loop. Hence Algorithm 1 terminates.

Let us analyze the complexity of Algorithm 1. Lines 1 and 2 are just instantiations. Line 3 activates a loop executed while Q is not empty. We have seen that $|Q| \leq n^n$. Lines 4 and 5 are executed in constant time. Line 6 does not depend on the size of F and complexity of Lines 7 to 13 depends on the choice of data structures. If we choose to use a hash table to represent the set S, where the length of collisions lists is proportional to n, these lines are executed in $O(n)$. Clearly Algorithm 1 runs in time $O(n^n.|F|)$ and allows us to conclude that the *class membership problem* is fixed-parameter tractable, where n is the parameter. □

6 Complexity of Clones on Ternary Domains

The meta-problem for MCSP on ternary domains can be treated more efficiently than the general meta-problem. Actually, we do not have to compute $[F]$, even a part of it, to know if there exists a combination of functions in F that leads to a constant function. To know if it is possible to produce such a constant function, it is sufficient to check if the functions in F verify the subsequent conditions. We first note the following fact.

Remark 12. Kernels of functions on a ternary domain are limited to only one equivalence class. This can be easily verified by the pigeonhole principle. Moreover, the size of these kernels can only be equal to 0, 2 or 3. Therefore we identify in the sequel the kernel of a unary function over a ternary domain with its singleton equivalence class.

Lemma 13. *Let F be a set of functions without constants. The clone $[F]$ contains a constant function f_c if and only if there exists two functions $f_a, f_b \in [F]$ such that $f_c = f_a \circ f_b$ and $\operatorname{ran} f_b \subseteq \ker f_a$.*

Proof. The only-if implication is obvious, therefore we focus on the if-implication. Let $f_c \in [F]$ be a constant function. Then f_c must be a composition of two functions f_a and f_b — possibly obtained by composition — since F does not contain any constant function. Without loss of generality, we can assume that f_a and f_b are non-constant functions such that $f_c = f_a \circ f_b$.

Suppose that for every equivalence class $[d]_{f_a} \in \ker f_a$, we have $\operatorname{ran} f_b \nsubseteq [d]_{f_a}$. Let $x, y \in \operatorname{ran} f_b$ such that, for every $[d]_{f_a}$ we have $\{x, y\} \nsubseteq [d]_{f_a}$. Then $f_a(x) \neq f_a(y)$, but since $x, y \in \operatorname{ran} f_b$, there must exist $x', y' \in D$ such that $f_b(x') = x$ and $f_b(y') = y$. Thus $(f_a \circ f_b)(x') \neq (f_a \circ f_b)(y')$, *i.e.* $f_c(x') \neq f_c(y')$. This is a contradiction with the fact that f_c is a constant function. □

From the aforementioned lemma we immediately derive the following corollary.

Corollary 14. *If there exist two functions f_a and f_b such that $\operatorname{ran} f_b = \ker f_a$, then the composition $f_a \circ f_b$ is a constant function.*

In addition to Lemma 13, we show some useful results on the range and kernel set of functions in the sequel.

Lemma 15. *Let $f, g \in F$. The following conditions hold:*
(i) $\operatorname{ran}(f \circ g) \subseteq \operatorname{ran} f$ *and* $\ker g \subseteq \ker(f \circ g)$*;*
(ii) if $\operatorname{ran} g \nsubseteq \ker f$ *and* $|\ker f| = 2$ *then* $\operatorname{ran}(f \circ g) = \operatorname{ran} f$*;*
(iii) if $\operatorname{ran} g \nsubseteq \ker f$ *and* $|\ker g| = 2$ *then* $\ker g = \ker(f \circ g)$.

Proof. Every unary function f is monotone, *i.e.*, if $A \subseteq B$ then $f(A) \subseteq f(B)$ holds for all subsets A, B of D. Since $\operatorname{ran} g \subseteq D$ and f is monotone, we have $f(\operatorname{ran} g) \subseteq f(D)$. Moreover, $f(\operatorname{ran} g)$ is $\operatorname{ran}(f \circ g)$ and $f(D)$ is $\operatorname{ran} f$. Therefore the inclusion $\operatorname{ran}(f \circ g) \subseteq \operatorname{ran} f$ holds. Let $x, y \in \ker g$. We have $g(x) = g(y)$, therefore $(f \circ g)(x) = (f \circ g)(y)$, *i.e.* $x, y \in \ker(f \circ g)$.

Let $\operatorname{ran} g \nsubseteq \ker f$ and $|\ker f| = 2$. We have to show that $\operatorname{ran} f \subseteq \operatorname{ran}(f \circ g)$. Let $y \in \operatorname{ran} f$. Suppose that for all $x \in \operatorname{ran} g$, the inequality $f(x) \neq y$ holds. Let $z \in \operatorname{ran} f$,

with $z \neq y$. So for all $x \in \operatorname{ran} g$, we have $f(x) = z$ since $|\ker f| = 2$, which implies $\operatorname{ran} g \subseteq \ker f$: contradiction with the assumption.

Let $\operatorname{ran} g \nsubseteq \ker f$. We have to show that $\ker(f \circ g) \subseteq \ker g$. Suppose that there exist $x, y \in \ker(f \circ g)$ such that $g(x) \neq g(y)$. Since $|\ker g| = 2$, we have $\{g(x), g(y)\} = \operatorname{ran} g$. However the equality $(f \circ g)(x) = (f \circ g)(y)$ holds, so we have the inclusion $\operatorname{ran} g \subseteq \ker f$, which is a contradiction. Thus, for all $x, y \in \ker(f \circ g)$, we have $x, y \in \ker g$. □

Corollary 16. *Let* $\{f, g\} = F$ *such that* $\operatorname{ran} g \nsubseteq \ker f$, $\operatorname{ran} f \nsubseteq \ker g$ *and* $|\ker f| = |\ker g| = 2$. *If* $\operatorname{ran} f = \operatorname{ran} g$ *then for every function* $h \in [F]$ *we have* $\operatorname{ran} h = \operatorname{ran} f$. *If* $\ker f = \ker g$ *then for every function* $h \in [F]$, *we have* $\ker h = \ker f$.

We need the notions of *circular* and *swap permutation* on a ternary domain to present our main result.

Definition 17. *A* ***circular permutation*** c *on* $D = \{0, 1, 2\}$ *is a permutation satisfying the condition* $c(x) = (x+k) \bmod |D|$ *with* $k \in D$, *for all* $x \in D$. *A* ***swap permutation*** s *on* $D = \{0, 1, 2\}$ *is a permutation satisfying the conditions* $s(x) = y$, $s(y) = x$ *and* $s(z) = z$, *for distinct* $x, y, z \in D$. *The set* $\{x, y\}$ *is called* $\operatorname{swap} s$.

We can now introduce the main result of this section, dividing it into two parts.

Proposition 18. *Let* F *a set of functions. If* F *satisfies one of the following conditions, then there exists a constant function in* $[F]$. *The conditions are:*

(i) there exists a constant function $f \in F$;
(ii) there exist $f, g \in F$ *(not necessarily distinct) such that* $\operatorname{ran} f = \ker g$;
(iii) there exist $f, c \in F$ *such that* $|\ker f| = 2$ *and* c *is a circular permutation;*
(iv) there exist $f, s \in F$ *such that* $|\ker f| = 2$ *and* s *is a swap permutation where* $\operatorname{swap} s \neq \operatorname{ran} f$ *and* $\operatorname{swap} s \neq \ker f$;
(v) there exist $f, s_1, s_2 \in F$ *such that* $|\ker f| = 2$ *and* s_1, s_2 *are swap permutations satisfying the condition* $\operatorname{swap} s_1 \neq \operatorname{swap} s_2$.

Proof. Case (i) is obvious. Case (ii) follows from Corollary 14. We notice here that the existence of f and g satisfying $\operatorname{ran} f \subseteq \ker g$ implies either $\operatorname{ran} f = \ker g$, or $|\ker g| = 3$, *i.e* g is a constant.

Case (iii) is obvious by (ii) if $\operatorname{ran} f = \ker f$. Otherwise, let $\operatorname{ran} f = \{x, y\}$ and $\ker f = \{y, z\}$ with $x, y, z \in D$ all different. Notice that $\ker(c^2 \circ f) = \ker(c \circ f) = \ker f$. There are two possibles cases: (1) $\operatorname{ran}(c \circ f) = \{y, z\}$, hence by (ii) $c \circ f$ is a constant function; (2) $\operatorname{ran}(c \circ f) = \{x, z\}$. It is easy to see that $\operatorname{ran}(c^2 \circ f) = \{y, z\}$. By (ii), $c^2 \circ f$ is a constant function.

Case (iv) is also obvious by (ii) if $\operatorname{ran} f = \ker f$. Otherwise, we have $s(\operatorname{ran} f) = \ker f$. Thus, for all $x \in D$ we have $(s \circ f)(x) = \ker f$, *i.e.* $\operatorname{ran}(s \circ f) = \ker f$, and we conclude by (ii) that $(f \circ s \circ f)$ is a constant function.

Case (v) is obvious by (ii) if $\operatorname{ran} f = \ker f$. Otherwise, since we have $\operatorname{swap} s_1 \neq \operatorname{swap} s_2$, the composition $s_1 \circ s_2$ necessarily produces a circular permutation. We can conclude by (iii). □

We will see now that the conditions listed in Proposition 18 are necessary to get a constant function in $[F]$.

Proposition 19. *Let F be a set of functions satisfying no condition from Proposition 18. Then there is no constant function in $[F]$.*

Proof. If F does not verify any condition of Proposition 18, then we are in one of these cases:

(1) F contains only permutations;
(2) for each $f \in F$ we have $|\ker f_i| = 2$ and for all $f_i, f_j \in F$ (eventually $f_i = f_j$), we have $\operatorname{ran} f_i \neq \ker f_j$ and $\operatorname{ran} f_j \neq \ker f_i$.;
(3) F satisfies Condition 2 and contains also swap permutations with the same swap set, such that for all $f_i \in F$ with $|\ker f_i| = 2$, for all swap permutations $s_k \in F$, we have $\operatorname{swap} s_k = \ker f_i$ or $\operatorname{swap} s_k = \operatorname{ran} f_i$. Notice that both are impossible because we have $\ker f_i \neq \operatorname{ran} f_i$.

Case (1) is obvious. If F is a set of permutations, then $[F]$ is a set of permutations, too. By the pigeonhole principle, case (2) implies $\ker f_i = \ker f_j$, or $\operatorname{ran} f_i = \operatorname{ran} f_j$, or both, for all $f_i, f_j \in F$. From Corollary 16, we know that $\operatorname{ran} f_i = \operatorname{ran} f_j$ for all $f_i, f_j \in F$ implies that every $f \in [F]$ verifies the equality $\operatorname{ran} f = \operatorname{ran} f_i$. Since $|\ker f_i| = 2$ implies $|\ker f| = 2$, so f cannot be a constant function. We can apply the same argument for the case where $\ker f_i = \ker f_j$ for all $f_i, f_j \in F$.

Like in case (2), we have for all $f_i, f_j \in F$ the equations $\ker f_i = \ker f_j$, or $\operatorname{ran} f_i = \operatorname{ran} f_j$, or both. We distinguish four cases. Let $\operatorname{ran} f_i = \operatorname{ran} f_j$, for all $f_i, f_j \in F$, and $\operatorname{swap} s_k = \operatorname{ran} f_i$. It is clear that $\operatorname{ran}(s_k \circ f_i) = \operatorname{ran}(f_i \circ s_k) = \operatorname{ran} f_i$. Since $|\ker f_i| = 2$, we also have $|\ker(s_k \circ f_i)| = |\ker(f_i \circ s_k)| = 2$, hence $s_k \circ f_i$ and $f_i \circ s_k$ cannot be constant functions. Now let $\operatorname{swap} s_k = \ker f_i$, for a $f_i \in F$. We must have $\ker f_i = \ker f_j$ for all $f_i, f_j \in F$ because otherwise there exists $f_k \in F$ such that $\operatorname{swap} s_k = \operatorname{ran} f_k$, and thus $\ker f_k = \operatorname{ran} f_i$ which constitutes a contradiction. Therefore we have $\ker(s_k \circ f_i) = \ker(f_i \circ s_k) = \ker f_i$. Since $|\ker f_i| = 2$, we conclude that there is no composition producing a constant function. With the same arguments, we see that we cannot get a constant function if $\ker f_i = \ker f_j$ for all $f_i, f_j \in F$, whenever $\operatorname{swap} s_k = \operatorname{ran} f_i$ or $\operatorname{swap} s_k = \ker f_i$. □

Theorem 20. *Given a set of functions F on a ternary domain, the problem to know whether the clone $[F]$ contains a constant function can be decided in polynomial time.*

Proof. By Propositions 18 and 19, we know that $[F]$ contains a constant function if and only if F satisfies one of the conditions in Proposition 18. The satisfiability test of each condition can be done in polynomial time. For case (i), we have to test for all $f \in F$ whether f is a constant function. This condition can be verified in time $O(|F|)$. For case (ii), we are looking for $f, g \in F$, for which we compute $\ker f$ and $\operatorname{ran} g$, such that $\operatorname{ran} g \subseteq \ker f$. The verification of condition (ii) is done in time $O(|F|^2)$. For case (iii), we are looking for $f \in F$ such that $|\ker f| = 2$ and for a circular permutation $c \in F$. This can be verified in time $O(|F|)$. For case (iv), we have to check for all $f, s \in F$ if $|\ker f| = 2$, if s is a swap permutation, followed by a check if $\operatorname{swap} s \neq \ker f$ and $\operatorname{swap} s \neq \operatorname{ran} f$. This can be verified in time $O(|F|^2)$. Finally for case (v), we are looking for $f \in F$ such that $|\ker f| = 2$ and for $s_1, s_2 \in F$ such that s_1 and s_2 are swap permutations, such that $\operatorname{swap} s_1 \neq \operatorname{swap} s_2$. This is verified in time $O(|F|^2)$. □

We have shown a polynomial-time algorithm concerning the meta-problem on a ternary domain. Despite its complexity in $O(|F|^2)$ which is less efficient than the general

meta-problem algorithm in $O(n^n . |F|)$, with the constant $n = 3$ for the ternary case, this algorithm use a method allowing to skip the computation, even partially, of the clone $[F]$. Thus, this method constitute a serious approach for obtaining polynomial-time algorithms more efficient than the general meta-problem algorithm.

7 Concluding Remarks

We analyzed the computational complexity of monotone constraint satisfaction problems, allowing also the disjunction connective to be applied. We obtained a complete characterization expressed by a Dichotomy Theorem, distinguishing between tractable and NP-complete instances. The tractability condition turned out to be the closure of the constraints under a constant function. Since the endomorphism set $\mathrm{End}\, S$ is equal to the clone $[F]$ generated from a set of unary functions F, it is also interesting to study the meta-problem of the tractability condition. This means, given a set of unary functions F, whether the clone $[F]$ contains a constant function. We showed that the meta-problem is NP-complete if the domain is part of the input, but it is fixed-parameter tractable, with an algorithm running in time $O(n^n\, |F|)$, if we consider the domain as a parameter. We performed a special complexity analysis for the meta-problem of the ternary domain, for which we deduced conditions ensuring the presence of a constant function in the clone $[F]$ without the necessity of computing (at least a part of) the functions in $[F]$.

References

1. Bulatov, A.A.: A dichotomy theorem for constraint satisfaction problems on a 3-element set. Journal of the Association for Computing Machinery 53(1), 66–120 (2006)
2. Cohen, D., Jeavons, P., Jonsson, P., Koubarakis, M.: Building tractable disjunctive constraints. Journal of the Association for Computing Machinery 47(5), 826–853 (2000)
3. Downey, R.G., Fellows, M.R.: Parametrized Complexity. Springer, Heidelberg (1999)
4. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. SIAM Journal on Computing 28(1), 57–104 (1998)
5. Jeavons, P.: On the algebraic structure of combinatorial problems. Theoretical Computer Science 200(1-2), 185–204 (1998)
6. Krasner, M.: Une généralisation de la notion de corps. Journal de Mathématiques pures et appliquées 17, 367–385 (1938)
7. Pöschel, R.: Galois connections for operations and relations. In: Denecke, K., et al. (eds.) Galois Connections and Applications, pp. 231–258. Kluwer, Dordrecht (2004)
8. Salomaa, A.: Composition sequences for functions over a finite domain. Theoretical Computer Science 292(1), 263–281 (2003)
9. Schaefer, T.J.: The complexity of satisfiability problems. In: Proceedings 10th Symposium on Theory of Computing (STOC 1978), San Diego, California, USA, pp. 216–226 (1978)
10. Yanov, Y.I., Muchnik, A.A.: On the existence of k-valued closed classes that have no bases. Doklady Akademii Nauk SSSR 127, 44–46 (1959) (in Russian)

Parameterized Complexity of Stabbing Rectangles and Squares in the Plane

Michael Dom[1], Michael R. Fellows[2,⋆], and Frances A. Rosamond[2,⋆]

[1] Institut für Informatik, Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
dom@minet.uni-jena.de
[2] PC Research Unit, Office of DVC (Research), University of Newcastle,
Callaghan, NSW 2308, Australia
michael.fellows, frances.rosamond@newcastle.edu.au

Abstract. The NP-complete geometric covering problem Rectangle Stabbing is defined as follows: Given a set of horizontal and vertical lines in the plane, a set of rectangles in the plane, and a positive integer k, select at most k of the lines such that every rectangle is intersected by at least one of the selected lines.

While it is known that the problem can be approximated in polynomial time with a factor of two, its parameterized complexity with respect to the parameter k was open so far—only its generalization to three or more dimensions was known to be W[1]-hard. Giving two fixed-parameter reductions, one *from* the W[1]-complete problem Multicolored Clique and one *to* the W[1]-complete problem Short Turing Machine Acceptance, we prove that Rectangle Stabbing is W[1]-complete with respect to the parameter k, which in particular means that there is no hope for fixed-parameter tractability with respect to the parameter k. Our reductions show also the W[1]-completeness of the more general problem Set Cover on instances that "almost have the consecutive-ones property", that is, on instances whose matrix representation has at most two blocks of 1s per row.

For the special case of Rectangle Stabbing where all rectangles are squares of the same size we can also show W[1]-hardness, while the parameterized complexity of the special case where the input consists of rectangles that do not overlap is open. By giving an algorithm running in $(4k+1)^k \cdot n^{O(1)}$ time, we show that Rectangle Stabbing is fixed-parameter tractable in the still NP-hard case where *both* these restrictions apply.

1 Introduction

Geometric covering problems arise in many applications and are intensively studied (see [8,9,14]). Here, we consider the problem 2-Dimensional Rectangle Stabbing (Rectangle Stabbing for short), which is defined as follows.

⋆ Supported by the Australian Research Council. This work was done in Jena, where Michael R. Fellows was supported by the Alexander von Humboldt-Foundation, Bonn, Germany, as a recipient of the Humboldt Research Award.

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 298–309, 2009.

(2-Dimensional) Rectangle Stabbing

Input: A set L of vertical and horizontal lines embedded in the plane, a set R of axis-parallel rectangles embedded in the plane, and a positive integer k.

Question: Is there a set $L' \subseteq L$ with $|L'| \leq k$ such that every rectangle from R is intersected (*"stabbed"*) by at least one line from L'?

Rectangle Stabbing is NP-complete (see [8]). Its optimization version, considered in the setting of polynomial-time approximation, asks for a *minimum-cardinality* set $L' \subseteq L$ to cover all rectangles from R.

Applications of Rectangle Stabbing range from radiotherapy [10] to embedded sensor networks, spatial data organization and statistical data analysis [1,11]. Also, the problem of stabbing arbitrary connected figures (instead of rectangles) in the plane with horizontal and vertical lines can easily be reduced to Rectangle Stabbing by replacing each figure by its bounding box. The same holds for the stabbing problem where only the rectangles in the plane are given and a minimum number of horizontal and vertical lines shall be inserted that stab all rectangles: any instance of this problem can be transformed into an instance of Rectangle Stabbing by inserting $O(|R|)$ lines. Without loss of generality, we can always assume that all given lines have integer coordinates.

The literature so far mainly considers the polynomial-time approximability of Rectangle Stabbing and its variants. Hassin and Megiddo [10] give a factor-$d2^{d-1}$ approximation for stabbing d-dimensional, identical objects with axis-parallel lines in the d-dimensional space. Gaur et al. [8] achieve a factor-d approximation for d-Dimensional Rectangle Stabbing, that is, for stabbing d-dimensional, axis-parallel hyperboxes with $(d-1)$-dimensional, axis-parallel hyperplanes; the two-dimensional case Rectangle Stabbing, hence, can be approximated with a factor of two. A similar result was obtained by Mecke et al. [16]; they give a factor-d approximation algorithm for a problem called d-C1P-Set Cover, which is a generalization of d-Dimensional Rectangle Stabbing. Weighted and capacitated versions of d-Dimensional Rectangle Stabbing have been considered by Even et al. [4] and by Xu and Xu [18], also leading to several approximation algorithms. A restricted, but still NP-complete variant of (2-Dimensional) Rectangle Stabbing is called Interval Stabbing; here, every rectangle in the input is intersected by at most one horizontal line (that is, every rectangle is a horizontal interval in the plane). Kovaleva and Spieksma [12,13] give constant-factor approximations for several variants of Interval Stabbing. Approximation algorithms for the more general variant of Interval Stabbing where the input contains horizontal *and* vertical intervals have been developed by Hassin and Megiddo [10].

Concerning the parameterized complexity of Rectangle Stabbing and d-Dimensional Rectangle Stabbing (that is, the question whether there is an algorithm running in $f(k) \cdot |(L, R, k)|^{O(1)}$ time), it is known that, on the one hand, d-Dimensional Rectangle Stabbing is W[1]-hard with respect to the parameter k for $d \geq 3$ [2]. On the other hand, two special cases of Rectangle Stabbing in two dimensions have been shown to be fixed-parameter tractable [2]: If each rectangle is intersected by at most b horizontal but arbitrarily many vertical lines

or at most b vertical but arbitrarily many horizontal lines, or if each horizontal line intersects at most b rectangles, then Rectangle Stabbing is fixed-parameter tractable with respect to the combined parameter b, k. The parameterized complexity of Rectangle Stabbing without restrictions, however, remained open so far [2].

The contributions of this paper are the following. In Section 3, we settle the question of Dom and Sikdar [2] for the parameterized complexity of (2-Dimensional) Rectangle Stabbing by proving its W[1]-hardness with respect to the parameter k as well as its membership in W[1]. Our proofs also show the W[1]-completeness of the more general problem 2-C1P-Set Cover (see Section 2), which was also open so far [2]. In Section 4, we consider the restriction of Rectangle Stabbing where all rectangles in the input are squares of the same size that do not intersect. After showing its NP-hardness, we prove that this variant is fixed-parameter tractable. Due to the lack of space, some proofs are omitted.

2 Preliminaries

In the framework of parameterized complexity [3,6,17], the running time of an algorithm is viewed as a function of two quantities: the size of the given problem instance *and* a *parameter*. Thus, a parameterized problem is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet and $\mathbb{N}$ is the set of positive integers; an instance of a parameterized problem is a pair (I, k), where k is called the parameter. A parameterized problem is said to be *fixed-parameter tractable (FPT)* with respect to the parameter k if there exists an algorithm for the problem running in $f(k) \cdot |I|^{O(1)}$ time, where f is a computable function only depending on k.

A parameterized problem π_1 is *fixed-parameter reducible* to a problem π_2 if there are two computable functions $f, g : \mathbb{N} \to \mathbb{N}$ and an algorithm that transforms an instance (I, k) of π_1 into an instance $(I', f(k))$ of π_2 in $g(k) \cdot |I|^{O(1)}$ time such that $(I', f(k))$ is a *yes*-instance for π_2 iff (I, k) is a *yes*-instance for π_1. The complexity hierarchy used for characterizing the hardness of parameterized problems is the W-*hierarchy* consisting of the classes $\mathrm{W}[1], \mathrm{W}[2], \ldots, \mathrm{W}[\mathrm{Sat}], \mathrm{W}[\mathrm{P}]$, interrelated as follows: $\mathrm{FPT} \subseteq \mathrm{W}[1] \subseteq \mathrm{W}[2] \subseteq \ldots \subseteq \mathrm{W}[\mathrm{Sat}] \subseteq \mathrm{W}[\mathrm{P}]$. There is strong evidence that all these subset inclusions are strict, which means that there are problems in W[1] that are presumably not fixed-parameter tractable and, in particular, that W[1]-*hard* problems are not fixed-parameter tractable [3,6,17]. To show that a problem is W[1]-hard (is in W[1]), one needs to exhibit a fixed-parameter reduction from a known W[1]-hard problem to the problem at hand (from the problem at hand to a problem known to be in W[1]). A problem is W[1]-*complete* if it is W[1]-hard and in W[1].

When considering an instance (L, R, k) of Rectangle Stabbing, let $L = V \cup H$, where $V = \{v_1, \ldots, v_n\}$ are the vertical lines, ordered from left to right, and $H = \{h_1, \ldots, h_m\}$ are the horizontal lines, ordered from top to bottom. For a rectangle $r \in R$, let $\mathrm{lx}(r), \mathrm{rx}(r), \mathrm{tx}(r), \mathrm{bx}(r)$ be the indices of the leftmost, rightmost, topmost and bottommost line intersecting r. We define the

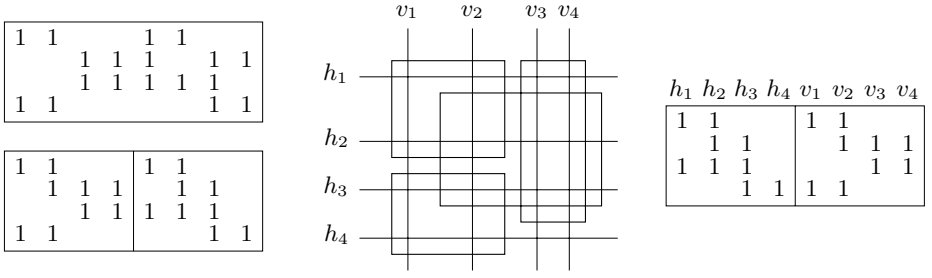


Fig. 1. Left top: A matrix having the 2-C1P, but not the 2-SC1P. Left bottom: A matrix having the 2-SC1P. Middle and right: Illustration of the equivalence between Rectangle Stabbing and 2-SC1P-Set Cover. In all figures of this paper, only the 1-entries of the matrices are displayed (that is, all 0-entries are omitted).

width $\mathrm{wh}(r) := \mathrm{rx}(r) - \mathrm{lx}(r) + 1$ and the height $\mathrm{ht}(r) := \mathrm{bx}(r) - \mathrm{tx}(r) + 1$ as the number of vertical and horizontal lines, respectively, intersecting r. A rectangle r is called a *square* if $\mathrm{wh}(r) = \mathrm{ht}(r)$.

All graphs that we consider are undirected; a graph $G = (V, E)$ is called k-*colorable* if there is a function $c : V \to \{1, \ldots, k\}$ satisfying $\forall\{u, v\} \in E : c(u) \neq c(v)$; the function c is then called a k-*coloring* for G.

For a simpler description of our algorithms and reductions, we will consider Rectangle Stabbing as a covering problem on binary matrices, which is a restricted version of the NP-complete problem Set Cover.[1]

Set Cover

Input: A binary matrix M and a positive integer k.

Question: Is there a set C' of at most k columns of M such that the submatrix M' of M that consists of these columns has at least one 1 in every row?

To introduce restricted versions of Set Cover, we need the following.

Definition 1

1. Given a binary matrix M, a block of 1s *in a row of M is a maximal set of consecutive 1-entries in this row.*

2. A binary matrix M has the d-consecutive-ones property (d-C1P) *if in every row of M there are at most d blocks of 1s.*

3. A binary matrix M with columns $c_1, \ldots, c_n$ has the separated d-consecutive-ones property (d-SC1P) *if the columns of M can be partitioned into d sets of consecutive columns $C^1 = \{c_1, \ldots, c_{j_1}\}$, $C^2 = \{c_{j_1+1}, \ldots, c_{j_2}\}$, $\ldots$, $C^d = \{c_{j_{d-1}+1}, \ldots, c_n\}$ such that for every $i \in \{1, \ldots, d\}$ the submatrix of M consisting of the columns of C^i has at most one block of 1s per row.*

See Fig. 1 for an illustration for the d-C1P and d-SC1P.

[1] Set Cover is usually defined as a subset selection problem; however, the equivalence of our definition and the more common definition of Set Cover as a subset problem can easily be seen by identifying columns with subsets and rows with elements to be covered.

If Set Cover is restricted by demanding that the input matrix M must have the d-C1P, then we call the resulting problem d-C1P-Set Cover; if M must have the d-SC1P, then we call the resulting problem d-SC1P-Set Cover. For an illustration of the following observation, see Fig. 1.

Observation 1. Rectangle Stabbing *and* 2-SC1P-Set Cover *are equivalent: There is a polynomial-time computable one-to-one mapping between instances of* Rectangle Stabbing *and instances of* 2-SC1P-Set Cover *that leaves the parameter k unchanged and maps yes-instances to yes-instances and no-instances to no-instances.*

3 W[1]-Completeness of Rectangle Stabbing

In this section, we prove that, for the parameter k, 2-SC1P-Set Cover is W[1]-hard and 2-C1P-Set Cover is in W[1], which implies the W[1]-completeness of Rectangle Stabbing.

3.1 W[1]-Hardness of Rectangle Stabbing

We give a fixed-parameter reduction from the problem Multicolored Clique defined below to 2-SC1P-Set Cover—the W[1]-hardness of Rectangle Stabbing then follows from the W[1]-hardness of Multicolored Clique [5] and the equivalence between 2-SC1P-Set Cover and Rectangle Stabbing.

Multicolored Clique
Input: An undirected k-colorable graph $G = (V, E)$, a positive integer k, and a k-coloring $c : V \to \{1, \ldots, k\}$ for G.
Question: Is there a size-k clique in G?

The basic scheme of the reduction. The basic approach of our reduction is similar to the one used in the W[1]-hardness proof for 3-SC1P-Set Cover [2]. However, due to the more restricted nature of 2-SC1P-Set Cover, the technical details are more involved.

We use the "Multicolored Clique reduction technique" [5], where the key idea is to use an alternative, equivalent formulation of Multicolored Clique: Given an undirected k-colorable graph $G = (V, E)$, a positive integer k, and a k-coloring $c : V \to \{1, \ldots, k\}$ for G, find a set $E' \subseteq E$ with $|E'| = \binom{k}{2}$ and a set $V' \subseteq V$ with $|V'| = k$ that satisfy the following three constraints:

Constraint 1: For every unordered pair $\{a, b\}$ of colors from $\{1, \ldots, k\}$, the edge set E' contains an edge whose endpoints are colored with a and b.
Constraint 2: For every color from $\{1, \ldots, k\}$, the vertex set V' contains a vertex of this color.
Constraint 3: If E' contains an edge $\{u, v\}$, then V' contains the vertices u and v.

Clearly, this formulation of Multicolored Clique is equivalent to the original definition. Given an instance (G, k, c) of Multicolored Clique, we construct

an equivalent instance (M, k') of 2-SC1P-Set Cover based on this alternative formulation. The standard approach for such a construction would be to create a matrix M with $|V| + |E|$ columns, one column for each vertex and each edge of G, and to set $k' = k + \binom{k}{2}$. The rows of M would have to be constructed in such a way that any solution C' for 2-SC1P-Set Cover on (M, k') corresponded to a solution (E', V') as described above for Multicolored Clique on the instance (G, k, c). That is, the rows of M would have to enforce that Constraints 1–3 are satisfied. The problem with this standard approach is that the resulting matrix M does not have the 2-SC1P. Therefore, we use a construction that is based on the same ideas, but involves more columns and rows. In order to obtain a matrix that has the 2-SC1P, we add not one, but *three* columns to M for every edge e in G. Hence, an instance (G, k, c) of Multicolored Clique is mapped to an instance (M, k'), where $k' = 3 \cdot \binom{k}{2} + k$.

To describe the details of M's construction, let the *color of an edge* $\{u, v\}$, denoted $d(\{u, v\})$, be the set of colors of its endpoints, that is, $d(\{u, v\}) := \{c(u), c(v)\}$. We assume that the edges $E = \{e_1, \ldots, e_{|E|}\}$ and vertices $V = \{v_1, \ldots, v_{|V|}\}$ of G are ordered in such a way that edges and vertices of the same color appear consecutively. For every edge color $\{a, b\}$, we define: $E_{\{a,b\}} := \{e \in E \mid d(e) = \{a, b\}\}$, $\text{first}(\{a, b\}) := \min\{p \in \{1, \ldots, |E|\} \mid d(e_p) = \{a, b\}\}$, and $\text{last}(\{a, b\}) := \max\{p \in \{1, \ldots, |E|\} \mid d(e_p) = \{a, b\}\}$. The details of the construction of M read as follows, see also Fig. 2.

The columns of M. The matrix M has $3 \cdot |E| + |V|$ columns, partitioned into two sets C^1 and C^2. The column set C^1 consists of two subsets of columns: a subset D^1 consisting of the columns $c^1_1, \ldots, c^1_{|E|}$, and a subset D^2 consisting of the columns $c^2_1, \ldots, c^2_{|E|}$. The column set C^2 also consists of two subsets of columns: the subset D^3 consisting of the columns $c^3_1, \ldots, c^3_{|V|}$, and the subset D^4 consisting of the columns $c^4_1, \ldots, c^4_{|E|}$.

These columns are ordered as follows in M. The leftmost $2 \cdot |E|$ columns of M are those from C^1, the remaining $|V| + |E|$ columns are those from C^2. The columns from C^1 are ordered in such a way that columns corresponding to edges of the same color appear consecutively. More precisely, for every edge color $\{a, b\}$, there are $2 \cdot |E_{\{a,b\}}|$ consecutive columns

$$c^1_{\text{first}(\{a,b\})}, \ldots, c^1_{\text{last}(\{a,b\})}, c^2_{\text{first}(\{a,b\})}, \ldots, c^2_{\text{last}(\{a,b\})}.$$

The columns from C^2 are ordered as follows: To the right of the columns from C^1, there are the $|V|$ columns $c^3_1, \ldots, c^3_{|V|}$. The rightmost $|E|$ columns of M, finally, are the columns $c^4_1, \ldots, c^4_{|E|}$. Intuitively speaking, every column $c^1_p \in C^1$, every column $c^2_p \in C^1$, and every column $c^4_p \in C^2$ one-to-one corresponds to the edge $e_p \in E$, and every column $c^3_q \in C^2$ one-to-one corresponds to the vertex $v_q \in V$.

The rows of M. The rows of M have to ensure that every solution C' for 2-SC1P-Set Cover on $(M, k' = 3 \cdot \binom{k}{2} + k)$ corresponds to a subset of edges and vertices of G satisfying Constraints 1–3. Since there are three columns in M for every edge in G, we need *four* types of rows:

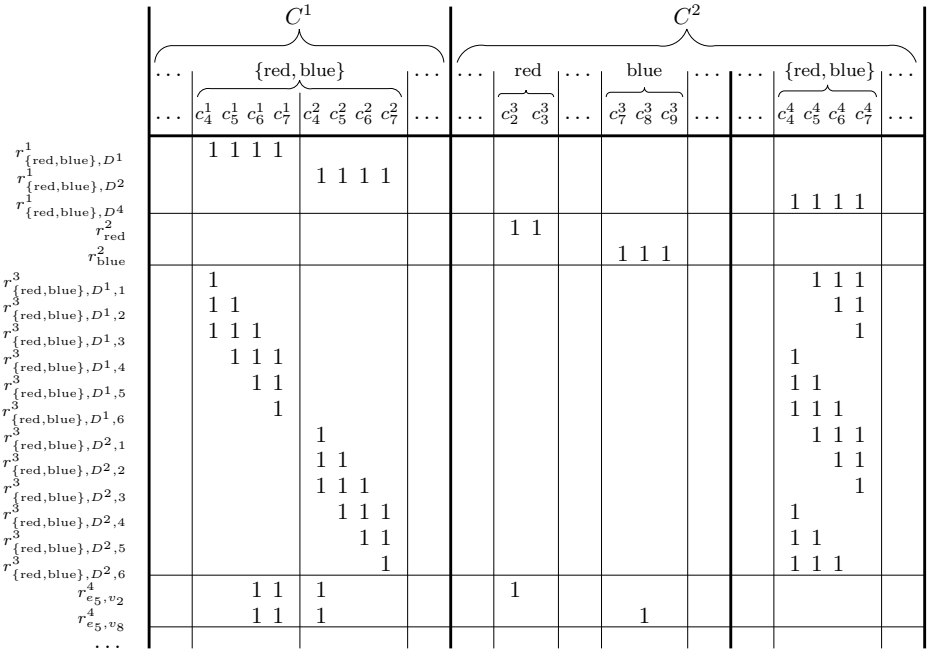


Fig. 2. Example for the construction of M in the W[1]-hardness proof for 2-SC1P-Set Cover. We assume that in G there are exactly two red vertices v_2, v_3 and exactly three blue vertices v_7, v_8, v_9, among vertices of other colors. Moreover, the only edges between red and blue vertices are e_4, e_5, e_6, e_7 with $e_5 = \{v_2, v_8\}$.

Rows of Type 1 and 2 ensure that any set of k' columns that forms a solution for 2-SC1P-Set Cover contains exactly $\binom{k}{2}$ columns from D^1—one of each edge color—, $\binom{k}{2}$ columns from D^2—one of each edge color—, $\binom{k}{2}$ columns from D^4—one of each edge color—, and k columns from D^3—one of each vertex color. Type-3 rows ensure that the columns chosen from D^1, D^2, and D^4 are consistent: if a solution contains the column c_j^1, then it must contain c_j^4, and *vice versa*; analogously, if a solution contains the column c_j^2, then it must contain c_j^4, and *vice versa*. Finally, Type-4 rows ensure that if a solution contains a column c_j^1 (and, due to the Type-3 rows, the column c_j^2) corresponding to an edge $e_j = \{u, v\}$, then it also contains the columns corresponding to the vertices u and v. See Fig. 2 for an illustration of the following construction details. The argument that the reduction works correctly is omitted.

Type-1 rows. For every edge color $\{a, b\}$, M contains three rows $r^1_{\{a,b\},D^1}$, $r^1_{\{a,b\},D^2}$, and $r^1_{\{a,b\},D^4}$.

For $x = 1, 2, 4$, the row $r^1_{\{a,b\},D^x}$ has a 1 in every column $c_j^x \in D^x$ with $d(e_j) = \{a, b\}$, and 0s in all other columns.

Type-2 rows. For every vertex color $a \in \{1, \ldots, k\}$, M contains a row r_a^2 which has a 1 in every column $c_j^3 \in D^3$ with $c(v_j) = a$, and 0s in all other columns.

Observe that the rows of the Types 1 and 2 together with the value of k' force every solution for 2-SC1P-Set Cover on (M, k') to contain *exactly one* column from each of D^1, D^2, and D^4 for every edge color and *exactly one* column from D^3 for every vertex color.

Type-3 rows. For every edge color $\{a, b\}$, M contains a set of $2 \cdot (|E_{\{a,b\}}| - 1)$ rows $r^3_{\{a,b\},D^1,i}$ and a set of $2 \cdot (|E_{\{a,b\}}| - 1)$ rows $r^3_{\{a,b\},D^2,i}$, where in both cases $1 \leq i \leq 2 \cdot (|E_{\{a,b\}}| - 1)$.

A row $r^3_{\{a,b\},D^x,i}$ with $x \in \{1, 2\}$ and $i \in \{1, \ldots, |E_{\{a,b\}}| - 1\}$ has a 1 in every column $c^x_j \in C^x$ with $d(e_j) = \{a, b\}$ and $j < \text{first}(\{a, b\}) + i$ and every column $c^4_j \in C^4$ with $d(e_j) = \{a, b\}$ and $j \geq \text{first}(\{a, b\}) + i$, and 0s in all other columns.

A row $r^3_{\{a,b\},D^x,i}$ with $i \in \{|E_{\{a,b\}}|, \ldots, 2 \cdot (|E_{\{a,b\}}| - 1)\}$ has a 1 in every column $c^x_j \in C^x$ with $d(e_j) = \{a, b\}$ and $j \geq \text{first}(\{a, b\}) + i - (|E_{\{a,b\}}| - 1)$ and every column $c^4_j \in C^4$ with $d(e_j) = \{a, b\}$ and $j < \text{first}(\{a, b\}) + i - (|E_{\{a,b\}}| - 1)$, and 0s in all other columns.

To see that the columns selected from D^x, $x \in \{1, 2\}$, and D^4 are consistent in every solution for 2-SC1P-Set Cover on (M, k'), observe that taking a column c^x_j into the solution, this column does not contain a 1 from the rows $r^3_{\{a,b\},D^x,j-\text{first}(\{a,b\})}$ and $r^3_{\{a,b\},j-\text{first}(\{a,b\})+|E_{\{a,b\}}|}$ (if existing). Hence, the single column from D^4 belonging to the solution must be c^4_j.

Type-4 rows. For every edge $e_p = \{v_{q_1}, v_{q_2}\} \in E$, the matrix M contains two rows $r^4_{e_p,v_{q_1}}$ and $r^4_{e_p,v_{q_2}}$.

For $i = 1, 2$, the row $r^4_{e_p,v_{q_i}}$ has a 1 in every column $c^1_j \in C^1$ with $d(e_j) = d(e_p)$ and $j > p$, every column $c^2_j \in C^2$ with $d(e_j) = d(e_p)$ and $j < p$, and the column $c^3_{q_i} \in C^3$, and 0s in all other columns.

