Md. Saidur Rahman
Satoshi Fujita (Eds.)

LNCS 5942

WALCOM: Algorithms and Computation

4th International Workshop, WALCOM 2010
Dhaka, Bangladesh, February 2010
Proceedings

Springer

Lecture Notes in Computer Science 5942

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

Md. Saidur Rahman Satoshi Fujita (Eds.)

WALCOM: Algorithms and Computation

4th International Workshop, WALCOM 2010
Dhaka, Bangladesh, February 10-12, 2010
Proceedings

Springer

Volume Editors

Md. Saidur Rahman
Bangladesh University of Engineering and Technology (BUET)
Department of Computer Science and Engineering
Dhaka 1000, Bangladesh
E-mail: saidurrahman@cse.buet.ac.bd

Satoshi Fujita
Hiroshima University
Graduate School of Engineering
Department of Information Engineering
Kagamiyama 1-4-1, Higashi-Hiroshima, 739-8527, Japan
E-mail: fujita@se.hiroshima-u.ac.jp

Library of Congress Control Number: 2009942780

CR Subject Classification (1998): F.2, G.2.1, G.2.2, G.4, I.1, I.3.5, E.1, B.8

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

ISSN 0302-9743
ISBN-10 3-642-11439-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-11439-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12836628 06/3180 5 4 3 2 1 0

Preface

WALCOM 2010, the 4th International Workshop on Algorithms and Computation, held during February 10–12, 2010 in Dhaka, Bangladesh, covered the areas of approximation algorithms, combinatorial algorithms, combinatorial optimization, computational Biology, computational geometry, data structures, graph algorithms, graph drawing, parallel and distributed algorithms, parameterized complexity, network optimization, online algorithms, randomized algorithms and string algorithms. The workshop was organized jointly by the Bangladesh Academy of Sciences (BAS) and Bangladesh University of Engineering and Technology (BUET), and the quality of the workshop was ensured by a Program Committee comprising 25 researchers of international repute from Australia, Bangladesh, Canada, France, Germany, Greece, Hong Kong, Hungary, India, Italy, Japan, Switzerland, Taiwan, UK and USA.

This volume contains 23 contributed papers and four invited papers presented at WALCOM 2010. The Call for Papers received an enthusiastic response, resulting in 60 submissions from 21 countries. The Program Committee thoroughly reviewed each of the 60 submissions and accepted 23 of them for presentation in the workshop after elaborate discussions on review reports. The image of the workshop was highly enhanced by the four invited talks of eminent and well-known researchers Tetsuo Asano of JAIST, Japan, Subir Kumar Ghosh of TIFR, India, Giuseppe Liotta of University of Perugia, Italy and János Pach of EPFL Lausanne, Switzerland and Rényi Institute Budapest, Hungary.

As editors of this proceedings, we would like to thank all the authors who submitted their papers to WALCOM 2010. Our sincere appreciation goes to the invited speakers for joining us and presenting their talks on recent research areas of computer science from which researchers of this field will be immensely benefited. We thank the members of the Program Committee and external reviewers for their wonderful job in reviewing the manuscripts. We acknowledge the Steering Committee members for their continuous encouragement. We also thank the advisory committee members M. Shamsher Ali, Naiyyum Choudhury and A.M.M. Safiullah for their inspiring support to this workshop. We are indebted to the Organizing Committee led by M. Kaykobad and Md. Monirul Islam for their excellent services that made the workshop a grand success. We thank M.A. Mazed for his prompt organizational support and appreciate Debajyoti and Rahnuma for their tireless effort for the workshop.

We would like to thank Springer for publishing these proceedings in their prestigious LNCS series. This workshop is in cooperation with Technical Committee on Computation, IEICE, and Special Interest Group for Algorithms, IPSJ. We acknowledge the EasyChair conference system—a free conference management

system that is flexible, easy to use, and has many features to make it suitable for various conference models. Finally, we thank our sponsors for their assistance and support.

February 2010

Md. Saidur Rahman
Satoshi Fujita

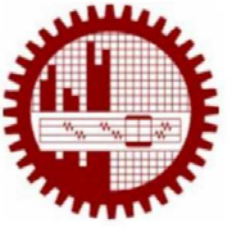
WALCOM Organization

WALCOM Steering Committee

Kyung-Yong Chwa	KAIST, Korea
Costas S. Iliopoulos	KCL, UK
M. Kaykobad	BUET, Bangladesh (Convenor)
Petra Mutzel	TU Dortmund, Germany
Shin-ichi Nakano	Gunma University, Japan
Subhas Chandra Nandy	Indian Statistical Institute, Kolkata, India
Takao Nishizeki	Tohoku University, Japan
Md. Saidur Rahman	BUET, Bangladesh
C. Pandu Rangan	IIT, Madras, India

WALCOM 2010 Organizers

BANGLADESH
ACADEMY OF SCIENCES



BANGLADESH
UNIVERSITY OF
ENGINEERING AND
TECHNOLOGY (BUET)

WALCOM 2010 Committees

Advisory Committee

M. Shamsher Ali	President, BAS
Naiyyum Choudhury	Secretary, BAS
A.M.M. Safiullah	Vice-Chancellor, BUET

Program Committee

Tetsuo Asano	JAIST, Japan
Therese Biedl	University of Waterloo, Canada
Sandip Das	Indian Statistical Institute, Kolkata, India
Hubert de Fraysseix	CNRS, France
Satoshi Fujita	Hiroshima University, Japan (Co-chair)
Subir Kumar Ghosh	TIFR, India
Ming-Yang Kao	Northwestern University, USA
Giuseppe Liotta	University of Perugia, Italy
Alejandro López-Ortiz	University of Waterloo, Canada
Meena Mahajan	The Institute of Mathematical Science, Chennai, India
Brendan D. McKay	Australian National University, Australia
Petra Mutzel	TU Dortmund, Germany
Hiroshi Nagamochi	Kyoto University, Japan
Shin-ichi Nakano	Gunma University, Japan
Subhas Chandra Nandy	Indian Statistical Institute, Kolkata, India
János Pach	EPFL Lausanne, Switzerland and Rényi Institute Budapest, Hungary
Leonidas Palios	University of Ioannina, Greece
Tomasz Radzik	King's College London, UK
Md. Saidur Rahman	BUET, Bangladesh (Co-chair)
William F. Smyth	McMaster University, Canada and Curtin University, Australia
Anand Srivastav	CAU Kiel, Germany
Takeshi Tokuyama	Tohoku University, Japan
Ryuhei Uehara	JAIST, Japan
Hsu-Chun Yen	National Taiwan University, Taiwan
W. Zang	The University of Hong Kong, Hong Kong

Organizing Committee

Reaz Ahmed	Muhammad Jawaherul Alam
Syed Ishtiaque Ahmed	Muhammad Masroor Ali
Shah Md. Rifat Ahsan	Md. Tanvir Al Amin
Md. Mostofa Akbar	Md. Faizul Bari

Sukarna Barua
Md. Shamsuzzoha Bayzid
Md. Shariful Islam Bhuyan
Naiyyum Choudhury
Shihabur Rahman Chowdhury
Anupam Das
Rajkumar Das
Masud Hasan
Mojahedul Hoque Abul Hasnat
A.S.M. Latiful Hoque
Md. Iqbal Hossain
Shahrear Iqbal
Md. Monirul Islam (Co-chair)
Md. Monirul Islam
Mohammad Mahfuzul Islam
Nusrat Sharmin Islam
Md. Humayun Kabir
Md. Rezaul Karim
M. Kaykobad (Co-chair)
M.A. Mazed
Momenul Islam Milton
Debajyoti Mondal
Md. Abu Sayeed Mondol
Tanaeem Muhammad Moosa
Mahmuda Naznin
Rahnuma Islam Nishat
Suraiya Parveen
Md. Anindya Tahsin Prodhan
A.K.M. Ashikur Rahman
M. Sohel Rahman
Md. Saidur Rahman (Secretary)
Md. Shaifur Rahman
Md. Wasi-ur-Rahman
Arup Raton Roy
Md. Abdus Sattar
Khaled Mahmud Shahriar
Nashid Shahriar
Rifat Shahriyar
Sadia Sharmin