Theorem 1. 2-SC1P-Set Cover, 2-C1P-Set Cover, *and* Rectangle Stabbing *are* W[1]*-hard with respect to the parameter* k.

Using a modification of the above construction, we can also show:

Theorem 2. *The restricted variant of* Rectangle Stabbing *where all rectangles in* R *are squares having the same width and the same height is* W[1]*-hard with respect to the parameter* k.

3.2 Membership of Rectangle Stabbing in W[1]

The membership of Rectangle Stabbing, and, more general, of 2-C1P-Set Cover, in W[1] can be shown in analogy to the proof given by Marx [15] for Dominating Set on intersection graphs of axis-parallel rectangles: One exhibits a fixed-parameter reduction to the W[1]-complete [3] problem Short Turing Machine Acceptance, defined as follows.

Short Turing Machine Acceptance
Input: The description of a nondeterministic Turing machine N and a positive integer k'.
Question: Can N stop within k' steps on the empty input string?

To reduce 2-C1P-Set Cover to Short Turing Machine Acceptance, one constructs, for a given instance (M, k) of 2-C1P-Set Cover, a nondeterministic Turing machine N that can stop after $k' = f(k)$ steps on the empty input string iff (M, k) is a *yes*-instance. Intuitively speaking, this Turing machine N nondeterministically decides in $f(k)$ steps whether the tuple (M, k), which is encoded into the internal states and the transition function of N, is a *yes*-instance of 2-C1P-Set Cover or not, and correspondingly it either stops after $f(k)$ steps or goes into an infinite loop (details are omitted).

Theorem 3. *For the parameter k,* 2-C1P-Set Cover*,* 2-SC1P-Set Cover *and* Rectangle Stabbing *are in* W[1].

4 Stabbing Nonoverlapping Squares of the Same Size

In this section, we consider the natural restriction of Rectangle Stabbing where no two rectangles from R "overlap". Two rectangles r_1, r_2 *overlap* if there exist a vertical line v and a horizontal line h that both intersect r_1 as well as r_2. Moreover, we further restrict the problem by demanding that all rectangles in R are squares of the same size, meaning that there is a number b such that for every rectangle r in R we have $\mathrm{wh}(r) = \mathrm{ht}(r) = b$. We call this restricted problem variant Disjoint b-Square Stabbing; it is equivalent to the stabbing problem where a set of unit squares in the plane is given (but no lines are given) and a minimum number of horizontal and vertical lines shall be inserted that stab all the squares.

Basic Observations. We show that, one the one hand, the problem Disjoint b-Square Stabbing is NP-complete for every $b \geq 2$, while, on the other hand, it is polynomial-time solvable for $b = 1$. To prove the NP-hardness, one can reduce from the NP-complete [7] problem Vertex Cover (details omitted).

Theorem 4. Disjoint b-Square Stabbing *is* NP*-complete for every $b \geq 2$.*

Complementing Theorem 4, one can see that Disjoint b-Square Stabbing is polynomial-time solvable for $b = 1$. To this end, observe that every instance of Disjoint 1-Square Stabbing is equivalent to an 2-SC1P-Set Cover instance (M, k) where the column set of M can be partitioned into two sets C^1 and C^2 of consecutive columns such that every row of M contains exactly one 1 in a column of C^1 and one 1 in a column of C^2. Such a matrix can be interpreted as a bipartite graph, and, hence, (M, k) corresponds to an instance of Vertex Cover on bipartite graphs. Vertex Cover on bipartite graphs, however, is known to be polynomial-time solvable.

Fixed-parameter tractability. We show that Disjoint b-Square Stabbing is in FPT with respect to the parameter k. To this end, we present a search-tree

algorithm, that is, a recursive algorithm that in each recursive step branches into a bounded number of cases. More precisely, in each step the algorithm first determines a subset of the given lines in such a way that every size-k solution for the Rectangle Stabbing instance must contain at least one of these lines, and then recursively tests which of these lines leads to the desired solution.

In order to bound the size of the above subsets of lines and, thus, the number of cases to branch into, in each step a set of *data reduction rules* is applied. Each of these rules takes as input an instance (L, R, k) of Rectangle Stabbing and outputs in polynomial time an instance (L', R', k') of Rectangle Stabbing such that $|L'| \leq |L|$, $|R'| \leq |R|$, $k' \leq k$, and (L', R', k') is a *yes*-instance iff (L, R, k) is a *yes*-instance. An instance to which none of the rules can be applied is called *reduced*. Our data reduction rules read as follows.

Rule 1: If there are two lines $l_1, l_2 \in L$ such that every rectangle in R that is intersected by l_2 is also intersected by l_1, then delete l_2.

Rule 2: If there are two rectangles $r_1, r_2 \in R$ such that every line in L that intersects r_1 also intersects r_2, then delete r_2.

Rule 3: If there are $k+2$ rectangles $r_1, \ldots, r_{k+2} \in R$ such that no horizontal line intersects more than one of these rectangles and, for each $i \in \{1, \ldots, k+1\}$ we have $\mathrm{lx}(r_i) \geq \mathrm{lx}(r_{k+2})$ and $\mathrm{rx}(r_i) \leq \mathrm{rx}(r_{k+2})$, then delete r_{k+2}.

While the correctness of Rules 1 and 2 is obvious, the correctness of Rule 3 follows from the fact that k horizontal lines cannot intersect all rectangles $r_1, \ldots, r_{k+1}$. Note that the data reduction rules may transform an instance of Disjoint b-Square Stabbing into an instance of Rectangle Stabbing where the rectangles have heights or widths smaller than b; anyway, we call such an instance a *reduced instance of* Disjoint b-Square Stabbing.

The following observation is an immediate consequence of Rule 1.

Observation 2. *In a reduced problem instance of* Rectangle Stabbing*, for every vertical line $v_j \in V$, there exist rectangles $r, r' \in R$ with $\mathrm{lx}(r) = j$ and $\mathrm{rx}(r') = j$.*

Observation 3. *After applying any sequence of data reduction rules to an instance of* Disjoint b-Square Stabbing*, for any two rectangles $r_1, r_2 \in R$, we have $\mathrm{lx}(r_1) > \mathrm{lx}(r_2) \Rightarrow \mathrm{rx}(r_1) \geq \mathrm{rx}(r_2)$ and $\mathrm{rx}(r_1) < \mathrm{rx}(r_2) \Rightarrow \mathrm{lx}(r_1) \leq \mathrm{lx}(r_2)$.*

The latter observation follows from the fact that every instance of Disjoint b-Square Stabbing has the property described in the observation, and none of the data reduction rules destroys the property. The next observation follows directly from Rule 3; the proof of Lemma 1 is omitted.

Observation 4. *In a reduced instance of* Disjoint b-Square Stabbing*, for every $j \in \{1, \ldots, n\}$ there are at most $k+1$ rectangles r with $\mathrm{lx}(r) = j$.*

Lemma 1. *For every rectangle r in a reduced instance, there are at most k rectangles r' with $\mathrm{rx}(r') < \mathrm{rx}(r)$ and $\mathrm{lx}(r') \geq \mathrm{lx}(r)$, and all these rectangles have $\mathrm{lx}(r') = \mathrm{lx}(r)$.*

Lemma 2. *Let r be a rectangle with* $\mathrm{wh}(r) > xk+1$ *for* $x \geq 2$ *in a reduced instance. Then there exists a rectangle* r' *with* $\mathrm{lx}(r') < \mathrm{lx}(r)$ *and* $(x-1)k+1 < \mathrm{wh}(r') \leq xk+1$.

Proof. We show that the lemma is true if r has minimum $\mathrm{lx}(r)$ under all rectangles whose width is greater than $xk+1$; this suffices to prove the lemma.

Observation 2 implies the existence of a rectangle r' with $\mathrm{rx}(r') = p$ for every $p \in \{\mathrm{lx}(r), \ldots, \mathrm{rx}(r)-1\}$. Due to Lemma 1, at most k of these rectangles can have $\mathrm{lx}(r') \geq \mathrm{lx}(r)$, and, hence, there exists $p \in \{\mathrm{rx}(r)-k-1, \ldots, \mathrm{rx}(r)-1\}$ such that there is a rectangle r' with $\mathrm{rx}(r') = p$ and $\mathrm{lx}(r') < \mathrm{lx}(r)$. For the width of r' we have $\mathrm{wh}(r') = \mathrm{rx}(r') - \mathrm{lx}(r') + 1 \geq \mathrm{rx}(r) - k - 1 - \mathrm{lx}(r') + 1 > \mathrm{rx}(r) - k - 1 - \mathrm{lx}(r) + 1 = \mathrm{wh}(r) - k - 1$. Due to the selection of r, no rectangle r' with $\mathrm{lx}(r') < \mathrm{lx}(r)$ can have $\mathrm{wh}(r') > xk+1$, and, hence, we have $(x-1)k+1 < \mathrm{wh}(r') \leq xk+1$. □

Lemma 3. *If a reduced instance contains a rectangle r with* $\mathrm{wh}(r) > 2k+1$, *then there exists a rectangle r' with the following properties.*

1. $k+1 < \mathrm{wh}(r') \leq 2k+1$.
2. *There are at least k rectangles r'' with* $\mathrm{rx}(r'') \in \{\mathrm{lx}(r'), \ldots, \mathrm{rx}(r')-1\}$.
3. *All rectangles r'' with* $\mathrm{rx}(r'') \in \{\mathrm{lx}(r'), \ldots, \mathrm{rx}(r')-1\}$ *have* $\mathrm{lx}(r'') \leq \mathrm{lx}(r')$.
4. *All rectangles r'' with* $\mathrm{rx}(r'') \in \{\mathrm{lx}(r'), \ldots, \mathrm{rx}(r')-1\}$ *have* $\mathrm{wh}(r'') \leq 2k+1$.

Proof. The existence of a rectangle r' with $k+1 < \mathrm{wh}(r') \leq 2k+1$ follows from Lemma 2. Select a rectangle r' in such a way that $k+1 < \mathrm{wh}(r') \leq 2k+1$ and $\mathrm{lx}(r')$ is minimum under this property. Clearly, r' fulfills properties 2 and 3 because of Observations 2 and 3, respectively. Now assume, for the sake of a contradiction, that there is a rectangle r'' with $\mathrm{rx}(r'') \in \{\mathrm{lx}(r'), \ldots, \mathrm{rx}(r')-1\}$ and $\mathrm{wh}(r'') > 2k+1$. Clearly, we have $\mathrm{lx}(r'') < \mathrm{lx}(r')$. Together with Lemma 2, this implies the existence of a rectangle r''' that has $\mathrm{lx}(r''') < \mathrm{lx}(r'') < \mathrm{lx}(r')$ and $k+1 < \mathrm{wh}(r''') \leq 2k+1$, contradicting the selection of r'. □

Theorem 5. Disjoint b-Square Stabbing *is solvable in* $(4k+1)^k \cdot n^{O(1)}$ *time.*

Proof. If all rectangles have width at most $2k+1$, the instance can be solved in $(2k+2)^k \cdot n^{O(1)}$ time [2]. Otherwise, there is a rectangle r' as described in Lemma 3, such that the vertical line going through $\mathrm{lx}(r')$ intersects more than k rectangles whose width is at most $2k+1$. Since no horizontal line intersects more than one of these rectangles, not all of them can be stabbed by horizontal lines. Therefore, the solution must contain a vertical line intersecting at least two of these rectangles. There are at most $4k+1$ such lines. □

5 Open Questions

Is Rectangle Stabbing in FPT when the input consists of nonoverlapping arbitrary rectangles? Is there a polynomial-size problem kernel for Disjoint b-Square Stabbing? Is d-Dimensional Rectangle Stabbing in W[1] when parameterized by both k *and* d?

Acknowledgement. We thank Dániel Marx, who pointed us to the approach for proving that Rectangle Stabbing is in W[1].

References

1. Călinescu, G., Dumitrescu, A., Karloff, H.J., Wan, P.-J.: Separating points by axis-parallel lines. Internat. J. Comput. Geom. Appl. 15(6), 575–590 (2005)
2. Dom, M., Sikdar, S.: The parameterized complexity of the rectangle stabbing problem and its variants. In: Preparata, F.P., Wu, X., Yin, J. (eds.) FAW 2008. LNCS, vol. 5059, pp. 288–299. Springer, Heidelberg (2008)
3. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
4. Even, G., Levi, R., Rawitz, D., Schieber, B., Shahar, S., Sviridenko, M.: Algorithms for capacitated rectangle stabbing and lot sizing with joint set-up costs. ACM Trans. Algorithms 4(3), Article 34 (2008)
5. Fellows, M.R., Hermelin, D., Rosamond, F.A., Vialette, S.: On the parameterized complexity of multiple-interval graph problems (manuscript, 2008)
6. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
8. Gaur, D.R., Ibaraki, T., Krishnamurti, R.: Constant ratio approximation algorithms for the rectangle stabbing problem and the rectilinear partitioning problem. J. Algorithms 43(1), 138–152 (2002)
9. Giannopoulos, P., Knauer, C., Whitesides, S.: Parameterized complexity of geometric problems. The Computer Journal 51(3), 372–384 (2008)
10. Hassin, R., Megiddo, N.: Approximation algorithms for hitting objects with straight lines. Discrete Appl. Math. 30, 29–42 (1991)
11. Koushanfar, F., Slijepcevic, S., Potkonjak, M., Sangiovanni-Vincentelli, A.: Error-tolerant multimodal sensor fusion. In: Proceedings of the IEEE CAS Workshop on Wireless Communications and Networking. IEEE CAS, Los Alamitos (2002)
12. Kovaleva, S., Spieksma, F.C.R.: Approximation of a geometric set covering problem. In: Eades, P., Takaoka, T. (eds.) ISAAC 2001. LNCS, vol. 2223, pp. 493–501. Springer, Heidelberg (2001)
13. Kovaleva, S., Spieksma, F.C.R.: Approximation algorithms for rectangle stabbing and interval stabbing problems. SIAM J. Discrete Math. 20(3), 748–768 (2006)
14. Langerman, S., Morin, P.: Covering things with things. Discrete Comput. Geom. 33(4), 717–729 (2005)
15. Marx, D.: Parameterized complexity of independence and domination on geometric graphs. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 154–165. Springer, Heidelberg (2006)
16. Mecke, S., Schöbel, A., Wagner, D.: Station location – complexity and approximation. In: Kroon, L.G., Möhring, R.H. (eds.) Proc. 5th ATMOS. IBFI Dagstuhl, Germany (2005)
17. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
18. Xu, G., Xu, J.: Constant approximation algorithms for rectangle stabbing and related problems. Theory Comput. Syst. 40(2), 187–204 (2007)

Straight-Line Grid Drawings of Label-Constrained Outerplanar Graphs with $O(n \log n)$ Area (Extended Abstract)

Md. Rezaul Karim[1], Md. Jawaherul Alam[2], and Md. Saidur Rahman[2]

[1] Dept. of Computer Science and Engineering,
Bangladesh University of Engineering and Technology(BUET), Dhaka-1000,
Bangladesh. Also Dept. of Computer Science and Engineering, University of Dhaka,
Dhaka-1000, Bangladesh
rkarim@univdhaka.edu
[2] Dept. of Computer Science and Engineering,
Bangladesh University of Engineering and Technology(BUET),
Dhaka-1000, Bangladesh
jawaherul@yahoo.com, saidurrahman@cse.buet.ac.bd

Abstract. A straight-line grid drawing of a planar graph G is a drawing of G on an integer grid such that each vertex is drawn as a grid point and each edge is drawn as a straight-line segment without edge crossings. Any outerplanar graph of n vertices with the maximum degree d has a straight-line grid drawing with area $O(dn \log n)$. In this paper, we introduce a subclass of outerplanar graphs, which we call label-constrained outerplanar graphs, that admits straight-line grid drawings with $O(n \log n)$ area. We give a linear-time algorithm to find such a drawing. We also give a linear-time algorithm for recognition of label-constrained outerplanar graphs.

Keywords: Planar Graph, Outerplanar Graph, Label-Constrained Outerplanar Graph, Dual Graph, Straight-Line Drawing, Grid Drawing.

1 Introduction

Recently automatic aesthetic drawings of graphs have created intense interest due to their broad applications in computer networks, VLSI layout, information visualization etc., and as a consequence a number of drawing styles have come out [NR04]. The most typical and widely studied drawing style is the "straight-line grid drawing" of a planar graph. A *straight-line grid drawing* of a planar graph G is a drawing of G on an integer grid such that each vertex is drawn as a grid point and each edge is drawn as a straight-line segment without edge crossings as illustrated in Fig. 1(d). The *area* of such a drawing is the area of the smallest rectangle that encloses the drawing. It is well known that a planar graph of n vertices admits a straight-line grid drawing on a grid of

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 310–321, 2009.

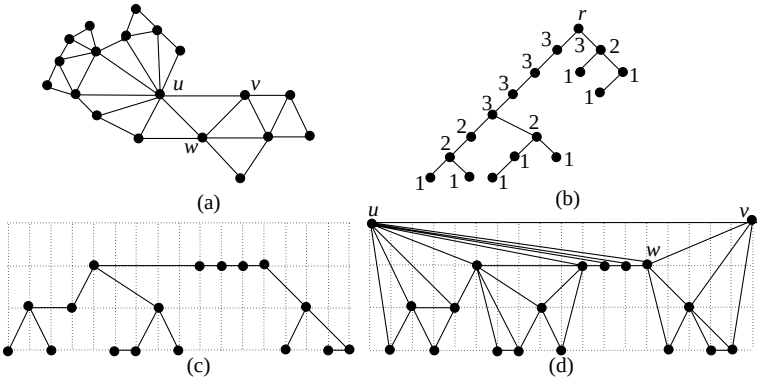


Fig. 1. (a) A label-constrained outerplanar graph G, (b) the dual rooted ordered tree T_r of G, (c) a straight-line grid drawing of T_r, and (d) a straight-line grid drawing of G.

area $O(n^2)$ [Sch90, FPP90]. A lower bound of $\Omega(n^2)$ on the area-requirement for straight-line grid drawings of certain planar graphs are also known [FPP90]. Although trees admit straight-line grid drawings with linear area [GR04b], it is generally thought that triangulations may require a grid of quadratic size. Hence finding nontrivial classes of planar graphs of n vertices richer than trees that admit straight-line grid drawings with area $o(n^2)$ is posted as an open problem in [BEGKLM04], and several recent works have addressed this open problem [GR04a, KR07, KR08] etc. The problem of finding straight-line grid drawings of outerplanar graphs with $o(n^2)$ area was first posed by Biedl in [Bie02], and Garg and Rusu showed that an outerplanar graph with n vertices and the maximum degree d has a planar straight-line drawing with area $O(dn^{1.48})$ [GR04a]. Di Battista and Frati showed that a "balanced" outerplanar graph of n vertices has a straight-line grid drawing with area $O(n)$ and a general outerplanar graph of n vertices has a straight-line grid drawing with area $O(n^{1.48})$ [DF06]. Recently Frati showed that a general outerplanar graph with n vertices admits a straight-line grid drawing with area $O(dn \log n)$, where d is the maximum degree of the graph [Fra07].

In this paper, we introduce a subclass of outerplanar graphs which has a straight-line grid drawing on a grid of area $O(n \log n)$. We give a linear-time algorithm to find such a drawing. We call this class "label-constrained outerplanar graphs" since a "vertex labeling" of the dual tree of this graph satisfies certain constraints. Fig. 1(a) illustrates a "label-constrained outerplanar graph" G, and a straight-line grid drawing of G with $O(n \log n)$ area is illustrated in Fig. 1(d). The "label-constrained outerplanar graphs" are richer than "balanced" outerplanar graphs. We also give a linear-time algorithm for recognition of a "label-constrained outerplanar graph."

The remainder of the paper is organized as follows. In Section 2, we give some definitions. Section 3 provides the drawing algorithm. Section 4 presents a linear-time algorithm for recognition of a "label-constrained outer planar graph," and Section 5 concludes the paper.

2 Preliminaries

In this section we give some definitions, introduce a labeling of a tree and define a class of outerplanar graphs which we call "label-constrained outerplanar graphs."

Let $G = (V, E)$ be a connected simple graph with vertex set V and edge set E. Throughout the paper, we denote by n the number of vertices in G, that is, $n = |V|$, and denote by m the number of edges in G, that is, $m{=}|E|$. We denote by $G - \{u, v\}$ a graph $G' = (V', E')$ where $V' = V(G) - \{u, v\}$ and $E' =$ set of edges induced by V' in G. A *path* in G is an ordered list of distinct vertices $v_1, v_2, ..., v_q \in V$ such that $(v_{i-1}, v_i) \in E$ for all $2 \leq i \leq q$. Vertices v_1 and v_q are the end-vertices of the path $v_1, v_2, ..., v_q$. We call a path *u-v path* if u and v are the end-vertices of the path.

A graph is *planar* if it can be embedded in the plane so that no two edges intersect geometrically except at a vertex to which they are both incident. A *plane graph* is a planar graph with a fixed embedding. A plane graph G divides the plane into connected regions called $faces$. A bounded region is called an *inner face* and the unbounded region is called the *outer face*. For a face f in G we denote by $V(f)$ the set of vertices of G on the boundary of face f.

A plane graph is an *outerplanar graph* if its all vertices lie on the outer face. Let G be an outerplanar graph. We now define the *dual tree* of G. The dual tree T of G is a tree whose vertices correspond to the inner faces of G, and two vertices x and y of T are adjacent if the faces of G corresponding to x and y share an edge in G. A *maximal outerplanar graph* is an outerplanar graph in which no edge can be added without losing outerplanarity. Clearly, each inner face of a maximal outerplanar graph has three edges. Therefore, a vertex of the dual tree T of a maximal outerplanar graph G has degree at most three, and hence T is a binary tree. It is easy to see that any outerplanar graph can be augmented in linear time to a maximal outerplanar graph by adding only linear number of extra edges.

We now define a vertex labeling of a rooted binary tree. Let T be a binary tree and let r be the root of T. Then the *labeling of a vertex u of T with respect to r*, which we denote by $L_r(u)$, is defined as follows:

(a) if u is a leaf node then $L_r(u) = 1$;
(b) if u has only one child q and $L_r(q) = k$ then $L_r(u) = k$;
(c) if u has two children s and t, and $L_r(s) = k$ and $L_r(t){=}k'$ where $k > k'$, then $L_r(u) = k$; and
(d) if u has two children s and t, and $L_r(s) = k$ and $L_r(t){=}k$ then $L_r(u) = k + 1$.

We denote by $L_r(T)$ the labeling of all the vertices of T with respect to r. Fig. 2 illustrates the vertex labeling of a binary tree T rooted at r where an integer value represents the label of the associated vertex. The following lemma is immediate from the labeling defined above.

Lemma 1. *Let T be a binary tree and let r be the root of T. Let u and v be two vertices of T such that u is an ancestor of v. If $L_r(u) = k$ and $L_r(v) = k'$, then $k \geq k'$.*

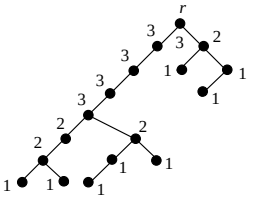


Fig. 2. Vertex labeling of a binary tree T

The following lemma gives an upper bound on the value of the vertex labeling. The proof of this lemma is omitted in this extended abstract.

Lemma 2. *Let T be a binary tree with n vertices and let r be the root of T. Assume that $L_r(r) = k$. Then $k = O(logn)$.*

We also have the following lemma whose proof is omitted in this extended abstract.

Lemma 3. *Let T be a binary tree and let r be the root of T. Assume that all the vertices of T are labeled with respect to r using vertex labeling. Let $V(k)$ be a set of vertices such that for all $v \in V(k)$, $L_r(v) = k$. Then any connected component of the subgraph of T induced by $V(k)$ is a path.*

A binary tree T is *ordered* if one child of each vertex v of T is designated as the *left child* and the other is designated as the *right child*. (Note that the left child or the right child of a vertex may be empty.) Let T be a rooted ordered binary tree. For any vertex $v \in T$, we call the subtree of T rooted at the left child (if any) of v the *left subtree* of v. Similarly we define the *right subtree* of v. We call an u-v path of T a *left-left path* if u is an ancestor of all the vertices of the path and each vertex except u of this path is the left child of its parent. Similarly we define a *right-right path* of T. We call a path of T a *cross path* if the path is neither a left-left path nor a right-right path. We call a maximal left-left path of T the *leftmost* path of T if one of the end vertex of the path is the root of T. Similarly we define the *rightmost* path of T. For any vertex $x \in T$, we call a path $x, v_1, ..., v_m$ the *left-right* path of x such that v_1 is the left child of x, and v_{i+1} is the right child of v_i where $1 \leq i \leq m-1$. Similarly we define the *right-left* path of x. We call $L_r(T)$ a *flat labeling* if any path induced by the vertices of T of the same label is either a left-left or a right-right path. We now have the following fact.

Fact 4. *Let T be an ordered binary tree and let r be the root of T. Let $L_r(T)$ be a flat labeling. Then for any vertex $x \in T$, each of the vertices of the left-right (right-left) path of x except the left (right) child of x has a smaller label than the label of x .*

Let G be a maximal outerplanar graph and let T be the dual tree of G. We convert T as a rooted ordered binary tree T_r by assigning its root r and ordering

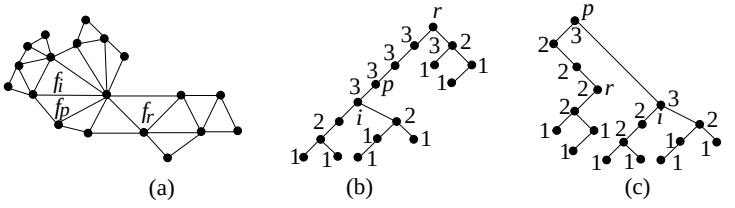


Fig. 3. (a) A maximal outerplanar graph G, (b) a rooted ordered binary dual tree T_r of G where the vertex r of T_r corresponds to the face f_r of G, and (c) another rooted ordered binary dual tree T_p of G where the vertex p of T_p corresponds to the face f_p of G

of the children of each vertex of T, as follows. Let r be a vertex of T such that r corresponds to an inner face f_r of G containing an edge (u, v) on the outerface. (Note that the degree of r is either one or two.) We regard r as the root of T_r. Let w be the vertex of f_r other than u, v such that u, v and w appear in the clockwise order on f_r. We call u and v the *poles of* f_r and w the *central vertex of* f_r. We also call u the *left vertex of* f_r and v the *right vertex of* f_r. The vertex of T corresponding to the face sharing the vertices v and w (if any) of f_r is the *right child of* r and the vertex of T corresponding to the face sharing the vertices u and w (if any) of f_r is the *left child of* r. Let p and q be two vertices of T_r such that p is the parent of q, and let f_p and f_q be the two faces of G corresponding to p and q in T_r. Let v_1, v_2 and v_3 be the vertices of f_q in the clockwise order such that v_1 and v_2 are also on f_p. Then v_1 and v_2 are poles of f_q, and v_3 is the central vertex of f_q. The vertex v_1 is the left vertex of f_q and the vertex v_2 is the right vertex of f_q. The vertex of T corresponding to the face sharing the vertices v_2 and v_3 (if any) of f_q is the right child of q and the vertex of T corresponding to the face sharing the vertices v_1 and v_3 (if any) of f_q is the left child of q. Thus we have converted the dual tree T of a maximal outerplanar graph G to a rooted ordered dual tree T_r.

We are now ready to give the definition of "label-constrained outerplanar graphs." Let G be a maximal outerplanar graph and let T be the dual tree of G. We call G a *label-constrained outerplanar graph* if T can be converted to a rooted ordered binary dual tree T_r such that $L_r(T_r)$ is a flat labeling. Fig. 3(a) illustrates a label-constrained outerplanar graph G since $L_r(T_r)$ a flat labeling as illustrated in Fig. 3(b). The $L_p(T_p)$ is not a flat labeling since T_p has a cross path induced by the label 2 vertices as illustrated in Fig. 3(c).

3 Drawing Algorithm

In this section we give a linear-time algorithm for finding a straight-line grid drawing of a label-constrained outerplanar graph with $O(n \log n)$ area.

Let G be a maximal outerplanar graph and let T_r be the rooted ordered binary dual tree of G taking a vertex r of T_r as the root. In [DF06], Di Battista and Frati defined a bijection function γ between the vertices of T_r and vertices of G

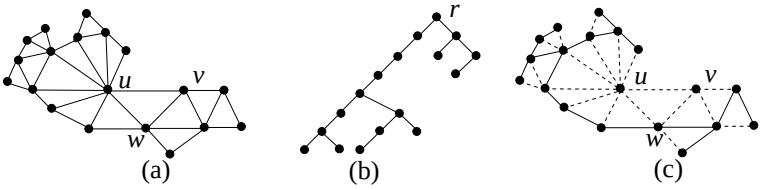


Fig. 4. (a) A maximal outerplanar graph G, (b) the dual rooted ordered tree T_r of G, and (c) a spanning-tree T' of $G - \{u, v\}$ is drawn by the solid lines

except for the poles of the face f_r corresponding to the root of T_r where each of the vertex of T_r is mapped to the central vertex of the corresponding face of G. We immediately have the following lemma from [DF06].

Lemma 5. *Let G be a maximal outerplanar graph and let T_r be the rooted ordered binary dual tree of G. Then G contains a copy of T_r which is a spanning tree T' of $G - \{u, v\}$, where u and v are the poles of the face f_r corresponding to the root of T_r.*

Fig. 4(b) illustrates the dual tree of G in Fig. 4(a) where the root r of T_r corresponds to the face f_r of G containing vertices u, v and w in the clockwise order. G contains a copy of T_r, which is a spanning tree T' of $G-\{u, v\}$, such that each of the vertices of T_r is mapped to the central vertex of the corresponding face of G as illustrated in Fig. 4(c) where the edges of T' are drawn by solid lines.

Our idea is as follows. We first draw the rooted ordered binary dual tree T_r of a label-constrained outerplanar graph G. We then put the poles of the face f_r corresponding to the root r of T_r, and add the edges of G which are not in T_r. The x-coordinates of the vertices of T_r are assigned in the order of the inorder traversal of T_r in the increasing order starting from 1. The y-coordinate of a vertex of T_r is the label of the vertex minus one. We now put the vertices of T_r at the calculated coordinates and add the required edges to complete the drawing of T_r. Fig. 1(c) illustrates a straight-line grid drawing of T_r in Fig. 1(b). We now put the left vertex and the right vertex of the face f_r of G at $(0, k)$ and at $(n-1, k)$ respectively, where $L_r(T_r) = k$. We now add the edges of G which are not in T_r using straight-line segments, and complete the drawing of G. We call the algorithm described above for drawing an outerplanar graph Algorithm **Draw-Graph**. We now have the following theorem.

Theorem 1. *Let G be a label-constrained outerplanar graph. Then Algorithm* **Draw-Graph** *finds a straight-line grid drawing of G with $O(n \log n)$ area in linear time.*

In the rest of the section, we give a proof of Theorem 1. We first show that Algorithm **Draw-Graph** produces a straight-line grid drawing of G. The following lemmas are immediate from the assignment of x-coordinates and y-coordinates of the vertices of T_r.

Lemma 6. *Let G be a label-constrained outerplanar graph and let T_r be a rooted ordered binary dual tree of G. Let u be a vertex of T_r, and let s and t be the left and the right child of u, respectively. Then the x-coordinate of any vertex of the subtree rooted at s is less than the x-coordinate of any vertex of the subtree rooted at t.*

Lemma 7. *Let G be a label-constrained outerplanar graph and let T_r be a rooted ordered binary dual tree of G. Let u and v be vertices of T_r where u is an ancestor of v. Then the y-coordinate of u is greater than or equal to the y-coordinate of v.*

We also have the following lemma, which is immediate from the definition of the left and right vertex of a face in a maximal outerplanar graph.

Lemma 8. *Let G be a maximal outerplanar graph and let T_r be a rooted ordered binary dual tree of G. Let q be any vertex of T_r, and let x and y be the left and the right child of q, respectively. Let f_q, f_x and f_y be the faces of G corresponds to the vertices q, x and y in T_r. Then the left vertex of f_q is the left vertex of f_x and the right vertex of f_q is the right vertex of f_y.*

The drawing of T_r is a straight-line grid drawing immediate from the Lemmas 6 and 7. We now show that each of the edges of G those are not in T_r can be drawn using straight-line segments without any edge crossings. An edge between the two pole vertices of the face corresponding to the root of T_r can be drawn using a straight-line segment without any crossings with the existing drawing of T_r since each of the pole vertices is placed above all the vertices of T_r. By Lemma 8, the left vertex of the face corresponding to the root of T_r is the left vertex of the faces corresponding to the vertices of the leftmost path of T_r. Therefore the left vertex of the face corresponding to the root of T_r is adjacent to all the vertices of the leftmost path of T_r in G. These edges can be drawn using straight-line segments without any edge crossings since the left vertex of the face corresponding to the root of T_r is placed strictly to the left and above of all the vertices on the leftmost path of T_r. Similarly, we can draw the edges between the right vertex of the face corresponding to the root of T_r and the vertices on the rightmost path of T_r using straight-line segments without any edge crossings. In a maximal outerplanar graph, the rest of the edges are between any vertex $v \in T_r$ and a vertex on the left-right or the right-left path of v. From the x-coordinate of any vertex $v \in T_r$ and by Fact 4, one can see that a vertex $v \in T_r$ is placed strictly to the left (right) and above of all the vertices of the right-left (left-right) path of v except the right (left) child of v. Thus all such edges can be drawn using straight-line segments without any edge crossings and hence the following lemma holds.

Lemma 9. *Let G be a label-constrained outerplanar graph. Then Algorithm* **Draw-Graph** *finds a straight-line grid drawing of G.*

Fig. 1(d) illustrates the straight-line grid drawing of G in Fig. 1(a).

Proof of Theorem 1: By Lemma 9, the drawing of G is a straight-line grid drawing. The height of the drawing of G is the label of the root of the dual tree

of G. By Lemma 2, the height of the drawing of G is $O(\log n)$. The width of the drawing is $O(n)$. Therefore the area of the drawing is $O(n \log n)$. One can easily see that the drawing of G can be found in linear time.

4 Recognition of a Label-Constrained Outerplanar Graph

In this section we give a linear-time algorithm for recognition of a label constrained outerplanar graph.

Let G be a maximal outerplanar graph and let T_r be a rooted ordered binary dual tree of G taking a vertex r as the root. (Note that r corresponds to an inner face f_r of G having an edge on the outer face.) From the definition of the vertex labeling of a binary tree in Section 2, one can easily see that $L_r(T_r)$ can be done by a bottom-up computation in linear time. The verification whether $L_r(T_r)$ is a flat labeling can also be done in linear time. In case $L_r(T_r)$ is not a flat labeling, $L_p(T_p)$ might be a flat labeling where T_p is a rooted ordered binary dual tree of G rooted at a vertex p of degree one or two other than r. Therefore to examine whether G is a label-constrained outerplanar graph or not, we have to compute the vertex labeling of the rooted ordered binary dual tree of G rooted at each vertex of degree one or degree two in the dual tree T of G and verify whether each such a labeling of vertices is a flat labeling or not. Hence the recognition of a label-constrained outerplanar graph requires $O(n^2)$ time by a naive approach.

Before presenting our linear-time recognition algorithm, we present the following observation. Fig. 3(b) and (c) illustrate T_r and T_p of a maximal outerplanar graph G in Fig. 3(a) where r and p correspond to the faces f_r and f_p of G respectively. Note that the vertex i is the left child of p in T_r in Fig 3(b); but it is the right child of p in T_p in Fig 3(c). Thus we can not get the rooted ordered binary dual tree T_p of G immediately from T_r by simply choosing the vertex p of T_r as the new root without taking care about the ordering of the children of each vertices. However one can observe that the ordering of the children of a vertex is changed from T_r to T_p only for the vertices on the r-p path of T_r. For the rest of the vertices, the ordering of the children is unchanged in both T_r and T_p. This change in the ordering of the children is presented in the following three lemmas, which are immediate from the definition of the ordering of the children of a vertex in the rooted ordered binary dual tree.

Lemma 10. *Let G be a maximal outerplanar graph and let T_r be a rooted ordered binary dual tree of G. Let s be the right (left) child of r in T_r. Let p be a vertex of T other than r such that the degree of p is one or two, and the vertex s remains as a child of r in T_p. Then s is the left (right) child of r in T_p.*

Lemma 11. *Let G be a maximal outerplanar graph and let T_r be a rooted ordered binary dual tree of G. Let s be a vertex of T_r other than r such that the degree of vertex s is one or two, and let t be the right (left) child of s in T_r. Then t is the left (right) child of s in T_s.*

Lemma 12. *Let G be a maximal outerplanar graph and let T_r be a rooted ordered binary dual tree of G. Let x be a degree three vertex such that s is the parent of x,*

and p and q are the left and the right children of x in T_r. Let y be a descendant of x in the left (right) subtree of x in T_r such that the degree of y is one or two. Then s (p) is the right child of x and q (s) is the left child of x in T_y.

We now can compute the labeling of a rooted ordered dual tree T_x of a maximal outerplanar graph G from a given labeling of a rooted ordered dual tree T_y of G using the Lemmas 10, 11, 12, where vertex x and y corresponds to the faces f_x and f_y of G, as in the following lemmas.

Lemma 13. *Let G be a maximal outerplanar graph and let T_r be a rooted ordered binary dual tree of G. Then $L_p(T_p)$ can be computed in constant time if $L_r(T_r)$ is given where p is a child of r in T_r and the degree of p is either one or two.*

Proof. Without loss of generality, we may assume that p is the left child of r in T_r. We also assume that q is the right child of r (if any) in T_r, and p has the right child s (if any) in T_r.The labeling of the vertices of the subtrees rooted at q and s is the same in both of the $L_r(T_r)$ and the $L_p(T_p)$. Therefore we need to compute the label of r and p with respect to p to compute $L_p(T_p)$. By Lemma 10, q becomes the left child of r in T_p, and the label of r is same as the label of q. A cross path is detected at r in T_p if q has the same label as its right child. By Lemma 11, s becomes the left child of p in T_p. Then r becomes the right child of p in T_p. The label of p is computed from the label of r and s. One can also determine in constant time whether there is a cross path at p or not. $\mathcal{Q}.\mathcal{E}.\mathcal{D}.$

Lemma 14. *Let G be a maximal outer planar graph and let T_r be a rooted ordered binary dual tree of G. Then $L_x(p)$ can be computed in constant time if $L_r(T_r)$ is given where p is a child of r, and x is a descendant of p and the degree of x is either one or two.*

Proof. Without loss of generality, we assume that p is the left child of r in T_r. We also assume that q is the right child of r (if any) in T_r. We need to compute the label of r and p with respect to x to compute $L_x(p)$. By Lemma 10, q becomes the left child of r in T_x, and the label of r is same as the label of q. A cross path is detected at r if q has the same label as its right child. We now have two cases to consider.

Case 1: The degree of p is two.
In this case, we assume that s is the child of p in T_r. The label of p would be same as r since r is the only child of p in T_x. One can also determine in constant time whether there is a cross path at p as illustrated in Fig. 5(i). Fig. 5(i)(b) and (d) illustrate the cases where s is the left and right child respectively, of p in T_r (the thick line represents the cross path).

Case 2: The degree of p is three.
In this case, we assume that s and t are the left and the right child of p in T_r. If x is in the left subtree of p in T_r, then by Lemma 12, r is the right child and t is the left child of p in T_x as illustrated in Fig. 5(ii)(b). Fig. 5(ii)(d) illustrates the case where x is in the right subtree of p in T_r. Then $L_x(p)$ can be computed from the labeling of r and t (s).

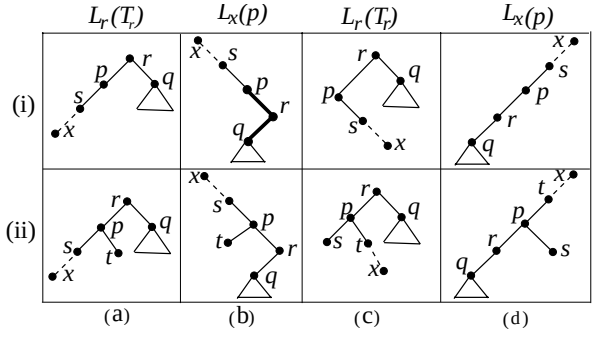


Fig. 5. Illustration of different cases of computing $L_x(p)$ from given $L_r(T_r)$

Thus we can compute $L_x(p)$ in constant time. Furthermore, one can determine in constant time whether there is a cross path at p or not. $\mathcal{Q}.\mathcal{E}.\mathcal{D}.$

We are now ready to present our algorithm. Let G be a maximal outerplanar graph and let T_r be a rooted ordered binary dual tree of G. We first compute $L_r(T_r)$. We have done if $L_r(T_r)$ is a flat labeling. Then G is a label-constrained outerplanar graph. We thus assume that $L_r(T_r)$ is not a flat labeling. We now have the following lemma whose proof is omitted in this extended abstract.

Lemma 15. *Let G be a maximal outerplanar graph and let T_r be a rooted ordered binary dual tree of G. If $L_r(T_r)$ is not a flat labeling and there exists a vertex $x \in T_r$ such that both of the subtrees of x contains a cross path, then G is not a label-constrained outerplanar graph.*

We thus assume that $L_r(T_r)$ is not a flat labeling and there exists no vertex $x \in T_r$ such that both the subtrees of x contain a cross path. In this case, each of the vertices of T_r at which a cross path is detected lies on a single path and r is an end vertex of this path. Let us assume that u is the other end vertex of this path, i.e., u is the farthest vertex from r at which a cross path is detected. Then for any vertex v of T_r which is neither u nor a descendant of u in T_r, $L_v(T_v)$ can not be a flat labeling since labeling is done in a bottom-up approach. We thus concentrate our attention only on the vertices of the subtree rooted at u in T_r as a probable new root for computing labeling. We now have two cases to consider.

Case 1: u is not the root in T_r.
In this case, we have two subcases to consider.

Subcase 1(a): The degree of u is two.
By Lemmas 13 and 14, we can compute $L_u(T_u)$ in the $O(l)$ time where l is the length of the r-u path. If a cross path is detected at any ancestor x of u in T_r during this step then for any descendant y of x in T_r, $L_y(T_y)$ can not be a flat labeling. Since u and all the descendants of u are also descendants of x in T_r, for any vertex v in T_r, $L_v(T_v)$ can not be a flat labeling. Hence G is not

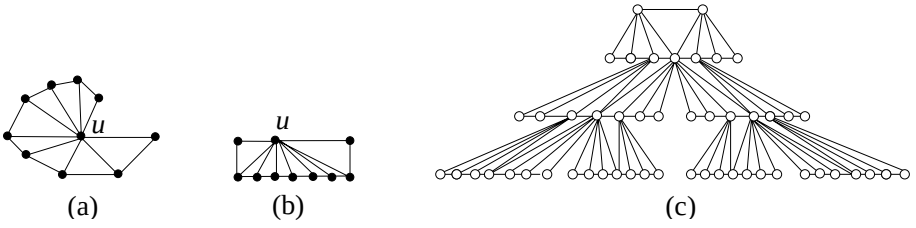


Fig. 6. (a) A trivial outerplanar graph G with maximum degree $n-1$, (b) a straight-line grid drawing of G with $O(n)$ area, and (c) an example of an outerplanar graph G which has maximum degree $O(n^{0.5})$

label-constrained. Thus we assume that no cross path is detected at any ancestor x of u. We have done if $L_u(T_u)$ is a flat labeling. Otherwise, a cross path is detected at u. In this case, we have to compute $L_x(T_x)$ for any descendant x of u in T_r with degree one or two and verify whether $L_x(T_x)$ is a flat labeling for recognizing G a label-constrained outerplanar graph, and this can be done in linear time by Lemma 13 and Lemma 14.

Subcase 1(b): The degree of u is three.
Let v be any vertex on the r-u path other than u then for all the descendants x of u in T_r the value of $L_x(v)$ is the same. We can compute $L_x(v)$ for all such vertices v on the r-u path other than u in $O(l)$ time where l is the length of the r-u path and x is any degree one or two descendant of u. Again, for any ancestor y of u, if a cross path is detected at y, then G can not be a label-constrained outerplanar graph. Otherwise, we have to compute $L_x(T_x)$ for any descendant x of u in T_r with degree one or two and verify whether $L_x(T_x)$ is a flat labeling for recognizing G a label-constrained outerplanar graph, and this can be done in linear time by Lemma 13 and Lemma 14.

Case 2: u is the root in T_r.
In this case, we have to compute $L_x(T_x)$ for any descendant x of r in T_r with degree one or two and verify whether $L_x(T_x)$ is a flat labeling for recognizing G a label-constrained outerplanar graph, and this can also be done in linear time by Lemma 13 and Lemma 14.

Thus one can recognize a label-constrained outerplanar graph in linear time. We have the following theorem.

Theorem 2. *Let G be a maximal outerplanar graph and let T be the dual tree of G. Then one can decide in linear time whether there exists a vertex r of T such that $L_r(T_r)$ is a flat labeling where T_r is the rooted ordered binary dual tree of G. Furthermore, one can find such a vertex r of T in linear time if such a vertex exists.*

5 Conclusion

In this paper we introduced a subclass of outerplanar graphs, which we call label-constrained outerplanar graphs. A graph in this class has a straight-line grid drawing on a grid of $O(n \log n)$ area, and the drawing can be found in linear

time. We gave an algorithm to recognize a label-constrained outerplanar graph in linear time. Our drawing algorithm is based on a very simple and natural labeling of a tree. The labeling is bounded by $O(\log n)$, and the labeling might be adopted for solving some other tree-related problems.

The previously best known area bound for an outerplanar graph is $O(dn \log n)$ due to [Fra07], where d is the maximum degree of the outerplanar graph G. This immediately gives an $O(n \log n)$ area bound if the maximum degree of G is bounded by a constant. But the maximum degree of an outerplanar graph is not always bounded by a constant. A trivial outerplanar graph may have the maximum degree $n - 1$ as illustrated in Fig. 6(a) although it requires $O(n)$ area for straight-line grid drawing as illustrated in Fig. 6(b). However, there are an infinite number of outerplanar graphs, as one illustrated in Fig. 6(c), which have the maximum degree $O(n^{0.5})$. A straight-line grid drawing of such a graph obtained by the algorithm of Frati [Fra07] requires $O(n^{1.5} \log n)$ area whereas our algorithm produces the drawings with $O(n \log n)$ area.

References

[BEGKLM04] Brandenburg, F., Eppstein, D., Goodrich, M.T., Kobourov, S., Liotta, G., Mutzel, P.: Selected open problems in graph drawing. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 515–539. Springer, Heidelberg (2004)

[Bie02] Biedl, T.C.: Drawing outerplanar graphs in $O(n \log n)$ area. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 54–65. Springer, Heidelberg (2002)

[DF06] Di Battista, G., Frati, F.: Small area drawings of outerplanar graphs. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 89–100. Springer, Heidelberg (2006)

[FPP90] de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10, 41–51 (1990)

[Fra07] Frati, F.: Straight-line drawings of outerplanar graphs in $O(dn \log n)$ area. In: Proc. of the 19th Canadian Conference on Computational Geometry, pp. 225–228 (2007)

[GR04a] Garg, A., Rusu, A.: Area-efficient drawings of outerplanar graphs. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 129–134. Springer, Heidelberg (2004)

[GR04b] Garg, A., Rusu, A.: A more practical algorithm for drawing binary trees in linear area with arbitrary aspect ratio. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 159–165. Springer, Heidelberg (2004)

[KR07] Karim, M.R., Rahman, M.S.: Straight-line grid drawings of planar graphs with linear area. In: Proc. of Asia-pacific Symposium on Visualization, 2007, pp. 109–112. IEEE, Los Alamitos (2007)

[KR08] Karim, M.R., Rahman, M.S.: Four-connected spanning subgraphs of doughnut graphs. In: Nakano, S.-i., Rahman, M. S. (eds.) WALCOM 2008. LNCS, vol. 4921, pp. 132–143. Springer, Heidelberg (2008)

[NR04] Nishizeki, T., Rahman, M.S.: Planar Graph Drawing. World Scientific, Singapore (2004)

[Sch90] Schnyder, W.: Embedding planar graphs on the grid. In: Proc. of the First ACM-SIAM Symp. on Discrete Algorithms, San Francisco, pp. 138–148 (1990)

Matched Drawability of Graph Pairs and of Graph Triples*

Luca Grilli[1], Seok-Hee Hong[2], Giuseppe Liotta[1],
Henk Meijer[3], and Stephen K. Wismath[4]

[1] Dip. di Ingegneria Elettronica e dell'Informazione, Università degli Studi di Perugia
{luca.grilli,liotta}@diei.unipg.it
[2] School of Information Technologies, University of Sydney
shhong@it.usyd.edu.au
[3] Science Department, Roosevelt Academy, the Netherlands
h.meijer@roac.nl
[4] Department of Mathematics and Computer Science, University of Lethbridge
wismath@cs.uleth.ca

Abstract. The contribution of this paper is twofold. It presents a new approach to the matched drawability problem of pairs of planar graphs and provides three algorithms based on this approach for drawing the pairs outerplane-outerpillar, outerplane-wheel and wheel-wheel. Further, it initiates the study of the matched drawability of triples of planar graphs: It presents an algorithm to compute a matched drawing of a triple of cycles and an algorithm to compute a matched drawing of a caterpillar and two universal level planar graphs. The results extend previous work on the subject and relate to existing literature about simultaneous embeddability and universal level planarity.

1 Introduction

Let $G_0, G_1, \ldots, G_{k-1}$ be a set of planar graphs such that for every pair G_i, G_j a vertex of G_i is matched to a distinct vertex of G_j. A matched drawing of $G_0, G_1, \ldots, G_{k-1}$ consists of k straight-line planar drawings $\Gamma_0, \Gamma_1, \ldots, \Gamma_{k-1}$ such that Γ_i represents G_i and for every pair G_i, G_j ($i \neq j, 0 \leq i, j \leq k-1$) the edges that describe the matched vertices are parallel.

Matched drawings can be regarded as a variant of the geometric simultaneous embeddings defined in [1]. In a geometric simultaneous embedding of two (or more) planar graphs, which have their vertex set in common, a planar straight-line drawing of each graph is computed such that every vertex has the same location in all drawings. If an edge is shared by two graphs, it is therefore represented by the same straight-line segment which makes it easy to visually discover common patterns in the drawings. However, the families of planar graphs that admit a geometric simultaneous embedding have been proven to be rather restricted (see [1,7,8]). Requiring that the same vertices share only one of their coordinates, as is the case for matched drawings of pairs of graphs, is a natural relaxation to consider.

* Research partially supported by the MIUR Project "MAINSTREAM: Algorithms for massive information structures and data streams" and by NSERC.

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 322–333, 2009.

The study of matched drawings fits within the application framework recently described by Collins and Carpendale [2] who, motivated by exploring relationships between relational sets of data, present a system that matches vertices of independent drawings. Collins and Carpendale [2] compute the drawings independent of one another, which may give rise to many crossings among the edges that describe correspondences between vertices in different visualizations. By studying matched drawings we explore the possibility of visualizing pairs of graphs such that, if there is a bijection between their vertices, this bijection is described by a set of parallel edges.

Matched drawings of a pair of planar graphs have been first defined and studied in [4]. The authors proved that not all planar pairs admit a matched drawing for any possible bijection between their vertices. They also described meaningful pairs of planar graphs that always admit a matched drawing. Among the different families of matched drawable graphs, they showed that every pair of trees with a bijection between their vertices admits a matched drawing while it is known that not all tree pairs admit a geometric simultaneous embedding [8]. In this paper we extend the results of [4] in different directions. Namely, we present new pairs of planar graphs that admit a matched drawing and we initiate the study of matched drawings for more than two planar graphs. More precisely, our main results can be outlined as follows.

1. We present a novel approach for computing matched drawings of pairs of planar graphs. This technique is based on a suitable coloring of the vertices of the graph and on characterizing vertex levelings of graphs that always admit a level planar realization. In this respect, our results can be related to the well-known research area about level planarity and about universal level planarity (see [5,6]). As application examples of the proposed technique, we prove that two wheel graphs, an outerplanar and a wheel, and an outerplanar and an outerpillar are pairs that admit a matched drawing.
2. We introduce and study the notion of triangular matched drawing for a triple of planar graphs. By using this concept we find new differences between matched drawability and simultaneous drawability. Namely, while it is known that triples of graphs with the same vertex set may not have a geometric simultaneous embedding even in the case that they are simple paths [4], it turns out that a triple consisting of a caterpillar and two universal level planar graphs [6] is always triangular matched drawable.
3. We use the drawing technique behind the result of (2), combined with a suitable sequence of geometric translations to show that every triple of cycles with the same vertex set admits a triangular matched drawing.

The remainder of the paper is organized as follows. Preliminaries can be found in Sect. 2. Our approach for computing matched drawings of pairs of graphs is described in Sect. 3. Matched drawings of more than two graphs are discussed in Sect. 4. Finally, Sect. 5 lists some open problems.

2 Preliminaries

We assume the reader is familiar with standard notions of graph drawing [3,10,11,12]. A *wheel* is a graph consisting of a cycle plus a vertex, called the *center* of the wheel

and connected to all the vertices of the cycle. A *fan* is a graph formed by a path plus a vertex, called the *apex*, connected to all the vertices of the path; a fan can be obtained from a wheel by removing an edge not incident to its center. An *outerplanar* graph is a graph that admits a planar embedding such that all vertices are on the same face. Such an embedding is called an *outerplanar embedding* and the face containing all the vertices is referred to as the *unbounded* or the *external* face. The other faces are called *bounded* or *internal* faces. An outerplanar graph is a *maximal outerplanar graph* if the addition of any edge destroys its outerplanarity. An *outerplane* graph is an outerplanar graph with a given outerplanar embedding. Let G be an outerplane graph. The *weak dual* G^* of G is a graph whose vertices are the internal faces of G and such that two vertices are adjacent if the corresponding internal faces in G share a common edge. If G is maximal and has at least three vertices, its internal faces are triangular and G^* is a tree whose nodes have degree at most three. Note that G always contains a vertex of degree two: a vertex $v \in G$ has degree two if and only if v belongs to a face that is a leaf of G^*. Throughout the paper n denotes the number of vertices of a graph.

3 Matched Drawings of Pairs of Graphs

Let G_0, G_1 be two planar graphs with the same number of vertices. We say that the pair G_0, G_1 defines a *matched pair*, denoted as $\langle G_0, G_1 \rangle$ if there is a bijection Φ from the vertices of G_0 to the vertices of G_1; two vertices u, v with $u \in G_0$ and $v \in G_1$ are *matched* if $\Phi(u) = v$. Let Γ_0 and Γ_1 be two straight-line planar drawings of G_0 and G_1, respectively. The pair of drawings $\langle \Gamma_0, \Gamma_1 \rangle$ is a *matched drawing* of $\langle G_0, G_1 \rangle$ if the edges $(u, \Phi(u))$ that describe the bijection are all parallel to a common direction. For reasons of concreteness and without loss of generality, we shall assume in the following that any two matched vertices have the same y-coordinate in $\langle \Gamma_0, \Gamma_1 \rangle$. A matched pair is *matched drawable* if it admits a matched drawing.

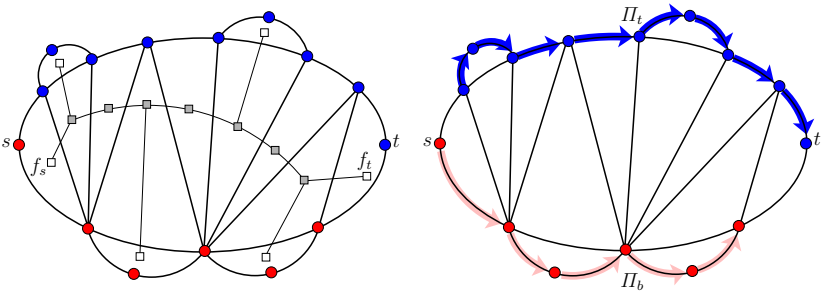
In [4] it has been proven that every matched pair $\langle T_0, T_1 \rangle$ where both T_0 and T_1 are trees is matched drawable. A matched drawing of two trees is constructed by a vertex addition strategy that alternately chooses the next vertex to be drawn from T_0 and from T_1: T_0 determines which vertex is placed in odd steps at y-coordinates $n - 1, n - 2, \ldots, \lfloor n/2 \rfloor + 1$ while T_1 determines which vertex is placed in even steps at y-coordinates $0, 1, \ldots, \lfloor n/2 \rfloor$. We extend this result by presenting new pairs of matched drawable graphs. This is done by using a drawing approach that looks similar but is also quite different from the one of [4]. The main idea behind our approach is to separate the role that the two graphs have in the computation. Based on the topology of one of the two graphs, the vertices are given one of two colors. Based on the topology of the other graph, the y-coordinates are defined. Finally, the matched drawing is computed. More precisely, our approach can be described as consisting of the following three main phases. Since there is a bijection between the vertices of G_0 and of G_1, we may as well assume that the vertex set of G_0 coincides with the vertex set of G_1 and denote this set as V.

1. **Coloring Phase.** Vertices of V are associated with labels in the set $\{B, T\}$. Such a labeling, called a *BT-labeling*, specifies that vertices labeled as B ("Bottom") will be drawn below those labeled as T ("Top"). The BT-labeling exclusively depends on the topology of G_1.
2. **Leveling Phase.** Let $Y = \{y_0, y_1, \ldots, y_{n-1}\}$ be a set of n distinct non-negative integers. By considering the topology of G_0 each vertex is assigned a number in set Y such that every vertex labeled B is given a number smaller than any vertex labeled T. We call this phase the *color-preserving level assignment*.
3. **Drawing Phase.** A matched drawing $\langle \Gamma_0, \Gamma_1 \rangle$ is computed; the y-coordinates of the vertices of V are those defined by the color preserving level assignment.

The Drawing Phase implies proving the following property: for any color-preserving level assignment of the vertices of G_1, there always exists a planar straight-line drawing of G_1 where every vertex is given a y-coordinate equal to its level number. This will be clarified in the following subsections. It is worth noting that the graphs of the matched pairs that we study in the next sections may not have level preserving straight-line drawings for all possible levelings of their vertices [6].

3.1 Matched Drawing of ⟨Outerplane, Outerpillar⟩

A *caterpillar* is a tree such that removing all of its leaves gives rise to a path; this path is called the *spine* of the caterpillar. An outerplanar graph G is an *outerpillar* if G is maximal and it has an outerplanar embedding whose weak dual is a caterpillar. An outerpillar with such an outerplanar embedding is called an *outerplane outerpillar*. Figure 1(a) shows a drawing of an outerplane outerpillar; the weak dual is also depicted and the vertices of its spine are in grey. We show that a matched pair ⟨outerplane, outerpillar⟩ is matched drawable. We assume that even the outerplane graph is maximal; if not, it can be made maximal by means of a suitable addition of edges between vertices of non-triangular faces. Also, the outerplanar embedding of the outerpillar, whose weak dual is a caterpillar, is assumed to be known. We describe how to construct a matched drawing by executing the three phases of our general methodology.



(a) (b)

Fig. 1. (a) A drawing of an outerplane outerpillar and of its weak dual. (b) An example of a bottom path (bold light color) and of a top path (bold darker color).

Coloring Phase. Let G and G^* be the outerplane outerpillar and its weak dual, respectively. Since G is maximal, G^* is a caterpillar with maximum vertex degree three (see Fig. 1(a)). Let f_s and f_t denote two leaf nodes adjacent to different end-vertices of the spine of G^*, respectively. Let s and t denote two vertices with degree two of f_s and f_t, respectively.

Vertices s and t are used to split the boundary of the external face of G into two disjoint oriented paths. The *bottom path*, denoted as Π_b consists of the sequence of vertices encountered when traversing counterclockwise the boundary of the external face of G starting from s to the vertex preceding t. The *top path*, denoted as Π_t, is the sequence of vertices encountered when traversing clockwise the boundary of the external face of G starting from the vertex that follows s to t. Figure 1(b) shows examples of top and bottom paths.

The BT-labeling of the vertices of V is defined by assigning the label T to each vertex of Π_t and the label B to each vertex of Π_b.

Leveling Phase. A color preserving level assignment of the vertices of V is computed by constructing a straight-line planar drawing of the outerplane graph that preserves the BT-labeling.

Lemma 1. *Let G be an outerplane graph with a given BT-labeling. G admits a straight-line planar drawing such that every vertex labeled B is drawn below any vertex labeled T.*

Sketch of Proof: A matched drawing is constructed with a vertex addition strategy. At each step a new vertex and its incident edges are drawn such that a new face is added to the current drawing. Vertices are assigned n distinct integer y-coordinates in the interval $[0, n-1]$. Each of these y-coordinates is initially marked as *unused*; it is marked as *used* when it is assigned to a vertex during the drawing procedure.

Let G^* be the weak dual of G, f_0 be the face of G corresponding to a leaf of G^*, and e_0 be an edge of f_0 that also belongs to the external face of G. Let u and v be the end-vertices of e_0. Vertices u and v are drawn at Step 0 and at Step 1. If both u and v have a B label, they are assigned the y-coordinates 0 and 1. If they both have a T-label, they are assigned the y-coordinates $n-1$ and $n-2$. Otherwise, the vertex with label B is given y-coordinate 0 and the other is given y-coordinate $n-1$; the x-coordinate of u is 0 and the x-coordinate of v is 1. At Step i ($2 \leq i \leq n-1$), a new vertex and its incident edges are drawn such that a new face is added to the current drawing. The new face to be added is chosen by performing a BFS visit of G^* that starts at f_0 (that is, at Step 2 the vertex of G other than u and v forming f_0 is drawn; at Step 3 a new face adjacent to f_0 is drawn, and so on).

At any Step i such that $1 \leq i \leq n-1$, the following two invariants are maintained: (i) for each edge e of the drawing there exists a triangular region called *safe region of* e and denoted as R_e such that one of the sides of R_e is e and at least two sides of R_e intersect every unused y-coordinate; (ii) for any two edges in the drawing either the interior of their safe regions do not intersect or one safe region is completely contained in the other.

The invariants can be easily verified at the end of Step 1. Assume that the invariant is maintained at Step $i-1$, and let w be the vertex of G processed at Step i. By adding

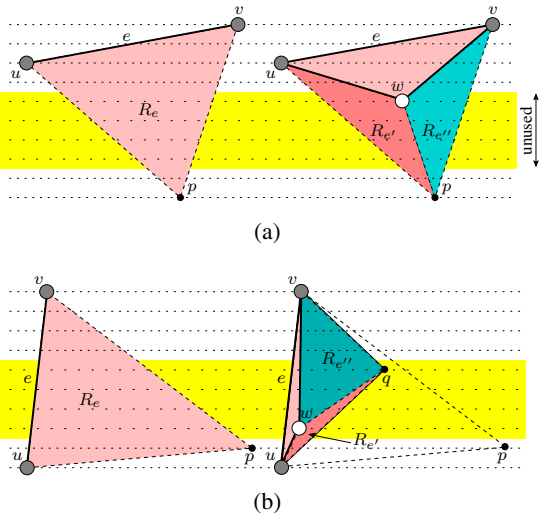


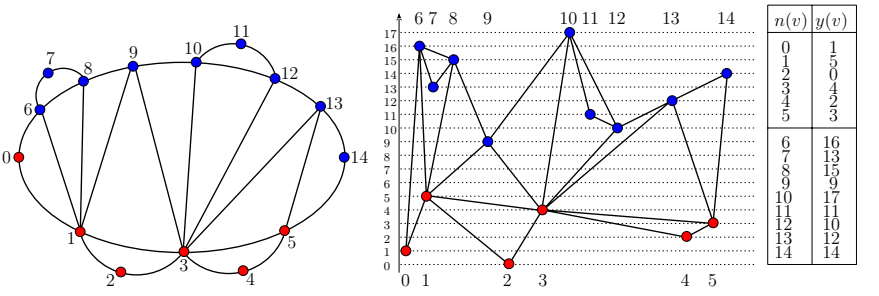
Fig. 2. Examples of regions R_e, $R_{e'}$ and $R_{e''}$. The unused y-coordinates are highlighted. (a) Vertices u, v have the same label T and w is also labeled with T. (b) Vertices u, v have labels B and T, respectively, and w is labeled with B.

vertex w to the drawing, a new face f of G is also drawn. Let $e = (u, v)$ be the edge of f that has been drawn at some previous step and let R_e be its safe region. Let u, v, p be the corners of R_e. Refer also to Fig. 2.

In order to draw w, we distinguish between two cases, depending on whether both end-vertices u and v have the same BT-label. If both u and v have the same BT-label, the construction is like the one depicted in Fig. 2(a). Vertex w is drawn inside R_e with the minimum or maximum unused y-coordinate based on its label (i.e. we choose the minimum if w is labeled B, else we choose the maximum). The safe region R_e of e does not change. The safe regions of edges $e' = (u, w)$ and $e'' = (w, v)$ are the triangular regions u, w, p and w, v, p, respectively. It is immediate to see that both invariants are maintained.

If u is labeled B and v is labeled T, the construction is like the one depicted in Fig. 2(b). Suppose that w is labeled B. Let q be a point inside R_e having the highest unused y-coordinate. Points u, v, and q define a triangle R'_e contained inside R_e such that R'_e contains all unused y-coordinates. Draw w inside R'_e at the lowest unused y-coordinate. We define the safe regions of edges (u, w) and (w, v) as the triangular regions u, w, q and w, v, q, respectively; the safe region of e stays the same. In this case, it is easy to see that both invariants are maintained. The case when w is labeled T is symmetric to the case when w is labeled B.

The planarity of the resulting drawing is a consequence of the two invariants and of the order by which the new vertices to be added are chosen. □



$n(v)$	$y(v)$
0	1
1	5
2	0
3	4
4	2
5	3
6	16
7	13
8	15
9	9
10	17
11	11
12	10
13	12
14	14

Fig. 3. Illustration of the construction in Lemma 2. (a) An outerplane outerpillar and the numbering of its vertices. (b) A planar straight-line drawing of the outerpillar that preserves the y-coordinate assignment shown in the table on the right.

Drawing Phase

Lemma 2. *Let G be an outerplane outerpillar and let Π_b and Π_t be a bottom and a top path of G, respectively. Assume that each vertex of G is given a distinct y-coordinate such that every vertex of Π_b has a y-coordinate that is smaller than the y-coordinate of any vertex of Π_t. G admits a planar straight-line drawing that preserves the given y-coordinates of its vertices.*

Sketch of Proof: We first prove the statement in the case that the weak dual of G is a path. We then show how the argument can be extended to the general case that the weak dual is a caterpillar with vertices of degree higher than two.

Let n_b be the size of Π_b. Visit Π_b from its source to its sink and denote any encountered vertex with an increasing integer, starting from 0. Similarly, visit Π_t from its source to its sink and denote any encountered vertex with an increasing integer, starting from n_b. Figure 3(a) shows an example of this numbering applied to the graph of Fig. 1(a). The y-coordinate assignment is shown on the right hand-side of Fig. 3(b). We compute a planar straight-line drawing of G that preserves the given y-coordinates in n_b steps.

At Step 0, vertex 0 is drawn at x-coordinate 0. Notice that the only vertex of Π_t adjacent to vertex 0 is vertex n_b (see Fig. 3(a)). Assign x-coordinate 1 to vertex n_b.

At Step i ($1 \leq i \leq n_b - 1$) the vertex of Π_b numbered i is drawn together with its adjacent vertices of Π_t that have not yet been drawn. Vertex i is the apex of a fan that contains one or more vertices of Π_t; denote as $\Pi_t(i)$ these vertices. Let k be the leftmost vertex of $\Pi_t(i)$. Observe that k is adjacent to some vertex that precedes vertex i along Π_b, and therefore k has been drawn at some previous step. Also, k has the largest x-coordinate among all vertices of the Π_t that have been drawn at steps preceding Step i. Let x_k be this x-coordinate.

Note that x_k is the largest x-coordinate of the current drawing. Vertex i is given any x-coordinate larger than x_k. Let $k+1$ be the next vertex of $\Pi_t(i)$ adjacent to vertex i. The x-coordinate of $k+1$ is any value larger than x_i. Assume that $\Pi_t(i)$ has at least another vertex $k+2$. We choose an x-coordinate x_{k+2} for $k+2$ that is "large enough" so that the edge $(i, k+2)$ does not cross any other edges drawn so far.

For example, the value x_{k+2} can be defined as follows. Let y^t_{max} and y^b_{max} (y^t_{min} and y^b_{min}) be the highest (lowest) y-coordinates of the top and bottom path, respectively. Consider the line ℓ through the two points (x_i, y^b_{max}) and (x_{k+1}, y^t_{min}). Let p be the point of ℓ having y-coordinate equal to y^t_{max}. The x-coordinate of $k+2$ is to the right of p. Any other vertices that follow $k+2$ and belong to $\Pi_t(i)$ are drawn with a similar technique.

What remains to consider is the case where the weak dual G^* of G has vertices of degree higher than two. We first remove all faces of G which correspond to the leaves of G^* and compute a drawing of the resulting graph as described above. We then re-insert the faces corresponding to the leaves of G^* by inserting a single vertex at a time. For a face with vertices u, w, v such that u and v have already been drawn proceed as follows. Note that u and v both belong to either Π_b or to Π_t and that they have consecutive x-coordinates. Assume first that u and v are in the top path and assume that x_u is left of x_v. Let e' be the rightmost edge incident to u and let e'' be the leftmost edge incident to v. Let p' be the point of e' having y-coordinate equal to y^t_{min}; let p'' be the point of e'' having y-coordinate equal to y^t_{min}. Choose for w an x coordinate in-between the x-coordinates of p' and p'' and such that u, v, and w are not collinear. The case where u and v both belong to Π_b is handled similarly. □

The coloring phase described in Sect. 3.1 together with Lemmas 1 and 2 implies the following.

Theorem 1. *Let $G_0 = (V, E_0)$ and $G_1 = (V, E_1)$ denote an outerplane and an outerpillar graph, respectively. The matched pair $\langle G_0, G_1 \rangle$ is matched drawable.*

3.2 Matched Drawing of ⟨Outerplane, Wheel⟩ and of ⟨Wheel, Wheel⟩

In this section, we describe two algorithms for computing a matched drawing of the matched pairs ⟨outerplane, wheel⟩ and ⟨wheel, wheel⟩, respectively. The algorithms follow the three main phases of the general approach previously introduced. For the pair ⟨outerplane, wheel⟩ these phases are as follows:

Coloring Phase. The BT-labeling of V depends only on the topology of the wheel: the center of the wheel is labeled as B, while all the remaining vertices are labeled as T.

Leveling Phase. A drawing Γ of the outerplane graph preserving the BT-labeling is computed as described in the proof of Lemma 1. The y-coordinates of the vertices of V are those of Γ.

Drawing Phase. A matched drawing of the pair ⟨outerplane, wheel⟩ is obtained by computing a planar straight-line drawing of the wheel such that the y-coordinates of the vertices are those defined in the leveling phase (see [9] for more details).

For the matched pair ⟨wheel, wheel⟩, the Coloring Phase and the Drawing Phase are the same as in the case ⟨outerplane, wheel⟩. The Leveling Phase is executed by computing a planar straight-line drawing of the first wheel such that the y-coordinates of the vertices define a color-preserving level assignment (see [9] for more details). As a consequence, we can state the following theorems; their proofs are omitted for space reasons; the interested reader can refer to [9].

Theorem 2. *Let $G_0 = (V, E_0)$ and $G_1 = (V, E_1)$ denote an outerplane and a wheel graph, respectively. The matched pair $\langle G_0, G_1 \rangle$ is matched drawable.*

Theorem 3. *Let $G_0 = (V, E_0)$ and $G_1 = (V, E_1)$ denote two wheels. The matched pair $\langle G_0, G_1 \rangle$ is matched drawable.*

4 Matched Drawings of Triples of Graphs

In this section, we extend the definition of matched drawing in order to allow the pairwise comparison of three n-vertex graphs. We observe that any number of universal level planar graphs have a matched drawing such that shared vertices have the same y-coordinate. It can be constructed by horizontally aligning the visualizations of the graphs. However, this type of visualization implies that the edges of the matching between two non-consecutively aligned drawings would overlap the in-between representations. Similarly to the classical edge-region crossing-free requirement of cluster planarity (see [10]) we add the constraint that the edges describing the matching of a pair of graphs never intersect the area occupied by the drawing of a third graph.

Let G_0, G_1, and G_2 be three graphs on the same vertex set V. The triple $\langle G_0, G_1, G_2 \rangle$ is a *matched triple*. Let L_0, L_1 and L_2 be three sets of n parallel lines with different slopes forming a set of triangles as illustrated in Fig. 4. For any $0 \leq i \leq 2$, the intersection points of all the pairs of lines in $L_i \times L_{(i-1) mod 3}$ form a set of n^2 elements, denoted as D_i, which is called the *diamond* associated with L_i and $L_{(i-1) mod 3}$. We assume that, for any $i \neq j$, $D_i \cap D_j = \emptyset$. A *triangular matched drawing* of $\langle G_0, G_1, G_2 \rangle$ is a triple $\langle \Gamma_0, \Gamma_1, \Gamma_2 \rangle$ of three straight-line planar drawings of G_0, G_1 and G_2, respectively, such that, for each vertex $v \in V$, the point representing v in Γ_i and the point representing v in $\Gamma_{(i+1) mod 3}$ lie on the same line of L_i ($0 \leq i \leq 2$). The vertices of the drawing Γ_i are a subset, of size n, of the diamond D_i ($0 \leq i \leq 2$). A matched triple $\langle \Gamma_0, \Gamma_1, \Gamma_2 \rangle$ is *triangular matched drawable*, or briefly *matched drawable*, if it admits a triangular matched drawing.

In [8], it was proved that there exist two trees that do not admit a geometric simultaneous embedding while in [4], it was proved that every matched pair of trees has a matched drawing. The next theorem contributes by shedding more light about the differences between the notion of matched drawability and simultaneous drawability. Namely, while it is known that triples of graphs with the same vertex set may not have a geometric simultaneous embedding even in the case that they are simple paths [4], it turns out that a matched triple consisting of a caterpillar and two *universal level planar graphs* is always triangular matched drawable.

A universal level planar graph G (also known as an *ULP graph* [6]) is a planar graph such that for any numbering of its vertices with n distinct values there exists a straight-line planar drawing of G such that every vertex has a y-coordinate equal to its given number. Trees that are ULP graphs are characterized in [5]; general universal planar graphs are characterized in [6].

4.1 Matched Drawings of a Caterpillar and Two Universal Level Planar Graphs

Let G_0, G_1, G_2 be a caterpillar and two ULP graphs, respectively. The general approach for computing a matched drawing $\langle \Gamma_0, \Gamma_1, \Gamma_2 \rangle$ is as follows. (1) The vertices of G_0 are

suitably assigned to a set L_0 of n parallel lines by ensuring that crossings between pairs of edges with no vertex in common are not possible (independently of the horizontal position of the vertices along the corresponding lines). Such an assignment implicitly defines a vertex numbering, i.e. the vertex assigned to line ℓ_j is numbered as j, ($0 \leq j \leq n-1$). (2) A planar straight-line drawing of the ULP-graph G_1 is computed by preserving the vertex numbering defined by L_0 [6]. Also, L_1 is defined as a set of n parallel lines with slope different from that of L_0 and such that each line of L_1 passes through a distinct vertex of G_1. (3) Similarly, a planar straight-line drawing of G_2 is computed by preserving the vertex numbering defined by L_1. The set L_2 is defined in such a way that each line passes only through a distinct vertex of G_2 and the lines of L_0, L_1 and L_2 form a set of parallel triangles as required in the definition of triangular matched drawing. Γ_0 is induced by the vertex numbering defined by L_2, which fixes the horizontal position of the vertices of G_0 along the lines of L_0.

The proof of the next theorem can be found in [9].

Theorem 4. *Let $\langle G_0, G_1, G_2 \rangle$ be a matched triple such that G_0 is a caterpillar, G_1 and G_2 are universal level planar graphs. The matched triple $\langle G_0, G_1, G_2 \rangle$ is triangular matched drawable.*

4.2 Matched Drawings of Three Cycles

The proof of Theorem 4 strongly relies on the acyclicity of graph G_0. The next theorem studies the case that the three graphs in the triple are all cycles.

Theorem 5. *A matched triple of cycles $\langle C_0, C_1, C_2 \rangle$ is triangular matched drawable.*

Sketch of Proof: Let (u, v) denote an edge that does not belong to all the three cycles. If such an edge does not exist, the three cycles are identical and the proof is trivial. Without loss of generality, suppose that $(u, v) \in C_0$ and $(u, v) \notin C_2$. Now, decompose each cycle as a path plus an edge connecting its end-vertices, where such an edge is referred to as the *broken edge*, as follows. Define Π_0 as the n-vertex path formed by the vertices encountered when traversing along C_0 from u to v. Moreover, define Π_1 and Π_2 as the paths obtained by removing an edge adjacent to vertex u from the cycle C_1 and C_2, respectively. As stated by Theorem 4, the three paths admit a triangular matched drawing, which can be easily computed as follows (see Fig. 4 for an illustration).

Let r_i be the direction orthogonal to the lines of L_i ($0 \leq i \leq 2$). For each $0 \leq i \leq 2$, travel along r_i and assign to the k-th encountered line of L_i the k-th vertex of the path Π_i. Now, place each vertex $z \in \Pi_i$ at a point of the diamond D_i by intersecting the pairs of lines in $L_i \times L_{(i-1) mod 3}$ that have been associated to z ($0 \leq i \leq 2$). By construction, each path is represented by a monotone chain, and thus the drawings Γ_0, Γ_1 and Γ_2 are planar. Note that, if w denotes the last vertex of the path Π_2, by construction $w \neq v$ (i.e. paths Π_0 and Π_2 have two distinct end-vertices). Thus, the last vertex v of the path Π_0 does not lie on the corner of the diamond D_0 (this corner is formed by the intersection of the lines of L_0 and L_2 corresponding to vertices v and w, respectively). At this point, the resulting drawings may not be planar because of the broken edges. However, the following line shift operations remove possible crossings introduced by

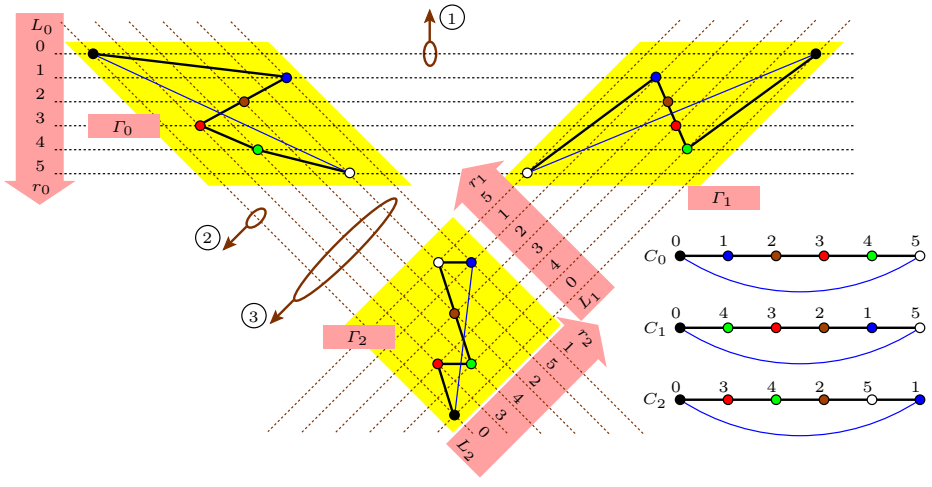


Fig. 4. Triangular matched drawing of three cycles

the broken edges while preserving the planarity of each path. More precisely, suppose that all three cycles are not planar, then proceed as follows:

1. Move the first line of L_0 in the direction opposite to r_0 until the broken edge of C_1 does not cross. Clearly, this operation does not affect path Π_2, but it lengthens one of the edges of Π_0.
2. Move the first line of L_2 in the direction opposite to r_2 until the cycle C_0 becomes planar. This operation does not affect cycle C_1, but it lengthens one of the edges of Π_2.
3. Move the first $n-1$ lines of L_2 in the direction opposite to r_2 until the cycle C_2 becomes planar. Note that, this motion does not affect cycle C_1. It modifies cycle C_0, but since there is no vertex on the corner of the diamond L_0, cycle C_0 remains planar. □

5 Conclusions and Open Problems

In this paper we have addressed the matched drawability problem for well-known families of planar graphs. In particular, we have extended to triples of graphs the concept of matched drawing, introduced in [4], and we have studied the drawability problem for pairs and triples of graphs. We have introduced a general approach for computing a matched drawing of a pair of graphs, and provided three algorithms based on this approach for drawing the pairs ⟨outerplane, outerpillar⟩, ⟨outerplane, wheel⟩ and ⟨wheel, wheel⟩. Finally, we have presented two algorithms for computing a matched drawing of a caterpillar and two ULP-graphs and of three cycles. We conclude with several interesting open problems for future research:

1. Is a matched pair of biconnected outerplane graphs matched drawable?
2. Is the matched triple formed by a tree and two ULP-graphs matched drawable?
3. Is every triple of outerpillar graphs matched drawable?

Acknowledgments

The research in this paper started during the *Bertinoro Workshop on Graph Drawing, BWGD 2008*. The authors gratefully acknowledge the other participants for the many interesting conversations about matched drawings of planar graphs.