External Reviewers

Alam, Muhammad Jawaherul
Binucci, Carla
Bishnu, Arijit
Claude, Francisco
Cohen, Elad
Cooper, Colin
Datta, Samir
Di Giacomo, Emilio
Didimo, Walter
Dorrigiv, Reza
El Ouali, Mourad
Fraser, Robert
Grilli, Luca
Islam, Md. Monirul
Jäger, Gerold
Karim, Md. Rezaul
Karmakar, Arindam
Kliemann, Lasse
Langetepe, Elmar
Langfeld, Barbara
Lin, Chun-Cheng
Misra, Neeldhara
Muthu, Rahul
Nimbhorkar, Prajakta
Romero, Jazmin
Ruiz Velasquez, Lesvia Elena
Salinger, Alejandro
Samee, Md. Abul Hassan
Sarma, Jayalal M.N.
Satti, Srinivasa Rao
Sauerland, Volkmar
Shibuya, Tetsuo
Sikdar, Somnath
Sun, Jonathan Z.
Zhao, Liang

WALCOM 2010 Sponsors

Table of Contents

Graph Algorithms I

Computational Biology and Strings

Combinatorial Optimization

Graph Algorithms II

Approximation Algorithms

Parameterized Complexity

Crossings between Curves with Many Tangencies

Jacob Fox[1,⋆], Fabrizio Frati[2], János Pach[3,⋆⋆], and Rom Pinchasi[4]

[1] Department of Mathematics, Princeton University, Princeton, NJ
jacobfox@math.princeton.edu
[2] Dipartimento di Informatica e Automazione, Roma Tre University, Italy
frati@dia.uniroma3.it
[3] EPFL Lausanne, Switzerland and Rényi Institute Budapest, Hungary
pach@cims.nyu.edu
[4] Mathematics Department,
Technion – Israel Institute of Technology, Haifa 32000, Israel
room@math.technion.ac.il

Abstract. Let $\mathcal{A}$ and $\mathcal{B}$ be two families of two-way infinite x-monotone curves, no three of which pass through the same point. Assume that every curve in $\mathcal{A}$ lies above every curve in $\mathcal{B}$ and that there are m pairs of curves, one from $\mathcal{A}$ and the other from $\mathcal{B}$, that are tangent to each other. Then the number of proper crossings among the members of $\mathcal{A} \cup \mathcal{B}$ is at least $(1/2 - o(1))m \ln m$. This bound is almost tight.

1 Introduction

Studying the incidence structure of a family of curves in the plane is a classical theme in combinatorial geometry with many applications in computational geometry. Venn diagrams were introduced in the 19th century to analyze logical relationships between various statements [9,7]. The incidence structure of non-overlapping circular disks was investigated by Koebe [2], while Erdős [4] raised several questions about tangencies between possibly overlapping congruent disks, including his famous problem on unit distances: How many pairs of points can be at distance one from each other in a set of n points in the plane? In other words, how many tangencies can occur among n unit diameter disks in the plane? These are hard questions, see [5] for a survey.

An equally tantalizing innocent-looking question was asked by Richter and Thomassen [6]. We say that two closed curves γ_1 and γ_2 in the plane *properly cross* if they share at least one point p (called a *crossing point*) such that γ_1 passes from one side to the other side of γ_2 in a small neighborhood of p. We say that two closed curves γ_1 and γ_2 in the plane *touch* or are *tangent to* each other, if they share exactly one point. This point is called the *point of tangency* of the two curves. We say that two closed curves are *intersecting* if they have at least

⋆ Research supported by an NSF Graduate Research Fellowship and a Princeton Centennial Fellowship.
⋆⋆ Research supported by NSF grant CCF-08-30272, by grants from NSA, OTKA, BSF, and SNF.

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 1–8, 2010.

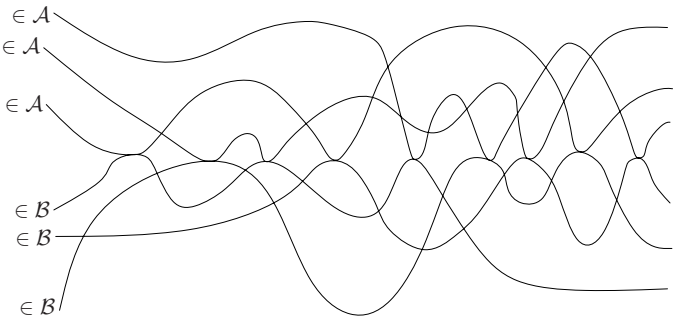


Fig. 1. Two intersecting families $\mathcal{A}$ and $\mathcal{B}$ of curves in general position such that no curve in $\mathcal{A}$ properly crosses any curve in $\mathcal{B}$

one point in common. A *family* $\mathcal{F}$ of closed curves is *intersecting* if every pair of them is intersecting. The family $\mathcal{F}$ is in *general position* if any *two* of its members share only a finite number of points and no *three* members pass through the same point. According to the Richter-Thomassen conjecture, any intersecting family of n closed curves in general position in the plane determines a total of at least $(1-o(1))n^2$ crossing points. This, of course, holds automatically if no two curves of the family touch each other, because then the number of crossing points is at least $2\binom{n}{2}$. Therefore, in order to settle the problem, we have to analyze families of curves with many tangencies.

In this note, we take the first step in this direction by studying the system of tangencies between *two* intersecting families $\mathcal{A}$ and $\mathcal{B}$ of curves in general position, with the property that no curve in $\mathcal{A}$ properly crosses any curve in $\mathcal{B}$ (see Fig. 1). In this case, we are going to prove that, if m denotes the number of pairs of touching curves (α, β) with $\alpha \in \mathcal{A}$ and $\beta \in \mathcal{B}$, the total number of crossing points in $\mathcal{F} = \mathcal{A} \cup \mathcal{B}$ divided by m tends to infinity, as $m \to \infty$. Consequently, if $|\mathcal{F}| = n$ and $m > \varepsilon n^2$ for some $\varepsilon > 0$, then the total number of crossing points in $\mathcal{F}$ is superquadratic in n.

For aesthetical reasons, we formulate our results for *two-way infinite x-monotone curves*, that is, for graphs γ_f of continuous functions $f : \mathbb{R} \to \mathbb{R}$. For simplicity, in the sequel, we use the term *curve* in this sense. We say that a curve γ_f lies *above* a curve γ_g if $f(x) \geq g(x)$ for all $x \in \mathbb{R}$.

For any family $\mathcal{F}$ of curves in general position, let $\mathrm{CN}(\mathcal{F})$ denote the number of crossing points.

Our main result is the following.

Theorem 1. *Let $\mathcal{A}$ and $\mathcal{B}$ be two families of two-way infinite x-monotone curves such that $\mathcal{A} \cup \mathcal{B}$ is in general position. Assume that every curve in $\mathcal{A}$ lies above every curve in $\mathcal{B}$ and that there are m pairs of curves, one from $\mathcal{A}$ and the other from $\mathcal{B}$, that touch. Then the sum of the numbers of proper crossings among the members of $\mathcal{A}$ and among the members of $\mathcal{B}$ satisfies*

$$CN(\mathcal{A}) + CN(\mathcal{B}) \geq \left(\frac{1}{2} - o(1)\right) m \ln m,$$

where the $o(1)$ *term goes to* 0 *as* m *tends to* ∞.

We say that $\mathcal{A}$ and $\mathcal{B}$ *completely touch* if every member of $\mathcal{A}$ touches every member of $\mathcal{B}$.