References

1. Brass, P., Cenek, E., Duncan, C.A., Efrat, A., Erten, C., Ismailescu, D.P., Kobourov, S.G., Lubiw, A., Mitchell, J.S.B.: On simultaneous planar graph embeddings. Comput. Geom. Theory Appl. 36(2), 117–130 (2007)
2. Collins, C., Carpendale, S.: Vislink: Revealing relationships amongst visualizations. IEEE Trans. on Visualization and Comput. Graph. 13(6), 1192–1199 (2007)
3. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing. Prentice Hall, Upper Saddle River (1999)
4. Di Giacomo, E., Didimo, W., van Kreveld, M., Liotta, G., Speckmann, B.: Matched drawings of planar graphs. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 183–194. Springer, Heidelberg (2008)
5. Fowler, J.J., Kobourov, S.G.: Characterization of unlabeled level planar trees. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 367–379. Springer, Heidelberg (2007)
6. Fowler, J.J., Kobourov, S.G.: Characterization of unlabeled level planar graphs. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 37–49. Springer, Heidelberg (2008)
7. Frati, F., Kaufmann, M., Kobourov, S.G.: Constrained simultaneous and near-simultaneous embeddings. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 268–279. Springer, Heidelberg (2008)
8. Geyer, M., Kaufmann, M., Vrt'o, I.: Two trees which are self-intersecting when drawn simultaneously. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 201–210. Springer, Heidelberg (2006)
9. Grilli, L., Hong, S.H., Liotta, G., Meijer, H., Wismath, S.K.: Matched drawability of graph pairs and of graph triples. Tech. Rep. RT-004-08, Dip. Ing. Elettr. e dell'Informaz., Univ. Perugia (2008)
10. Kaufmann, M., Wagner, D. (eds.): Drawing graphs: methods and models. Springer, Heidelberg (2001)
11. Nishizeki, T., Rahman, M.S.: Planar Graph Drawing. World Scientific, Singapore (2004)
12. Sugiyama, K.: Graph Drawing and Applications for Software and Knowledge Engineers. World Scientific, Singapore (2002)

An Improved Upward Planarity Testing Algorithm and Related Applications

Sarmad Abbasi[1], Patrick Healy[2], and Aimal Rextin[2]

[1] National University of Computer and Emerging Sciences, Lahore, Pakistan
sarmad.abbasi@nu.edu.pk
[2] Computer Science Department, University of Limerick, Ireland
patrick.healy@ul.ie,
aimal.tariq@ul.ie

Abstract. We consider the standard algorithm to test the upward planarity of embedded digraphs by Bertolazzi *et al.* [3]. We show how to improve its running time from $O(n + r^2)$ to $O(n + r^{\frac{3}{2}})$, where r is the number of sources and sinks in the digraph. We also discuss 2 applications of this technique: finding a certificate of correctness of an implementation of our upward planarity testing algorithm; and improving the running time of getting a quasi-upward planar drawing for an embedded digraph with minimum number of bends.

1 Introduction

A digraph $G = (V, E)$ is upward planar if it has a planar drawing with all edges pointing upward [6]. An upward planarity testing algorithm checks if a planar digraph G has a drawing that is simultaneously upward and planar. Although an upward planar digraph must be both upward and planar, these two properties alone do not guarantee upward planarity. In fact, Garg and Tamassia [7] showed that upward planarity testing is NP-complete for general digraphs, so efficient upward planarity testing algorithms have been developed for some restricted versions of the problem. Hutton and Lubiw [9] presented an $O(n^2)$ upward planarity testing algorithm for single-source digraphs, this algorithm was later optimized to $O(n)$ by Bertolazzi *et al.* [4]. An $O(n + r^2)$ time algorithm was developed by Bertolazzi et al. to test the upward planarity of digraph with a fixed embedding [3], where r is the number of sources and sinks in the digraph and n is the number of vertices of digraph. An *st-digraph* G is a directed acyclic graph (DAG) with a single source s, a single sink t and an edge (s, t). Di Battista *et al.* and Kelly independently showed that a digraph is upward planar if and only if it is a spanning subgraph of a planar st-digraph [1,10].

In this paper, we present an improved algorithm for testing the upward planarity of an embedded digraph. The algorithm by Bertolazzi *et al.* [3] works by solving a c-matching problem. The flow network for this problem has a particular structure which allows for the maximum flow to be found in $O(r^{\frac{3}{2}})$ time [11]. However, it takes another $O(r^2)$ time because we also need to check for a

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 334–344, 2009.

feasible external face [3]. We reduce the running time of the algorithm by giving an efficient way to find a feasible external face. Since checking the upward planarity of an embedded digraph is also used as a subroutine in other algorithms, for example [8,5], improving the time taken by this algorithm may also make other algorithms efficient.

A program is *certifying* if, along with its output, the program gives a certificate that the output is correct. A *certificate* is used by human users to ensure that the output is correct for a particular input. For example, the planarity testing algorithm of the class library `LEDA` implements this idea [12]. It presents a planar drawing of a graph G when G is determined to be planar and when G is determined to be non-planar it shows a subdivision of K_5 or $K_{3,3}$ inside G. Hence users can be certain that `LEDA` computed the correct answer for that instance. We argue that an upward planarity testing algorithm should also present a certificate of correctness for the computed output. We present one possible way that such certificate of correctness can be presented to a user.

In a *quasi-upward drawing* Γ of a digraph, a horizontal line through each vertex in Γ locally splits the incoming edges from the outgoing edges [2]. In a quasi-upward drawing, a *bend* is a point on an edge e where the tangent of e is horizontal. Quasi-upward planarity is a generalization of upward planarity; and all digraphs with a bimodal planar embedding have a quasi-upward planar drawing. Let G_ϕ be an embedded digraph with a fixed external face. A quasi-upward planar drawing of G_ϕ with minimum bends can be computed by finding the minimum cost flow in a network $\mathcal{N}$ [2]. We also present how the practical running time of computing a quasi-upward planar drawing with minimum number of bends can be improved.

1.1 Preliminaries

We assume basic familiarity with graph theory. Let $B = (V, W, E)$ be a bipartite graph. Let $S \subseteq W$, we let $N(S)$ denote the *neighborhood* of S, i.e $N(S) = \{v \in V : (v, w) \in E \text{ and } w \in S\}$. Let $c : W \to \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers. For each vertex $w \in W$, we call $c(w)$ the *capacity* of w. The *capacitated matching* or *c-matching* M of B with respect to c is a set $M \subseteq E$ and has the following properties: (a) a vertex $v \in V$ is incident to at most 1 edge in M; and (b) vertex $w \in W$ is incident to at most $c(w)$ edges in M. We also define a function $I_M : V \cup W \to \mathbb{N}$, such that for a $v \in V \cup W$, $I_M(v)$ equals the number of edges incident to v in M. A vertex $v \in V$ is called *matched* if $I_M(v) \neq 0$, otherwise it is called *free*. A vertex $w \in W$ is called *saturated* if $I_M(w) = c(w)$, and it is called *unsaturated* if $0 < I_M(w) < c(w)$. A c-matching M is called a *perfect* capacitated matching if

$$|M| = \sum_{w \in W} c(w).$$

A path $P = w_0, v_1, w_1 \ldots, w_k, v_{k+1} = v$ is called an *augmenting path* for a c-matching M if $(w_i, v_{i+1}) \in E \setminus M$ and $(v_i, w_i) \in M$, such that $I_M(v) = 0$ and $I_M(w_0) < c(w)$. We allow a non-free vertex $w \in W$ to repeat in a augmenting

path. Given M and an augmenting path P, we can obtain a c-matching $M' = M \oplus P$, such that $(v_i, w_{i-1}) \in M'$ for all v_i appearing on P and $(v_i, w_i) \in M'$ for all v that do not appear on P. Note that $|M'| = |M| + 1$.

A perfect c-matching can computed by changing $B = (V, W, E)$ to a flow network N and finding the maximum flow in it. We do this by first orienting every edge e of B from V to W such that each edge has a capacity $u(e) = 1$, adding a source s and a sink t, adding an edge $e' = (s, v)$ between s and every vertex $v \in V$ with capacity $u(e') = 1$, and finally an edge $e'' = (w, t)$ from every $w \in W$ to t with capacity $u(e'') = c(w)$. If the maximum flow has a value equal to $\sum_{w \in W} c(w)$ then B has a perfect c-matching with respect to c. We can obtain this perfect c-matching by taking all edges between V and W with non zero flow in N. The following theorem is from [11].

Theorem 1. *Let $N = (V, E)$ be network. The maximum flow in N can be found in $O(\sqrt{|V|} \cdot |E|)$ time if either out-degree$(v) \leq 1$ or in-degree$(v) \leq 1$ for every vertex of N.*

Corollary 1. *In a bipartite graph $B = (V, E)$, one can find a c-matching with respect to a function c in $O(\sqrt{|V|} \cdot |E|)$ time.*

In a digraph, a *source* vertex has only outgoing edges and a *sink* vertex has only incoming edges. We denote the set of sources and sinks in a digraph G by S and T respectively, we also let $K = S \cup T$. An *embedding* is an equivalence class of planar drawings for a graph G, such that an embedding ϕ has a fixed circular order of edges around each vertex of G. A graph G with a given embedding is denoted by G_ϕ and we call it an *embedded* digraph. Two drawings with the same embedding have the same set of faces F. We can draw G_ϕ in a planar fashion with any face in F as the external face. An embedded digraph G_ϕ is *bimodal* if the edges incident to each vertex in G_ϕ can be partitioned into consecutive incoming and outgoing edges. We can check if a planar digraph G_ϕ is bimodal in $O(n)$ time. Assume that the vertex v has the edges e_1 and e_2 incident to it in a face f of an embedded digraph G_ϕ, such that e_1 and e_2 are consecutive edges in the facial boundary of f. The triplet $\langle e_1, v, e_2 \rangle$ is a *switch* if both e_1 and e_2 either point toward v or away from v. The number of switches in a face f in an embedded DAG is even, and is denoted by $2n_f$.

Lemma 1 (Bertolazzi *et al.* [3]). *Let F be the set of faces in a bimodal embedded digraph G_ϕ. Then*

$$\sum_{f \in F} (n_f - 1) + 2 = |S| + |T|.$$

A mapping of sources and sinks to the faces of G_ϕ is a *consistent assignment* if the following conditions are satisfied:

- Each source or sink is mapped to exactly one face.
- A face h has $n_h + 1$ sources or sinks mapped to it.
- Every other face f has $n_f - 1$ sources or sinks mapped to it.

The embedded digraph G_ϕ cannot be upward planar if it is non-planar or non-bimodal or has directed cycles. Bimodality, planarity and acyclicity can be tested in linear time. Hence, the $O(n + r^2)$-time embedded upward planarity testing algorithm by Bertolazzi *et al.* [3] first ensures that G_ϕ satisfies these properties and then checks if it satisfies the following theorem:

Theorem 2 (Bertolazzi et al. [3]). *Let G_ϕ be a planar and bimodal embedded DAG. G_ϕ is upward planar if and only if we can find a consistent assignment of the sources or sinks of G_ϕ to its faces.*

Bertolazzi *et al.* showed that there exists a consistent assignment for G_ϕ if a bipartite graph B_{G_ϕ} has a perfect c-matching. The graph $B_{G_\phi} = (K, F, E)$ has an edge (v, f) if $v \in K$ is incident to $f \in F$ in G_ϕ. Bertolazzi *et al.* show that $|F|$ is $O(r)$, where r is the number of sources and sinks in G. The graph B_{G_ϕ} is planar and hence it has $O(r)$ edges. The graph B_{G_ϕ} can be constructed in $O(n)$ time. Given B_{G_ϕ}, one can check if a consistent assignment exists for G_ϕ by using the following 2 step procedure:

1. Define function $c : F \to \mathbb{N}$, such that $c(f) = n_f - 1$ for every face f in G_ϕ. Then find the c-matching in B_{G_ϕ} with respect to c. If perfect c-matching does not exist then G_ϕ is not upward planar, otherwise each face f can act as an internal face in an upward planar drawing of G_ϕ.
2. If the previous test is successful, then we try to find a feasible external face, if it exists. We let $c(h) = n_h + 1$ for a face h and check if we can have a perfect c-matching. If we get a perfect c-matching then G_ϕ is upward planar, otherwise we revert back to $c(h) = n_h - 1$ and let $c(g) = n_g + 1$ for a face g, we then find the maximum c-matching. We repeat this until we find a perfect c-matching or we run out of faces, in which case G_ϕ is not upward planar.

Bertolazzi *et al.* proposed an $O(r^2)$ method for step 1 [3], by converting it from a matching problem to a maximum flow problem on a network N' as discussed earlier. However, we conclude from Corollary 1 that step 1 can be done in $O(r^{\frac{3}{2}})$ time because B_{G_ϕ} is planar. Bertolazzi *et al.* proposed an $O(r^2)$ method for step 2 [3], which essentially corresponds to iterativly checking each face if it can act as the external face. We show in the next section how it can be done in linear time.

A quasi-upward planar drawing with minimum bends for an embedded digraph G_ϕ with a fixed external face can be computed by finding the minimum cost flow in a network $\mathcal{N}$ [2]. So we will breifly describe min cost flows. Assume we have a network $N = (V, E)$, such that each edge e has an associated value $c(e)$ called *cost* and $u(e)$ called *capacity*. We also have a integer $d(v)$ with each vertex v called *demand*, if $d(v) < 0$ then v is a *supply-node*. The *minimum cost flow* problem computes a function $x : E \to \mathbb{N}$, called *flow*, that minimizes

$$\sum_{e \in E} c(e)x(e).$$

We let $Val(x)$ represent the sum of flow "supplied" by the the supply-nodes.

Let $R_{(\mathcal{N},x)}$ be the *residual graph* for the network $\mathcal{N}$ with respect to the flow x. The residual graph $R_{(\mathcal{N},x)}$ has the same vertices as $\mathcal{N}$, for each edge $e = (u, v)$ in $\mathcal{N}$, $R_{(\mathcal{N},x)}$ has an edge $e' = (u, v)$ and an edge $e'' = (v, u)$. The edge e' has a cost $c(e') = c(e)$ and an upper bound on flow $u(e') = u(e) - x(e)$, while the edge e'' has a cost $c(e'') = -c(e)$ and an upper bound on flow $u(e'') = x(e)$. The cost of a path P in $R_{(\mathcal{N},x)}$ is defined as

$$cost(P) = \sum_{\forall e \in P} c(e),$$

while the capacity of a path P is the minimum $u(e)$ for all edges e on P.

Theorem 3. *Let x be a flow of minimum cost of value $Val(x)$ in $\mathcal{N}$ and let P be a the minimum cost path in $R_{(\mathcal{N},x)}$. Let x' be a flow in $R_{(\mathcal{N},x)}$ with non-zero flow only along P. Then the flow x'' is the minimum cost flow of value $Val(x) + Val(x')$, such that*

$$x''(e) = x(e) + x'(e) \quad \textit{for all } e \in E$$

Theorem 3 leads to *Successive Path Augmentation* algorithm to find the minimum cost flow [11].

2 An Improved Upward Planarity Testing Algorithm

Let F denote the set of faces of G_ϕ and $K = S \cup T$ denote the sources and sinks in G_ϕ. Assume we are given a bipartite graph $B_{G_\phi} = (K, F, E_B)$ such that B_{G_ϕ} has an edge (s, f) if and only if the source or sink s is incident on the face f in G_ϕ. Furthermore, we are given a function $c : F \to \mathbb{N}$, such that $c(f) = n_f - 1$ for each $f \in F$. We note that B_{G_ϕ} is planar and hence has $O(n)$ edges [3]. If B_{G_ϕ} has no perfect c-matching then G_ϕ does not have a consistent assignment. Similarly, if B_{G_ϕ} has a perfect c-matching, then each face f in G_ϕ can be assigned $n_f - 1$ sources or sinks. We can find if B_{G_ϕ} has a perfect c-matching with respect to the function c in $O(r^{\frac{3}{2}})$ time according to Corollary 1.

Assume that M is a perfect c-matching in B_{G_ϕ}. The embedded digraph G_ϕ is upward planar if we can show that there exists a face h, such that h can be assigned $n_h + 1$ sources or sinks. From Lemma 1 there will be two unmatched sources/sinks, say, s and s'. Suppose a face f is *reachable* from s. That is, there exists an alternating path P given by

$$s = s_0, f_1, \ldots, s_{k-1}, f_k = f$$

satisfying $(s_{i-1}, f_i) \in E_B \setminus M$ and $(s_{i+1}, f_i) \in M$. Similarly, suppose f is also reachable from s' via another alternating path P' given by

$$s' = s'_0, f'_1, \ldots, s'_{l-1}, f'_l = f$$

satisfying $(s'_{i-1}, f'_i) \in E_B \setminus M$ and $(s'_{i+1}, f'_i) \in M$. We claim that the first common vertex between P and P' is a face. Suppose on the contrary that the first common vertex between P and P' is a source or sink $\hat{s}$. Let $\hat{f}$ be the preceding vertex of $\hat{s}$ in P, and $\hat{f}'$ be the preceding vertex of $\hat{s}$ on P'. Then $\hat{f} \neq \hat{f}'$. But this implies that both $\hat{f}$ and $\hat{f}'$ are matched to the same source/sink $\hat{s}$, contradicting the fact that M matches a source/sink to only one face.

A simple modified version of BFS or DFS can search for all faces reachable from s and s' in linear time. Let $\hat{P}$ be the alternating sub-path of P from s to $\hat{f}$ and $\hat{P}'$ be the alternating path from s' to $\hat{f}$. We let $c(\hat{f}) = n_{\hat{f}} + 1$ and apply these paths to obtain a capacitated matching, such that $I_M(\hat{f}) = n_{\hat{f}} + 1$ and $I_M(f) = n_f - 1$ for all $f \neq \hat{f}$. We obtain a consistent assignment for G_ϕ by assigning a source/sink v to a face f in G_ϕ if the edge $(v, f) \in M$.

It only remains to be shown that if the set of faces reachable from both s and s' are disjoint then there is no consistent assignment. In this case, let R_1 and R_2 be the set of faces reachable via alternating paths from s and s' respectively. Then $R_1 \cap R_2 = \emptyset$. We show that

$$|N(R_1)| > \sum_{f \in R_1} n_f - 1$$

where $N(R_1)$ is the neighborhood of R_1 in B_{G_ϕ}. Each face $f \in R_1$ must have been assigned to $n_f - 1$ different sources/sinks. This accounts for $\sum_{f \in R_1} n_f - 1$ sources/sinks in the neighborhood of R_1. Since s is also included in this neighborhood its total size is at least $\left(\sum_{f \in R_1} n_f - 1\right) + 1$. This implies that in a consistent assignment the external face must belong to R_1. A similar argument shows that in a consistent assignment

$$|N(R_2)| > \sum_{f \in R_2} n_f - 1$$

and the external face must belong to R_2. But this is impossible as $R_1 \cap R_2 = \emptyset$. Hence we have the following theorem.

Theorem 4. *If B_{G_ϕ} has a perfect c-matching then we can check if G_ϕ has a consistent assignment in linear time.*

The improved upward planarity testing algorithm is shown in Algorithm 1. We can construct B_{G_ϕ} in $O(n)$ time. We can check in $O(r^2)$ time if B_{G_ϕ} has a perfect c-matching with respect to a function c, such that $c(f) = n_f - 1$ for each face $f \in G_\phi$. Moreover, checking for a feasible external face takes linear time. Hence, the time taken by our algorithm is $O(n + r^{\frac{3}{2}})$.

3 Application: A Certificate of Non-upward Planarity

We can compute a certificate that justifies the decision made by an upward planarity testing algorithm for embedded digraphs by looking at it as a matching

Algorithm 1. `Improved Upward Planarity Checking`

Input: A planar, acyclic and bimodal embedded digraph G_ϕ
Output: True if G_ϕ is upward planar; False otherwise
Construct the bipartite graph $B_{G_\phi} = (K, F, E_B)$
For each $f \in F$ set $c(f) = n_f - 1$.
Find the maximum c-matching M in B_{G_ϕ}
if $|M| < |S| + |T| - 2$ **then**
    Return False
end if
u' and v' are the two vertices in B_{G_ϕ} that are not saturated
Run BFS from u' coloring each face as red.
RUN BFS from v'
if reached a red face vertex **then**
    Return True
else
    Return False
end if

problem. This certificate can be used to ensure that an implementation of the algorithm computes the correct answer for a given input. An upward planar drawing of the embedded digraph G_ϕ is the certificate when G_ϕ is upward planar. When G_ϕ is cyclic then we can highlight a directed cycle in a planar drawing. Similarly, we could zoom in on a non-bimodal vertex in a planar drawing to show that G_ϕ is not bimodal. However, it is not immediately clear what proof to use when no consistent assignment of G_ϕ is found.

Let us consider the bipartite graph B_{G_ϕ} defined in Section 2. The following simple corollary of Hall's theorem provides us with a proof that B_{G_ϕ} does not have a perfect c-matching.

Theorem 5 (Hall). *A bipartite graph $G = (V, W, E)$ has a matching that saturates all vertices in W if and only if $|N(S)| \geq |S|$ for all $S \subseteq W$.*

Corollary 2. *If B_{G_ϕ} does not have a perfect c-matching then there exists $F' \subset F$ such that*

$$|N(F')| < \sum_{f \in F'} (n_f - 1)$$

The subset F' of faces serves as proof that B_{G_ϕ} does not have a perfect matching. In terms of G_ϕ, this implies that there is a subset of faces F' such that the number of sources or sinks incident to F' are strictly less than the capacities of F'. In a planar drawing of G_ϕ, we can highlight the switches in all the faces in F' and also the sources and sinks appearing in these faces. Such a drawing can be examined visually for moderate sized graphs by a user to verify that G_ϕ is indeed non-upward planar; for larger graphs, one might use a simple linear-time verification algorithm.

Let M be a maximum c-matching in B_{G_ϕ} that is not perfect. It is obvious that B_{G_ϕ} will have at least one face f that is not saturated. We find all vertices

of B_{G_ϕ} that are reachable from f with respect to M, that is, all those vertices that have an alternating path from f. Since M is maximum, none of the possible alternating paths will be an augmenting path. Let F' be faces that are reached by an alternating path from f. Note that every vertex in $N(F')$ will be reachable from f because of the way B_{G_ϕ} is constructed. Since f is not saturated and is part of F', we conclude that $|N(F')| < \sum_{f \in F'}(n_f - 1)$.

Even if B_{G_ϕ} does have a perfect matching the assignment problem can still be unsolvable if no face can serve as an external face in G_ϕ by accepting the remaining two sources/sinks. One could give $O(n)$ certificates by letting $c(f) = n_f + 2$ for each of the $O(n)$ faces. However, we can present a single proof by showing that there is an *external face conflict* as shown in Section 2. An embedded digraph G_ϕ has an external face conflict if there are two disjoint sets of faces, $R_1 \subseteq F$ and $R_2 \subseteq F$, with $R_1 \cap R_2 = \emptyset$ such that

$$|N(R_1)| > \sum_{f \in R_1} n_f - 1 \text{ and } |N(R_2)| > \sum_{f \in R_2} n_f - 1.$$

The first condition implies that the external face must belong to R_1 and the second condition implies that the external face must belong to R_2 thereby proving that G_ϕ is not an upward planar graph. These faces can be highlighted for the user in different colors. We can find R_1 (and R_2 analogously) as follows.

Let M be a perfect c-matching for B_{G_ϕ} with respect to c. Let s_1 and s_2 be the two unassigned sources or sinks. We find the set of all vertices reachable from s_1 via alternating paths from s_1. Let R_1 be the faces that are reachable via alternating paths from s_1. Each face $f \in R_1$ will have been assigned $n_f - 1$ different sources/sinks. This accounts for $\sum_{f \in R_1} n_f - 1$ sources/sinks in the neighborhood of R_1. Since s is also included in this neighborhood its total size is at least $\left(\sum_{f \in R_1} n_f - 1\right) + 1$. Hence, $|N(R_1)| > \sum_{f \in R_1} n_f - 1$.

Since we can find a certificate that a given embedded digraph is non-upward planar by running a modified BFS on B_{G_ϕ} and because B_{G_ϕ} has $O(n)$ edges we have the following theorem.

Theorem 6. *We can find a certificate that the embedded upward planarity testing algorithm computes the correct answer in $O(n)$ time.*

4 Application: Improving Quasi-upward Planarity Testing

We now show how to improve the practical running time of computing a quasi-upward planar drawing with minimum bends for an embedded digraph. Bertolazzi *et al.* showed that this can be done by solving a minimum cost network $\mathcal{N}$ [2]. The flow network $\mathcal{N}$ is constructed in the following manner. The vertices of $\mathcal{N}$ consists of the sources/ sinks of G_ϕ and the faces of G_ϕ. Let f be a face in G_ϕ then $d(f) = n_f + 1$ if f is the external face and $d(f) = n_f - 1$ otherwise. Note that if f is a directed cycle and is an internal face then $d(f) = -1$, i.e it is a

supply-node. Each source or sink v has a demand of $d(v) = -1$. There is an edge $e = (v, f)$ in $\mathcal{N}$ with $u(e) = 1$ and $c(e) = 0$ if the source or sink v is incident to the face f. There is an edge e in $\mathcal{N}$ with $u(e) = \infty$ and $c(e) = 2$ between every pair of adjacent faces.

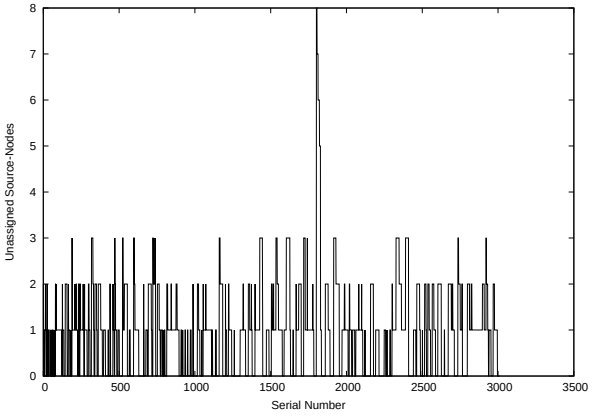
We now use c-matching to reduce the number of iterations needed to find a minimum cost flow in $\mathcal{N}$. Let $B_{G_\phi} = (K, F, E)$ be a bipartite graph, such that K is the set of sources and sinks in G_ϕ while F is the set of faces in G_ϕ. Assume that we also have a function $cap : F \to \mathbb{N}$ such that $cap(f) = n_f + 1$ when f is the external face, $cap(f) = 0$ when f is an internal face with 0 switches, and $cap(f) = n_f - 1$ when f is an internal face with positive number of switches. We find the maximum c-matching M in B_{G_ϕ} with respect to the function *cap*. If $|M| = |S| + |T|$ and none of the facial boundaries of G_ϕ is a directed cycle, then G_ϕ is upward planar, which means that G_ϕ has a quasi-upward planar drawing with no bends. A quasi-upward planar drawing of G_ϕ has positive number of bends in the following 2 cases: when none of the facial boundaries of G_ϕ are directed cycles and $|M| < |S| + |T|$, or when at least one facial boundary of G_ϕ is a directed cycle. We can map M to a flow x in $\mathcal{N}$ in the following manner: if for a source or sink v and a face f the edge $(v, f) \in M$, then we add a unit flow on the edge (v, f) in $\mathcal{N}$. The flow x will have a value of $|M|$, and the total cost of x is 0 because $x(e)$ is positive only for edges with 0 cost. We argue that this will be the minimum cost flow of value $|M|$ in $\mathcal{N}$.

Lemma 2. *If x is the flow obtained from running c-matching on B_{G_ϕ} with respect to the function cap, then f is the minimum cost flow of value $Val(x) = |M|$.*

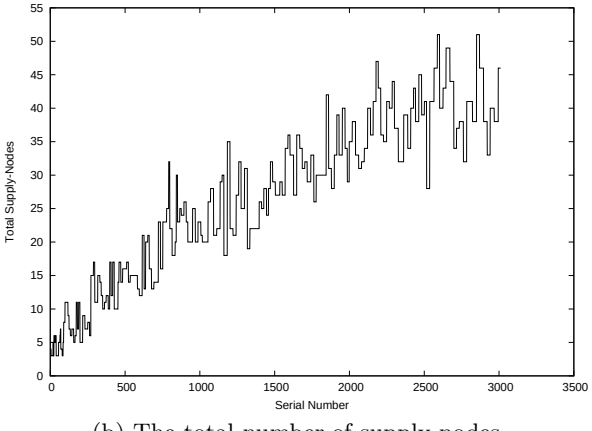
Proof. Assume we have a flow x' in $\mathcal{N}$ such that $Val(x') = Val(x)$ and $c(x) > c(x')$. This means that $c(x') \leq -1$ because $c(x) = 0$, but there are no negative cost edges in $\mathcal{N}$. Hence we have a contradiction. □

Let U be those supply-nodes with 0 *out-flow* with respect to the flow x. We experimentally check the value of $|U|$ by finding the maximum c-matching for B_{G_ϕ} for 3012 randomly generated planar bimodal connected digraphs. For each digraph G in our test-suite, we first find a bimodal embedding G_ϕ and then find the maximum c-matching in B_{G_ϕ} with respect to the function *cap*. We find that mostly $|U|$ is below 2 and very rarely goes above 3. The detailed result of our experiment is given in Figure 1. Hence, it is reasonable to believe that in practical situations $|U|$ will be a small number.

We then modify the network $\mathcal{N}$ so that it has a single supply-node $\hat{s}$ and a single demand-node $\hat{t}$. The modified network remains to be $O(n)$ in size, and the modification will take $O(n)$ time. We can find the minimum cost flow in $\mathcal{N}$ by repeatedly taking the cheapest augmenting path from $\hat{s}$ to $\hat{t}$ until the maximum flow value is attained [11]. We usually find the cheapest augmenting path in $O(n \log n)$ time by using Dijkstra's algorithm. However, we can do it in $O(n)$ time by modifying $\mathcal{N}$ to a new network $\mathcal{N}'$. $\mathcal{N}'$ is constructed by replacing every edge (h, g), where h and g are faces in G_ϕ, by a dummy vertex v and the edges $e_1 = (h, v)$ and $e_2 = (v, g)$. Moreover, we let $u(e_1) = u(e_2) = u(e)$ and



(a) The number of supply-nodes whose out-flow is 0



(b) The total number of supply-nodes

Fig. 1. Results

$c(e_1) = c(e_2) = 1$. We can now find the cheapest augmenting path in $O(n)$ time by using a modified BFS algorithm.

It takes $O(n)$ time to construct the flow network $\mathcal{N}$. Hence, we can find the minimum bend drawing for G_ϕ in $O(n+r^{\frac{3}{2}}+|U|n)$. We have seen experimentally that $|U| \approx O(1)$, so practically one can expect the running time be be close to $O(n+r^{\frac{3}{2}})$. Recall that the algorithm described by Bertolazzi *et al.* has a running time of $O(n^2 \log n)$ [2].

5 Conclusion

We showed that testing if an embedded digraph is upward planar can be done in $O(n + r^{\frac{3}{2}})$ time by presenting an efficient way to find a feasible external face. We also showed how to compute a certificate of correctness for an embedded

upward planarity testing algorithm. Moreover, we showed that we can improve the practical running time of computing a quasi-upward planar drawing with minimum number of bends by using our improved upward planarity testing algorithm.

References

1. Di Battista, G., Tamassia, R.: Algorithms for plane representations of acyclic digraphs. Theor. Comput. Sci. 61, 175–198 (1988)
2. Bertolazzi, P., Di Battista, G., Didimo, W.: Quasi-upward planarity. Algorithmica 32(3), 474–506 (2002)
3. Bertolazzi, P., Di Battista, G., Liotta, G., Mannino, C.: Upward drawings of triconnected digraphs. Algorithmica 12(6), 476–497 (1994)
4. Bertolazzi, P., Di Battista, G., Mannino, C., Tamassia, R.: Optimal upward planarity testing of single-source digraphs. SIAM J. Comput. 27(1), 132–169 (1998)
5. Chan, H.: A parameterized algorithm for upward planarity testing. In: Proceedings of ESA, pp. 157–168 (2004)
6. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall, Englewood Cliffs (1999)
7. Garg, A., Tamassia, R.: On the computational complexity of upward and rectilinear planarity testing. SIAM J. Comput. 31(2), 601–625 (2001)
8. Healy, P., Lynch, K.: Fixed-parameter tractable algorithms for testing upward planarity. In: Proceedings of SOFSEM, pp. 199–208 (2005)
9. Hutton, M.D., Lubiw, A.: Upward planar drawing of single-source acyclic digraphs. SIAM J. Comput. 25(2), 291–311 (1996)
10. Kelly, D.: Fundamentals of planar ordered sets. Discrete Math 63(2-3), 197–216 (1987)
11. Mehlhorn, K.: Graph algorithms and NP-completeness. Springer, New York (1984)
12. Mehlhorn, K., Näher, S.: LEDA: a platform for combinatorial and geometric computing. Cambridge University Press, New York (1999)

Spherical-Rectangular Drawings

Mahdieh Hasheminezhad[1], S. Mehdi Hashemi[1], and Brendan D. McKay[2]

[1] Department of Computer Science, Faculty of Mathematics and Computer Science
Amirkabir University of Technology, Tehran, Iran
m.hashemi@aut.ac.ir, hashemi.aut.ac.ir
[2] Department of Computer Science, Australian National University, Canberra,
ACT 0200, Australia
bdm@cs.anu.edu.au

Abstract. We extend the concept of rectangular drawing to drawings on a sphere using meridians and circles of latitude such that each face is bounded by at most two circles and at most two meridians. This is called spherical-rectangular drawing. Special cases include drawing on a cylinder, a cone, or a lattice of concentric circles on the plane. In this paper, we prove necessary and sufficient conditions for cubic planar graphs to have spherical-rectangular drawings, and show that one can find in linear time a spherical-rectangular drawing of a subcubic planar graph if it has one.

Introduction

A *plane graph* is a planar graph with a given planar embedding and a *plane-embedded graph* is a plane graph with a given face designated as the external face.

A rectangular drawing of a plane graph G is a crossing-free drawing of G on the plane in which each edge is drawn as a horizontal or vertical segment, and each face is drawn as a rectangle.

On a sphere with fixed North and South poles, a *meridian* is a semicircle with ends at the poles of the sphere. A *spherical-rectangular* drawing of a plane graph is a crossing-free drawing of the graph on a sphere such that edges do not have any bends (where an edge changes its direction abruptly) except at the poles, and each face is bounded by at most two circles of latitude and two meridians. We will present the definition more precisely in the next section. We say that a planar graph has a spherical-rectangular drawing if at least one of the plane graphs corresponding to its embeddings has a spherical-rectangular drawing. We are interested in characterization of plane and planar graphs with spherical-rectangular drawings.

In a spherical-rectangular drawing, if the North and South poles are in the interior of the same face, the drawing is equivalent to a rectangular drawing. Thomassen [8] proved a necessary and sufficient condition for plane-embedded graphs with vertices of degree 3 except for four vertices of degree 2 on the external face to have rectangular drawings. Rahman et al. [5,6,7] studied the problem

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 345–356, 2009.

on planar graphs with vertices of degree 2 and 3, and found necessary and sufficient conditions to check in linear time whether these graphs have rectangular drawings. Miura et al. [4] found an $O(n^{1.5}/\log n)$ algorithm for plane-embedded graphs with maximum degree 4.

In this paper we solve the problem of spherical-rectangular drawability in linear time for planar graphs with maximum degree 3. In Sections 2 and 3, we state our results on cubic and subcubic planar graphs, and in Section 4 we outline the proof of the main theorem of the paper. Solving the problem for planar graphs of maximum degree 4 is an interesting problem for future work.

In this paper, we consider graphs to be multigraphs, that is, graphs which may have multiple edges and loops.

1 Spherical-Rectangular Drawings

On a sphere, a *meridional arc* is a connected part of a meridian and a *circular arc* is a connected part of a circle of latitude. We denote the South pole and the North pole of the sphere by S and N, respectively, and define the concepts rectangle, sector and lens on a sphere as follows:

- A *rectangle* is a closed curve consisting of two disjoint non-zero length circular arcs and two (possibly coincident) non-zero length meridional arcs (see Figure 1(a,b)).
- A *sector* is a closed curve consisting of one non-zero length circular arc and two distinct non-zero length meridional arcs (see Figure 1(c)).
- A *lens* is a closed curve consisting of two distinct meridional arcs meeting at the poles (see Figure 1(d)).

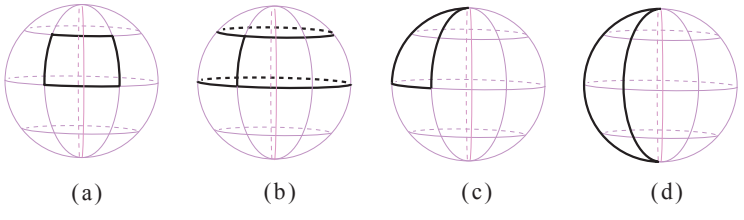


Fig. 1. (a) A rectangle with distinct meridional arcs, (b) a rectangle with coincident meridional arcs, (c) a sector, (d) a lens

A *spherical-rectangular drawing* of a plane graph G is a crossing-free drawing of G on a sphere such that each edge is drawn as a circular arc, a meridional arc or two meridional arcs connected at one of the poles, and the boundary of each face of G is drawn as a rectangle, circle, sector or lens (Figure 2).

For a plane graph G, a vertex, edge or face of G is called an *object* of G. In a spherical-rectangular drawing of G, we say that an object x *is drawn at* S if:

- x is a vertex and x is drawn at S,
- x is an edge and S is in the interior of x, or
- x is a face and S is inside x.

Being drawn at N is defined similarly.

Let G be a plane graph, and let x and y be objects of G. An *xy-drawing* of G is a spherical-rectangular drawing such that x is drawn at N and y is drawn at S. We say G is *xy-drawable* if G has an xy-drawing. Obviously G is xy-drawable iff G is yx-drawable.

As we mentioned a spherical-rectangular drawing is equivalent to a rectangular drawing on the plane if S and N are in the interior of the same face, which means that one face is drawn at both poles (Figure 2(e)). It can be considered that a spherical-rectangular drawing is a drawing on a cone or a lattice of concentric circles if at least one of the objects drawn at the poles is a face (Figure 2(b,c,d,e)), and a drawing on a cylinder if both of the objects drawn at the poles are faces (Figure 2(d,e)). A different investigation of drawings on a cylinder using lines parallel to the axis and circles perpendicular to the axis is presented in [2].

Note that in a spherical-rectangular drawing, an edge drawn at a pole consists of any two distinct meridional arcs which meet at the pole, and thus may have a bend at the pole. We can also ask for when the two meridional arcs can be required to be part of the same great circle passing through the pole. This question is more complex and we do not have a complete answer. If we have an edge e drawn at one of the poles and there is a vertex or face drawn at the other pole, we can always draw the graph in such a way that there is no bend in e. But when we have two edges drawn at the poles, there is not always a drawing with no bends. In Figure 3 we give an example.

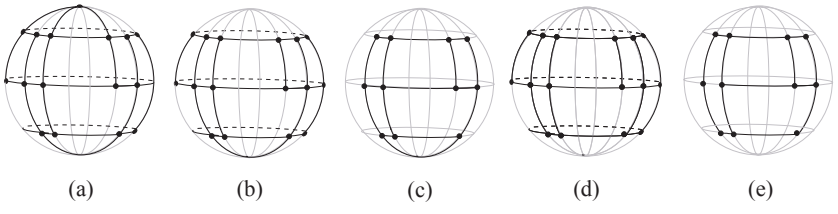


Fig. 2. Spherical-rectangular drawings with different objects drawn at the poles

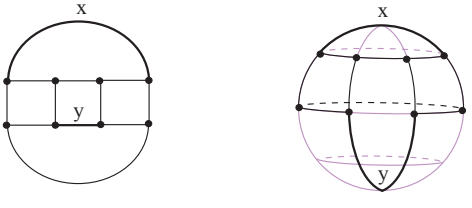


Fig. 3. Every xy-drawing of this graph has a bend at one of the poles

We will consider a number of related questions for a planar graph G.

1. If an embedding of G is given, then we can ask:
   (a) Given objects x and y, is G xy-drawable? This is answered in Theorem 1 for biconnected cubic graph G after which we explain the case of cubic graphs which are not biconnected.
   (b) Does G have any objects x and y, such that G is xy-drawable? This is achieved in linear time by Algorithm S-R(G) for biconnected cubic graph G.
2. If an embedding of G is not given, we can ask whether G has an embedding for which questions (a) and (b) have affirmative answers. This is answered in Theorems 3 and 4 for cubic graph G.

These questions are considered for subcubic graphs in Section 3.

2 Cubic Graphs

2.1 Cubic Plane Graphs

Let G be a plane graph. A cycle of G with length k is called a *proper k-cycle* if it is not the boundary of a face. We denote the planar dual of G by G^* and the object of G^* corresponding to an object x of G by x^*.

The following theorem gives necessary and sufficient conditions for a cubic plane graph G to have an xy-drawing for objects x and y.

Theorem 1. *Let G be a biconnected cubic plane graph with more than two vertices, and let x and y be objects of G. Then G is xy-drawable iff x^* and y^* are on opposite sides of each proper 2-cycle and proper 3-cycle of G^* and one of the following holds (possibly with the roles of x and y reversed).*

1- x and y are non-adjacent faces;
2- x is a face and y is a vertex which is not on the boundary of x.
3- x is a face and y is an edge with no end-vertex on the boundary of x;
4- x and y are edges with no common end-vertices;
5- x is an edge and y is a vertex which is not an end-vertex of x.
6- x and y are distinct vertices.

Proof. A sketch of the proof is given in Section 4.

Let G be a cubic plane graph, connected but not necessarily biconnected, and let x and y be two objects of G. If G is xy-drawable, then for each bridge e of G, if g is the unique face whose boundary includes e, then g should be drawn as a rectangle that has exactly one meridional arc which is e, and x and y are in the different components of $G \setminus \{e\}$ (see Figure 4(a)).

Let G_x be the component including x, G_y be the component including y, and let g_x and g_y be the faces of G_x and G_y, respectively, subdivided by end-vertices of e. Then G_x is $g_x x$-drawable and G_y is $g_y y$-drawable. Conversely, if G_x is $g_x x$-drawable and G_y is $g_y y$-drawable, then G is xy-drawable. If G has several bridges,

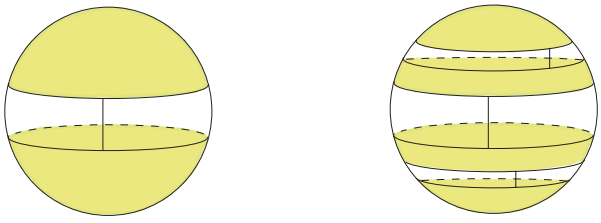


Fig. 4. Spherical-rectangular drawings of graphs with bridges

then applying the same logic to each bridge shows that a spherical-rectangular drawing has a form as shown in Figure 4(b). This decomposition can be carried out in linear time and then a linear implementation of Theorem 1 applied to each biconnected component can be used to test if G is xy-drawable and find such a drawing if it exists.

We next present a linear-time algorithm to find two objects x and y of a biconnected cubic plane graph G such that G is xy-drawable. The extension to the general case of connected G will be presented in the full version of the paper.

Let G be a plane-embedded graph and let C be a cycle of G. Then $G(C)$ is the subgraph of G consisting of C and all the vertices and edges inside C, and $G_o(C)$ is the subgraph of G consisting of C and all the vertices and edges outside C. In each case we consider the subgraph embedded in the plane with external face bounded by C.

Algorithm. S-R(G)
Input: a biconnected cubic plane graph G.
Output: two objects x and y of G such that G is xy-drawable, or $\emptyset$ if there is no such pair of objects.

Begin

1. Embed G^* in the plane and let C_0 be the boundary of the external face. If G^* has two vertices joined by more than two edges, return $\emptyset$.

2. If G consists of two vertices x and y connected by 3 edges, return $\{x, y\}$, and if G is the complete graph K_4, return $\{x, y\}$ where x is a face and y is the vertex which does not lie on x.

3. If G^* has no proper 2-cycles or 3-cycles return $\{x, y\}$, where x and y are any two non-adjacent vertices of G.

4. Let C be a shortest proper cycle of G^*. Set $x = \text{Proc}(G^*(C))$ and $y = \text{Proc}(G^*_o(C))$. If $x = \emptyset$ or $y = \emptyset$, return $\emptyset$; otherwise return $\{x, y\}$.

End

Procedure Proc(H)

Input: a biconnected plane-embedded graph H whose internal faces have size at least 3.

Output: a vertex x which is in the interior of all proper 2-cycles and 3-cycles of H and not on the boundary of the external face, or $\emptyset$ if there is no such vertex.

Begin

1. If H has two vertices joined by more than two edges, return $\emptyset$.
2. Set $C_0 =$ the boundary of the external face of H.
3. For each proper 2-cycle C of H, if $H(C) \subseteq H(C_0)$ set $C_0 = C$, otherwise if $H(C_0)$ is not a subgraph of $H(C)$ return $\emptyset$.
4. For each proper 3-cycle C of H, if $H(C) \subseteq H(C_0)$ set $C_0 = C$, otherwise if $H(C_0)$ is not a subgraph of $H(C)$ return $\emptyset$.
5. Return x, where x is a vertex in the interior of C_0.

End

Theorem 2. *We can find in linear time a spherical-rectangular drawing of a connected cubic plane graph if it has one.*

Proof. If the graph is biconnected, we apply Algorithm S-R. The correctness of the algorithm is a consequence of Theorem 1. To implement the algorithm in linear time, we embed G^* on the integer grid with bounded bends per edge using a suitable linear-time algorithm (such as [9]) so that questions of whether a short cycle or object lie inside or outside a specified short cycle can be answered in constant time. We find the 2-cycles and 3-cycles by deleting one of each pair of parallel edges, applying the linear-time algorithm of [3] to find the remaining 2-cycles and 3-cycles, then adjusting the list to reincorporate the deleted edges.

The case that G has bridges relies on a decomposition as in Figure 4(b) and will be treated in the full version of the paper.

2.2 Cubic Planar Graphs

For a planar graph G and an embedding ρ of G, G_ρ is the plane graph corresponding to embedding ρ of G.

In this section, we consider the case of a cubic planar graph whose embedding is not specified. The following theorem shows that to test whether a biconnected cubic planar graph has a spherical-rectangular drawing it is enough to test one of the plane graphs corresponding to its embeddings.

Theorem 3. *Let G be a biconnected cubic planar graph. Then either all the plane graphs corresponding to embeddings of G have spherical-rectangular drawings or none of them do.*

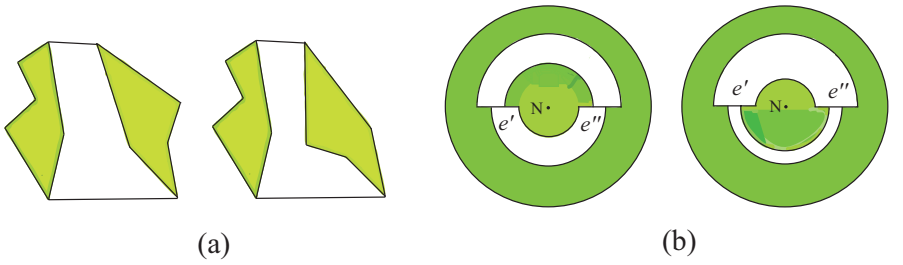


Fig. 5. (a) Flipping over one side of a 2-edge-cut, (b) flipping over one part of a drawing

Specifically, for two embeddings ρ_1 and ρ_2 of G, if f_1 and h_1 are objects of G_{ρ_1} such that G_{ρ_1} is f_1h_1-drawable, then G_{ρ_2} is f_2h_2-drawable for two objects f_2 and h_2 such that: (1) if f_1 is a face, then f_2 is a face with the same boundary as f_1 (possibly reversed), otherwise $f_2 = f_1$; and (2) similarly for h_1 and h_2.

Proof. It follows from Lemma 4.2 of [1] that any planar embedding of G can be obtained from a single fixed embedding by a finite number of operations which consist of flipping over one side of a 2-edge-cut (see Figure 5(a)). Suppose this operation applied at 2-edge-cut $\{e', e''\}$ takes embedding ρ_1 to embedding ρ_2.

Now take a spherical-rectangular drawing of G_{ρ_1}. Without losing generality, we can assume that e' and e'' are drawn as meridional arcs. This allows us to make a spherical-rectangular drawing of G_{ρ_2} by flipping over one part of the drawing as in Figure 5(b).

Theorem 3 does not hold in the case that the graph is not biconnected. For a connected cubic planar graph G with at least one bridge, let A be the set of bridges of G. Define $T'(G)$ to be the graph with the components of $G \setminus A$ as its vertices and two vertices adjacent iff their corresponding components are incident to the same bridge of G. Let H be a component of $G \setminus A$ and ρ be an embedding of H. A face f of H_ρ is a *candidate face* if the boundary of f includes an end-vertex of a bridge of G. The following theorem gives necessary and sufficient conditions for G to have a spherical-rectangular drawing.

Theorem 4. *Let G be a connected cubic planar graph with at least one bridge. Then G has a spherical-rectangular drawing iff all of the following hold:*

(1) $T'(G)$ is a path.
(2) Let $G^1, G^2, \ldots, G^k$ be the components of $G \setminus A$ in the order given by the path $T'(G)$, and let ρ_i be an arbitrary embedding of G^i. Then:
  (a) $G^1_{\rho_1}$ has a candidate face h_1, and an object x such that $G^1_{\rho_1}$ is xh_1-drawable,
  (b) $G^k_{\rho_k}$ has a candidate face f_k, and an object y such that $G^k_{\rho_k}$ is f_ky-drawable, and
  (c) For $i = 2, \ldots, k-1$, $G^i_{\rho_i}$ has two different candidate faces f_i and h_i such that $G^i_{\rho_i}$ is f_ih_i-drawable.

Proof. The theorem is a consequence of Theorem 3.

A corollary of the proof of Theorem 4 is that, if a connected cubic planar graph G has a spherical-rectangular drawing then, for an arbitrarily chosen embedding of each biconnected component of G, there is a spherical-rectangular drawing of G that induces the chosen embedding of each biconnected component.

Theorem 5. *Let G be a connected cubic planar graph. Then we can find in linear a spherical-rectangular drawing of G if there is one.*

3 Subcubic Planar Graphs

A graph whose vertices have degree at most 3 is called *subcubic*. Let v be a degree 2 vertex of a graph G with neighbours u and w. Remove v and add edge uw. We call this operation *smoothing* v. Define G_s to be the result of smoothing all degree 2 vertices of G.

Theorem 6. *A connected subcubic plane graph G has a spherical-rectangular drawing iff at least one of the following holds:*

(a) G has a rectangular drawing,
(b) G_s has a spherical-rectangular drawing, and
(c) G has a vertex or edge y on the boundary of a face x such that G is xy-drawable. (In this case face x is drawn as a sector. (Figure 2(c))

Proof. A graph with a spherical-rectangular drawing does not have any vertex of degree 1, so we consider only the subcubic graphs with minimum degree 2.

The sufficiency is obvious. To prove the necessity, note that a face of a spherical-rectangular drawing has a $3\pi/2$ angle other than at a pole only in the cases that the drawing is a rectangular drawing or there is a face x drawn at one of the poles and a vertex or edge on the boundary of x drawn at the other pole. So if those cases do not happen, each vertex of degree 2 not drawn at a pole has an angle of π. Therefore, every spherical-rectangular drawing of G corresponds to a spherical-rectangular drawing of G_s.

The first case of the theorem is characterized by Rahman et al. [5,6,7] and the second case in Section 2. The third case is characterized by the following theorem if G_s does not have any 2-edge-cuts.

Let x be a face of a plane graph G and let C be a cycle of G. An *x-leg* of C is an edge of $G \setminus C$ incident to C and lying on the same side of C as x. Note that a chord of C on the same side as x is counted as two legs. Two cycles C_1 and C_2 of G are *x-independent* if $G(C_1)$ and $G(C_2)$ have no common vertices when G is embedded in the plane with external face x.

Theorem 7. *Let G be a subcubic plane graph of minimum degree at least 2 such that G_s does not have any 2-edge-cuts, and let x be a face of G. Then G has a vertex or edge y on the boundary of x such that G is xy-drawable iff all the following hold:*

(a) all the proper 3-cycles of $(G_s)^*$ include x^*;
(b) there are at least two degree 2 vertices on the boundary of x;
(c) G has at most three pairwise x-independent cycles with three x-legs and for every two x-independent cycles with three x-legs, at least one of them has a degree 2 vertex lying on x.

Theorems 6 and 7 provide a linear-time algorithm to test if a connected subcubic plane graph which is a subdivision of a 3-connected cubic graph has a spherical-rectangular drawing. In the full version of the paper, we will discuss the subdivisions of general cubic with or without a specified embedding and show that it can also be handled in linear time. In summary we have the following.

Corollary 1. *Let G be a connected subcubic planar graph. We can find in linear time a spherical-rectangular drawing of G if there is one.*

4 Proof of Theorem 1

Let x and y be faces of a plane graph G and let C be a cycle of G. Then C is called an *xy-bad cycle* if x and y are on the same side of C and C has less than four x-legs.

Proof. First consider the case that x and y are faces. Suppose G is xy-drawable. Let C be a cycle of G that does not separate x and y. Then the vertices at the ends of north-most and south-most circular arcs of C must be incident to x-legs. So C has at least four x-legs and is not an xy-bad cycle . The absence of xy-bad cycles shows that x and y are not adjacent. If there is a proper 2-cycle or 3-cycle C with x and y on the same side, choose such a cycle with the minimum number of vertices on the other side. Then, the boundary of the part of G that lies on the latter side of C is an xy-bad cycle. Therefore, no such proper 2-cycles and 3-cycles exists.

Conversely, suppose that x and y are non-adjacent and on opposite sides of each proper 2-cycle and 3-cycle of G^*. If G has an xy-bad cycle C, the edges of G^* corresponding to the x-legs of C induce a proper 2-cycle or 3-cycle with x and y on the same side, except for two cases: (a) C has a chord on the same side as x and y, and (b) C has three x-legs whose end-vertices not lying on C are the same. In both cases (a) and (b) all the faces on the same side as x are adjacent and so are x and y. Therefore, G does not have any xy-bad cycles.

If there is a face g which is adjacent to both x and y, then the common boundary of g and x is an edge e_x, and the common boundary of g and y is an edge e_y. Subdivide e_x and e_y with new vertices and join them with an edge e crossing g. Cut the graph through e. By Theorem 2.5 in [5], the resulting graph has a rectangular drawing which we can easily convert to an xy-drawing of G. Otherwise, since G is cubic and has no xy-bad cycles, each face adjacent to x shares exactly one edge with x. Define $f_1, f_2, ..., f_k, f_{k+1} = f_1$ to be the faces adjacent to x in cyclic order. The absence of xy-bad cycles implies $f_i \cap f_j$ is empty for $j \neq i-1, i, i+1$. Therefore, the edges which are on the boundary of

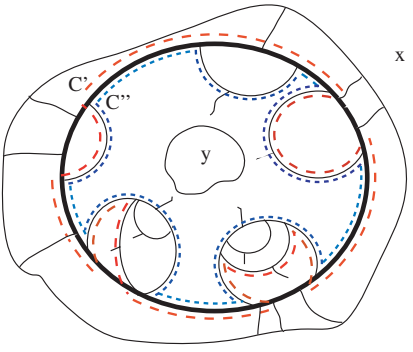


Fig. 6. The bold cycle is C and the dashed cycles are C' and C''

exactly one of $f_1, f_2, ..., f_k, x$ induce a cycle C which separates x and y, and has no common edge with x or y.

Let G_x be the subgraph of G consisting of C and all the vertices and edges on the same side of C as x and let g_x be the face of G_x with boundary C. Define G_y and g_y similarly. It is clear that G_x has no xg_x-bad cycles. If G_y has any yg_y-bad cycles, they must have connected intersection with C. By using maximal yg_y-bad cycles of G_y and analogous to the method used in [5], we can construct two cycles C' and C'' such that the following conditions hold:

(a) C' and C'' separate x and y, they are disjoint from the boundaries of x and y, and all edges of $C' \setminus C''$ are on the same side of C'' as x.

(b) Let G' be the subgraph of G consisting of C' and all the vertices and edges on the same side of C' as x and let g' be the face of G' with boundary C'. Define G'' and g'' similarly. Then G' has no xg'-bad cycles and G'' has no yg''-bad cycles.

(c) The resulting graph after removing common edges of C' and C'' from $[G \setminus (G' \cup G'')] \cup (C' \cup C'')$ consists of some isolated vertices and t components G_1, G_2,..., G_t which each have exactly four degree 2 vertices lying on the boundary of the same face and satisfy the conditions of Theorem 2.5 in [5].

Then we find spherical-rectangular drawings of G', G'' and G_i, $i = 1, ..., t$, and combine these drawings to produce a spherical-rectangular drawing of G.

An illustration of how to construct C' and C'' is given in Figure 6.

Now we consider all other cases when x and y are not necessarily faces. We will use two following operations.

Let v be a degree d vertex of a plane graph G, and let $v_1, \ldots, v_d$ be the neighbors of v in clockwise order around v. Replace v with the cycle $u_1u_2, u_2u_3, \ldots, u_du_1$ and replace edges $vv_1, vv_2, \ldots, vv_d$ with edges $u_1v_1, u_2v_2, \ldots, u_dv_d$. We call this operation *replacement of a vertex with a cycle* (see Figure 7(a)).

Let e be an edge of G with end-vertices v_1 and v_2. Replace e with two parallel edges u_1u_2 and two edges v_1u_1 and v_2u_2. We call this operation *replacement of an edge with a cycle* (see Figure 7(b)).

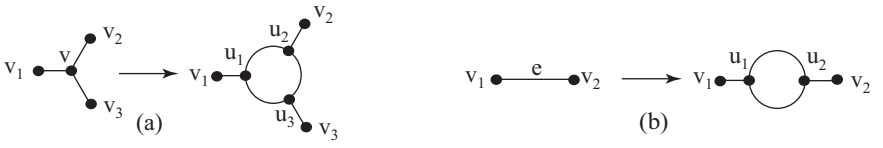


Fig. 7. (a) Replacement of a vertex with a cycle, (b) replacement of an edge with a cycle

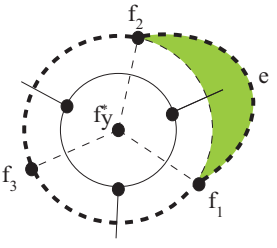


Fig. 8. The dashed lines are edges of G_0^*. Vertices x^* and f_y^* are on the same side of the thick dashed cycle when they are on opposite sides of the cycle f_1f_2, f_2f_1.

Let G_0 be the graph obtained from G by replacing each of x and y with a cycle unless they are faces already. Define $f_x = x$ if x is a face, and the face bounded by the cycle replacing x if x is not a face. Define f_y similarly. It is clear that G is xy-drawable iff G_0 is f_xf_y-drawable.

In order to complete the proof, we consider the various cases of whether x and y are faces, vertices or edges. In each case, we can show that the following conditions are equivalent: (1) f_x^* and f_y^* are non-adjacent and on the opposite sides of each proper 2-cycle and 3-cycle of G_0^*; (2) one of the conditions 1–6 holds, and x^* and y^* are on the opposite sides of each proper 2-cycle and 3-cycle of G^*.

For example if x is a face and y is a vertex, then f_x and f_y are adjacent iff y is on the boundary of x. G^* is a subgraph of G_0^*, namely $G^* = G_0^* \setminus \{f_y^*\}$. Suppose that x^* and y^* are on opposite sides of each proper 2-cycle and 3-cycle of G^*. Let C be a proper 2-cycle or 3-cycle of G_0^*. If C is a proper 2-cycle or 3-cycle of G^*, then f_x^* and f_y^* are on opposite sides of C. Otherwise C either includes f_y^* or $C = f_1f_2, f_2f_3, f_3f_1$ where f_1, f_2, f_3 are the vertices adjacent to f_y^*. In the first case, without loss of generality we can suppose $C = f_y^*f_1, f_1f_2, f_2f_y^*$. Since C is proper there are two parallel edges between f_1 and f_2 such that f_1f_2, f_2f_1 is a proper 2-cycle. Let e be the edge between f_1 and f_2 such that $e, f_1f_y^*, f_y^*f_2$ is not the boundary of a face. Cycle e, f_1f_3, f_3f_2 in G^* is a proper 3-cycle such that x^* and y^* are on the same side of the cycle when they are on opposite sides of cycle f_1f_2, f_2f_1 (see Figure 8). So this case does not happen. If $C = f_1f_2, f_2f_3, f_3f_1$, then f_x^* and f_y^* are on opposite sides of C since y is not on the boundary of x by Condition 2.

The fact that $G^* = G_0^* \setminus \{f_y^*\}$ proves that, if f_x^* and f_y^* are on opposite sides of each proper 2-cycle or 3-cycle of G_0^*, then x^* and y^* are on opposite sides of each proper 2-cycle or 3-cycle of G^*.

The other cases can be proved in similar fashion.

Acknowledgement

The authors would like to acknowledge that Maryam Tahmasbi contributed to the first part of the proof of Theorem 1.

References

1. Di Battista, G., Tammasia, R.: On-line planarity testing. SIAM Journal on Computing 5, 956–997 (1996)
2. Hasheminezhad, M., Hashemi, M.S., Tahmasbi, M.: Ortho-radial drawings of graphs. Australasian Journal Combinatorics (to appear)
3. Kowalik, L.: Short cycles in planar graphs. In: Bodlaender, H.L. (ed.) WG 2003. LNCS, vol. 2880, pp. 284–296. Springer, Heidelberg (2003)
4. Miura, K., Haga, H., Nishizeki, T.: Inner rectangular drawings of plane graphs. International Journal of Computational Geometry and Applications 16, 249–270 (2006)
5. Rahman, M.S., Nakano, S., Nishizeki, T.: Rectangular grid drawing of plane graphs. Computational Geometry 10, 203–220 (1998)
6. Rahman, M.S., Nakano, S., Nishizeki, T.: Rectangular drawings of plane graphs without designated corners. Computational Geometry 21, 121–138 (2002)
7. Rahman, M.S., Nishizeki, T., Ghosh, S.: Rectangular drawings of planar graphs. Journal of Algorithms 50, 62–78 (2004)
8. Thomassen, C.: Plane representations of graphs. Progress in Graph Theory, 43–69 (1984)
9. Zhang, H., He, X.: Compact visibility representation and straight-line grid embedding of plane graphs. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) WADS 2003. LNCS, vol. 2748, pp. 493–504. Springer, Heidelberg (2003)

The Exemplar Breakpoint Distance for Non-trivial Genomes Cannot Be Approximated

Guillaume Blin[1], Guillaume Fertin[2], Florian Sikora[1], and Stéphane Vialette[1]

[1] Université Paris-Est, IGM-LabInfo - UMR CNRS 8049, France
{gblin,sikora,vialette}@univ-mlv.fr
[2] LINA - UMR CNRS 6241 - Université de Nantes - France
guillaume.fertin@univ-nantes.fr

Abstract. A promising and active field of comparative genomics consists in comparing two genomes by establishing a one-to-one correspondence (i.e., a matching) between their genes. This correspondence is usually chosen in order to optimize a predefined measure. One such problem is the Exemplar Breakpoint Distance problem (or EBD, for short), which asks, given two genomes modeled by signed sequences of characters, to keep and match exactly one occurrence of each character in the two genomes (a process called *exemplarization*), so as to minimize the number of breakpoints of the resulting genomes. Bryant [6] showed that EBD is **NP**-complete. In this paper, we close the study of the approximation of EBD by showing that no approximation factor can be derived for EBD considering non-trivial genomes – i.e. genomes that contain duplicated genes.

1 Introduction

Comparative genomics is a recent and active field of bioinformatics. One of the problems arising in this domain consists in comparing two species, and more specifically to look for conserved sets of genes between their genomes: a set of genes that is conserved in the same order during the evolution suggests that it participates to the same biological process. Finding conserved sets of genes in genomes is usually done by optimizing a given (dis)similarity measure. Many such measures have already been studied in the recent past: number of breakpoints, of adjacencies, of conserved intervals, of common intervals, etc. In this paper, we focus on the *number of breakpoints* between genomes.

All these measures are well-defined when genomes do not contain duplicates, and can usually be computed in polynomial time in this case. However, this assumption does not hold biologically. Moreover, by definition, the above mentioned measures do not apply when genes appear several times in a genome. A way to overcome this difficulty is to start from two genomes and to obtain a one-to-one correspondence (i.e., a matching) between their genes, in order to end up with a permutation on which the measure can then be computed. Among all possible matchings, the choice goes to the one that optimizes the studied measure. There exists several ways to achieve the desired matching. In this paper,

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 357–368, 2009.

we are interested in the so-called *exemplar* model [9], where, for any gene family, exactly one gene is kept (and thus matched) in the genome. The motivation for this model is that the gene which is kept is assumed to be the ancestral gene, from which the other copies have derived.

Following the notations from Blin et al. [5], given an alphabet Σ of elements called *gene families*, a genome G on Σ is a sequence of signed elements of Σ, where the sign represents the DNA strand on which the gene lies. Each occurrence of an element of Σ in G is called a *gene*. For any gene family $g \in \Sigma$, we denote by $occ(G, g)$ the number of genes ($+g$ and $-g$) that appear in G. Let also $occ(G) = \max\{occ(G, g) \mid g \in \Sigma\}$.

For any genome G, a gene family g is said to be *trivial* if $occ(G, g) = 1$. Otherwise, g is said to be *non-trivial*. A gene belonging to a trivial (resp. non-trivial) family is said to be trivial (resp. non-trivial). A genome is called *trivial* if it only contains trivial genes, i.e. if it is a signed permutation. For convenience, we will use characters to represent each gene. In the following, given a genome G over an alphabet Σ, let $\chi(i, g, G)$ denote the i^{th} occurrence of character $g \in \Sigma$ in G (not taking signs into account). When there is no ambiguity, we will simply use $\chi(i, g)$. Moreover, we will refer to the i^{th} character of G as $G[i]$. We will also note $G[i] < G[j]$ for any $i < j$, that is when $G[i]$ appears before $G[j]$. For any gene g in G, we denote by $\overline{g}$ the gene with opposite sign. As introduced by Chen et al. [7], a genome G is called an *s-span* genome if all the genes from the same gene family g are within distance at most s in G. For example, let $G = +a\ -d\ +c\ -b - d - a\ +e\ +b\ -b$. We have $occ(G, a) = 2$, $occ(G) = 3$, $\chi(2, b) = +b$ and G is a 5-span genome.

Given a trivial genome G, we say that gene $g = G[i]$ immediately precedes $g' = G[j]$ iff $j = i+1$. Given two trivial genomes G_1 and G_2, if gene g immediately precedes gene g' in G_1 while neither (i) g immediately precedes g' nor (ii) $\overline{g'}$ immediately precedes $\overline{g}$ in G_2, then they constitute a *breakpoint* in G_2. The breakpoint distance between two trivial genomes G_1 and G_2 is then defined as the number of breakpoints in G_2.

As previously mentioned, the breakpoint distance is not well defined for non-trivial genomes. The idea is then to establish a matching between genes of G_1 and G_2, in order to get back to a signed permutation on which the breakpoint distance can be computed. The *exemplar* model, introduced by Sankoff [9], is one of several ways to construct the matching, which consists in keeping, for each gene family, only one occurrence of its genes in G_1 and in G_2. This raises the following problem, named Exemplar Breakpoint Distance problem, or EBD for short.

Given two genomes G_1 and G_2, built over the same alphabet Σ, and an integer k, the Exemplar Breakpoint Distance problem asks whether it is possible to establish an exemplar matching of G_1 and G_2, such that the breakpoint distance between the resulting genomes is at most k.

Bryant [6] showed that EBD is **NP**-complete, even when one of the genomes is trivial, and the other has genes that appear at most twice in each genome. Concerning (in)approximability results, Angibaud et al. [2] proved that EBD is

APX-hard under the same conditions. Chen et al. [7] also showed that there exists no approximation algorithm for EBD, even when both genomes have genes that appear at most three times. However, this result does not completely close the question of EBD potential approximation. Indeed, whether EBD can be approximated on genomes that contain at most 2 copies of each gene remains unknown, and was actually raised as an open question by Chen et al. [7] and Angibaud et al. [4]. In this paper, we will answer this open question by showing that no approximation factor can be derived for EBD, even when considering genomes in which each gene occurs at most twice.

Chen et al. [7] also provided a logarithmic approximation ratio for the particular case in which one of the genomes is an s-span genome, with $s = O(\log m)$, $m = |\Sigma|$. It should also be noted that Nguyen et al. [8] designed a divide-and-conquer heuristic method in order to compute the Exemplar Breakpoint Distance while Angibaud et al. [3] proposed an exact method based on transforming the problem into a 0-1 linear programming problem.

In order to prove that EBD is not approximable, we will prove that a particular subproblem of EBD – called the Zero Exemplar Breakpoint Distance problem (ZEBD for short) – is **NP**-complete. This decision problem asks whether there exists an exemplar matching of two genomes, such that the breakpoint distance between the resulting genomes is equal to zero. For sake of readability, for any $1 \leq p \leq q$, we will denote ZEBD(p, q) the ZEBD problem in which $occ(G_1) = p$ and $occ(G_2) = q$. It is easy to see that ZEBD$(1, q)$ can be solved in linear time, for any $q \geq 1$. Chen et al. [7] showed that ZEBD$(3, 3)$ is **NP**-complete. Angibaud et al. [4] also showed that ZEBD$(2, q)$ is **NP**-complete, but with a value of q unbounded due to their reduction. Hence, the remaining unknown cases concern the complexity of ZEBD$(2, q)$, with fixed q. We will answer this question, by proving that ZEBD$(2, 2)$ (and thus, ZEBD$(2, q)$ for any $q \geq 2$) is **NP**-complete.

In this paper, we thus focus on ZEBD. More precisely, we first complete and close the study of the complexity of ZEBD, by proving that ZEBD is **NP**-complete, even when both genomes contain at most two occurrences of each gene. This result thus provides a full characterization of the polynomial and **NP**-complete cases for ZEBD, and also answers an open question raised by Chen et al. [7] and Angibaud et al. [4]. It also proves that no approximation factor can be derived for EBD, even when considering genomes in which each gene occurs at most twice; that is the simpliest case considering non-trivial genomes. We then propose to overcome this difficulty by studying the fixed-parameter tractability of ZEBD which was also leaved as an open question in [7].

2 Inapproximability of Exemplar Breakpoint Distance

In this section, we prove that ZEBD$(2, 2)$ is **NP**-complete. This result implies that EBD does not admit any approximation unless $\mathbf{P} = \mathbf{NP}$, even when both genomes contain at most two occurrences of each gene.

Theorem 1. ZEBD$(2, 2)$ *is* **NP***-complete.*

It is easy to see that ZEBD is in **NP**. In order to prove its **NP**-hardness, we propose a reduction from 3-SAT : let $V_n = \{x_1, x_2, \ldots, x_n\}$ be a set of n boolean variables, and $C_q = \{c_1, c_2, \ldots, c_q\}$ be a collection of q clauses, where each clause is a disjunction of three literals taken from V_n. The 3-SAT problem asks whether there exists an assignment of each variable of V_n such that each clause is satisfied. Let $\mathcal{I} = (C_q, V_n)$ be an instance of 3-SAT. From $\mathcal{I}$, we will build an instance $\mathcal{I}' = (G_1, G_2)$ of ZEBD, such that $occ(G_1) = occ(G_2) = 2$. In our construction, all genes carry a positive sign, which is omitted for sake of clarity. Moreover, for convenience, for any $1 \leq i \leq q$ and $1 \leq j \leq 3$, we let L_i^j denote the j^{th} literal of clause c_i in C_q. Moreover, for any $1 \leq k \leq n$, let N_{x_k} (resp. $N_{\overline{x_k}}$) denote the number of occurrences of x_k (resp. $\overline{x_k}$) in C_q. For each clause $c_i \in C_q$, $1 \leq i \leq q$, we first build a pair of sequences (U_i, V_i), with $U_i = \mathbf{U_i^1}\ d_i\ \mathbf{U_i^2}\ d_i\ \mathbf{U_i^3}\ t_i$ and $V_i = \mathbf{V_i^1}\ d_i\ \mathbf{V_i^2}\ d_i\ \mathbf{V_i^3}\ t_i$, such that

$$U_i = \overbrace{m_i^1\ \mathbf{T_{L_i^1}}\ p_i^1\ \mathbf{F_{L_i^1}}\ a_i\ m_i^1}^{U_i^1}\ d_i\ \overbrace{a_i\ m_i^2\ \mathbf{T_{L_i^2}}\ p_i^2\ \mathbf{F_{L_i^2}}\ b_i\ m_i^2}^{U_i^2}\ d_i\ \overbrace{b_i\ m_i^3\ \mathbf{T_{L_i^3}}\ p_i^3\ \mathbf{F_{L_i^3}}\ m_i^3}^{U_i^3}\ t_i$$
$$V_i = \underbrace{p_i^1\ \mathbf{F_{L_i^1}}\ m_i^1\ \mathbf{T_{L_i^1}}\ p_i^1\ a_i}_{V_i^1}\ d_i\ \underbrace{p_i^2\ a_i\ \mathbf{F_{L_i^2}}\ m_i^2\ \mathbf{T_{L_i^2}}\ p_i^2\ b_i}_{V_i^2}\ d_i\ \underbrace{p_i^3\ b_i\ \mathbf{F_{L_i^3}}\ m_i^3\ \mathbf{T_{L_i^3}}\ p_i^3}_{V_i^3}\ t_i$$

Let us now define formally $T_{L_i^j}$ and $F_{L_i^j}$ for all $1 \leq i \leq q$ and $1 \leq j \leq 3$. Let $T_{L_i^j} = T^1_{L_i^j}\ T^2_{L_i^j}$ (resp. $F_{L_i^j} = F^1_{L_i^j}\ F^2_{L_i^j}$), defined as follows:

- if (i) L_i^j is the first occurrence of x_k or $\overline{x_k}$ in C_q, and (ii) N_{x_k} and $N_{\overline{x_k}}$ are both strictly positive, then $T^1_{L_i^j} = y_k^1$ and $F^1_{L_i^j} = y_k^2$; otherwise, $T^1_{L_i^j}$ and $F^1_{L_i^j}$ are empty.
- if L_i^j is the l^{th} occurrence of x_k (resp. $\overline{x_k}$), let $p = l + 1$ if $l < N_{x_k}$ (resp. $l < N_{\overline{x_k}}$), and $p = 1$ otherwise. If $N_{x_k} > 1$ (resp. $N_{\overline{x_k}} > 1$), then $T^2_{L_i^j} = x_k^l$ (resp. $\overline{x_k}^l$) and $F^2_{L_i^j} = x_k^p$ (resp. $\overline{x_k}^p$) ; otherwise, $T^2_{L_i^j}$ and $F^2_{L_i^j}$ are empty.

Genomes G_1 and G_2 are then defined as $G_1 = U_1\ U_2\ \ldots U_q$ and $G_2 = V_1\ V_2\ \ldots V_q$. Clearly, this construction can be carried out in polynomial time, and it can also be seen that $occ(G_1) = occ(G_2) = 2$.

We now give an intuitive description of the different elements of this construction: each clause $c_i \in C_q$, $1 \leq i \leq q$, is represented by a pair (U_i, V_i) of sequences. Those sequences are both composed of three subsequences representing the literals of c_i. More precisely, the pair (U_i^j, V_i^j) represents a selection mechanism of the j^{th} literal of c_i. For each variable $x_k \in V_n$, the set $\{x_k^1, x_k^2, x_k^3, \ldots, x_k^{N_{x_k}}\}$ is used to propagate the selection of the literal x_k all over C_q, in order to make sure that if a literal satisfies a clause, then it satisfies all clauses where it appears. Similarly, the set $\{\overline{x_k}^1, \overline{x_k}^2, \overline{x_k}^3, \ldots, \overline{x_k}^{N_{\overline{x_k}}}\}$ is a propagation mechanism for $\overline{x_k}$. Finally, for each variable $x_k \in V_n$ such that N_{x_k} and $N_{\overline{x_k}}$ are both strictly positive, the pair (y_k^1, y_k^2) in both G_1 and G_2 is a control mechanism that guarantees that a variable x_k cannot be TRUE and FALSE simultaneously.

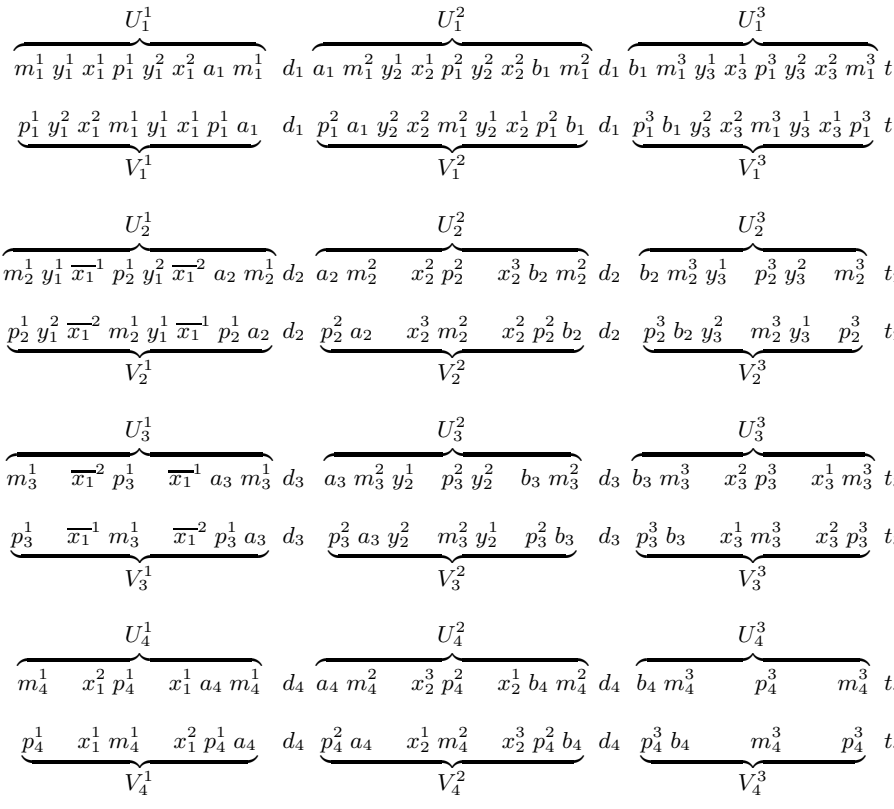


Fig. 1. The instance (G_1, G_2) of ZEBD(2, 2) obtained starting from $C_q = \{x_1 \vee x_2 \vee x_3), (\overline{x_1} \vee x_2 \vee \overline{x_3}), (\overline{x_1} \vee \overline{x_2} \vee x_3), (x_1 \vee x_2 \vee x_4)\}$

Figure 1 illustrates an example of an instance (G_1, G_2) of ZEBD(2, 2) obtained from our construction, starting from $C_q = \{(x_1 \vee x_2 \vee x_3), (\overline{x_1} \vee x_2 \vee \overline{x_3}), (\overline{x_1} \vee \overline{x_2} \vee x_3), (x_1 \vee x_2 \vee x_4)\}$.

In the following, let us denote by S_k (resp. S_d) the set of characters that are kept (resp. deleted) in an exemplarization of G_1 and G_2, in a given solution for ZEBD. By definition, exactly one occurrence of each gene family must be kept. We also note that by construction, since, for $1 \leq i \leq q$, there is only one occurrence of t_i in G_1 and in G_2 , characters of U_i may only be matched with characters of V_i. Moreover, for a given $1 \leq i \leq q$ and a given $1 \leq j \leq 3$, in U_i^j, $\chi(1, m_i^j) < p_i^j < \chi(2, m_i^j)$, whereas, in V_i^j, $\chi(1, p_i^j) < m_i^j < \chi(2, p_i^j)$. Those properties induce the following lemma.

Lemma 1. *In any solution for* ZEBD *on* (G_1, G_2)*, for any* $1 \leq i \leq q$ *and* $1 \leq j \leq 3$*, either (a)* $\{\chi(1, m_i^j, U_i^j), \chi(2, p_i^j, V_i^j)\} \subseteq S_k$ *or (b)* $\{\chi(2, m_i^j, U_i^j), \chi(1, p_i^j, V_i^j)\} \subseteq S_k$.

Lemma 2. *In any solution for* ZEBD *on* (G_1, G_2)*, for any* $1 \leq i \leq q$ *and* $1 \leq j \leq 3$*, at least one of* $\chi(1, m_i^1, U_i^1)$, $\chi(1, m_i^2, U_i^2)$, $\chi(1, m_i^3, U_i^3)$ *belongs to* S_k.

Proof. By contradiction, let us suppose that none of $\chi(1, m_i^1, U_i^1)$, $\chi(1, m_i^2, U_i^2)$, $\chi(1, m_i^3, U_i^3)$ belongs to S_k. Then, by Lemma 1, $\{\chi(2, m_i^j, U_i^j), \chi(1, p_i^j, V_i^j)\} \subseteq S_k$ for all $1 \leq j \leq 3$. Since in U_i, $\chi(1, a_i) < \chi(2, m_i^1, U_i^1) < \chi(1, d_i) < \chi(2, a_i) < p_i^2$, whereas in V_i, $m_i^1 < \chi(1, a_i) < \chi(1, d_i) < \chi(1, p_i^2, V_i^2) < \chi(2, a_i) < m_i^2$, we conclude that $\{\chi(1, a_i, U_i),\ \chi(2, a_i, V_i)\} \subseteq S_d$. Therefore, $\{\chi(1, d_i, U_i), \chi(1, d_i, V_i)\} \subseteq S_d$, whereas $\{\chi(2, a_i, U_i),\ \chi(1, a_i, V_i)\} \subseteq S_k$.

Moreover, since in U_i, $\chi(1, b_i) < \chi(2, m_i^2, U_i^2) < \chi(2, d_i) < \chi(2, b_i) < p_i^3$, whereas in V_i, $m_i^2 < \chi(1, b_i) < \chi(2, d_i) < \chi(1, p_i^3, U_i^3) < \chi(2, b_i) < m_i^3$, we conclude that $\{\chi(1, b_i, U_i), \chi(2, b_i, V_i)\} \subseteq S_d$. Thus, $\{\chi(2, d_i, U_i),\ \chi(2, d_i, V_i)\} \subseteq S_d$, whereas $\{\chi(2, b_i, U_i), \chi(1, b_i, V_i)\} \subseteq S_k$. Consequently, none of the occurrences of d_i can be kept in the exemplarization, a contradiction. □

Lemma 3. *Let* $I = \{(i_1, j_1), (i_2, j_2), \ldots (i_p, j_p)\}$ *such that* $\forall\ 1 \leq m \neq n \leq p$, $L_{i_m}^{j_m} = L_{i_n}^{j_n}$*. Then either (a)* $\{\chi(1, m_{i_m}^{j_m}, U_{i_m}^{j_m}), \chi(1, m_{i_n}^{j_n}, U_{i_n}^{j_n})\} \subseteq S_k$*, or (b)* $\{\chi(2, m_{i_m}^{j_m}, U_{i_m}^{j_m}), \chi(2, m_{i_n}^{j_n}, U_{i_n}^{j_n})\} \subseteq S_k$.