Theorem 2. *For every* $n > 2$*, there exist two completely touching* n*-member families* $\mathcal{A}$ *and* $\mathcal{B}$ *of two-way infinite* x*-monotone curves such that* $\mathcal{A} \cup \mathcal{B}$ *is in general position, every curve in* $\mathcal{A}$ *lies above every curve in* $\mathcal{B}$*, and*

$$CN(\mathcal{A}) + CN(\mathcal{B}) \leq \left(\frac{3}{4} + o(1)\right) n^2 \log_2 n.$$

Comparing Theorems 1 and 2, we obtain that if $c(n)$ denotes the minimum number of crossing points in the union $\mathcal{A} \cup \mathcal{B}$ of two completely touching n-member families of curves, $\mathcal{A}$ and $\mathcal{B}$, such that all the members of $\mathcal{A}$ are above all the members of $\mathcal{B}$, then we have:

$$(1 - o(1))n^2 \ln n \leq c(n) \leq \left(\frac{3}{4} + o(1)\right) n^2 \log_2 n = \left(\frac{3}{4 \ln 2} + o(1)\right) n^2 \ln n.$$

This shows that Theorem 1 is tight up to a multiplicative factor of roughly $\frac{3}{4 \ln 2} \approx 1.082$.

In Sections 2 and 3 of this note, we establish Theorems 1 and 2, respectively. In the final section, we make some concluding remarks. In particular, we formulate a combinatorial result of independent interest on alternations in certain sequences over finite alphabets (Theorem 3), which can also be used to prove Theorem 1.

2 Levels – Proof of Theorem 1

The *lower* k*-level* of a family $\mathcal{F}$ of curves is the closure of the set of all points that lie on exactly one member of $\mathcal{F}$ and strictly above exactly $k-1$ members (see Fig. 2). Let $\ell_k(\mathcal{F})$ denote the number of all proper crossings among members of $\mathcal{F}$ that lie on the lower k-level of $\mathcal{F}$. Analogously, the *upper* k*-level* of a family $\mathcal{F}$ of curves is the closure of the set of all points that lie on exactly one member of $\mathcal{F}$ and strictly below exactly $k-1$ members. Let $u_k(\mathcal{F})$ denote the number of all proper crossings among members of $\mathcal{F}$ that lie on the upper k-level of $\mathcal{F}$. Note that each proper crossing among two members of a family $\mathcal{F}$ of curves in general position lies on two consecutive levels, so that we have

$$\sum_{k=1}^{|\mathcal{F}|} \ell_k(\mathcal{F}) = \sum_{k=1}^{|\mathcal{F}|} u_k(\mathcal{F}) = 2\mathrm{CN}(\mathcal{F}). \tag{1}$$

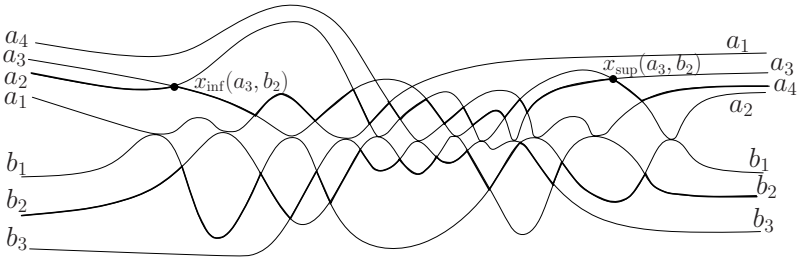


Fig. 2. Two families $\mathcal{A}$ and $\mathcal{B}$ of curves, with $|\mathcal{A}| = 4$ and $|\mathcal{B}| = 3$. The lower 2-level of $\mathcal{A}$ and the upper 2-level of $\mathcal{B}$ are shown by thick lines. Black dots show $x_{\inf}(a_3, b_2)$ and $x_{\sup}(a_3, b_2)$ when $k = 2$.

Theorem 1 can be easily deduced from the following lemma.

Lemma 1. *Let $k > 1$ and $\mathcal{A}$ and $\mathcal{B}$ be two families of two-way infinite x-monotone curves, each of cardinality at least k, such that $\mathcal{A} \cup \mathcal{B}$ is in general position. Assume that every curve in $\mathcal{A}$ lies above every curve in $\mathcal{B}$ and that there are m pairs of curves, one from $\mathcal{A}$ and the other from $\mathcal{B}$, that touch. Then, we have*

$$\ell_1(\mathcal{A}) + u_1(\mathcal{B}) \geq m - 1,$$

and

$$\ell_k(\mathcal{A}) + u_k(\mathcal{B}) \geq 2\frac{m}{k} - 4k.$$

Proof: We may assume without loss of generality that all crossing points between members of $\mathcal{A} \cup \mathcal{B}$ have distinct x-coordinates and that all of these values belong to the open interval $0 < x < 1$.

Note that, as x varies between the x-coordinates of two consecutive points at which a member of $\mathcal{A}$ touches a member of $\mathcal{B}$, the lowest curve of $\mathcal{A}$ or the highest curve of $\mathcal{B}$ must change. This yields the inequality

$$\ell_1(\mathcal{A}) + u_1(\mathcal{B}) \geq m - 1.$$

Fix $k > 1$. For any $0 \leq \xi \leq 1$ which is not the x-coordinate of an intersection point, let $\mathcal{A}_k(\xi)$ denote the kth lowest curve in $\mathcal{A}$ at the vertical line $x = \xi$ and let $\mathcal{B}_k(\xi)$ denote the kth highest curve in $\mathcal{B}$ at the vertical line $x = \xi$. Analogously, $\mathcal{A}_{\leq k}(\xi)$ denotes the family consisting of the k lowest curves in $\mathcal{A}$ at the vertical line $x = \xi$ and $\mathcal{B}_{\leq k}(\xi)$ denotes the family consisting of the k highest curves in $\mathcal{B}$ at the vertical line $x = \xi$.

For $\xi = 0$ or $\xi = 1$, the number of pairs $(a, b) \in \mathcal{A}_{\leq k}(\xi) \times \mathcal{B}_{\leq k}(\xi)$ is k^2. A pair $(a, b) \in \mathcal{A} \times \mathcal{B}$ is said to be *internally touching* if a and b touch each other and

$$(a, b) \notin (\mathcal{A}_{\leq k}(0) \times \mathcal{B}_{\leq k}(0)) \cup (\mathcal{A}_{\leq k}(1) \times \mathcal{B}_{\leq k}(1)).$$

Let I stand for the number of internally touching pairs (a, b). Clearly, we have $I \geq m - 2k^2$. For any internally touching pair (a, b), let (see Fig. 2)

1. $x_{\inf}(a,b)$ be the infimum of all x-values for which $\mathcal{A}_k(x)=a$ and $b\in\mathcal{B}_{\le k}(x)$, or $a\in\mathcal{A}_{\le k}(x)$ and $\mathcal{B}_k(x)=b$, and let
2. $x_{\sup}(a,b)$ be the supremum of all x-values for which $\mathcal{A}_k(x)=a$ and $b\in\mathcal{B}_{\le k}(x)$, or $a\in\mathcal{A}_{\le k}(x)$ and $\mathcal{B}_k(x)=b$.

Obviously, we have $x_{\inf}(a,b)<x_{\sup}(a,b)$ as the x-coordinate of the touching point between a and b lies strictly between these two numbers. It is also clear that the numbers $x_{\inf}(a,b)$ and $x_{\sup}(a,b)$ are x-coordinates of crossing points lying on the kth lowest level of $\mathcal{A}$ or on the kth highest level of $\mathcal{B}$.