Proof. Let us first suppose that $L_{i_1}^{j_1} = x_k$. Then, by construction, $x_k^r < p_{i_r}^{j_r} < x_k^{r+1}$ in $U_{i_r}^{j_r}$ and $x_k^{r+1} < m_{i_r}^{j_r} < x_k^r$ in $V_{i_r}^{j_r}$, for any $1 \leq r < p$ and $x_k^p < p_{i_p}^{j_p} < x_k^1$ in $U_{i_p}^{j_p}$ and $x_k^1 < m_{i_p}^{j_p} < x_k^p$ in $V_{i_p}^{j_p}$. If $\chi(1, m_{i_r}^{j_r}, U_{i_r}^{j_r}) \in S_k$, for a given $(i_r, j_r) \in I$ such that $r < p$, x_k^{r+1} in $U_{i_r}^{j_r}$ must be deleted. Therefore, the two occurrences of x_k^{r+1} in $U_{i_{r+1}}^{j_{r+1}}$ and $V_{i_{r+1}}^{j_{r+1}}$ must be kept. Consequently, $\chi(1, m_{i_{r+1}}^{j_{r+1}}, U_{i_{r+1}}^{j_{r+1}}) \in S_k$. By induction on r, we have $\chi(1, m_{i_p}^{j_p}, U_{i_p}^{j_p}) \in S_k$. This implies that x_k^1 in $U_{i_p}^{j_p}$ must be deleted, which in turn means that x_k^1 in $U_{i_1}^{j_1}$ must be kept, and thus $\chi(1, m_{i_1}^{j_1}, U_{i_1}^{j_1}) \in S_k$. The induction can then be continued from $U_{i_1}^{j_1}$, and we conclude that $\chi(1, m_{i_n}^{j_n}, U_{i_n}^{j_n}) \in S_k$ for any $1 \leq i \leq p$. This proves case (a) of the above lemma, when $L_{i_1}^{j_1} = x_k$.

Moreover, if $\chi(1, m_{i_r}^{j_r}, U_{i_r}^{j_r}) \in S_d$, for a given $(i_r, j_r) \in I$ such that $r > 1$, then $\chi(2, m_{i_r}^{j_r}, U_{i_r}^{j_r}) \in S_k$ and x_k^r in $U_{i_r}^{j_r}$ must be deleted. Therefore, the two occurrences of x_k^r in $U_{i_{r-1}}^{j_{r-1}}$ and $V_{i_{r-1}}^{j_{r-1}}$ must be kept. Consequently, $\chi(2, m_{i_{r-1}}^{j_{r-1}}, U_{i_{r-1}}^{j_{r-1}}) \in S_k$. By induction on r, $\chi(2, m_{i_1}^{j_1}, U_{i_1}^{j_1}) \in S_k$ and, thus, this implies that x_k^1 in $U_{i_1}^{j_1}$ must be deleted, which in turn means that x_k^1 in $U_{i_p}^{j_p}$ must be kept, and thus $\chi(2, m_{i_p}^{j_p}, U_{i_p}^{j_p}) \in S_k$. The induction can then be continued from $U_{i_p}^{j_p}$, and we conclude that $\chi(2, m_{i_n}^{j_n}, U_{i_n}^{j_n}) \in S_k$ for any $1 \leq i \leq p$. This proves case (b) of the above lemma, when $L_{i_1}^{j_1} = x_k$.

By a similar reasoning, it is possible to prove that the same holds when $L_i^j = \overline{x_p}$. □

Lemma 4. $\forall\ (i,j),(i',j')$ *such that (1)* $L_i^j = \overline{L_{i'}^{j'}}$ *and (2)* L_i^j *and* $\overline{L_{i'}^{j'}}$ *are the first occurrences of the corresponding variable, only one of* $\chi(1, m_i^j, U_i^j)$ *and* $\chi(1, m_{i'}^{j'}, U_{i'}^{j'})$ *may be kept.*

Proof. Let $L_i^j = \overline{L_{i'}^{j'}} = x_k$ and suppose L_i^j (resp. $L_{i'}^{j'}$) is the first occurrence of x_k (resp. $\overline{x_k}$). If $\chi(1, m_i^j, U_i^j)$ is kept, then y_k^2 in U_i^j (resp. V_i^j) must be deleted, and therefore of y_k^2 in $U_{i'}^{j'}$ (resp. $V_{i'}^{j'}$) must be kept. In that case, it can be seen that $\chi(1, m_{i'}^{j'}, U_{i'}^{j'})$ must be deleted. The proof is similar if we choose to keep $\chi(1, m_{i'}^{j'}, U_{i'}^{j'})$. □

Thanks to the four above lemmas, we can prove the following theorem (the proof is omitted due to space constraints, and is devoted to the full version of the paper).

Theorem 2. *Let* $\mathcal{I}$ *be an instance of* 3-SAT*, and let* $\mathcal{I}' = (G_1, G_2)$ *be the instance of* ZEBD$(2,2)$ *constructed from* $\mathcal{I}$*. There exists a truth assignment that satisfies each clause in* $\mathcal{I}$ *iff there exists a zero breakpoint distance exemplarization in* $\mathcal{I}'$*.*

Altogether, this proves that ZEBD$(2,2)$ is **NP**-complete.

3 Exponential-Time Algorithms for ZEBD

It can be easily seen that, for two genomes G_1 and $G_2 = g_1\ g_2\ \ldots\ g_n$, if ZEBD is answered positively, the induced exemplarization is either (1) a common subsequence of G_1 and G_2 or (2) a common subsequence between G_1 and $\overleftarrow{G_2} = \overline{g_n}\ \ldots \overline{g_2}\ \overline{g_1}$. Therefore, any algorithm that answers ZEBD should check both cases. For simplicity, we will only discuss case (1) in this section. Checking case (2) just requires to run the same algorithm on $(G_1, \overleftarrow{G_2})$, instead of (G_1, G_2), which does not change the complexity.

Let G_1 and G_2 be two genomes defined over a set of m gene families, and such that $occ(G_1) = k_1$ and $occ(G_2) = k_2$. A brute-force algorithm for answering to ZEBD of G_1 and G_2 consists in computing all the possible exemplarizations of G_1 and G_2, and then determining whether one of them leads to a zero breakpoint distance. Since for any gene family there are at most k_i occurrences in G_i, $1 \leq i \leq 2$, there are at most $(k_1)^m$ (resp. $(k_2)^m$) exemplarizations of G_1 (resp. G_2). Moreover, given two exemplar genomes (each of length m), one can check in $O(m)$ whether their breakpoint distance is equal to zero. Therefore, on the whole the time complexity of the brute force algorithm is $O(m \cdot (k_1 \cdot k_2)^m)$ time. In the following, we present two fixed parameter algorithms, which give more feasible solutions.

Theorem 3. *There exists an* $O(m2^m)$ *algorithm for solving* ZEBD*, where* m *is the number of gene families of the input genomes.*

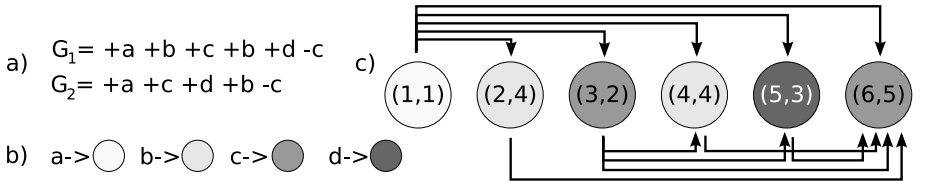


Fig. 2. a) Two genomes G_1 and G_2; b) A coloring function C; c) The corresponding graph (V, A)

Proof. The key idea is to decrease the time complexity of the brute-force algorithm by using a color-coding like method (see [1]). Color-coding is a technique to design fixed-parameter algorithms for several **NP**-complete subgraph isomorphism problems. This technique is based on a random coloring of each vertex of the input graph, using a small set of colors, and looking for a *colorful* path in the graph, that is a path whose vertices carry different colors and all colors are used.

Given two genomes G_1 and G_2 over a set Σ of m gene families, we build a directed graph (V, A) defined as follows. For each pair $(G_1[i], G_2[j])$ of genes of the same gene family that carry the same sign, add a vertex (i, j) to V. For all $\{(i, j), (p, q)\} \in V^2$ such that $i < p$ and $j < q$, add an edge from (i, j) to (p, q) in A. Finally, let $C : \Sigma \rightarrow \{c_1, c_2, \ldots, c_m\}$ be a function that assigns a unique color to each gene family. For each vertex $(i, j) \in V$, we assign to (i, j) the color $C(f(G_1[i]))$, where $f(G[i])$ denotes the gene family corresponding to $G[i]$. An illustration is given in Figure 2.

It can be seen that looking for a colorful path of length m in the graph (V, A) is equivalent to finding an zero breakpoint distance exemplarization of G_1 and G_2, since (i) genes belonging to the same family carry the same color, and (ii) the order is preserved by construction of A. It can be easily seen that there exists a dynamic programming algorithm in $O(m2^m)$ to answer that question. Indeed, we only need to maintain the set of colors already selected in the current path, instead of the current selected vertices of the graph. Therefore, ZEBD is fixed parameter tractable with respect to the number m of gene families. □

We now propose a second fixed-parameter algorithm for solving ZEBD, defined as follows. As previously, we transform the ZEBD problem into the problem of finding a path in a directed acyclic graph (or DAG, for short). But instead of having a vertex for each possible match between two genes of G_1 and G_2, in this DAG, a vertex will represent a common subsequence between the two genomes (i.e. a sequence formed from the original sequence by deleting some of the elements without changing the relative order and signs of the remaining elements). The complexity of this algorithm will stem from the construction of this graph, which is, as we will see, exponential on the span s of the input genomes, rather than on m.

Suppose, w.l.o.g., $|G_1| \leq |G_2|$. The algorithm is based on two functions : *CommonSubsequenceSet* (cf. Figure 3) and *BuildGraph* (cf. Figure 4). The function *CommonSubsequenceSet* consists in (1) computing – for all the common

```
1 Function CommonSubsequenceSet (G1,G2) {
2 V = ∅; j = 1;
3 for (i = 1; i ≤ |G1|; i = i + s) do
4     foreach common subsequence α_j^i between G1[i..i + s − 1] and G2 do
5         V = V ∪ {v_j^i} such that v_j^i = (α_j^i, s_j^i, e_j^i) where s_j^i (resp. e_j^i) is the
          starting (resp. ending) position of α_j^i in G2;
6     end
7     j + +;
8 end
9 return V }
```

Fig. 3. Function CommonSubsequenceSet over two s-span genomes G_1 and G_2

subsequences between non overlapping segments of G_1 of size s (i.e. $G_1[i..i+s-1]$ for $i = \{1, s+1, 2s+1, \ldots\}$) and G_2 – the starting and ending positions of the common subsequences in G_2 ; and (2) add a vertex represented as a triplet $(\alpha_j^i, s_j^i, e_j^i)$ (formally defined in Figure 3) for each of those common subsequences to a set V. This set will be returned by the function.

Function *BuildGraph* consists in building a DAG $G = (V, A)$ from two s-span genomes G_1 and G_2. The set of vertices V is obtained by calling the *CommonSubsequenceSet* function on G_1 and G_2. *BuildGraph* then considers each pair of vertices $(v_j^i, v_{j+1}^{i'})$ and $(v_j^i, v_{j+2}^{i'})$, and adds an edge to A iff the corresponding common subsequences are compatible, i.e. if they have no gene of the same family in common (exemplar solution) and do not overlap. In the resulting graph (as illustrated in Figure 5), a path is an exemplarization of both G_1 and G_2. In such a graph, if one decomposes each vertex v_j^i into a path of length $|\alpha_j^i|$ then, as we will show, the existence of a path of length m induces that there exists a zero breakpoint distance exemplarization of G_1 and G_2.

Before proving the correctness and complexity of the above algorithm, let us prove an interesting property on the set V returned by *CommonSubsequenceSet*. In the following, δ_j will refer to the set $\{v_j^i | v_j^i \in V\}$.

Lemma 5. *Given two s-span genomes G_1 and G_2, the function CommonSubsequenceSet, ran on (G_1, G_2), returns a set V of size less than or equal to $n2^s s$, where $n = |G_1|$.*

Proof. Given any segment of size s of G_1, there are at most 2^s different subsequences. Moreover, for each such subsequence α_j^i, there are at most s positions in G_2 for the first (resp. the last) gene of α_j^i since G_2 is a s-span genome; namely $\alpha_j^i[1]$ (resp. $\alpha_j^i[|\alpha_j^i|]$). Since in the function *CommonSubsequenceSet*, a vertex is added to V for each $(\alpha_j^i, s_j^i, e_j^i)$, V is of cardinality at most $\frac{|G_1|}{s} 2^s s^2$. □

By construction, given any sequence of vertices along a path in the DAG obtained by $BuildGraph(G_1, G_2)$, the order of the vertices in the path is similar to the one of the corresponding genes in both G_1 and G_2. Moreover, by definition, for

```
1  Function BuildGraph (G1,G2) {
2  V = CommonSubsequenceSet(G1, G2);
3  foreach (v_j^i, v_{j+1}^{i'}) ∈ V^2 do
4      if e_j^i < s_{j+1}^{i'} then
5          //the order is respected;
6          if α_j^i and α_{j+1}^{i'} have no gene of the same family in common then
7              //the concatenation is exemplar;
8              A = A ∪ {(v_j^i, v_{j+1}^{i'})};
9          end
10     end
11 end
12 foreach (v_j^i, v_{j+2}^{i'}) ∈ V^2 do
13     if e_j^i < s_{j+2}^{i'} then
14         //by construction, the concatenation is exemplar;
15         A = A ∪ {(v_j^i, v_{j+2}^{i'})};
16     end
17 end
18 return G = (V, A) }
```

Fig. 4. Function BuildGraph over two s-span genomes G_1 and G_2

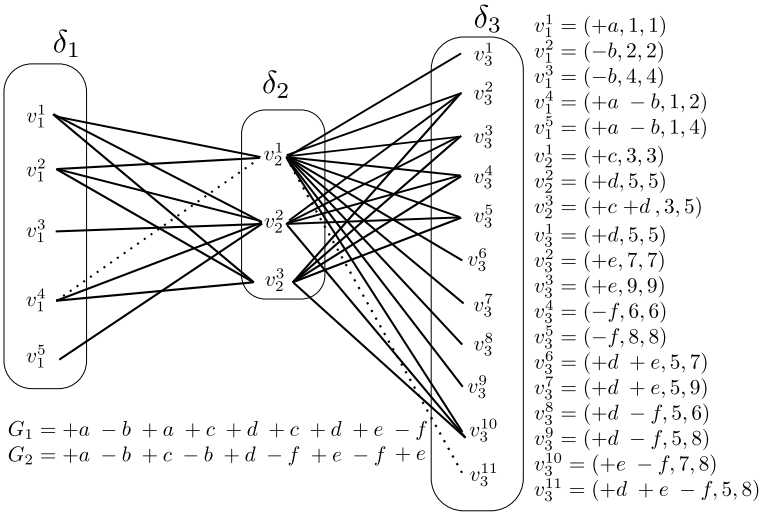


Fig. 5. Resulting graph from $BuildGraph(G_1, G_2)$ with $G_1 = +a\ -b\ +a\ +c\ +d\ +c\ +d\ +e\ -f$ and $G_2 = +a\ -b\ +c\ -b\ +d\ -f\ +e\ -f$. The dotted path corresponds to a positive solution for ZEBD. Links between δ_1 and δ_3 have not been drawn. All arcs are drawn without arrow, but should be understood from left to right.

any j the set of genes involved in any common subsequence of δ_j is disjoint from the one for $\delta_{j'}$ with $j' \geq j+2$. Therefore, a path in the DAG corresponds to an exemplarization of both G_1 and G_2. Hence, for any s-span genome G_1 and G_2, ZEBD is positively answered iff there exists a path of length m in the DAG obtained by $BuildGraph(G_1, G_2)$.

Let us now analyze the time complexity of our algorithm. The function $CommonSubsequenceSet(G_1, G_2)$ has a worst time complexity in $O(n2^s s)$, where n is the length of G_1. Indeed, for each value of i (which is bounded by $\frac{n}{s}$), according to Lemma 5 there are at most 2^s different subsequences in G_1 and s^2 pairs (s_j^i, e_j^i) for any such given subsequence in G_2. Moreover, any set δ_j, $1 \leq j \leq \frac{n}{s}$, is of size at most $2^s s^2$.

Function $BuildGraph(G_1, G_2)$ has a worst time complexity in $O(n2^{2s}s^3)$. Indeed, there are less than $(2^s s^2)^2$ pairs $(v_j^i, v_{j+1}^{i'})$ in V, for each of $1 \leq j \leq \frac{n}{s}$.

Finally, finding a longest path in the resulting DAG from $BuildGraph(G_1, G_2)$ may be done polynomially in the size of the graph. On the whole, the algorithm has a time complexity in $O(n2^{2s}s^3)$.

Altogether, the above results lead to the following theorem.

Theorem 4. *There exists an $O(n2^{2s}s^3)$ algorithm for solving* ZEBD*, where n is the length of the shortest input genome, and s is the span of the input genomes.*

4 Conclusion

In this paper, we have given several results concerning the Exemplar Breakpoint Distance and Zero Exemplar Breakpoint Distance problems. We first proved that that EBD cannot be approximated *at all* as soon as both genomes contain duplicates; this was done by showing that ZEBD(2,2) is **NP**-complete. This last result fills the remaining gaps concerning the knowledge of the complexity landscape of ZEBD. In particular, this answers an open question from Chen et al. [7] and Angibaud et al. [4]. We have also provided two fixed-parameter algorithms for ZEBD: one parameterized by the number of gene families, and the other by their span. Two questions remain open:

(1) since ZEBD$(1, q)$, $q \geq 1$, is polynomial, there may exist approximation algorithms for EBD$(1, q)$ (we recall that EBD$(1, q)$ is **APX**-hard [2]). For instance, is it possible to approximate EBD$(1, q)$ within a *constant* ratio ?

(2) as mentioned by Chen et al. [7], for problems whose optimal value could be equal to zero, it is sometimes better to look for so-called *weak approximations*. The natural question is thus the following: what can be said concerning weak approximations for EBD(p, q) ? Chen et al. [7] gave a negative result in the general case but restricting our study to particular values of p and q could lead to positive results.

References

1. Alon, N., Yuster, R., Zwick, U.: Color coding. Journal of the ACM 42(4), 844–856 (1995)
2. Angibaud, S., Fertin, G., Rusu, I.: On the approximability of comparing genomes with duplicates. In: Nakano, S.-i., Rahman, M. S. (eds.) WALCOM 2008. LNCS, vol. 4921, pp. 34–45. Springer, Heidelberg (2008)

3. Angibaud, S., Fertin, G., Rusu, I., Thévenin, A., Vialette, S.: A pseudo-boolean programming approach for computing the breakpoint distance between two genomes with duplicate genes. In: Tesler, G., Durand, D. (eds.) RECMOB-CG 2007. LNCS (LNBI), vol. 4751, pp. 16–29. Springer, Heidelberg (2007)
4. Angibaud, S., Fertin, G., Rusu, I., Thévenin, A., Vialette, S.: On the approximability of comparing genomes with duplicates. Journal of Graph Algorithms and Applications, Extended version of [2] (2008), http://www.arxiv.org/abs/0806.1103
5. Blin, G., Chauve, C., Fertin, G., Rizzi, R., Vialette, S.: Comparing genomes with duplications: a computational complexity point of view. ACM/IEEE Trans. Computational Biology and Bioinformatics 14(4), 523–534 (2007)
6. Bryant, D.: The complexity of calculating exemplar distances. In: Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment, and the Evolution of Gene Families, pp. 207–212. Kluwer Academic Publisher, Dordrecht (2000)
7. Chen, Z., Fu, B., Zhu, B.: The approximability of the exemplar breakpoint distance problem. In: Cheng, S.-W., Poon, C.K. (eds.) AAIM 2006. LNCS, vol. 4041, pp. 291–302. Springer, Heidelberg (2006)
8. Thach Nguyen, C., Tay, Y.C.: Louxin Zhang. Divide-and-conquer approach for the exemplar breakpoint distance. Bioinformatics 21, 2171–2176 (2005)
9. Sankoff, D.: Genome rearrangement with gene families. Bioinformatics 15(11), 909–917 (1999)

The Minimal Manhattan Network Problem in Three Dimensions

Xavier Muñoz[1,⋆], Sebastian Seibert[2,⋆⋆], and Walter Unger[3,⋆⋆⋆]

[1] UPC. Matemàtica Aplicada 4. E-08034 Barcelona
xml@ma4.upc.edu
[2] ETH Zürich, Department Informatik, ETH Zentrum, CH-8092 Zürich
sseibert@inf.ethz.ch
[3] RWTH Aachen, Lehrstuhl für Informatik I, D-52056 Aachen
quax@cs.rwth-aachen.de

Abstract. For the Minimal Manhattan Network Problem in three dimensions (MMN3D), one is given a set of points in space, and an admissible solution is an axis-parallel network that connects every pair of points by a shortest path under L_1-norm (Manhattan metric). The goal is to minimize the overall length of the network.

Here, we show that MMN3D is $\mathcal{NP}$- and $\mathcal{APX}$-hard, with a lower bound on the approximability of $1 + 2 \cdot 10^{-5}$.

This lower bound applies to MMN2-3D already, a sub-problem in between the two and three dimensional case. For MMN2-3D, we also develop a 3-approximation algorithm which is the first algorithm for the Minimal Manhattan Network Problem in three dimensions at all.

1 Introduction

The Minimal Manhattan Network Problem (MMN) has recently attracted some attention. Originally, it was stated in two dimensions, but it generalizes naturally.

One is given a finite set of n points in (k-dimensional) space, and as distance measure the L_1-norm is used, also called Manhattan metric. A *rectilinear path* is a path that consists solely of axis-parallel line segments. A shortest rectilinear path is called a *Manhattan path*. A Manhattan path connecting two points $p_1 = (x_1, y_1, z_1)$ and $p_2 = (x_2, y_2, z_2)$ will always be a (k-dimensional) staircase, and its length is the L_1-distance between p_1 and p_2, i.e. $|x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2|$ (in the three-dimensional case). The task is to construct a network that connects each pair of points by a Manhattan path. The goal is to minimize the overall length of the network.

⋆ Partially supported by the Ministerio de Educación y Ciencia, Spain, and the European Regional Development Fund under project TEC2005-03575 and by the Catalan Research Council under project 2005SGR00256."

⋆⋆ Partially supported by SNF grant 200021-109252/1.

⋆⋆⋆ Partially supported by the EU within the 6th Framework Programme under contract 001907 (DELIS).

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 369–380, 2009.

This problem is part of a wide range of network problems with applications in VLSI design and related areas. In general, given a set of points in a plane or in a higher dimensional space, one is asked for a minimal cost network of a certain type. Steiner trees and spanners are prominent examples of such networks, and the MMN can be seen as a special case of finding a spanner. The problem of finding cheap spanners has numerous applications, and it has been widely studied, especially metric subproblems [1,4,5,7,10].

In a recent sequence of papers [9,11,2,6,12] approximation algorithms for the two-dimensional case (MMN2D) have been studied, where 1.5 is the best approximation ratio achieved so far[1]. One has to note, however, that there is still the possibility that the MMN2D is solvable optimally in polynomial time.

A generalized version of the MMN3D is considered in [8]. For a given set P of points, a transitive relation $R \subseteq P \times P$, and a length bound $B > 0$, it is shown to be NP-hard to decide whether there is a network that connects all pairs $\{p, q\} \in R$ by Manhattan paths and has total length $\leq B$.

Here, we give the first hardness proof for MMN3D by showing that at least in three dimensions, the original problem is $\mathcal{NP}$ hard. Moreover, we show that there is no polynomial time approximation algorithm with a ratio better than $1 + 2 \cdot 10^{-5}$, unless $\mathcal{P} = \mathcal{NP}$, and consequently no approximation scheme exists. This proof applies in fact to a sub-case already, defined as MMN2-3D below. Here, the points are distributed in three dimensions, but the critical connections need to be made in (multiple) planes only.

Also, we develop a first approximation algorithms for three dimensions, in that we get a 3-approximation[2] for MMN2-3D. Thus, MMN2-3D presents the first case of the Minimal Manhattan Network Problem where both a lower bound and an approximation algorithm are known.

The remainder of the paper is structured as follows: in the next section, a few preliminaries and basic observations will be stated. In Section 3, we describe how an MMN2-3D instance is constructed for a given SAT formula. This will be used in the subsequent section to derive the claimed lower bound on the approximability of MMN2-3D, which includes $\mathcal{NP}$-hardness. Afterwards, we show how to obtain approximation algorithms for MMN2-3D.

2 Preliminaries

Let P be the set of points given. We will consider positive integer valued coordinates only, so we can identify each point p_i with a triple (x_i, y_i, z_i) of positive integers. A solution S is given as a set of line segments.

Next, we want to transfer an observation that is well known from the two-dimensional case.

Each pair $\{p, q\}$ of points spans a cuboid $C(p, q)$ inside which (including the surface) all possible shortest Manhattan paths between those two points lie. Here,

[1] There exists an unpublished 1.25-approximation algorithm on MMN2D [13].
[2] This becomes a 2.5-approximation with the result from [13].

we need to consider only cuboids with axis parallel sides, so in the following, by "cuboid" we always mean a cuboid with axis parallel sides, and the same holds for planes.

Assume that there exists another point r inside $C(p,q)$ (again including the surface). Any admissible solution needs to contain shortest paths from p to r, and from r to q. Those two paths together form a shortest one from p to q already. Consequently, the pair $\{p,q\}$ is not critical for the correctness of a solution.

If we look at it the other way round, we call a pair $\{p,q\}$ and its cuboid *critical* if the cuboid contains no other point besides p and q.

Remark 1. A solution S is admissible for P iff, for every critical pair $\{p,q\}$ in P, a shortest path from p to q is contained in S.

Easy though this observation is, one has to keep it always in mind. Because of it, in the following, for each point we need to discuss its connections with few selected others, only. Most of the other points will not form a critical pair with the currently considered one. They lay "behind" other points (in Manhattan metric).

Another rather obvious fact is that, if the cuboid $C(p,q)$ degenerates into a line segment, then that line segment is a necessary part of every admissible solution. This will be used to construct certain fixed parts of the solutions, and to separate other parts.

One feature of our construction will be that all critical cuboids will be in fact rectangles. That is, for each point we only have to consider the axis parallel planes it belongs to in order to see which connecting line segments it needs. This greatly facilitates arguing which amount of lines segments is needed and sufficient, respectively. Also this gives rise to the definition of MMN2-3D as that sub-case of MMN3D where all critical cuboids are just rectangles. In other words, the points are spread in three dimensions but the critical connections are always two-dimensional, hence the name.

3 Construction of Hard Instances

Our proof is going to be a reduction from 3SAT, that is, we have to construct, for a given formula, a corresponding set of points as an MMN2-3D instance.

We assume that we are given a formula $\Phi = \phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_m$ with clauses $\phi_i = l_{i,1} \vee l_{i,2} \vee l_{i,3}$ over the variable set $x_1, \ldots, x_k$.

The point set P to be constructed will be contained in a large cuboid C_0 all of whose corners are in P. In fact, most of the points will lie on the surface of C_0, giving some sort of "cage" inside which the essential decisions about placing line segments take place. In Figure 1, an example is given for the formula $(x \vee \overline{y}) \wedge (y \vee z) \wedge (\overline{x} \vee \overline{z})$. Note that for readability, it is not true to scale. We can see a long sequence of upright plane segments. For each variable, there will be a subsequence of such "frames". The frames with five vertical lines are separators between the variables. For each clause, we will use a horizontal plane to place the respective points on it.

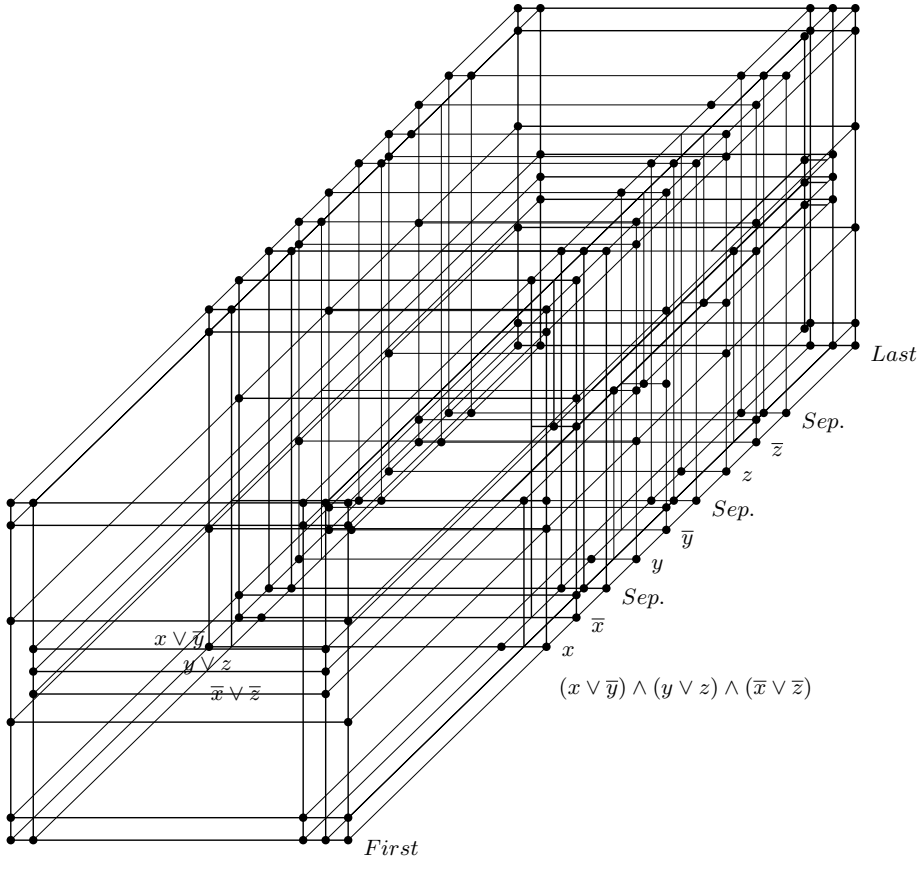


Fig. 1. Overview

We begin by describing the vertical frames first. More precisely, we construct subsets of points with identical z-coordinate, and we analyze the necessary and optional line segments between them.

A separator frame is shown in Figure 2 (a). It causes a fixed grid of necessary lines, as do the last and first frame shown in Figure 2 (b) and (c), respectively. Please note that in Figure 2 (c), the points $k_1, k_2, \ldots, k_m$ are drawn as open circles. This will always indicate that the respective points are not in the plane which is depicted by the current figure. Here, the points $k_1, k_2, \ldots, k_m$ are lying slightly (distance 1) in front of the last frame. They are shown in this figure for later orientation only.

Note that in the figures some large distances are used, annotated as b and b', and later a and a'. We will set concrete values for these later, when we calculate a lower bound on approximability. For lines that are drawn close to each other in the figures, we generally assume the distance between them to be 1, if not stated differently.

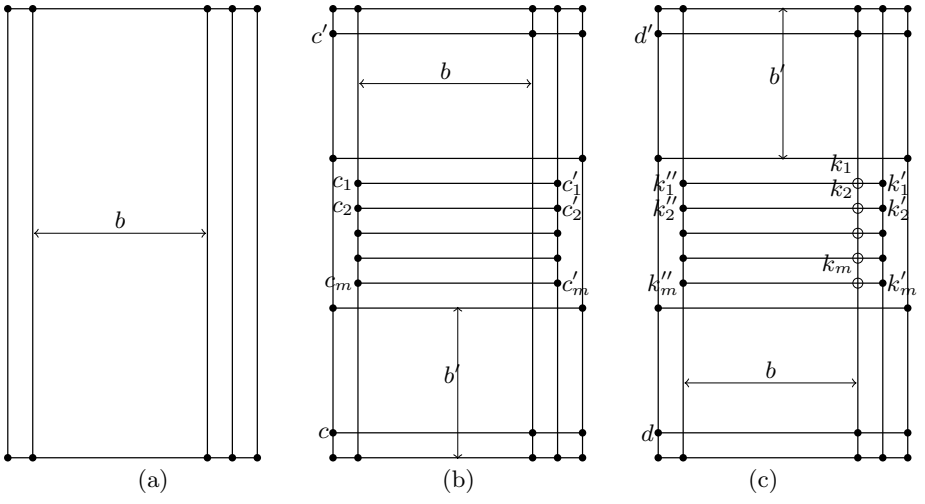


Fig. 2. Separator, first, and last frame

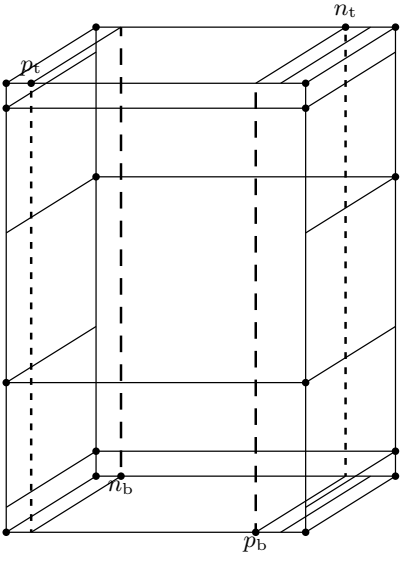


Fig. 3. Variable setting

For each variable x_j, we construct an even number of frames. Those for positive and negative occurrences of x_j are constructed alternatingly. If there are more positive or more negative occurrences, some of the frames will not be used in any clause, but this situation does not appear in the formulas we will use in the reduction in Section 4.

Let us look at a pair of positive and negative occurrence, as depicted in Figure 3.

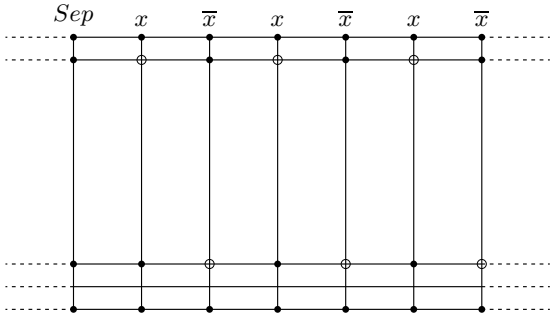


Fig. 4. Variable block from above

The solidly drawn lines on the outside of the cuboid are necessary lines (partly caused by the connections from first to last frame). The only point where there is some choice is in connecting $p_{\mathrm{t}}, p_{\mathrm{b}}, n_{\mathrm{t}}$, and n_{b} with each other. Here, two vertical lines are sufficient, but only if either both dotted or both dashed lines are chosen. This is the essential mechanism to represent the setting of variable values. We will interpret the variable to be set to true, iff in the frame for a positive occurrences, the left vertical line is used, the dotted choice in the figure. Note that the horizontal lines shown in the figure could be used to obtain a solution "mixed from dotted and dashed lines". But we will see below that this would make things only more expensive.

Now, in general, each variable occurs more than once each, positive and negative. But then, the above pattern is just repeated. In the end, the complete block for one variable, including one separator, looks from above as shown in Figure 4.

Next, since a frame for a variable is used for an occurrence of that variable in a certain clause, we will add few more points to it, the essential ones for the interaction with that clause. The case of negative and positive occurrences are shown in Figure 5 (a) and (b), respectively. In the following, we describe the case of a negative occurrence, since the other one is just a mirrored version.

For each clause $\phi_i = l_{i,1} \vee l_{i,2} \vee l_{i,3}$, a horizontal plane is reserved. At exactly that level, in the frame for the occurrence, a point $l'_{i,j}$ is added as shown in Figure 5 (a). And as its counterpart, another point $l_{i,j}$ is added on the left side, at distance 1 from the bottom.

Also, a point exactly above $l'_{i,j}$ is added at the top level, which fixes the upward connections from $l'_{i,j}$. And a point opposite $l_{i,j}$ on the right hand side generates a solid line between itself and $l_{i,j}$.

The focus however lies on the connection from $l'_{i,j}$ to $l_{i,j}$. Here, potentially an expensive additional line segment is needed. If the right vertical line is chosen in the variable assignment, this already produces a connection from $l'_{i,j}$ to $l_{i,j}$. Otherwise, an additional segment is needed, and by choosing b to be smaller than b', we ascertain that the better choice is to use the horizontal segment from $l'_{i,j}$ to the left.

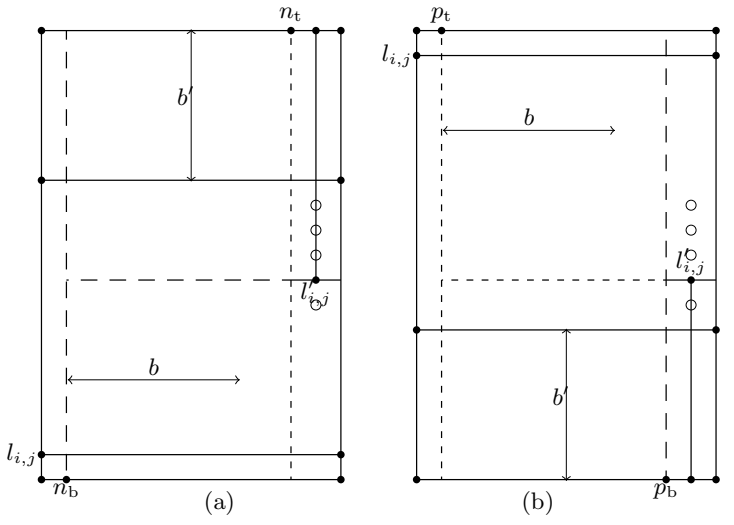


Fig. 5. A negative and a positive occurrence

The important point is that only in the latter case, a line segment in the plane of the clause is added. The intended interpretation is that this corresponds to the clause being satisfied by the variable occurrence under consideration.

Note that in the end, we will apply our construction to formulas where every variable is used exactly twice positive and exactly twice negative. Then, every variable setting will cause in two of the four frames for that variable the addition of a horizontal line segment as just described.

Now, we consider the mixed use of dotted and dashed lines. That would mean, here, to go down from n_t, then move to left, and down to n_b. But then, in *every* frame for this variable, the dashed horizontal line would be needed to connect $l'_{i,j}$ to $l_{i,j}$, not only in every other frame. For each such dashed line, we might save a segment of the same length in a clause plane, as we will see later, but not more. So, since we cannot gain anything by it, we will rule out those "mixed choices" in the following.[3]

Before we move on to the clauses, we have to consider the following. Different points $l'_{i,j}$ and $l_{i,j}$ from different occurrences (even from different variables) might form critical pairs. With respect to the $l_{i,j}$ this is no problem. All of them lie on two lines behind each other, surrounded by c, c', d, and d' on the first and last frame. Thus, they are connected by necessary lines. Also, for $(i,j) \neq (i',j')$, the cuboid $C(l'_{i,j}, l_{i',j'})$ is not critical since it contains $l_{i,j}$.

The points $l'_{i,j}$ and $l'_{i',j'}$ however are on different horizontal planes. In Figure 5, the potential positions of other $l'_{i',j'}$ are shown as open circles. Now, the points

[3] Note that using the horizontal line at distance 1 from the bottom could be of no use at all: the line segment of length 1 from n_b upwards would help for no other connection. This is essentially the same as drawing the line from n_t downwards all the way.

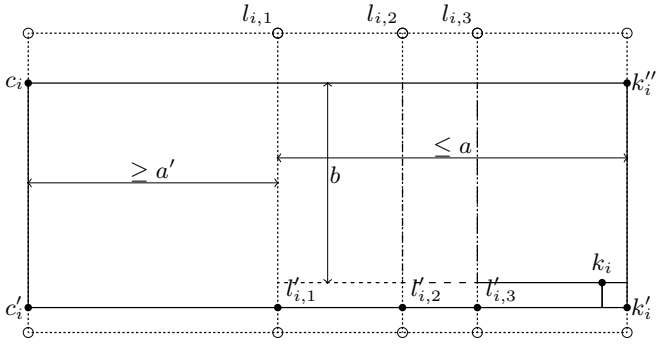


Fig. 6. A clause

$c'_1, \ldots, s'_m$ and $k'_1, \ldots, k'_m$ on the first and last frame, respectively, guarantee that there are connections in z-direction. What is missing to connect different $l'_{i,j}$ can only be small vertical segments. And here, by choosing the distance between clause planes to be just 1, we make sure that all of these small pieces together will not affect the overall optimum by much.

Finally, let us look at the horizontal plane for a single clause, see Figure 6. The only point we did not discuss before is k_i. First, we note that $k_1, \ldots, k_m$ lie above each other. Also at the bottom most level, and at distance 1 above the bottom (the plane of the $l_{i,j}$), we add points below the k_i, as we do at the top most level and at distance 1 below the top. That way, all we have to consider in the following are the connections that k_i needs in its own horizontal plane.

The distance between k_i and k'_i is just 1 in both coordinates, so that we need not to worry about where exactly the connections to the right and front will be done. The only essential connection is from k_i to c_i. Here, it is important to note that we chose the distance a' to be much larger than a (and a' is slightly larger than b). By a, we mean the distance between the last frame and the first one which represents any variable occurrence. In other words, there is the huge distance a' between the first frame and the next one, and then all other frames are close to each other.

The effect of this is that the connection from k_i to c_i is cheap, if one of the variable occurrences has a connection from $l'_{i,j}$ towards $l_{i,j}$ in the current plane. Then, only the connection from k_i to the left is needed, costing at most a. Otherwise, a connection from k_i to the line connecting c_i and k''_i is needed, costing at least b.

4 The Lower Bound on Approximability

In order to obtain a lower bound, we use a result by Berman, Karpinski, and Scott about a SAT variant where each variable occurs only a limited number of times. To this end, let (3,B2)-SAT be the restriction of SAT to inputs where each clause contains exactly 3 variables, and each variable occurs exactly twice positive and twice negative. MAX-(3,B2)-SAT is the corresponding maximization problem.

Theorem 1. *[3] There exists a family of MAX-(3,B2)-SAT instances, containing* $1016n$ *clauses for some* n*, such that it is* $\mathcal{NP}$*-hard to distinguish between instances where* $(1016-\varepsilon)n$ *clauses can be satisfied and those where at most* $(1015+\varepsilon)n$ *clauses can be satisfied (for arbitrary small* $\varepsilon < \frac{1}{2}$*).* □

We are going to show that, using our construction from the previous section, we can proof now the following theorem.

Theorem 2. *There exists a family of MMN2-3D instances, containing* $194l+4$ *points for some* l*, such that it is* $\mathcal{NP}$*-hard to distinguish between instances where a solution of cost less than* $(48260+\delta)L$ *exists and those where every solution costs at least* $(48261-\delta)L$ *(for* $L = \frac{l^4}{254}$ *and arbitrary small* $\delta < \frac{1}{2}$*).*

Proof. Given a MAX-(3,B2)-SAT instance Φ with $1016n$ clauses, we apply our construction from the previous section and obtain a point set P as an MMN2-3D instance. We have to show that, if there is a variable assignment satisfying $(1016-\varepsilon)n$ clauses, we obtain a point set P with a Manhattan Network of cost at most $(48260+\delta)L$. And if at most $(1015+\varepsilon)n$ clauses of Φ can be satisfied at the same time, every Manhattan Network for P will cost at least $(48261-\delta)L$.

What we have seen already, when constructing P, is that there are a lot of necessary lines in any Manhattan Network for P. And if we want a minimum cost solution for P, only a few choices remain. First, we have some choices of vertical line segments in the variable frames, and we have seen that these influence how expensive the additional line segments needed in the clause planes will be.

Now, one could think it a good idea to "cheat" in the variable frames in order to save a lot of cost for the clauses. That is, one could consider investing a little bit of vertical lines in the variable frames in order to switch in the middle of a variable block from one choice to the other. Then, some of the variable occurrences would represent the variable assignment to true while other would correspond to the assignment to false. But here is the respect where the choice of formulas from MAX-(3,B2)-SAT comes into play. Every variable has just four occurrences, two positive and two negative, and the frames for those alternate in our construction. If we add the consideration that any optimal assignment will satisfy at least two of the four clauses a variable occurs in, we see that by the described way of cheating at most one more "satisfied clause" can be gained. More precisely, we might save a line segment of length b in the clause plane, but we pay with a vertical line segment of length at least b' in a variable frame. Since we will choose $b' > b$, this would result in a loss overall.

So, what we have is that a minimal Manhattan Network always represents a variable assignment, and for those clauses satisfied by this assignment, only line segments of cost at most a need to be added in the corresponding plane (besides the necessary segments). For the non-satisfied clauses however, we need additional line segments of cost at least b.

What remains is to calculate the resulting cost of a minimal solution, depending on the number of satisfiable clauses. Any (3,B2)-SAT formula has, for a natural number l, exactly $m = 4l$ clauses and $3l$ variables who occur four times

each. Depending on l, we get the dimensions left open so far in our construction. a is just the number of frames that follow at distance 1 from each other before the last (i.e. all but the first frame) which is $15l + 1$. We choose $b = l^3$ and $a' = b' = l^3 + 1$.

From this, it is easy to calculate the length of all the necessary line segments. Let us look closer at the few places with a choice. In a frame for a variable occurrence (Figure 5), we always have four necessary horizontal lines of full width of cost $b + 3 = l^3 + 3$ each. Likewise, three full height vertical lines are necessary, if we include the one with optional placement (dotted or dashed). Each costs $2b' + m + 1 = 2l^3 + 4l + 3$. We have argued that in a minimal solution, every other frame for a variable occurrence needs the dashed horizontal line of cost b, so we can count $1/2b$ per frame. The short horizontal segment through $l'_{i,j}$ of length 2 is needed in any case. Some uncertainty comes in here only with respect to the vertical connections of $l'_{i,j}$. The part of length b' above the necessary horizontal line is always present, but how long the part below is, is uncertain. Not only because of the horizontal plane on which $l'_{i,j}$ lies, but also because we do not know how much below $l'_{i,j}$ will be needed for connecting to other points $l'_{i',j'}$. But remember that the horizontal connections to those $l'_{i',j'}$ are always there, so we need to worry about the vertical part only. And this will have a short length, namely between 1 and m.

The largest uncertainty occurs in the clause planes, see Figure 6. If the line placement in the variable frames corresponds to an assignment satisfying the current clause, only a left to right connection of length $\leq a$ is needed besides the necessary line segments. Otherwise, we know that a connection of length b is needed (in addition to the length 1 pieces from k_i to the front and right).

If all is added up, we get for the whole construction a necessary cost of an optimal solution of $190l^4 + O(l^3)$, and there are flexible costs depending mainly on the number α of non-satisfied clauses (by an optimal assignment), which are between $16l + \alpha(l^3 + 1)$ and $108l^2 + 4l + \alpha(l^3 + 15l + 1)$. Here, we see why we used l^3 as the main dimension. That way all the small pieces we are uncertain about are one order smaller and can be "neglected". (Remember that α is at most in the order of l.)

More precisely, we fix a small $\varepsilon' > 0$, and we have that, for sufficiently large l, the costs of an optimal solution are between $190l^4 + \alpha l^3$ and $190l^4 + \alpha l^3 + \varepsilon' l^4$.

Now, we apply this to the formulas from Theorem 1. There, we have $m = 1016n$, i.e., $l = 254n$.

If at least $(1016 - \varepsilon)n$ clauses can be satisfied ($\alpha \leq \varepsilon n$), the corresponding minimal Manhattan Network costs at most

$$190l^4 + \varepsilon' l^4 + \alpha l^3 \leq ((190 + \varepsilon')254n + \varepsilon n)l^3 = (48260 + \delta)L,$$

if we set $\delta = 254\varepsilon' + \varepsilon$ and $L = nl^3 = \frac{l^4}{254}$.

If at most $(1015+\varepsilon)n$ clauses can be satisfied ($\alpha \geq (1-\varepsilon)n$), the corresponding minimal Manhattan Network costs at least

$$190l^4 + \alpha l^3 \geq (190 \cdot 254n + (1-\varepsilon)n)l^3 \geq (48261 - \delta)L. \qquad \square$$

The result from Theorem 2 implies not only $\mathcal{NP}$-hardness of MMN2-3D and MMN3D but also the nonexistence of a polynomial time approximation scheme.

Corollary 1. *For the three-dimensional Minimum Manhattan Network problem, there exists no polynomial time approximation algorithm with a ratio of* $1 + 2 \cdot 10^{-5}$ *or better, unless* $\mathcal{P} = \mathcal{NP}$. □

5 Approximation Algorithms in Three Dimensions

Let us show how any approximation algorithm for MMN2D can be used for MMN2-3D. With the result from [12], the following gives a 3-approximation algorithm on MMN2-3D.

Theorem 3. *For every polynomial-time* α*-approximation algorithm on MMN2D, there is a polynomial-time* 2α*-approximation algorithm on MMN2-3D.*

Proof. The idea is simply to run the given algorithm on any axis-parallel plane on which at least two points lie, and to put together all resulting partial solution.

Since all critical cuboids are assumed to be rectangles, this will produce a valid solution. Obviously, the running time grows at most by $O(n)$, compared to the given algorithm: Each point belongs to three axis-parallel planes, thus at most $\frac{3}{2}n$ planes need to be considered.

For the approximation ratio, we consider an optimal solution S_{opt}. In every considered plane E, the intersection S_E of S_{opt} and E gives an admissible solution. Thus, the algorithm's solution A_E in that plane is at most a factor of α more expensive than S_E. If we put together all A_E, these may at worst be disjoint, while in putting together all S_E, every line from S_{opt} is counted at most twice (it belongs to two axis-parallel planes). Thus, the sum of the A_E costs is at most 2α times the cost of S_{opt}. □

We note, that there is already a 2.5-approximation algorithm on MMN2-3D, because there exists a unpublished 1.25-approximation algorithm on MMN2D [13]. This new approximation algorithm is submitted in parallel to this paper.

6 Conclusion

In this paper, we have given a hardness proof for the Minimum Manhattan Network problem, including a first lower bound on approximability. Also, we have introduced a first approximation algorithm for a three dimensional subcase.

It is certainly desirable to transfer the lower bound to MMN2D. This, however, seems rather unlikely if one analyses the construction. There seems to be no way to combine the effects of independent choices (for the variables) like we did in the clause planes, if one is restricted to two dimensions. The problem is that the demand, that *all* pairs of points need to be connected by a shortest path,

limits the ability to let different information (the effects of placement choices) pass each other independently in one plane.

On the other hand, approximation algorithms for MMN3D are needed. We expect that at least the algorithm from [9] will be adaptable to three dimensions, though not trivially (and the ratio can be at most 8 since one has to deal with octants instead of quadrants). Another approach would be to reduce MMN3D to MMN2-3D by introducing auxiliary points. However, it is not clear how to bound the cost incurred by this by some multiple of the optimum.

References

1. Althöfer, I., Das, G., Dobkin, D., Joseph, D., Soares, J.: On Sparse Spanners of Weighted Graphs. Discrete Comput. Geom. 9, 81–100 (1993)
2. Benkert, M., Shirabe, T., Widmann, F., Wolff, A.: The Minimum Manhattan Network Problem—Approximations and Exact Solution. Computational Geometry: Theory and Applications 35(3), 188–208 (2006)
3. Berman, P., Karpinski, M., Scott, A.D.: Approximation Hardness of Short Symmetric Instances of MAX-3SAT Electronic Colloquium on Computational Complexity Report No. 49 (2003), http://eccc.hpi-web.de/eccc-reports/2003/TR03-049/
4. Chandra, B., Das, G., Narasimhan, G., Soares, J.: New Sparseness Results on Graph Spanners. Internat. J. Comput. Geom. Appl. 5, 125–144 (1995)
5. Chen, D., Das, G., Smid, M.: Lower bounds for computing geometric spanners and approximate shortest paths. Discrete Applied Math. 110, 151–167 (2001)
6. Chepoi, V., Nouioua, K., Vaxès, Y.: A rounding algorithm for approximating minimum Manhattan networks. Theoret. Comp. Sci. 390, 56–69 (2008)
7. Das, G., Narasimhan, G.: A Fast Algorithm for Constructing Sparse Euclidian Spanners. Internat. J. Comput. Geom. Appl. 7, 297–315 (1997)
8. Engels, B.: The Transitive Minimum Manhattan Subnetwork Problem in 3 Dimensions. Submitted to Discrete Applied Mathematics: Proceedings of the 6th Cologne Twente Workshop 2007 (November 12, 2007)
9. Gudmundsson, J., Levcopoulos, C., Narasimhan, G.: Approximating a Minimum Manhattan Network. Nordic J. Computing 8, 219–232 (2001)
10. Gudmundsson, J., Levcopoulos, C., Narasimhan, G.: Fast Greedy Algorithms for Constructing Sparse Geometric Spanners. SIAM J. Computing 31, 1479–1500 (2002)
11. Kato, R., Imai, K., Asano, T.: An improved algorithm for the minimum manhattan network problem. In: Bose, P., Morin, P. (eds.) ISAAC 2002. LNCS, vol. 2518, pp. 344–356. Springer, Heidelberg (2002)
12. Seibert, S., Unger, W.: A 1.5-approximation of the minimal manhattan network problem. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 246–255. Springer, Heidelberg (2005)
13. Seibert, S., Unger, W.: Refined Analysis of the Minimal Manhattan Network Problem and a 1.25 Approximation (submitted for publication)

Shape Matching by Random Sampling*

Helmut Alt and Ludmila Scharf

Institute of Computer Science, Freie Universität Berlin
{alt,scharf}@mi.fu-berlin.de

Abstract. In order to determine the similarity between two planar shapes, which is an important problem in computer vision and pattern recognition, it is necessary to first match the two shapes as good as possible. As sets of allowed transformation to match shapes we consider translations, rigid motions, and similarities. We present a generic probabilistic algorithm based on random sampling for matching shapes which are modelled by sets of curves. The algorithm is applicable to the three considered classes of transformations. We analyze which similarity measure is optimized by the algorithm and give rigorous bounds on the number of samples necessary to get a prespecified approximation to the optimal match within a prespecified probability.

1 Introduction

Matching two geometric shapes under transformations and evaluating their similarity is one of the central problems in computer vision systems where the evaluation of the resemblance of two images is based on their geometric shape and not color or texture. Because of its significance the problem has been widely covered in the literature, see [4,15] for surveys.

We assume that the shapes are modeled by sets of plane curves. As possible classes of transformations we will consider *translations*, *rigid motions* (rotation and translation) and *similarities* (rotation, scaling, and translation). Our objective is to develop an algorithm which allows an efficient implementation and whose result comes close to human similarity perception.

Several similarity measures and algorithms are known to match two curves, especially polygonal curves. One of the most widely used similarity measures is the Hausdorff distance which is defined for any two compact sets A and B. Alt et al. describe in [2,4] efficient algorithms for computing the Hausdorff distance and minimizing it under translations and rigid motions for arbitrary sets of line segments. One of the drawbacks of the Hausdorff distance is that it is very sensitive to noise. A few similarity measures are defined for pairs of curves, which capture the relative course of two curves: Fréchet distance [3], turning function distance [7], and dynamic time warping distance [12]. There are only few generalizations of those distances to sets of curves, see [5,14,19].

* This research was supported by the European Union under contract No. IST-511572-2, Project PROFI, and by the Priority Programme 1307 "Algorithm Engineering" of Deutsche Forschungsgemeinschaft (DFG).

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 381–393, 2009.

The method we introduce is close to an intuitive notion of "matching", i.e., find one or more candidates for the best transformations, that when applied to the shape B map the most similar parts of the two shapes to each other. The major idea is to take random samples of points from both shapes and give a "vote" for that transformation (translation, rigid motion, or similarity) matching one sample with the other. If that experiment is repeated frequently, we obtain by the votes a certain probability distribution in the space of transformations. Maxima of this distribution indicate which transformations give the best match between the two shapes. The matching step of our algorithm is, therefore, a voting scheme. The idea of random sampling for geometric problems with an analysis similar to ours has been used in a more general context by Cheong et al. in [10] and a similar random sampling method for symmetry detection in 3D shapes with a different clustering method was described by Mitra et al. in [16].

Related methods in the image processing community are the generalized Hough transform, also called pose clustering [1,18], the Radon transform [20] and the RANSAC algorithm [13]. In contrast to those methods we do not consider a discrete set of features that describe shapes, but work with continuous curves. Our method is independent of the choice of parameterization and discretization grid in transformation space. In addition, we give rigorous bounds on the runtime (number of experiments) necessary to obtain the optimal match within a certain approximation factor with a prespecified probability. We consider this as the major contribution of this paper, the analysis leads to a better understanding of this kind of heuristic techniques. In fact, our algorithm is not meant to be directly applied to shape comparison problems arising in practice. For such purposes it makes sense to modify our technique and enhance it with heuristic methods, which we did (see [6]) within a shape retrieval system developed in the EU-funded project PROFI. Its major application, in cooperation with the industrial PROFI-partner Thomson-Compumark in Antwerp, is to identify (illegal) similarities between new *trademark* designs and existing trademarks of various companies in a large trademark database.

2 The Probabilistic Algorithm

We assume that *shapes* are modelled by finite sets of rectifiable curves in the plane, and that for each curve a random point under uniform distribution can be generated in constant time. This is the case for line segments, which would be the most common representation in practice, but also for curves for which a natural parameterization is explicitly given.

Given two shapes $A, B \subset \mathbb{R}^2$, a class of allowed transformations $\mathcal{T}$ and a certain parameter δ, we want to find a transformation $t \in \mathcal{T}$ which lets $t(B)$ *match best* A within a tolerance of δ. The definition of what exactly a good match means is given in section 3. We follow an intuitive notion of a "good match": two shapes are similar if they can be mapped to each other in such a way that large parts of them are close. We assume that the underlying metric in the plane is a piecewise algebraic function, e.g., an L_p metric.

Depending on the class of allowed transformations the algorithm generates ordered *random samples* S_A, S_B of the shapes A and B of appropriate size so that there is a unique transformation mapping S_B to S_A. For example, the samples consist of one point for translations and of two points for similarity maps. We denote by $S_B \xrightarrow[\delta]{t} S_A$ the fact that transformation t maps every element of S_B into the δ-neighborhood of the corresponding element of S_A. For a pair of samples S_A, S_B we define the corresponding *δ-region* in the space of transformations as the set of transformations t such that $S_B \xrightarrow[\delta]{t} S_A$.

The idea of the *probabilistic approach* is quite simple. We describe the algorithm in a generic way and give the details for different classes of transformations below:

1. Take random samples S_A from A and S_B from B and record the corresponding δ-region.
2. Repeat this experiment many times, say N.
3. Return the points of $\mathcal{T}$ covered by the largest number of δ-regions as candidates for good transformations.

The idea behind this algorithm is, that the transformations, that map large parts of the two shapes close to each other, should receive significantly more votes than others.

The size of a random sample within one experiment and the shape and the size of a δ-region depend on the class of transformations allowed:

For **translations** a random sample consists of a single randomly selected point of each shape, $S_A = a \in A$ and $S_B = b \in B$. Two points determine uniquely a translation mapping one point to the other. The transformation space is two-dimensional and a δ-region in translation space corresponding to a sample pair (a, b) is a δ-neighborhood of the translation vector $t = a - b$ with respect to the same metric as used for points in image space.

In case of **rigid motions** the transformation space is three-dimensional. A rigid motion $t = (\alpha, v_x, v_y)$ is defined by a rotation angle α and a translation $v = (v_x, v_y)$ and maps a point $b \in \mathbb{R}^2$ to the point $t(b) = Mb + v$, where $M = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} = \begin{pmatrix} m_1 & -m_2 \\ m_2 & m_1 \end{pmatrix}$ is the rotation matrix. For computational reasons we consider the four dimensional parameterization by (m_1, m_2, v_x, v_y) and restrict it to a three dimensional variety by the constraint $m_1^2 + m_2^2 = 1$, i.e., $\det(M) = 1$. Observe that for two points a, b for every rotation angle α there exists a unique translation vector v_α, such that the rigid motion $t = (\alpha, v_\alpha)$ maps b to a. For four points a_1, a_2, b_1, b_2 there is, in general, no rigid motion that maps b_1 to a_1 and b_2 to a_2. Therefore, we use a single random point of each shape $a \in A$ and $b \in B$ as a sample in one random experiment and record the δ-region $\{(M, v) \mid dist(M \cdot b + v, a) \leq \delta\}$, where all matrices M of the form given above are allowed.

For **similarity maps** the transformation space is four-dimensional. A similarity map $t = (\alpha, k, v_x, v_y)$ is defined by a rotation angle α, a scaling factor k, and a translation vector $v = (v_x, v_y)$. t maps a point $b \in \mathbb{R}^2$ to a point $t(b) = Mb + v$, where $M = \begin{pmatrix} k\cos\alpha & -k\sin\alpha \\ k\sin\alpha & k\cos\alpha \end{pmatrix} = \begin{pmatrix} m_1 & -m_2 \\ m_2 & m_1 \end{pmatrix}$.

A random sample from the shapes contains two points $S_A = (a_1, a_2)$ of A, and two points $S_B = (b_1, b_2)$ of B, which determine a unique similarity transformation t mapping b_1 to a_1 and b_2 to a_2. Although a standard way to parameterize the space of similarity transformations is by (α, k, v_x, v_y), for computational reasons it is more convenient to use the parameterization (m_1, m_2, v_x, v_y) where $m_1 = k\cos\alpha$ and $m_2 = k\sin\alpha$. For general L_p metric a δ-region is then bounded by algebraic surfaces, and for the L_1 and L_∞ metrics it is a convex polytope bounded by four pairs of parallel hyperplanes.

3 Analysis

3.1 Hit Probability in Transformation Space

In this section we analyse the measure of resemblance optimized by the algorithm and bound the number of experiments needed to get an ε-approximation of the maximum of that measure.

First we introduce some formal notation and definitions. Let Ω denote the sample space, i.e., the set of all sample pairs (S_A, S_B). By the definition of our random experiment, the samples of two shapes are drawn independently and uniformly, therefore, we have a uniform distribution on Ω.

Let $\mathcal{T} \subset \mathbb{R}^d$ denote the d-dimensional transformation space. We define a function $p_\delta : \mathcal{T} \to \mathbb{R}$ as the probability that a transformation vector t is covered by a δ-region corresponding to a randomly selected sample. We will call $p_\delta(t)$ the *hit probability* of transformation t. The set of samples yielding a δ-region that covers a transformation t is $M_\delta(t) = \{(S_A, S_B) \in \Omega | S_B \xrightarrow[\delta]{t} S_A\}$, and $p_\delta(t) = \frac{|M_\delta(t)|}{|\Omega|}$, where $|\cdot|$ denotes the Lebesgue measure. Consequently, we have

Remark 1. The hit probability $p_\delta(t)$ in the transformation space induced by the generic algorithm described in Section 2 has its maximum at the transformation maximizing the Lebesgue measure of the set $M_\delta(t)$ defined as

$$M_\delta(t) = \{(S_A, S_B) \in \Omega | S_B \xrightarrow[\delta]{t} S_A\}.$$

We can interpret the Lebesgue measure of the set $M_\delta(t)$ as a measure of resemblance associated with a transformation t. Intuitively, this should reflect the perceived notion of "closeness" of two shapes.

Let us discuss the meaning of Remark 1 for the different classes of transformations.

Translations and rigid motions. The sample space is in this case $\Omega = A \times B$ and $M_\delta(t) = \{(a, b) \in A \times B \mid dist(a, t(b)) \leq \delta\}$. To maximize the measure of this set means to find a transformation that maps largest possible parts of the shapes into proximity of each other.

In the case of translations, it can be observed that for $\delta \to 0$ the resulting probability distribution corresponds to the normalized *generalized Radon transform* of the shape A with respect to shape B as defined in [20].

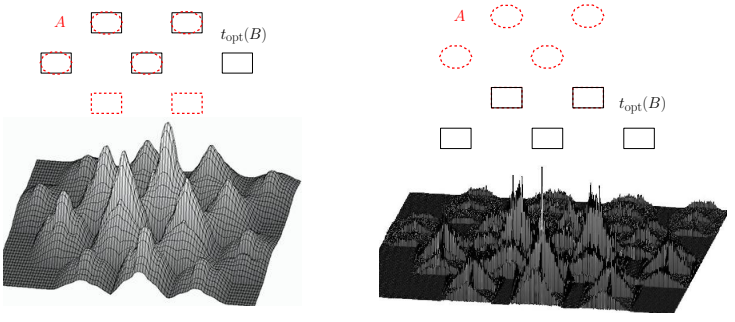


Fig. 1. Matching A (dashed lines) with B (solid lines) with large (left) and small (right) values of δ and the graphs of the corresponding functions $p_\delta(t)$ in translation space

Similarity maps. In case of similarity maps a sample taken from one shape consists of two random points, the sample space is then $\Omega = A^2 \times B^2$. By Remark 1 the similarity map with maximum coverage by the δ-regions is the one maximizing the measure of the set

$$M_\delta(t) = \left\{(a_1, a_2, b_1, b_2) \in A^2 \times B^2 \mid dist(t(b_1), a_1) \leq \delta \text{ and } dist(t(b_2), a_2) \leq \delta\right\}.$$

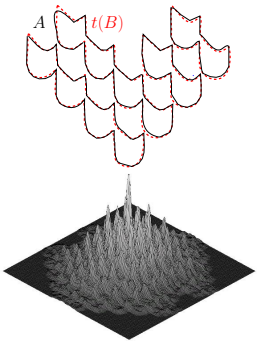
This measure is less intuitive with respect to matching shapes than the one in the previous cases. A simple consideration shows, however, that maximizing the measure of $M_\delta(t)$ also means to maximize the measure of $M'_\delta(t) = \{(a, b) \in A \times B \mid dist(t(b), a) \leq \delta\}$. The measure of the set $M_\delta(t)$ is exactly $\left|M'^2_\delta(t)\right| = |M'_\delta(t)|^2$. Since the measure of a set is always non-negative, both functions have maxima at the same values of t.

The role of the parameter δ. In the description of the algorithm we introduced a parameter δ, which defines how far apart two samples are allowed to be and still be considered close. The choice of δ, therefore, should be specified by the user and controls the trade-off between the quality of match and the size of the parts matched. With a small value of δ our algorithm would find a transformation which maps nearly congruent parts of two shapes to each other. A large value of δ leads to a transformation which gives a rough match but for larger parts of the shapes, see Figure 1.

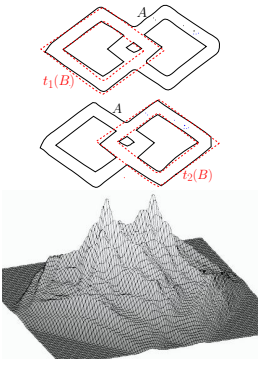
For nearly congruent shapes, however, a small δ already leads to a complete matching, see Figure 2(a). If shape B or parts of it are nearly congruent to some parts of A, then with a small value of δ we detect these occurrences as shown in Figures 2(b) and 2(c). For this purpose it might be worth to consider several local maxima of p_δ.

3.2 Approximation of the Hit Probability

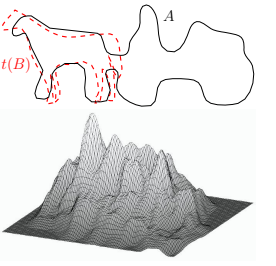
In this section we determine how many samples are needed in order to approximate the function $p_\delta(t)$ in the transformation space within a certain accuracy ε with high probability and analyze the total running time of the algorithm.



(a) Nearly congruent shapes yield a function $p_\delta(t)$ with a clear maximum.



(b) Shape B occurs twice in A which results in two (almost) equally large local maxima of $p_\delta(t)$.



(c) A rough complete-partial match of horse and carriage shapes from the MPEG7-Shape B data set results in a less distinct maximum of $p_\delta(t)$ than in case of nearly congruent shapes.

Fig. 2. Matched shapes and the corresponding function $p_\delta(t)$ in translation space

In order to find a transformation covered by the highest number of δ-regions corresponding to the samples, we consider the arrangement of these δ-regions, i.e., the subdivision of the transformation space induced by the boundaries of the regions. All transformations in the same cell of the arrangement have the same region coverage. Therefore, it is sufficient to traverse the arrangement and take the nodes with the highest number of δ-regions that contain this node.

We will show that the fraction of δ-regions covering the deepest cell of the arrangement gives a good approximation to the maximum value of $p_\delta(t)$. Let the random variable $Z(t)$ denote the number of δ-regions produced by N random experiments that cover t. Let $\tilde{p}_\delta(t)$ denote the ratio of the number of the observed δ-regions that cover t to the total number of samples, that is $\tilde{p}_\delta(t) = \frac{Z(t)}{N}$. $\tilde{p}_\delta(t)$ is an estimate of $p_\delta(t)$.

Theorem 1. *Given two shapes A and B, i.e., finite sets of rectifiable curves with total lengths L_A, L_B, respectively, and a tolerance value $\delta > 0$, for any ε, η, $0 < \varepsilon, \eta < 1$, and for the transformation classes translations, rigid motions, and similarities, the following holds: Let t_{app} be a transformation maximizing $\tilde{p}_\delta(t)$ after some number N of random experiments and t_{opt} a transformation maximizing $p_\delta(t)$, and let $m = \max(L_A, L_B, n\delta)$, where n is the total number of curves in A and B. Then there exists a constant c such that for $N \geq c\frac{m^2}{\varepsilon^2\delta^2} \ln\left(\max(\frac{1}{\eta}, \frac{m^2}{\varepsilon^2\delta^2})\right)$ the probability that $|\tilde{p}_\delta(t_{\text{app}}) - p_\delta(t_{\text{opt}})| \geq \varepsilon p_\delta(t_{\text{opt}})$ is at most η.*

Observe that the relative error with respect to $p_\delta(t)$ is also the relative error with respect to $|M_\delta(t)|$, which is the measure of resemblance underlying our algorithm. For the proof of the theorem we first show that for a fixed transformation t the probability of a bad estimate of $p_\delta(t)$ falls exponentially with N (Lemma 1). Then we define some experiment dependent transformations t and show that also for those t the probability of a bad estimate depends exponentially on N (Lemma 2). Finally we argue that it is sufficient to compute the estimate $\tilde{p}_\delta$ for a finite set of experiment dependent transformations t in order to get a good approximation of the maximum of p_δ. These considerations are valid for all transformation classes considered.

Lemma 1. *For all $0 < \varepsilon, \nu < 1$, for a sample S of size N, and any transformation represented by some vector $t \in \mathbb{R}^d$ the following holds:*

- $p_\delta(t) \leq \nu \Rightarrow P(\tilde{p}_\delta(t) > (1+\varepsilon)\nu) \leq e^{-\frac{\varepsilon^2 \nu N}{3}}$
- $p_\delta(t) \geq \nu \Rightarrow P(|\tilde{p}_\delta(t) - p_\delta(t)| > \varepsilon p_\delta(t)) \leq 2e^{-\frac{\varepsilon^2 \nu N}{4}}$.

Proof. If $p_\delta(t) \leq \nu$:

$$\begin{aligned}
P(\tilde{p}_\delta(t) > (1+\varepsilon)\nu) &= P(Z(t) > (1+\varepsilon)\nu N) = P(e^{rZ(t)} \geq e^{r(1+\varepsilon)\nu N}) \quad \text{for all } r > 0 \\
&\leq \tfrac{E(e^{rZ(t)})}{e^{r(1+\varepsilon)\nu N}} \quad \text{by the Markov inequality [17]} \\
&\leq \frac{e^{(e^r-1)p_\delta(t)N}}{e^{r(1+\varepsilon)\nu N}} \qquad \text{since } r \leq e^r - 1 \\
&\leq \left(\tfrac{e^{(e^r-1)}}{e^{r(1+\varepsilon)}}\right)^{\nu N} = (e^{\varepsilon-(1+\varepsilon)\ln(1+\varepsilon)})^{\nu N} \quad \text{for } r = \ln(1+\varepsilon) \\
&\leq e^{-\frac{\varepsilon^2 \nu N}{3}} \qquad \text{for } 0 < \varepsilon < 1 \ .
\end{aligned}$$

In case $p_\delta(t) \geq \nu$:

$$\begin{aligned}
P(|\tilde{p}_\delta(t) - p_\delta(t)| > \varepsilon p_\delta(t)) &= P(|Z(t) - p_\delta(t)N| > \varepsilon p_\delta(t)N) \\
&= P(|Z(t) - E(Z(t))| > \varepsilon E(Z(t))) \\
&\leq e^{-\frac{\varepsilon^2 E(Z(t))}{2}} + e^{-\frac{\varepsilon^2 E(Z(t))}{4}}
\end{aligned}$$

by the simplified Chernoff bound [17, Thm. 4.4,4.5]. Since $p_\delta(t) \geq \nu$, we get

$$P(|\tilde{p}_\delta(t) - p_\delta(t)| > \varepsilon p_\delta(t)) \leq 2e^{-\frac{\varepsilon^2 p_\delta(t) N}{4}} \leq 2e^{-\frac{\varepsilon^2 \nu N}{4}},$$

which concludes the proof. □

We associate with each cell C of the arrangement $\mathcal{A}$ of δ-regions a so-called *witness point*, i.e., a point that lies on a lowest-dimensional face F of $\mathcal{A}$ that contributes to the boundary of C. Observe, that F is in general a connected component of the intersection of k boundaries of δ-regions with $1 \leq k \leq d$. Thus, by considering all k-subsets of δ-regions for all k, $1 \leq k \leq d$, and taking a point in each connected component of the intersection of those k region boundaries we can be sure to have at least one witness point for each cell of the arrangement.

The following lemma states approximation bounds for the witness points:

Lemma 2. *For all ε, ν, $0 < \varepsilon, \nu < 1$, and a sample set S of size $N \geq \frac{2d}{\varepsilon\nu} + d$, for any witness point $t \in \mathbb{R}^d$ of the arrangement of the δ-regions corresponding to the samples in S, the following holds:*

- $p_\delta(t) \leq \nu \Rightarrow P(\tilde{p}_\delta(t) > (1+\varepsilon)\nu) \leq e^{-\frac{\varepsilon^2(N-d)\nu}{12}}$
- $p_\delta(t) \geq \nu \Rightarrow P(|\tilde{p}_\delta(t) - p_\delta(t)| > \varepsilon p_\delta(t)) \leq 2e^{-\frac{\varepsilon^2\nu(N-d)}{16}}$.

Proof. Observe that Lemma 1 cannot be applied to the witness points directly since they depend on the experiment, i.e., the chosen samples. However, since they depend on at most d samples, the remaining $\geq N-d$ samples are "random" for them and we can apply Lemma 1 replacing N by $N-d$. More specifically:

Let $S_1, \ldots, S_i \in S$, $1 \leq i \leq d$, be the sample pairs whose δ-regions induce the witness point t. Consider the sample set $Q = S \setminus \{S_1, \ldots, S_i\}$, $|Q| = N - i$. Let $Z_Q(t)$ denote the number of the δ-regions that cover t in samples Q and $\tilde{p}_{\delta Q}(t) = Z_Q(t)/(N-i)$. Since we consider closed regions, $Z_Q(t) = Z(t) - i$, $\tilde{p}_{\delta Q}(t) \leq \tilde{p}_\delta(t)$ and

$$\tilde{p}_{\delta Q}(t) = \tfrac{Z(t)-i}{N-i} = \tfrac{Z(t)}{N}\tfrac{N}{N-i} - \tfrac{i}{N-i} \geq \tilde{p}_\delta(t) - \tfrac{i}{N-i} \geq \tilde{p}_\delta(t) - \tfrac{d}{N-d} \ .$$

Therefore,

$$|\tilde{p}_\delta(t) - p_\delta(t)| \leq |\tilde{p}_{\delta Q}(t) - p_\delta(t)| + |\tilde{p}_\delta(t) - \tilde{p}_{\delta Q}(t)| \leq |\tilde{p}_{\delta Q}(t) - p_\delta(t)| + \tfrac{d}{N-d} \ .$$

In case $p_\delta(t) \leq \nu$:

$$\begin{aligned} P(\tilde{p}_\delta(t) > (1+\varepsilon)\nu) &\leq P\left(\tilde{p}_{\delta Q}(t) + \tfrac{d}{N-d} > (1+\varepsilon)\nu\right) = P\left(\tilde{p}_{\delta Q}(t) > (1+\varepsilon)\nu - \tfrac{d}{N-d}\right) \\ &\leq P\left(\tilde{p}_{\delta Q}(t) > \left(1+\tfrac{\varepsilon}{2}\right)\nu\right) \qquad \text{for } N \geq \tfrac{2d}{\varepsilon\nu} + d \\ &\leq e^{-\frac{(\varepsilon/2)^2(N-d)\nu}{3}} \qquad \text{by Lemma 1} \\ &= e^{-\frac{\varepsilon^2(N-d)\nu}{12}} \end{aligned}$$

If $p_\delta(t) \geq \nu$:

$$\begin{aligned} P(|\tilde{p}_\delta(t) - p_\delta(t)| > \varepsilon p_\delta(t)) &\leq P\left(|\tilde{p}_{\delta Q}(t) - p_\delta(t)| + \tfrac{d}{N-d} > \varepsilon p_\delta(t)\right) \\ &\leq P\left(|\tilde{p}_{\delta Q}(t) - p_\delta(t)| > \tfrac{\varepsilon}{2} p_\delta(t)\right) \qquad \text{for } N \geq \tfrac{2d}{\varepsilon\nu} + d \\ &\leq 2e^{-\frac{(\varepsilon/2)^2\nu(N-d)}{4}} = 2e^{-\frac{\varepsilon^2\nu(N-d)}{16}} \ . \end{aligned}$$

□

In the above lemmata we used an additional parameter ν for the smallest value of $p_\delta(t)$ which we want to approximate well enough. Next, we eliminate this parameter:

Lemma 3. *For any two shapes A and B of total length L_A and L_B, respectively, and for the following classes of transformations: translations, rigid motions, and similarities, there exists a transformation t such that $p_\delta(t) \geq \frac{\delta^2}{m^2}$, where $m = \max(L_A, L_B, n\delta)$ and n is the total number of curves in A and B.*

Proof. Let s_a denote a part of one of the curves of A of length δ if there exists one, otherwise s_a denotes the longest curve of A. The length of s_a is at least $\frac{L_A}{n} \leq \delta$ in the second case. Similarly s_b denotes a subcurve of length δ or the longest curve of B with length at least $\frac{L_B}{n} \leq \delta$. Let v denote the translation vector that maps the center of s_b to the center of s_a. For an arbitrary point p_a of s_a and an arbitrary point p_b of s_b it holds that $dist(p_a, p_b + v) \leq \delta$, therefore v is covered by the δ-region corresponding to p_a, p_b. Thus, in case of translations $s_a \times s_b$ is a subset of $M_\delta(v)$ and $p_\delta(v) \geq \frac{|s_a \times s_b|}{|\Omega|} \geq \frac{\min(\delta^2,\ \delta L_A/n,\ \delta L_B/n,\ L_A L_B/n^2)}{L_A L_B} \geq \frac{\delta^2}{m^2}$.

For rigid motions the same argument as above shows that the rigid motion t with rotation angle 0 and translation vector v is covered by every δ-region corresponding to an arbitrary point in s_a and an arbitrary point in s_b.

In case of similarity maps the shape B can be scaled by the factor $\frac{\delta}{D_B}$, where D_B is the diameter of B, so that the diameter of the scaled shape B is δ. If A contains a connected component of length at least δ, then we can place the scaled shape B in such a way that for any point of a part of A of length δ the distance to any point of the scaled B is at most δ. Therefore, the measure of the set $M_\delta(t)$ for that t is at least $L_B^2 \cdot \delta^2$. The corresponding value of p_δ is $p_\delta(t) = \frac{|M_\delta(t)|}{|\Omega|} \geq \frac{L_B^2 \cdot \delta^2}{L_A^2 L_B^2} = \frac{\delta^2}{L_A^2} \geq \frac{\delta^2}{m^2}$. Otherwise, observe that the largest connected component of A must have length at least $\frac{L_A}{n} \leq \delta$. For the transformation t that maps the scaled B to the largest component of A the measure of $M_\delta(t)$ is then at least $\frac{L_A^2}{n^2} L_B^2$ and $p_\delta(t) \geq \frac{1}{n^2} \geq \frac{\delta^2}{m^2}$. □

Now we can prove Theorem 1:

Proof. (Of Theorem 1) Any witness point lies on the boundary of a k-subset of δ-regions. Furthermore, any k-subset yields a system of constantly many polynomial equations of constant degree. There are constantly many connected components of the solution set of such system of equations, and with each connected component we associate a witness point. Therefore, there are at most $c_0 \sum_{k=1}^{d} \binom{N}{k} \leq c_0 N^d$ witness points, where c_0 is a constant. Then the probability that there exists a witness point t with $p_\delta(t) \geq \nu$ and $|\tilde{p}_\delta(t) - p_\delta(t)| > \varepsilon p_\delta(t)$ or with $p_\delta(t) < \nu$ and $\tilde{p}_\delta(t') > (1+\varepsilon)\nu$ is, according to Lemma 2, at most $c_0 N^d 2e^{-\frac{\varepsilon^2 \nu (N-d)}{16}}$. A straightforward calculation shows that for $N \geq \frac{c_1}{\varepsilon^2 \nu} \ln\left(\frac{1}{\varepsilon^2 \nu}\right)$ with some suitable constant c_1 this value is at most $e^{-\frac{\varepsilon^2 \nu (N-d)}{32}}$, which is less than η for $N \geq \frac{32}{\varepsilon^2 \nu} \ln \frac{1}{\eta} + d$. So the probability that there exists a witness point, for which the estimate of $p_\delta(t)$ is bad in the sense described above, is at most $\eta/2$ for

$$N \geq \frac{c_2}{\varepsilon^2 \nu} \ln\left(\max\left(\frac{1}{\eta}, \frac{1}{\varepsilon^2 \nu}\right)\right) \tag{1}$$

for some constant c_2.

By Lemma 3 for any two shapes and the transformation classes considered there always exists a transformation t such that $p_\delta(t) \geq \frac{\delta^2}{m^2}$, where $m = \max$

$(L_A, L_B, n\delta)$. The maximum of p_δ is then also greater or equal $\frac{\delta^2}{m^2}$ and we can choose the value ν as $\nu = \frac{\delta^2}{m^2}$.

Let t^* be a witness point of the cell of the arrangement containing t_{opt}. Plugging the value of ν in formula (1) we obtain that after $N = O\left(\frac{m^2}{\varepsilon^2\delta^2} \ln\left(\max(\frac{1}{\eta}, \frac{m^2}{\varepsilon^2\delta^2})\right)\right)$ experiments for all witness points, in particular for t^* and t_{app}, and additionally for t_{opt} it holds with probability at least $1 - \eta/2$ that $|\tilde{p}_\delta(t) - p_\delta(t)| \leq \varepsilon p_\delta(t)$. Combining these error bounds we get

$$\begin{aligned}
\tilde{p}_\delta(t_{\text{app}}) &\geq \tilde{p}_\delta(t^*) && \text{since } t_{\text{app}} \text{ maximizes } \tilde{p}_\delta(t)\\
&= \tilde{p}_\delta(t_{\text{opt}}) && \text{for } t_{\text{opt}} \text{ is in the cell witnessed by } t^*\\
&\geq (1-\varepsilon)p_\delta(t_{\text{opt}}) && \text{with probability } \geq 1 - \eta/2
\end{aligned}$$

and

$$\begin{aligned}
\tilde{p}_\delta(t_{\text{app}}) &\leq (1+\varepsilon)p_\delta(t_{\text{app}}) && \text{with probability } \geq 1 - \eta/2\\
&\leq (1+\varepsilon)p_\delta(t_{\text{opt}}) && \text{since } t_{\text{opt}} \text{ maximizes } p_\delta(t)
\end{aligned}$$

Therefore, $|\tilde{p}_\delta(t_{\text{app}}) - p_\delta(t_{\text{opt}})| \leq \varepsilon p_\delta(t_{\text{opt}})$ with probability at least $1 - \eta$. □

Running time. The running time of the algorithm consists of the time needed to generate N random samples denoted by $T_{\text{gen}}(n, N)$, where n is the number of curves in the shape, and the time needed to determine the depth of the arrangement of N δ-regions denoted by $T_{\text{depth}}(N)$. A random point on a curve can be generated in constant time. For generating a random point from a set of n curves we first select a curve randomly with probability proportional to the relative length of the curve and then take a random point from the selected curve. If we first record the curve lengths and record the corresponding probabilities to allow for binary search during the generation process we get preprocessing time linear in n and $O(\log n)$ generation time for a single point. Therefore, $T_{\text{gen}}(n, N) = O(n + N \log n)$.