For any $0<\xi<1$, there are at most k internally touching pairs (a,b) with $x_{\inf}(a,b)=\xi$. Indeed, for any $a\in\mathcal{A}$ such that $a=\mathcal{A}_k(\xi+\varepsilon)$, say, for all sufficiently small $\varepsilon>0$, all curves $b\in\mathcal{B}$ with $x_{\inf}(a,b)=\xi$ must belong to the set $\mathcal{B}_{\le k}(\xi)$. This is a set of size k. Thus, the number of distinct x-coordinates ξ at which either $\mathcal{A}_k(\xi)$ or $\mathcal{B}_k(\xi)$ changes is at least $2I/k$. That is, we have

$$\ell_k(\mathcal{A})+u_k(\mathcal{B})\ge\frac{2I}{k}\ge 2\frac{m-2k^2}{k}=2\frac{m}{k}-4k. \qquad \square$$

A similar argument was used in [1].

Now we are in a position to establish Theorem 1.

Proof of Theorem 1: Assume without loss of generality that $|\mathcal{A}|\ge|\mathcal{B}|$ and that every curve in $\mathcal{A}\cup\mathcal{B}$ participates in at least one touching pair. This implies that any two members of $\mathcal{A}$ properly cross at least once and any two members of $\mathcal{B}$ properly cross at least once. Hence, we have

$$\mathrm{CN}(\mathcal{A})+\mathrm{CN}(\mathcal{B})\ge\binom{|\mathcal{A}|}{2}+\binom{|\mathcal{B}|}{2}.$$

This completes the proof in the special case where $m\le|\mathcal{A}|^2/\ln|\mathcal{A}|$, because then the term $\binom{|\mathcal{A}|}{2}$ already exceeds the desired lower bound. In particular, since the total number m of touching pairs is at most $|\mathcal{A}||\mathcal{B}|$, we are done if $|\mathcal{B}|\le|\mathcal{A}|/\ln|\mathcal{A}|$.

From now on, we can assume that

$$m>|\mathcal{A}|^2/\ln|\mathcal{A}|$$

and

$$|\mathcal{A}|/\ln|\mathcal{A}|\le|\mathcal{B}|\le|\mathcal{A}|.$$

Let $\varepsilon>0$ be a very small constant. Set $K=m^{\frac{1}{2}-\varepsilon}$, and add up $\ell_k(\mathcal{A})+u_k(\mathcal{B})$ for all $1\le k\le K$. Note that we can apply Lemma 1, since the last two inequalities imply that $K\le|\mathcal{B}|$. In view of (1), we obtain

$$\begin{aligned}\mathrm{CN}(\mathcal{A})+\mathrm{CN}(\mathcal{B})&\ge\frac{1}{2}\sum_{k=1}^{K}(\ell_k(\mathcal{A})+u_k(\mathcal{B}))\ge\frac{1}{2}\left(m-1+\sum_{k=2}^{K}\left(2\frac{m}{k}-4k\right)\right)\\&\ge\frac{1}{2}\left(m-2K(K+1)+3+2m\sum_{k=2}^{K}\frac{1}{k}\right)=\left(\frac{1}{2}-\epsilon-o(1)\right)m\ln m.\end{aligned}$$

Letting $\epsilon \to 0$, we can conclude that $\mathrm{CN}(\mathcal{A})+\mathrm{CN}(\mathcal{B})$ is at least $(\frac{1}{2}-o(1))m\ln m$, as required. □

3 Constructive Upper Bound – Proof of Theorem 2

Let $c(n)$ denote the minimum number of crossing points in the union of any two completely touching n-member families of curves $\mathcal{A}\cup\mathcal{B}$, where all members of $\mathcal{A}$ are above all members of $\mathcal{B}$.

We need the following:

Lemma 2. *For any pair of positive integers i and j, we have*

$$c(ij) \le i^2 c(j) + j^2 c(i).$$

Proof: Let $(\mathcal{A}', \mathcal{B}')$ be a pair of completely touching i-member families of curves with

$$\mathrm{CN}(\mathcal{A}') + \mathrm{CN}(\mathcal{B}') = c(i).$$

Replace each curve $\gamma \in \mathcal{A}'\cup\mathcal{B}'$ by j curves that closely follow γ. For any $\alpha \in \mathcal{A}'$ and for any $\beta \in \mathcal{B}'$, let each of the j curves corresponding to α touch each of the j curves corresponding to β in a small neighborhood of the point where α and β touch each other. This can be achieved by introducing $c(j)$ crossings near each point of tangency between α and β. Apart from the crossings introduced in the neighborhoods of these points, the j new curves corresponding to an "old" curve $\gamma \in \mathcal{A}'\cup\mathcal{B}'$ are disjoint.

Denote the family of ij curves obtained from the members of $\mathcal{A}'$ by $\mathcal{A}$, and the family of ij curves obtained from $\mathcal{B}'$ by $\mathcal{B}$. Since the number of tangencies between $\mathcal{A}'$ and $\mathcal{B}'$ is i^2, there are at most $i^2c(j)$ crossings among the members of $\mathcal{A}\cup\mathcal{B}$ that occur near these touching points. On the other hand, in a small neighborhood of each crossing between two members of $\mathcal{A}'$ or two members of $\mathcal{B}'$, we create j^2 crossings in $\mathcal{A}$ or in $\mathcal{B}$. Therefore, there are $j^2c(i)$ crossings among members of $\mathcal{A}\cup\mathcal{B}$ that occur near crossings in $\mathcal{A}'$ or $\mathcal{B}'$. In view of the fact that each crossing in $\mathcal{A}\cup\mathcal{B}$ occurs in a small neighborhood of either a touching point or a crossing point in $\mathcal{A}'\cup\mathcal{B}'$, we obtain that $c(ij) \le \mathrm{CN}(\mathcal{A}) + \mathrm{CN}(\mathcal{B}) \le i^2c(j) + j^2c(i)$, as required. □

Using the fact $c(2) = 3$, by repeated application of Lemma 2 with $j = 2$ and $i = 2, 2^2, \ldots, 2^{k-1}$. we obtain that $c(2^k) \le \frac{3}{4}k4^k$. Starting with a completely touching pair of 2-member families of curves, after $k-1$ iterations we obtain a completely touching pair $(\mathcal{A}, \mathcal{B})$ of 2^k-member families with $m = 2^{2k}$ touching pairs. Thus, there exists a configuration with only $\frac{3}{4}k4^k = \frac{3}{8}m\log_2 m$ crossings, meeting the requirements. This completes the proof of Theorem 2.

4 Concluding Remarks

The assumption in Theorem 1 that the curves are two-way infinite is not important. If we have a family $\mathcal{F} = \mathcal{A} \cup \mathcal{B}$ of arbitrary x-monotone curves such that, for any pair of curves $\alpha \in \mathcal{A}$, $\beta \in \mathcal{B}$ which can be met by a vertical line, α lies above β, we can make each curve *two-way infinite* without destroying this property, by adding only at most

$$2\binom{|\mathcal{A}|}{2} + 2\binom{|\mathcal{B}|}{2} < |\mathcal{F}|^2$$

crossings.

One can give an alternative proof of Theorem 1 by reducing it to a combinatorial statement about sequences. Let $(x_1, \ldots, x_m)$ be a sequence of m elements taken from a finite alphabet Φ. For any pair of distinct elements $a, b \in \Phi$, define the number of *alternations* of a and b in the sequence, as the largest number t such that there is a subsequence $(x_{i(0)}, x_{i(1)}, \ldots, x_{i(t)})$ of length $t+1$ with $1 \leq i(0) < i(1) < \ldots < i(t) \leq m$ such that its elements alternate between a and b (or between b and a). That is,

$$x_{i(0)} = x_{i(2)} = \ldots = a, \qquad x_{i(1)} = x_{i(3)} = \ldots = b,$$

or

$$x_{i(0)} = x_{i(2)} = \ldots = b, \qquad x_{i(1)} = x_{i(3)} = \ldots = a.$$

This number t is denoted by $\mathrm{alt}_{\{a,b\}}(x_1, \ldots, x_n)$.