In order to determine the depth of the arrangement of N δ-regions we can construct this arrangement and during the construction keep record of the depth of the cells. Then at the end of the construction algorithm we know the depth of the deepest cell. For general metrics L_p and the considered classes of transformations the boundaries of δ-regions are algebraic hypersurfaces. By Basu et al. [9], the arrangement of such surfaces can be constructed and traversed in $T_{\text{depth}}(N) = O(N^{d+1})$ time.

Summarizing these results and using Theorem 1 we obtain the following theorem for all three classes of transformations considered:

Theorem 2. *For any two shapes A and B represented by finite sets of rectifiable curves in the plane, for all transformation classes considered, and for any ε, η, $0 < \varepsilon, \eta < 1$, the following holds: Let t_{opt} denote the transformation maximizing $p_\delta(t)$, L_A, L_B the total lengths of the curves in A and B, respectively, n the total number of curves in both shapes, and $m = \max(L_A, L_B, n\delta)$. Then there exists*

a constant c, such that for $N \geq c\frac{m^2}{\varepsilon^2\delta^2} \ln\left(\max(\frac{1}{\eta}, \frac{m^2}{\varepsilon^2\delta^2})\right)$ the generic probabilistic algorithm with probability at least $1-\eta$ computes a transformation t_{app} such that $|\tilde{p}_\delta(t_{\text{app}}) - p_\delta(t_{\text{opt}})| \leq \varepsilon p_\delta(t_{\text{opt}})$ in time $O(n + N \log n + N^{d+1})$, where d is the dimension of the transformation space.

The theorem only states that with high probability the numerical value obtained by $\tilde{p}_\delta(t_{\text{app}})$ is close to $p_\delta(t_{\text{opt}})$ which is a measure for the closeness of the two shapes. t_{app} and t_{opt} need not be close in transformation space. But it is also easily possible to derive that the transformation t_{app} is "good" in the sense that $p_\delta(t_{\text{app}})$ is close to the optimum, since $\tilde{p}_\delta(t_{\text{app}})$ is close to $p_\delta(t_{\text{app}})$.

Observe that, at least for sufficiently small values of δ, the runtime of the algorithm depends much more on the parameters ε and η than on the combinatorial input size n, which is only needed in the preprocessing and the drawing of random samples.

For translations and for similarities in combination with the L_1 and L_∞ metrics the running time of the algorithm can be improved: In case of translations, the δ-regions are pseudo-disks and their arrangement can be constructed straightforwardly in time $O(N^2)$. For similarities in combination with the L_1 or L_∞ metric the δ-regions in transformation space are bounded by a constant number of 3-dimensional hyperplanes. Using the algorithm of Edelsbrunner et al. [11] the arrangement of N such δ-regions can be constructed in $O(N^4)$ time. For translations, further speed-up can be achieved in combination with the depth approximation algorithm by Aronov and Har-Peled [8] resulting in running time $T_{\text{depth}}(N) = O(N\varepsilon^{-2} \log N)$.

4 Conclusions

In this paper we presented a probabilistic approach for matching two shapes which comes close to the human notion of match and is easy to implement. In this paper, we only considered the transformations: translations, rigid motions, and similarities, since those are the ones most commonly used for shape matching. However, our approach is much more general. Elaborating the details of similar analyses for homothety, shear and affine transformations is part of our ongoing work.

A minor problem occurs for the described variant for matching under similarity transformations in practice. As we observed in Lemma 3 a transformation t_1 that scales the shape B down to a shape of diameter δ and maps the scaled B to some position on the shape A has a measure of resemblance of $\frac{\delta^2}{m^2}$. On the other hand, if the shapes A and B are similar, then the transformation t_2 that matches best the shape B to the shape A has approximately the same measure of resemblance, since for every point b of B there is a segment $s_a(b)$ of A of length (approximately) δ such that b is in the δ-neighborhood of every point in $s_a(b)$. Because of this overrating of shrinking transformations other, reasonable transformations are likely to be missed, This problem can easily be avoided by setting a lower bound for the allowed scaling factor to some constant times $\frac{\delta}{D_B}$,

where D_B is the diameter of the shape B. Samples yielding smaller scaling factor will be discarded. By this restriction of the scaling factor we could achieve good experimental results. A similar problem arises in matching under affine transformations, where the analogue to a lower bound on the scaling factor is a lower bound on the value of the determinant of the linear transformation matrix.

We also considered some variants of random sample generation for rigid motions and similarities. For rigid motions, in addition to a point of the shape we take the direction of the tangent line at that point and restrict the corresponding δ-region so that not only points but also the sample directions are close. For similarity transformations, a sample of one shape consists of one point, the direction of the tangent line at that point and the (eventually interpolated) curvature at that point instead of a sample consisting of two points. The idea for rigid motions is to reduce the search space, and in case of similarity transformations this alternative approach is not affected by the downscaling problem described above. These approaches are best suitable for shapes where the tangent slopes actually contribute to the shape characterization, as opposed to shapes with noisy contours or shapes composed of many sparse and small parts.

In general the probabilistic algorithm presented here is robust to noise, deformations and cracks in the representation of shapes and does not require shapes to be modelled by a single contour line. It is applicable to the problem of complete and partial matching. As was mentioned before, for practical purposes we enhanced this algorithm with various heuristics (see [6]) but even in the simple form presented here we observed reasonable matching results in experiments with the MPEG7-Shape B data set and a selection of trademark images.

References

1. Aguado, A.S., Montiel, E., Nixon, M.S.: Invariant characterisation of the Hough transform for pose estimation of arbitrary shapes. Pattern Recognition 35, 1083–1097 (2002)
2. Alt, H., Behrends, B., Blömer, J.: Approximate matching of polygonal shapes. Annals of Mathematics and Artificial Intelligence 13, 251–265 (1995)
3. Alt, H., Godau, M.: Computing the Fréchet distance between two polygonal curves. Internat. J. Comput. Geom. Appl. 5, 75–91 (1995)
4. Alt, H., Guibas, L.J.: Discrete geometric shapes: Matching, interpolation, and approximation. In: Handbook of computational geometry. Elsevier, Amsterdam (1999)
5. Alt, H., Rote, G., Wenk, C., Efrat, A.: Matching planar maps. J. of Algorithms, 262–283 (2003)
6. Alt, H., Scharf, L., Scholz, S.: Probabilistic matching and resemblance evaluation of shapes in trademark images. In: Proc. of the 6th ACM International Conference on Image and Video Retrieval, pp. 533–540 (2007)
7. Arkin, E.M., Chew, L.P., Huttenlocher, D.P., Kedem, K., Mitchell, J.S.B.: An efficiently computable metric for comparing polygonal shapes. IEEE Trans. Pattern Anal. Mach. Intell. 13(3), 209–216 (1991)
8. Aronov, B., Har-Peled, S.: On approximating the depth and related problems. In: Proc. 16th ACM-SIAM Symposium on Discrete Algorithms, pp. 886–894 (2005)

9. Basu, S., Pollack, R., Roy, M.-F.: Computing roadmaps of semi-algebraic sets. In: Proc. 28th ACM Symposium on Theory of Computing, pp. 168–173 (1996)
10. Cheong, O., Efrat, A., Har-Peled, S.: On finding a guard that sees most and a shop that sells most. In: Proc. 15th ACM-SIAM Symposium on Discrete Algorithms, pp. 1091–1100 (2004)
11. Edelsbrunner, H., O'Rourke, J., Seidel, R.: Constructing arrangements of lines and hyperplanes with applications. SIAM J. Comput. 15(2), 341–363 (1986)
12. Efrat, A., Fan, Q., Venkatasubramanian, S.: Curve Matching, Time Warping, and Light Fields: New Algorithms for Computing Similarity between Curves. J. Math. Imaging Vis. 27(3), 203–216 (2007)
13. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM 24(6), 381–395 (1981)
14. Hagedoorn, M., Overmars, M., Veltkamp, R.: A new visibility partition for affine pattern matching. In: Nyström, I., Sanniti di Baja, G., Borgefors, G. (eds.) DGCI 2000. LNCS, vol. 1953, pp. 358–370. Springer, Heidelberg (2000)
15. Latecki, L.J., Veltkamp, R.C.: Properties and performances of shape similarity measures. In: Proc. of Int. Conf. on Data Science and Classification (2006)
16. Mitra, N.J., Guibas, L.J., Pauly, M.: Partial and approximate symmetry detection for 3d geometry. ACM Trans. Graph. 25(3), 560–568 (2006)
17. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press, Cambridge (2005)
18. Stockman, G.: Object recognition and localization via pose clustering. Computer Vision, Graphics, and Image Processing 40, 361–387 (1987)
19. Tanase, M., Veltkamp, R.C., Haverkort, H.: Multiple polyline to polygon matching. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 60–70. Springer, Heidelberg (2005)
20. Toft, P.: The Radon Transform - Theory and Implementation. PhD thesis, Department of Mathematical Modelling, Technical University of Denmark (1996)

Object Caching for Queries and Updates

Philip Little and Amitabh Chaudhary

Department of Computer Science & Engineering,
University of Notre Dame
{plittle1,achaudha}@cse.nd.edu

Abstract. We consider a new online problem, called caching for queries and updates, which encompasses three expansions to standard caching: requests can be for multiple data objects (or "file bundles"), requests can be queries that return results much smaller than the data they access, or there may be updates to the data at the source. Different combinations of these expansions arise in modern applications such as scientific computing on data-grids and middle-tier caching in web-based databases. We present a randomized online $(2\alpha + 2)$-competitive algorithm for this problem, given any α-competitive algorithm for the well studied object caching (multi-size paging) problem. This is the first known online algorithm for the combined problem and for several simpler combinations of the three extensions. This algorithm is both space-efficient and computationally tractable and has bounded overhead for control communication.

1 Introduction

We present an online competitive algorithm for a new abstract caching problem, which we call *caching for queries and updates.* This problem combines three expansions to caching, which have arisen in modern network-bound applications, into one seamless whole. These expansions are with respect to the baseline problem of *object caching* [1, 2, 3], described as follows: There is a remote data server that hosts data objects of varying sizes and a local cache, at or close to a client, which can hold only a few objects at any given time due to its limited size. The cache receives client requests for objects, and it responds to each request either (1) by forwarding the object to the client if it happens to have a local copy, or (2) else by first evicting some other objects to make space, then loading a copy of the required object from the server, and then forwarding it to the client. If the cache does not have a local copy it may choose another option: (3) getting a copy of the object from the server and just forwarding it to the client without actually storing a local copy. This last option does not affect the contents of the cache and is called the *bypass option.* The objective is to make eviction and bypass decisions that minimize the cost in terms of network traffic, i.e., the number of bytes sent on the network between the server and the cache. Object caching, both with and without the bypass option, is well studied and has known online competitive algorithms. [1, 2, 3]. The three expansions we address are:

S. Das and R. Uehara (Eds.): WALCOM 2009, LNCS 5431, pp. 394–405, 2009.

Multi-object Requests. Often single requests are for multiple objects in the cache, as in scientific data-grids in which computational jobs process several files concurrently and can only be executed when all files are present in the cache—termed *file-bundle caching* [4, 5]. These also occur, e.g., in table caching for web-databases where the requests, which are database queries, perform *joins* on multiple tables [6]. No online competitive algorithms are currently known for this; simple object caching algorithms (as in [1, 2, 3]) break down as they are not designed to track or maintain the right combination of objects.

Query Processing Requests. These occur in web-databases using table or column caching. Here, instead of requesting an entire object (table or column), a request is essentially a (read-only) database query that accesses the object but forwards to the client only the final processed result—often much smaller in size than the object accessed [7, 6]. Competitive online algorithms are known when the query accesses a single object [7], but not when it involves a join on multiple objects.

Dynamic Source Data. In databases for e-commerce or science, often the server independently receives updates to objects, which effectively make the corresponding copies in the cache stale [8, 9, 10, 6]. To answer client requests using the latest updates but without incurring substantial network costs, several mechanisms, including query-shipping and update-shipping (described in Sect. 1.1), have been designed. There are, however, no known competitive online algorithms that optimize their use.

1.1 Problem Description

As in object caching, in the problem of caching for queries and updates we have a single remote server which hosts objects of different sizes and a single local cache of limited size which at any given time can maintain copies of only some of the objects at the server. The cache receives a sequence of client queries, each of which accesses a *bundle* (subset) of objects on the server, and requires a processed query result to be forwarded to the client. The server meanwhile receives a sequence of updates, each of which affects a specific object. As soon as an update is received, the server copy of the affected object is updated, but any copy on the cache becomes stale unless a deliberate action is taken to update it. The objective is to service all queries in the sequence with the latest updated data, such that the network traffic incurred is minimized.

To reduce network traffic and yet allow queries to be serviced with the latest data, we use the complementary mechanisms of *query shipping* and *update shipping*. When a query arrives, if the cache contains the latest updated copies of all objects in the query's bundle, then the query is executed in cache and serviced without incurring any network traffic between server and cache. (We ignore the traffic between cache and client as it is both negligible and the same irrespective of the choices we make.) Else, we have two options. The first option is to send the specification of the query from the cache to the server, execute the query on the server (with the latest updated copies), and then send the query result from

server to cache to be forwarded to the client. This is called query result shipping or simply query shipping.

The other option is to first load into cache all objects in the query's bundle that are currently not in cache. For this some of the current objects may have to be evicted to create sufficient space. Next, some of the objects in the query's bundle that are currently in cache, but not freshly loaded, may need to be updated. So for each update that affects one such object, which arrived after the object was loaded but before the query arrived, and which has not yet been applied to the object copy in cache, we send the update specification from the server to the cache. These updates are then applied to the corresponding object copies in cache. This is called update shipping. The query is now executed in cache and the result forwarded to the client. The cost incurred for loading an object, or shipping a query, or shipping an update is the corresponding number of bytes sent on the network between the server and the cache in each case. A solution to the problem is a sequence of *online decisions* of loading and evicting objects, and shipping queries and updates, which services all queries with the latest updated data, and minimizes the sum of the costs incurred. (For a formal definition of online algorithms and competitive ratios please see [11].)

Aside from loading objects and shipping queries and updates, the server and cache may need to communicate simply to inform the other of the events at their ends or to decide which choices to make. We call such communication *control communication* and distinguish it from *data communication*, which is used to transfer the data in loading objects, or shipping queries or updates. We assume that cost of control communication is relatively insignificant, and ignore it in the problem definition. Our online algorithm, however, does guarantee bounds on the number of control messages used (see [12]).

Formal Definition. An instance of the *caching for queries and updates problem* consists of (using $\mathcal{P}(\cdot)$ to denote the powerset)

- a set W of objects at the server, and for each object $o \in W$ a size $s(o) \in \mathbb{R}$ and a load cost $l(o) \in \mathbb{R}$—the sizes are normalized such that the smallest object has size 1;
- a size $k \in \mathbb{R}$ of the cache;
- a set $\mathcal{U}$ of updates, and for each update $u \in \mathcal{U}$, an object $o(u) \in W$ updated by u and a cost $y(u) \in \mathbb{R}$ of shipping u;
- a set $\mathcal{Q}$ of queries, and for each query $q \in \mathcal{Q}$, a set (bundle) $B(q) \in \mathcal{P}(W)$ of objects accessed by q and a cost $y(q) \in \mathbb{R}$ of shipping q; and
- an online sequence of events $\sigma = e_1, \ldots, e_i, \ldots$, in which each $e_i \in (\mathcal{U} \cup \mathcal{Q})$ is either an update or a query.

The objective is, at each event e_i, to decide (online), the objects to load into cache, the objects to evict from cache, the updates to ship to cache, and the queries to ship to the server, if at all, such that the sum of the costs of loading objects, shipping updates, and shipping queries is minimized, and the following constraints are met: (1) the sum of the sizes of the loaded objects at any given

time is at most k, and (2) each query q in σ is serviced with the latest updated data before the next event in σ is revealed. To ensure that a query q is serviced with the latest updated data at least one of the following must be true: (1) q is shipped to the server, or (2) for each object $o \in B(q)$ accessed by q, o is in the cache when q is serviced, and for each update u such that u updates o (i.e., $o(u) = o$) and u is an event that occurs after o was last loaded and before q, u is shipped to the cache.

Our Contributions. Our primary contribution is an online randomized algorithm, which we call DynamicBypass, for the problem of caching for queries and updates, which has a competitive ratio of $(2\alpha + 2)$ given any α-competitive online algorithm for the object caching problem (with the bypass option). Given the $O(\lg^2 k)$-competitive algorithms known for object caching, when the load cost of an object is either a fixed cost (fault model) or proportional to the size (bit model) [2], our algorithm yields $O(\lg^2 k)$-competitive algorithms for the corresponding cost models. Further, the amount of state stored at any moment depends only on the objects currently in the cache and the outstanding updates rather than all the objects ever affected by updates or accessed by queries in the sequence σ. This is achieved through randomization. Lastly, the total number of control messages is bounded by the number of actual data messages, i.e., $O(L + U + Q)$, in which L is the number of loads, U is the number of updates shipped, and Q is the number of queries shipped by the algorithm.

Approach Overview. The caching for queries and updates problem can be viewed as encompassing subproblems of type *rent-versus-buy* at two different levels: (1) choosing between loading objects into cache (buying) versus shipping corresponding queries (renting), which we call the *outside subproblem*, and (2) choosing between shipping updates to cache (buying) versus shipping corresponding queries (renting), which we call the *inside subproblem*. The outside subproblem involves making eviction decisions as well. We use the classic ski-rental approach for the rent-versus-buy decisions. There are, however, two kinds of obstacles that need to be overcome: In both subproblems the rent-versus-buy decisions are not independent for different objects. Queries can access multiple objects and overlap "sub-bundles" in common with other queries. This adds a combinatorial element to the decision-making, and a crucial question is allocating the rental cost of a query among the objects in its bundle. We use different approaches for the two subproblems: In the inside subproblem we convert this into a minimum-weighted vertex cover problem on bipartite graphs (this special case is polynomial-time solvable); and in the outside subproblem we find that, given some minimal constraints, even a fairly arbitrary allocation of costs works.

The second obstacle is that the solution to one subproblem influences the solution to the other and vice versa. Bounding the performance of the integrated algorithm, derived by combining the solutions for the subproblems (two independent algorithms, oblivious to each other), is nontrivial. We achieve this surprising

result by introducing an hypothetical algorithm that mimics the offline optimal for one subproblem and our algorithm for the other, facilitating comparison. This technique should have applicability in other similar situations.

2 Related Work

The caching for queries and updates problem, has not been addressed before. It is however a generalization of several well-studied problems in caching, which we now describe. The fundamental caching problem of *paging* occurs in virtual memory hierarchies and is for objects of uniform size and with uniform load cost. The cache can hold at most k objects at any time. If an object is in cache when requested (cache hit) no cost is incurred, otherwise (cache miss) the object must be loaded from slow memory to cache incurring the load cost—there is no bypass option. The objective is to make online eviction decisions to minimize the cost. Deterministic online k-competitive algorithms (best possible) and a randomized online H_k-competitive algorithm (again, best possible) are known (see [11, 13]). ($H_k = \Theta(\lg k)$ is the kth harmonic number.) These lower bounds on best possible competitive ratios also apply to any problem that generalizes paging, such as caching for queries and updates.

The *multi-size caching* problem or *object caching* generalizes paging by allowing data objects that differ in size. It naturally arises in web proxy caching. Best-possible (k-competitive) deterministic algorithms are known for this problem without the bypass option [1, 3]. Randomized $O(\lg^2(k))$-competitive algorithms for this problem are known when the bypass option is allowed [2].

The *file-bundle caching* problem extends multi-size caching to allow requests involving groups of objects called *bundles*. All of the objects in a request must be present in the cache simultaneously for the cache to satisfy the request. For this problem, which arises in scientific data processing, empirical performance of heuristics have been studied [4, 5], but no prior online algorithm is known.

The *bypass-yield caching* problem extends multi-size caching to requests that are queries which access a single object but forward to the client only the final processed result. This final result may contain data much less than the accessed object. A query may be bypassed, i.e., it is shipped to the server and the result shipped directly back to the client. This option is useful for queries on infrequently accessed objects. An $O(k)$-competitive deterministic algorithm and an $O(\log^2 k)$-competitive randomized algorithm are know for this problem [7].

In the *push-pull partitioning* problem, instead of the cache we have a *replica* of the server, which has no space constraints. (This is not a caching problem as it does not generalize paging.) Queries arriving at the replica access multiple data objects, and updates arriving at the server modify the server copies of the objects. The objective is to use query shipping and update shipping to service all queries with the latest updated data such that the network communication cost is minimized. Heuristics and offline solutions have been proposed [8, 9, 10] but no prior online algorithms are known for this problem as well.

For conciseness we represent the different problems by $\mathcal{P}$ followed by a list of problem parameters. $\mathcal{P}\{k\}$ denotes paging, and in general, $\mathcal{P}\{k,\dots\}$ denotes a problem where the cache is space-constrained—k is the ratio of the size of cache to the size of the smallest object. Similarly, "m" denotes multiple object sizes; "f" denotes file bundles; "b" denotes the bypass option; "q" denotes queries and query shipping; and "u" denotes updates at server.

3 The DynamicBypass Algorithm

In the algorithm DynamicBypass we distinguish between two kinds of query. If a query requires at least one object that is not currently in the cache, it is called an *outside query*. DynamicBypass uses the subroutine OutQ (see Fig. 1 and 2) to decide between shipping the query or loading the missing objects into the cache—solving the outside subproblem. In the pseudocode, CACHE represents the set of objects currently in cache, $\mathcal{A}_{\text{obj}}$ is any given algorithm for object caching with bypass, and the labels are used to limit the number of control messages used (they play no role in the competitive analysis). If not all of the required objects are in the cache when OutQ finishes, the query is shipped. A query is an *inside query* if every object it requires is in the cache. A query may be an inside query when it arrives or may begin as an outside query but become an inside query during the execution of OutQ. DynamicBypass uses the subroutine InQ (see Fig. 3-6) to decide whether to satisfy an inside query by shipping outstanding updates or by shipping the query—solving the inside subproblem. The next sections explore these subroutines in further detail.

Algorithm: OutQ on cache

Invocation: User query q accessing bundle $B(q)$ and shipping cost $y(q)$
$y \leftarrow y(q)$
while $\big(B(q) \setminus \mathsf{CACHE}\big) \neq \emptyset$ *and* $y > 0$ **do**
  $B \leftarrow B(q) \setminus \mathsf{CACHE}$
  foreach $o \in B$ **do**
    if $y \geq l(o)$ **then**
      Invoke $\mathcal{A}_{\text{obj}}$ with next input o with probability 1
      $y \leftarrow y - l(o)$
    else
      Invoke $\mathcal{A}_{\text{obj}}$ with next input o with probability $y/l(o)$
      $y \leftarrow 0$
    Maintain CACHE according to $\mathcal{A}_{\text{obj}}$
    if $\mathcal{A}_{\text{obj}}$ *chooses to load* o *and evict set of objects* X **then**
      DATA MSG to server: "Load o into cache"
      CONTROL MSG to server: "Loading o, evicting X"
if $\big(B(q) \setminus \mathsf{CACHE}\big) = \emptyset$ *and* $y(q) > 0$ **then**
  Invoke InQ on cache with q
else
  DATA MSG to server: "Ship back result of q"

Fig. 1. OutQ on cache

Algorithm: ServiceLoadRequest on server

Invocation: DATA MSG from cache to load object o
DATA MSG to cache: "Loading o"; object o data
Set $label(o)$ to UpdateShipped

Fig. 2. ServiceLoadRequest on server

Algorithm: InQ on cache

Invocation: By OutQ on cache with query q accessing bundle $B(q)$ and shipping cost $y(q)$
foreach $o \in B(q)$ **do**
  if $label(o)$ *is QueryShipped* **then**
    CONTROL MSG to server: "Invoke InQ with q"
    exit
Process q in cache and forward result to client

Fig. 3. InQ on cache

Algorithm: ServiceServerRequest on cache

Invocation: DATA MSG from cache shipping update u or query q result
if DATA MSG *from server is shipped update* **then**
  Wait to receive entire set P of shipped updates for the query q
  foreach $u \in P$ **do**
    Apply u to $o(u) \in$ CACHE
    Set $label(o(u))$ to UpdateShipped
  Process q in cache and forward result to client
else
  /* DATA MSG from server is shipped query q result */
  Forward query q result to client

Fig. 4. ServiceServerRequest on cache

3.1 Outside Queries

During the OutQ subroutine (Fig. 1) DynamicBypass essentially ignores the presence of data updates, which are handled later in the InQ subroutine. The costs OutQ incurs, correspond to the costs of outside queries, and are the costs DynamicBypass would incur if all updates were removed from a sequence. (OutQ solves $\mathcal{P}\{k, m, q, f\}$.), Let σ_Q designate the queries of sequence σ without the updates. We shall compare OutQ(σ_Q) (equivalent to OutQ(σ)) to OPT$\{k, m, q, f\}(\sigma_Q)$.

OutQ decides randomly (with weighted probabilities) which objects to load (see Fig. 1). This simulates the use of counters to determine which objects are most costly to not have in cache, but is more memory-efficient since its memory usage does not scale with the number of distinct objects in σ. When an outside query arrives, OutQ attributes a portion of its shipping cost to each of the missing objects. If the amount of shipping cost that has not yet been attributed is greater than the cost of loading the object currently in consideration, it attributes a

Algorithm: ServiceUpdate on server

Invocation: Update u affecting object o and shipping cost $y(u)$
if $o \in$ CACHE **then**
  Add update vertex v with weight $y(u)$ to interaction graph G
  if *label*(o) *is UpdateShipped* **then**
    Set *label*(o) to QueryShipped
    CONTROL MSG to cache: "Label o QueryShipped"

Fig. 5. ServiceUpdate on server

Algorithm: InQ on server

Invocation: CONTROL MSG from cache with query q accessing bundle $B(q)$ and shipping cost $y(q)$
/* Maintains U_{t-1}, the minimum-weighted vertex cover from the last execution of this subroutine. */
/* Update the interaction graph */
Add query vertex v with weight $y(q)$ to interaction graph G
foreach $u \in V(G)$ **do**
  /* Find interacting updates for query q */
  if $o(u) \in B(q)$ *and* u *at server arrives before* q *at cache* **then**
    Add edge (u, v) to G
/* Compute min vertex cover for bipartite graph G using polynomial algorithms */
Compute the minimum-weighted vertex cover $U_t \subseteq V(G)$ for G
if $q \notin U_t$ **then**
  /* Vertex cover consists of new updates */
  foreach $u \in U_t \setminus U_{t-1}$ **do**
    DATA MSG to cache:"Shipping u"; update u data
    if u *was last outstanding update for* $o(u)$ **then**
      Set *label*(o) to UpdateShipped
      CONTROL MSG to cache: "Label o UpdateShipped"
else
  /* Vertex cover consists of new query */
  Process q on server
  DATA MSG to cache: "Shipping q result"; q result data

Fig. 6. InQ on server

portion equal to the cost of loading the object and loads the object into the cache. If the load cost is the greater quantity, it attributes the remaining shipping cost to the object and loads the object with a probability proportional to the ratio of that amount to the load cost of the object. Thus the sum of all the shipping costs attributed to objects equals the total shipping costs of outside queries in σ (whether or not those queries are shipped by DynamicBypass).

When OutQ decides to load an object, it sends a request for that object to the caching algorithm $\mathcal{A}_{\text{obj}}$ (which may be any algorithm for $\mathcal{P}\{k, m, b\}$), which decides whether to load or bypass the request for the object and what objects (if any) to evict. When $\mathcal{A}_{\text{obj}}$ is loading objects, it may evict others that the query requires. In this case, DynamicBypass will attribute any unattributed shipping

costs to these objects and load then with the corresponding probabilities. OutQ will continue until all objects required by the query are simultaneously in the cache or the entire shipping cost of the query has been attributed to objects. Note it may make counterintuitive decisions by loading only some of the objects required by a query and not others, or by evicting some of the required to load others. These do not negatively affect the competitive ratio, but reduce the complexity of the algorithm. The following theorem summarizes the analysis of DynamicBypass for outside queries. See [12] for the proof.

Theorem 1. *Given any α-competitive online algorithm $\mathcal{A}_{\text{obj}}$ for $\mathcal{P}\{k, m, b\}$, $\mathrm{E}[\mathsf{OutQ}(\sigma_Q)] \leq 2\alpha \cdot \mathsf{OPT}\{k, m, q, f\}(\sigma_Q)$ for any query sequence σ_Q. (*OutQ *is 2α-competitive for $\mathcal{P}\{k, m, q, f\}$.)*

3.2 Inside Queries

InQ solves the problem of servicing queries which only access objects that are currently cached. Let σ_I designate all updates and those queries in σ that are inside queries for DynamicBypass (including queries which only become inside queries after OutQ processes them). For each inside query, InQ must decide whether to ship all outstanding updates to objects accessed by the query (outstanding updates that *interact* with the query) or to ship the query itself. Recall that the cache contains only a subset of the objects at the server at any given time and may contain different objects at different times. Yet, the inside subproblem can be mapped directly to the push-pull partitioning problem $\mathcal{P}\{m, q, f, u\}$ [8] in which the replica (which corresponds to the cache) stores (possibly outdated) copies of *all* objects at the server. The replica, thus, does not make any load or eviction decisions, and just chooses between shipping queries and updates. The mapping is by giving an object in the inside subproblem a unique name each time it is loaded, and using that in all future references. In the following discussion we assume such a mapping when evaluating the cost of DynamicBypass in processing σ_I.

InQ essentially computes, at each query in σ_I, the decisions the corresponding offline optimum $\mathsf{OPT}\{m, q, f, u\}$, denoted $\mathsf{OPT}_{\mathsf{In}}$, would make if the event sequence terminated with the current query—the *incremental optimum* for this point in the sequence. InQ mimics the incremental optimum by shipping any outstanding updates that the incremental optimum would have shipped or shipping the current query if that is what the incremental optimum would do. These decisions may differ from those of subsequent incremental optima, since a decision that is optimal for a sequence terminating with one event may not be optimal in light of further events. This may cause InQ to do extra work (such as shipping a query initially and then later also shipping all interacting updates). Nevertheless, we show that InQ incurs data communication costs no more than twice those of $\mathsf{OPT}_{\mathsf{In}}$ (see Theorem 2).

Figures 3, 4, and 6 address inside queries. The heart of InQ is in its representation of the input sequence σ as a graph, which we call the *interaction graph*. An interaction graph at any stage of the input sequence consists of a node v_u

for each update u and a node v_q for each query q received till now. Each node has a weight equal to the shipping cost associated with it. An update node v_u has an edge with a query node v_q if and only if u and q interact, i.e., q accesses the object that u modifies, and that object was kept in the cache in the entire period between u and q. There are no other edges; thus, the graph is bipartite. We can show that for any set of valid shipping decisions, the set of nodes that correspond to the updates or queries that are shipped form a valid vertex cover in the interaction graph. This implies that to know what an incremental optimum does, InQ needs to compute the minimum-weighted vertex cover of the interaction graph. Since the graph is bipartite, this can be done in *polynomial-time* by a transformation to maximum flow problem (see e.g., [14]). Further, we can show that, for any incremental optimum, the update nodes included in the vertex cover and query nodes excluded from it will not affect the other parts of the cover when it is recomputed for later events. We can therefore remove these nodes from the interaction graph to form the *remainder interaction graph* for more efficient computation. (To keep the pseudocode concise, we have not included this additional efficiency in it.) The interaction graph can be maintained at either the server or cache; here we have described the algorithm with the graph maintained at the server.

Let $\mathsf{InQ}(\sigma_I)$ and $\mathsf{OPT}_{\mathsf{In}}(\sigma_I)$ designate the cost InQ and $\mathsf{OPT}_{\mathsf{In}}$ incur, respectively, in satisfying inside queries. The following theorem summarizes the analysis of DynamicBypass for inside queries. See [12] for the proof.

Theorem 2. $\mathsf{InQ}(\sigma_I) \leq 2\mathsf{OPT}_{\mathsf{In}}(\sigma_I)$. *(*InQ *is 2-competitive for push-pull partitioning.)*

3.3 Analysis of DynamicBypass for All Queries

Now we bound the competitive ratio for DynamicBypass for an entire sequence. The corresponding optimal offline algorithm is $\mathsf{OPT}\{k, m, q, f, u\}$. We prove the following theorem.

Theorem 3. *Given any α-competitive online algorithm $\mathcal{A}_{\text{obj}}$ for $\mathcal{P}\{k, m, b\}$,* $\mathrm{E}[\mathsf{DynamicBypass}(\sigma)] \leq (2\alpha + 2) \cdot \mathsf{OPT}\{k, m, q, f, u\}(\sigma)$ *for any sequence σ. (*DynamicBypass *is $(2\alpha + 2)$-competitive for caching for queries and updates.)*

Recall that $\mathsf{OutQ}(\sigma_Q)$ is what DynamicBypass pays to handle outside queries during the sequence σ, and this is at most 2α times what $\mathsf{OPT}\{k, m, q, f\}$ pays for the sequence σ_Q of queries. Clearly, $\mathsf{OPT}\{k, m, q, f\}(\sigma_Q) \leq \mathsf{OPT}\{k, m, q, f, u\}(\sigma)$. This means that $\mathsf{DynamicBypass}(\sigma_Q) \leq 2\alpha\mathsf{OPT}\{k, m, q, f, u\}(\sigma)$. Also recall that the optimal offline algorithm for minimizing the costs incurred in handling inside queries (given some sequence of loads and evictions) is $\mathsf{OPT}_{\mathsf{In}}$, and DynamicBypass pays at most 2 times what $\mathsf{OPT}_{\mathsf{In}}$ pays for handling inside queries ($\mathsf{InQ}(\sigma_I) \leq 2\mathsf{OPT}_{\mathsf{In}}(\sigma_I)$). Note, however, that $\mathsf{OPT}_{\mathsf{In}}$ minimizes the cost to handle σ_I in terms of DynamicBypass's cache. Since $\mathsf{OPT}\{k, m, q, f, u\}$ may not have the same objects cached at every point, we cannot simply sum $\mathsf{OPT}\{k, m, q, f\}(\sigma_Q)$ and $\mathsf{OPT}_{\mathsf{In}}(\sigma_I)$ for a lower bound

on $\mathsf{OPT}\{k, m, q, f, u\}(\sigma)$. We use a hypothetical algorithm Composite to relate the performance of DynamicBypass to $\mathsf{OPT}\{k, m, q, f, u\}$. Suppose there is an algorithm Composite that does the following:

Rule 1. Composite handles outside queries as OutQ would handle them.
Rule 2. Composite never ships updates to an object that is not in the cache of $\mathsf{OPT}\{k, m, q, f, u\}$.
Rule 3. Composite always ships outstanding updates to an object in the cache of DynamicBypass when $\mathsf{OPT}\{k, m, q, f, u\}$ loads the object.
Rule 4. Composite always ships updates to an object in its cache when $\mathsf{OPT}\{k, m, q, f, u\}$ ships them.
Rule 5. Composite ships any inside query that cannot be satisfied from the cache after rules 1-4 have been followed.

Theorem 4. $\mathsf{Composite}(\sigma_I) \leq \mathsf{OPT}\{k, m, q, f, u\}(\sigma)$ *for any sequence* σ

Proof. Rules 2-5 are sufficient to handle all inside queries. The costs Composite incurs in following these rules is no more than the cost $\mathsf{OPT}\{k, m, q, f, u\}$ incurs over the entire sequence σ: Following rule 2 will not in itself cost Composite anything. Rule 3 will involve Composite shipping some updates when $\mathsf{OPT}\{k, m, q, f, u\}$ loads corresponding objects, but since the updates cannot exceed the load costs, Composite will not pay more for these updates than $\mathsf{OPT}\{k, m, q, f, u\}$ pays for these loads. Rule 4 will only involve Composite shipping updates that $\mathsf{OPT}\{k, m, q, f, u\}$ ships at the same cost. Any queries that Composite is forced to ship because of rule 5 cannot be satisfied from $\mathsf{OPT}\{k, m, q, f, u\}$'s cache, and thus are shipped at the same cost by $\mathsf{OPT}\{k, m, q, f, u\}$. Thus rules 2-5 together will cost Composite no more than $\mathsf{OPT}\{k, m, q, f, u\}(\sigma)$, the total cost of $\mathsf{OPT}\{k, m, q, f, u\}$.

Observe that $\mathsf{Composite}(\sigma) = \mathsf{Composite}(\sigma_Q) + \mathsf{Composite}(\sigma_I)$. Since Composite handles outside queries exactly like DynamicBypass, $\mathsf{Composite}(\sigma_Q) = \mathsf{OutQ}(\sigma_Q) = 2\alpha\mathsf{OPT}\{k, m, q, f, u\}(\sigma)$. Since $\mathsf{OPT}_{\mathsf{In}}$ is optimal for handling inside queries in terms of DynamicBypass's (and thus Composite's) cache, $\mathsf{OPT}_{\mathsf{In}}(\sigma_I) \leq \mathsf{Composite}(\sigma_I) \leq \mathsf{OPT}\{k, m, q, f, u\}(\sigma)$. This means $\mathsf{InQ}(\sigma_I) \leq 2\mathsf{Composite}(\sigma_I)$. Since the total costs of DynamicBypass are $\mathsf{OutQ}(\sigma_Q) + \mathsf{InQ}(\sigma_I) \leq \mathsf{Composite}(\sigma_Q) + 2\mathsf{Composite}(\sigma_I) =\leq 2\alpha\mathsf{OPT}\{k, m, q, f, u\}(\sigma) + 2\mathsf{OPT}\{k, m, q, f, u\}(\sigma)$, we conclude that $\mathsf{DynamicBypass}(\sigma) \leq (2\alpha + 2)\mathsf{OPT}\{k, m, q, f, u\}(\sigma)$.

Conclusions. We have presented a new abstract caching problem, caching for queries and updates, and a $(2\alpha + 2)$-competitive randomized online algorithm for it, given any α-competitive online algorithm for the object caching problem with the bypass option. A useful line of further investigation will be to extend this algorithm to the scenario with multiple servers and clients.

References

[1] Cao, P., Irani, S.: Cost-aware www proxy caching algorithms. In: Proceedings of the USENIX Symposium on Internet Technologies and Systems, Berkeley, CA, USA, USENIX Association, p. 18 (1997)
[2] Irani, S.: Page replacement with multi-size pages and applications to web caching. In: Proceedings of the ACM Symposium on Theory of Computing, pp. 701–710. ACM, New York (1997)
[3] Young, N.E.: Online file caching. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, Philadelphia, PA, USA, pp. 82–86. Society for Industrial and Applied Mathematics (1998)
[4] Otoo, E., Rotem, D., Shoshani, A.: Impact of admission and cache replacement policies on response times of jobs on data grids. Cluster Computing 8, 293–303 (2005)
[5] Otoo, E., Rotem, D., Romosan, A.: Optimal file-bundle caching algorithms for data-grids. In: Proceedings of the 2004 ACM/IEEE conference on Supercomputing, p. 6. IEEE Computer Society Press, Washington (2004)
[6] Luo, Q., Krishnamurthy, S., Mohan, C., Pirahesh, H., Woo, H., Lindsay, B.G., Naughton, J.F.: Middle-tier database caching for e-business. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 600–611. ACM, New York (2002)
[7] Malik, T., Burns, R., Chaudhary, A.: Bypass caching: Making scientific databases good network citizens. In: Proceedings of the International Conference on Data Engineering, pp. 94–105. IEEE Computer Society Press, Washington (2005)
[8] Bagchi, A., Chaudhary, A., Goodrich, M.T., Li, C., Shmueli-Scheuer, M.: Achieving communication efficiency through push-pull partitioning of semantic spaces to disseminate dynamic information. IEEE Transactions on Knowledge and Data Engineering 18(10), 1352–1367 (2006)
[9] Olston, C., Loo, B.T., Widom, J.: Adaptive precision setting for cached approximate values. ACM SIGMOD Record 30, 355–366 (2001)
[10] Olston, C., Widom, J.: Best-effort cache synchronization with source cooperation. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 73–84. ACM, New York (2002)
[11] Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, New York (1998)
[12] Little, P.: Online algorithms for dynamic data and query shipping in object caching. Master's thesis, University of Notre Dame (2008)
[13] McGeoch, L., Sleator, D.: A strongly competitive randomized paging algorithm. Algorithmica 6, 816–825 (1991)
[14] Hochbaum, D.: Approximation Algorithms for NP-hard Problems. PWS Publishing Company (1997)

Author Index