Define the *alternation number* of the sequence $(x_1, \ldots, x_m)$, as

$$\sum_{\{a,b\} \subseteq \Sigma} \mathrm{alt}_{\{a,b\}}(x_1, \ldots, x_m),$$

where the sum is taken over all unordered pairs $\{a, b\}$ of distinct elements from Φ.

Theorem 1 can also be proved using the following result, which is perhaps of independent interest.

Theorem 3. *Let $(x_1, \ldots, x_m)$ be a sequence of length m over an alphabet Φ. Assume that there exists an absolute constant $c > 0$ such that for all $1 \leq z \leq m$, every z consecutive elements of the sequence contain at least $c\sqrt{z}$ distinct symbols. Then the alternation number of the sequence $(x_1, \ldots, x_m)$ is at least $dm \log m$, for a suitable constant $c' > 0$, depending only on c.*

Salazar [8] verified the Richter-Thomassen conjecture in the special case when any pair of curves have at most k points in common, for a fixed constant k. The best known general bound is due to Mubayi [3], who proved that any family of n closed curves in general position in the plane determines at least $\left(\frac{4}{5} + o(1)\right) n^2$ intersection points.

References

1. Chan, T.M.: On levels in arrangements of curves, II: A simple inequality and its consequences. Discrete & Computational Geometry 34(1), 11–24 (2005)
2. Koebe, P.: Kontaktprobleme der konforman abbildung. Berichte über die Verhandlungen d. Sächs. Akademie der Wissenschaften Leipzig 88, 141–164 (1936)
3. Mubayi, D.: Intersecting curves in the plane. Graphs and Combinatorics 18(3), 583–589 (2002)
4. Erdös, P.: On sets of distances of n points. The American Mathematical Monthly 53, 248–250 (1946)
5. Pach, J., Agarwal, P.K.: Combinatorial Geometry. Wiley, New York (1995)
6. Richter, R.B., Thomassen, C.: Intersection of curves systems and the crossing number of $C_5 \times C_5$. Discrete & Computational Geometry 13, 149–159 (1995)
7. Ruskey, F., Weston, M.: Venn diagram survey. Electronic Journal of Combinatorics DS#5 (2005)
8. Salazar, G.: On the intersections of systems of curves. Journal of Combinatorial Theory Series B 75, 56–60 (1999)
9. Venn, J.: On the diagrammatic and mechanical representation of propositions and reasonings. Philosophical Magazine and Journal of Science, Series 5 10(59) (1880)

Constant-Work-Space Algorithm for a Shortest Path in a Simple Polygon

Tetsuo Asano[1], Wolfgang Mulzer[2], and Yajun Wang[3]

[1] School of Information Science, JAIST, Japan
[2] Department of Computer Science, Princeton University, USA
[3] Microsft Research, Beijing, China

Abstract. We present two space-efficient algorithms. First, we show how to report a simple path between two arbitrary nodes in a given tree. Using a technique called "computing instead of storing", we can design a naive quadratic-time algorithm for the problem using only constant work space, i.e., $O(\log n)$ bits in total for the work space, where n is the number of nodes in the tree. Then, another technique "controlled recursion" improves the time bound to $O(n^{1+\varepsilon})$ for any positive constant ε. Second, we describe how to compute a shortest path between two points in a simple n-gon. Although the shortest path problem in general graphs is NL-complete, this constrained problem can be solved in quadratic time using only constant work space.

1 Introduction

We present two polynomial-time algorithms in a computational model which we call *constant-work-space computation*, which is also known as "log-space" algorithms. In this model, the input is given as a read-only array, and the algorithm can access an arbitrary array element in constant time, This is a difference from the strict data-streaming model where the input can be read only once in a sequential manner. Chan and Chen [7] give algorithms in different computational models varying from a multi-pass data-streaming model to the random access constant-work-space model in our paper.

One of the most important constant-work-space algorithms is a selection algorithm by Munro and Raman [9] which runs in $O(n^{1+\varepsilon})$ time using work space $O(1/\varepsilon)$ for any small constant $\varepsilon > 0$. A polynomial-time algorithm for determining connectivity of two arbitrarily specified nodes in a graph by Reingold [10] is also another breakthrough in this area. See also [2,3,4,5] for applications to image processing. Constant-work-space algorithms for geometric problems are also known. Asano and Rote [1] give efficient algorithms for drawing Delaunay triangulation and Voronoi diagram of a planar point set, and they also show how the Euclidean minimum spanning tree for a planar point set can be constructed quickly in this model.

Here, we focus on the efficiency of algorithms in the constant work space model. Using two geometric problems we showcase some techniques for designing space-efficient algorithms. One technique, named " **computing instead of**

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 9–20, 2010.

storing", is applied to the problem of finding a simple path between two nodes in a tree. A simple solution in a standard computational model with linear work space goes as follows: compute an Eulerian path between the two nodes and count how often each edge appears on the path. Removing those edges that appear more than once gives us a desired simple path. We can implement this idea without using any extra array. Instead of storing a count in each edge, we compute it whenever it is needed, which takes linear time. In this way we can compute a simple path in quadratic time without using any extra array.

Then we describe another technique named "**controlled recursion**" which limits the depth of recursion by a predetermined value determined by the amount of work space. Using this technique the running time of the algorithm is improved into $O(n^{1+\varepsilon})$ for any small positive constant ε using work space of size $O(1/\varepsilon)$.

The above algorithms can be extended to an algorithm for finding a shortest path between two points in a simple polygon. A naive application leads to a polynomial-time algorithm, but a more careful implementation yields a quadratic-time algorithm.

2 Finding a Simple Path on a Tree Using Eulerian Tours

As a warm up, consider a simple problem: Let T be a tree with n nodes. Given two nodes s and t, find a simple path with no node visited more than once from s to t. This is our first problem.

Here is a simple naive algorithm. It is well known that any tree has an Eulerian tour visiting every edge exactly twice. Let A be such a tour. A simple path between s and t is obtained by considering the portion of A between s and t and removing redundant edges where an (undirected) edge is redundant if it appears twice on A. Thus, if we know how to generate an Eulerian tour, it is easy to reform a part of the tour into a simple path by counting the number of occurrences of each edge. Unfortunately, in our constant work space model no extra array can be used for the counts.

Thus, we apply the technique "**computing instead of storing**" in which whenever we need a value we compute it instead of storing it. Whenever we extend a path by an edge e to generate an Eulerian path between two nodes, we generate the Eulerian path to count how often the edge e appears. If it appears exactly once, we report the edge.

We introduce some terminology for a formal description of the algorithm. We assume that a tree is given by adjacency lists on a read-only array. Let $\mathrm{Adj}(u)$ be the adjacency list of a node u. The following two functions suffice to generate an Eulerian tour.

FirstNeighbor(u)**:** given a node u, return the first node in the adjacency list $\mathrm{Adj}(u)$.

NextNeighbor(u, v)**:** Given a node u and one of its adjacent nodes, say v, return the next node in its adjacency list $\mathrm{Adj}(u)$. If v is the last node, return the first node in the list.

The function **FirstNeighbor** can easily be performed in constant time, but the time required for **NextNeighbor** depends on which data structure we assume. If the tree is given by a doubly-connected edge list [6], then it takes constant time. If a naive data structure is assumed, we may need to search all of $\mathrm{Adj}(u)$ for the next element, which takes time $O(\Delta)$ where Δ is the maximum degree of a node in the tree.

Given a tree T, a starting node s, and a target node t, the following function **FindFeasibleSubtree** finds the subtree of s which contains t, in other words, it tells us which edge to follow toward the target t. In the algorithm we successively find the next edge following an Eulerian path starting from s. We start from an edge (s, u) incident to s and follow an Eulerian path by applying the function **NextNeighbor**. If we come back to its twin edge (u, s) before finding the target node t, it means the subtree of s rooted at u does not contain the target node t.

Algorithm 1. Finding a simple path from s to t.

Input: A tree T and two nodes s and t in T.
Output: A simple path from s to t.
begin
  `currentNode` $= s$;
  repeat
    report `currentNode`;
    `currentNode` $=$ FindFeasibleSubtree(`currentNode`);
  until *currentNode* $= t$
end
function FindFeasibleSubtree(u, t) // returns a child of u whose subtree contains t
begin
  for *each node* v *in* $\mathrm{Adj}(u)$ **do**
    if *SubtreeSearch*(u, v, t) **then return** v;
end
function SubtreeSearch(u, v, t) // checks whether the subtree of u rooted at v contains t
begin
  `currentNode` $= u$; `neighbor` $= v$;
  repeat
    `nextNode` $=$NextNeighbor(`neighbor`, `currentNode`);
    `currentNode` $=$ `neighbor`; `neighbor` $=$ `nextNode`;
  until *(currentNode* $= t$ *or (currentNode* $= v$ *and neighbor* $= u$*))*
  return (`currentNode` $= t$);
end

Lemma 1. *Given a tree T, a starting node s, and a target node t, **Algorithm 1** reports a simple path from s to t in $O(n^2 d)$ time using only constant work space, where n is the number of nodes of T and depends on which data structure is used: if the tree is given by a doubly-connected edge list then $d = O(1)$. If a naive data structure is assumed then $d = O(\Delta)$, where Δ is the maximum node degree in T.*

Figure 1 illustrates how the search proceeds.

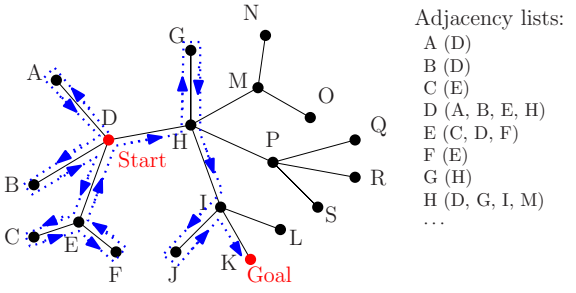


Fig. 1. Canonical traversal of a tree given by adjacency lists

We have shown that a simple path on a tree can be found in quadratic time if a tree is represented by an appropriate data structure. Further improvement on its running time is possible. A key idea is a "**controlled recursion**," which was also used by Munro and Raman [9] for finding a median . The controlled recursion is an algorithmic technique which controls the recursion depth so that the depth never exceeds a predetermined constant, which reflects the amount of work space.

Suppose $O(k)$ work space is available. Then, we design algorithms, $\mathcal{A}_1$, $\mathcal{A}_2$, ..., $\mathcal{A}_k$ such that $\mathcal{A}_i$ calls $\mathcal{A}_{i-1}$ for $i = k, k-1, \ldots, 2$. As describe above, the algorithm $\mathcal{A}_1$, finds a simple path between two nodes in a tree by removing redundant edges in quadratic time.

The algorithm $\mathcal{A}_2$ uses a decomposition of the Eulerian path from s to t. Let $s = u_0, u_1, \ldots, u_m = t$ be the Eulerian path from s to t, which is a sequence of nodes in which a node may appear more than once. We decompose the path into $O(\sqrt{m})$ blocks, $B_1, B_2, \ldots, B_{\sqrt{m}}$. For the first block $B_1 = \{s = u_0, u_1, \ldots, u_{\sqrt{m}} = v\}$ we find the lowest common ancestor of v and t in B_1 using a binary search. The binary search starts at a node x which is the $\sqrt{m}/2$-th step from u_0 toward v. Now we have three nodes x, v, and t. For each of them we compute its first occurrence and the last occurrence, denoted by $F(x), L(x), F(v), L(v), F(t)$ and $L(t)$, respectively. As is easily seen, the node x is a common ancestor of v and t if and only if $F(x) < F(v) \leq L(v) \leq F(t) \leq L(t) < L(x)$ holds. In the binary search, if $F(v) < F(x)$, that is, if we visit the node v before x when we walk along the Eulerian path from s to t, then we have to walk back to satisfy $F(x) < F(v)$. If $F(x) < F(v)$ but $L(x) < L(t)$, then we have to walk toward t to satisfy $L(t) < L(x)$. In each test we halve the number of steps starting from $\sqrt{m}/2$. Whenever we find a common ancestor of v and t, we compare it with the current lowest common ancestor of v and t. It is easy since it suffices to compare the number of steps from s. The node of the longer steps is lower than the other. Then, we walk toward t to find a possible lower common ancestor.

In this way we can find the lowest common ancestor u of v and t. Each step of the binary search is done in $O(n)$ time for the Eulerian tour from s to s. Thus, it is done in $O(n \log n)$ time in total.

Finally, we compute a simple path from s to u, which is the initial part of the simple path from s to t. The simple path is computed by removing redundant edges from the Eulerian path from s to u. Here note that its length is at most $\sqrt{m}$. Thus, if we use the function $\mathcal{A}_1$, it returns the simple path in $O(\sqrt{m}^2) = O(m) = O(n)$ time.

In the next block B_2 we can start from the lowest common ancestor we just computed. In the same way we can extend the simple path toward t. In this way we extend the simple path $O(\sqrt{m})$ times. Since each iteration is done in $O(n \log n)$ time, the total time we need is $O(n^{3/2} \log n)$.

The algorithm $\mathcal{A}_3$ partitions the Eulerian path from s to t into $O(n^{1/3})$ blocks and finds the lowest common ancestor by the binary search, and finally returns the Eulerian path from the starting node to the lowest common ancestor by using the algorithm $\mathcal{A}_2$. The first part is done in $O(n^{4/3} \log n)$ time and the second part is done in $O((n^{2/3})^{3/2} \log n^{2/3} n^{1/3}) = O(n^{4/3} \log n)$ time in total. Thus, it runs in $O(n^{4/3} \log n)$ time.

The algorithm $\mathcal{A}_k$ partitions the Eulerian path from s to t into $n^{1/(k+1)}$ parts of length $O(n^{k/(k+1)})$. It runs in $O(n^{1+1/(k+1)} \log^{k-1} n)$.

Lemma 2. *Algorithm $\mathcal{A}_k$ finds the simple path between any two nodes in a tree of size n stored in a read-only storage by doubly-connected edge lists in $O(n^{1+1/(k+1)} \log^k n)$ time using $O(k)$ work storage.*

Proof. The lemma is easily proved using induction on k. Note that the amount of work space is now $O(k)$ instead of constant. It is because Algorithm $\mathcal{A}_k$ successively calls algorithms $\mathcal{A}_{k-1}$, $\mathcal{A}_{k-2}$, $\mathcal{A}_1$ in order. □

Now, if we set

$$\varepsilon = \frac{1}{k+1}, \tag{1}$$

then the time complexity of Algorithm $\mathcal{A}_k$ is $O(n^{1+\varepsilon} \log^{1/\varepsilon} n / \log n)$. Suppose we have

$$n^{\varepsilon} = \log^{1/\varepsilon} n. \tag{2}$$

Then, we have

$$\varepsilon = \sqrt{\frac{\log \log n}{\log n}}.$$

Now, the time complexity of Algorithm $\mathcal{A}_k$ is

$$O(n^{1+2\varepsilon}). \tag{3}$$

This means the following theorem.

Theorem 1. *Given two nodes s and t in a tree T with n nodes in a read-only storage and any positive constant δ, there is an algorithm which finds the simple path from s to t in T in $O(n^{1+\delta})$ time using only $O(1/\delta)$ amount of work space.*

In the theorem we assumed a doubly-connected edge list for a given tree. If the tree is given in a simple list, then the basic operation to find the next or previous edge takes time proportional to the length of the adjacency list. Thus, the time complexity becomes $O(n^{1+\delta}\triangle)$, where $\triangle$ is the maximum node degree in T.

3 Shortest Paths in Polygons

Dijkstra's algorithm for finding a shortest path between two specified vertices in a weighted graph is one of the most popular and important algorithms. It can find such an path in $O(n^2)$ time using a simple data structure that maintains the current distance from a source vertex to each vertex during the progress of the algorithm. Is it still possible to find such a shortest path in the constant-work-space model where the input graph is given by a read-only array and only a constant number of storage cells of length $O(\log n)$ is available as work space? Unfortunately, no polynomial-time algorithm for the shortest path problem is known in the model. In this paper we consider a restricted version of the problem:

Geometric Shortest Path within a Simple Polygon
Given a simple polygon P with n vertices and two points s and t in the interior of P, find the shortest path between s and t within the polygon P.

A linear-time algorithm is known for the problem if $O(n)$ work space is allowed. It works as follows: Given a simple polygon P, we first partition its interior into triangles using Chazelle's linear-time algorithm [8]. Then, we compute the dual graph G^* of the triangulation: the vertices of G^* correspond to the triangles, and two vertices are adjacent if their corresponding triangles share a triangular edge. Since G^* is a tree, any two vertices in G^* are connected by a unique simple path. Given two points s and t to be interconnected, we locate them in the triangulation and thus in G^*. Consider the unique path in G^* between the triangle containing s to the triangle containing t. It defines a sequence of triangular edges hit by the path. Let $(e_0, e_1, \ldots, e_m)$ be this edge sequence. We walk along the sequence while keeping the visibility angle from the starting point s and two vertices v_{low} and v_{high} that determine the visibility.

Whenever the visibility angle vanishes at a triangular edge, we choose a vertex v, either v_{low} or v_{high}, depending on which direction the path bends, and output the edge (s, v) as a part of the shortest path and repeat the same operation after replacing s with v. Obviously, every step is done in constant time. Thus, the algorithm runs in $O(n)$ time.

3.1 A Shortest-Path Algorithm Using a Dual Graph

We adapt the algorithm to use constant work space. That is, we develop an algorithm for triangulating a given simple polygon and then finding a unique path in

the dual graph. The difficulty here is, of course, that we cannot store any intermediate result. To overcome the difficulty, we will use a canonical triangulation of a simple polygon and also a canonical traversal of the tree.

Our canonical triangulation is the *constrained Delaunay triangulation*. For a point set S, three points of S determine a *Delaunay triangle* if and only if the circle defined by the three points contains no point of S in its proper interior. Such a circle passing through three points of S is called an *empty circle*. Delaunay triangles partition the convex hull of the set S, and the resulting structure is called the *Delaunay triangulation* of S.

Now we describe how to extend this notion to a simple polygon P. The vertices of P define a set V of points and the edges define a set E of line segments. Constrained Delaunay edges are defined using the notion of a *chord*. A chord is an open line segment between two polygon vertices that does not intersect the boundary any edge in E. A pair (p, q) of vertices defines a constrained Delaunay edge if and only if there is a third point r in V such that(i) (p, q) is a chord; (ii) (p, r) and (q, r) are chords or polygon edges; and (iii) the circle through p, q, r does not contain any other point $s \in V$ that is visible from r.

It is known that a constrained Delaunay triangulation $\mathrm{DT}(P)$ is uniquely defined for any simple polygon whose vertices are in general position. Once we have a $\mathrm{DT}(P)$, we define its dual graph $\mathrm{DT}(P)^*$: vertices are triangles, and two vertices are adjacent if and only if their corresponding triangles share an edge. Since a simple polygon is simply connected, $\mathrm{DT}(P)^*$ is always a tree.

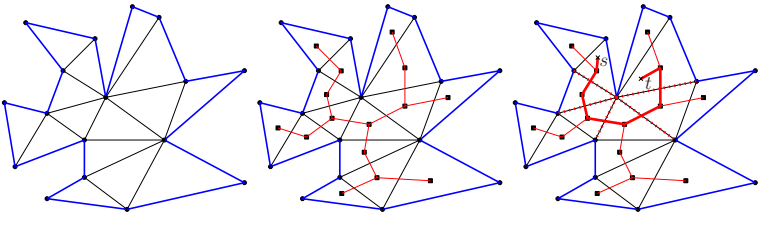


Fig. 2. The unique path on the dual graph and its corresponding sequence of Delaunay edges. (a) Constrained Delaunay triangulation of a given simple polygon, (b) the dual graph of the triangulation (a tree), and (c) the unique path between s and t.

Once we have a path in $\mathrm{DT}(P)^*$, we walk along the path while extending the visibility angle. Let $(e_0, e_1, \ldots, e_m)$ be an edge sequence corresponding to the path. We first compute the visibility angle defined by the starting point s and the first edge e_0. Then, we take the intersection between the current visibility angle with that defined by the next edge e_1. We keep the intersection as the current visibility angle. If the intersection becomes empty, we know that the line segment between the current vertex and previous starting point must be a part of the shortest path. So, we output that line segment and then we start the new

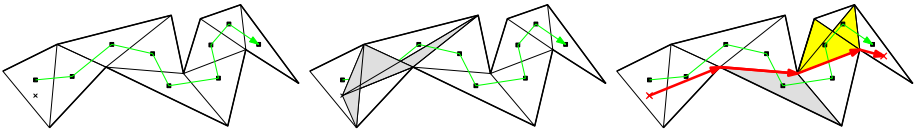


Fig. 3. Finding a shortest path along a sequence of Delaunay edges. (a) The unique path in the dual graph, (b) extension of visibility region from the starting point until it vanishes, and (c) shortest path within the simple polygon.

propagation of the visibility angle from the current vertex. Figure 3 illustrates how this algorithm proceeds.

Finally, we need to describe how to implement the function **NextNeighbor**() for $\mathrm{DT}(P)^*$. By the definition of the dual graph and the fact that each $\mathrm{DT}(P)^*$ has maximum degree at most 3, the next neighbor is found by finding the clockwisely or or counter-clockwisely next Delaunay edge as shown in Figure 4. Hence, we need to find a third vertex of a Delaunay triangle for a given edge. Given a Delaunay edge (u, v), we want to find a vertex w such that

(1) w is visible from the edge (u, v), and
(2) the circle defined by three points u, v, and w is empty, that is, it does not contain any other vertex visible from the edge (u, v).

A vertex w is visible from the edge (u, v) when there is no edge intersecting a line segment $\overline{uw}$ or $\overline{vw}$. Thus, we can find in $O(n^2)$ time a vertex with which a Delaunay edge forms a Delaunay triangle.

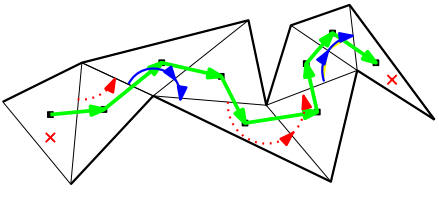


Fig. 4. Walking along a path in the dual graph while finding the clockwisely next Delaunay edge (solid or blue arrow) or counterclockwisely next edge (dotted or red arrow)

Theorem 2. *There is a constant-work space algorithm for finding a shortest path between arbitrary two points in a simple n-gon P in time $O(n^{3+\varepsilon})$ for any small constant $\varepsilon > 0$.*

Proof. Such a shortest path can be found by applying the function **NextNeighbor**() $O(n^{1+\varepsilon})$ times. Since it takes $O(n^2)$ time for each call of the function, we obtain the bound in the theorem. □

3.2 A Shortest-Path Algorithm Using Point Location

The algorithm given above is obtained by a direct adaptation of the algorithm for reporting a simple path between two nodes in a tree. The dual graph, which is a tree, gives us a correct direction toward a given target point. In this section we show that there is a more direct way to find a correct direction. Suppose we are in some triangle A in $\mathrm{DT}(P)$. Removing A divides P into at most 3 parts. We need to find the part that contains t. For this, we find an edge e_t just above the target point t, which is the first polygon edge hit by the vertical ray emanating upward from t. Then, the part containing t is the one whose boundary contains the edge e_t, which is found by walking along the boundary. This kind of operation is called *point location* in computational geometry. It takes $O(n)$ time since the total length of the boundary is $O(n)$.

Now, we know which way to go from any triangle. Unfortunately, finding adjacent triangles in a canonical triangulation can be slow. The next idea for efficiency is to use the trapezoidal decomposition instead of the constrained Delaunay triangulation. That is, we partition the interior of a given simple polygon by drawing a vertical chord at each vertex toward the interior of the polygon. This decomposition is canonical and easy to compute. Moreover, it inherits the same property as the triangulation used to have for shortest paths.

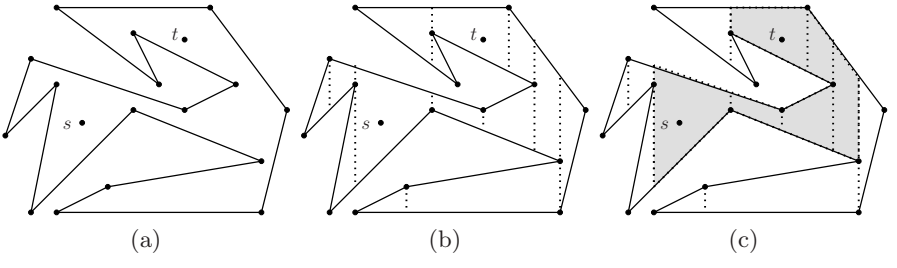


Fig. 5. Trapezoidal decomposition of a simple polygon for finding a shortest path in a simple polygon. (a) A simple polygon and two internal points s and t to be interconnected within the polygon. (b) Trapezoidal decomposition of P. (c) A sequence of trapezoids between two containing s and t.

The trapezoidal decomposition defined above is uniquely determined for any simple polygon. In general, degeneracies can cause one trapezoid to be adjacent to arbitrarily many trapezoids, as shown in Figure 6. Hence, we perform a symbolic perturbation to avoid this issue: each vertex of P and the two points s and t all have integral coordinates with $O(\log n)$ bits. Then, each integral point (x, y) is treated as a point $(x + y\varepsilon, y)$, ie, it is shifted to the right by $y\varepsilon$ for a small parameter ε such that $y^*\varepsilon < 1$ for the largest y-coordinate y^* of a vertex. After this perturbation, no two vertices share the same x-coordinate, as shown to the right in Figure 6.

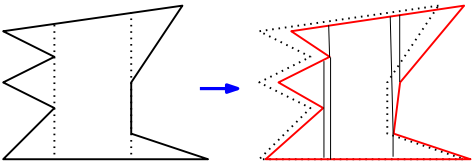


Fig. 6. Removing degeneracies by shifting vertices to the right. An original polygon is given to the left. The conversion results in the right polygon in which no two vertices share the same x-coordinate.

From now on, we assume that no two vertices have the same x-coordinate. This implies that any trapezoid is adjacent to at most four other trapezoids. Our first goal is to find a sequence of trapezoids between the two containing s and t. For the purpose, it suffices to find a correct neighbor at each trapezoid. More formally, suppose we are in a trapezoid T of which we know that it appears in the sequence. Since T is adjacent to at most four trapezoids, we want to determine which one lies on the correct path.

A characterization of a trapezoid is given in Figure 7. For a trapezoid T, two polygon edges $e_a(T)$ and $e_b(T)$ bound T from above and below, respectively. The vertical sides of T are denoted by $v_l(T)$ and $v_r(T)$, the left and right sides, respectively. At a trapezoid T we have to determine which way we should go toward the target point t. To that end, we traverse the corresponding boundary to find which part contains the edge e_t just above t.

By the observation above we now know that we can find the correct next trapezoid toward the target t in $O(n)$ time without using any extra array. Since the length of the trapezoid sequence is $O(n)$, the total time we need to find the sequence is $O(n^2)$.

We still need to describe how to find the shortest path from s to t, but this just works as in the previous algorithm: we know how to walk on the sequence

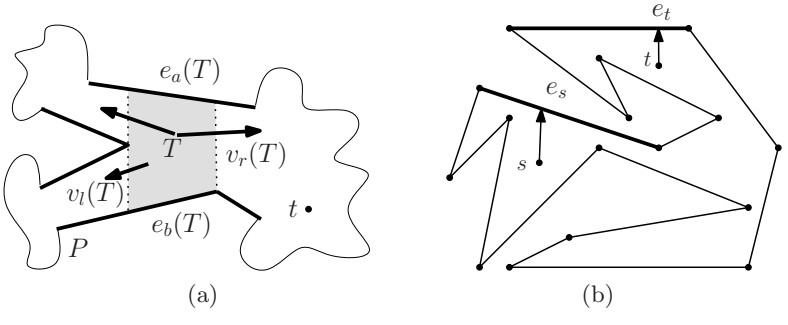


Fig. 7. Characterization of a trapezoid T by two polygon edges bounded from above and below and two vertical sides. (a) A trapezoid adjacent to three trapezoids. (b) A polygon edge e_s just above the point s and a polygon edge e_t just above t.

using $O(n)$ time at each step. To find a shortest path we just maintain the visible part of vertical sides of those trapezoids in the sequence. Whenever the entire side becomes invisible, we create a new bending point and recompute the visible part.

Given an arbitrary point q in the interior of P, we can determine a trapezoid containing q as follows: first find the polygon edges which are hit by a vertical ray emanating from q upward. If we find the edge among them that is closest to q, it is the top edge $e_a(T)$ of the trapezoid T containing q. In a similar fashion we can find a polygon edge $e_b(T)$ just below q which is the bottom edge of the trapezoid that contains q. Then, we compute the left and right vertical sides of the trapezoid T, denoted by $v_l(T)$ and $v_r(T)$, respectively. We start with four endpoints of $e_a(T)$ and $e_b(T)$. $v_l(T)$ is initially determined by the rightmost of the two left endpoints of $e_a(T)$ and $e_b(T)$. The initial value of $v_r(T)$ is similarly determined. Then, we scan each polygon vertex. If it lies in the current trapezoid and its incident polygon edge enters the trapezoid from its left, then we update the value $v_l(T)$ to be the x-coordinate of the vertex. If it lies in T and its incident edge enters T from the right, we update $v_r(T)$. In this way we can obtain the trapezoid in $O(n)$ time.

A trapezoid is specified in this way. Then, how can we find trapezoids adjacent to a given trapezoid? Suppose we want to find a trapezoid T_r which shares a right boundary with T. To do this, take a point q which is located to the right of the side at a small enough distance. Using the point q, the trapezoid T_r is computed in the same manner is described above. Thus, once we have a trapezoid, we can find trapezoids adjacent to it in $O(n)$ time.

Theorem 3. *Given an n-gon P and two arbitrary points in P, we can find a shortest path between them within P in $O(n^2)$ time in the constant work space model.*

4 Concluding Remarks

We have presented a constant-work-space algorithm for finding a shortest path between two arbitrary points in a simple polygon in polynomial-time. A number of geometric problems are open in the constant work space model. For example, does there exist an efficient constant-working-space algorithm for computing the visibility polygon from a point in a simple polygon. Another interesting direction is to investigate time-space trade-offs: how much work space is need to find a shortest path in a simple polygon in linear time?

Acknowledgments

This work of T.A. was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research on Priority Areas and Scientific Research (B).

References

1. Asano, T., Rote, G.: Constant-Working-Space Algorithms for Geometric Problems. In: Proc. CCCG 2009, Vancouver, pp. 87–90 (2009)
2. Asano, T.: Constant-Working-Space Algorithms: How Fast Can We Solve Problems without Using Any Extra Array? In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, p. 1. Springer, Heidelberg (2008)
3. Asano, T.: Constant-Working-Space Algorithms for Image Processing. In: Nielsen, F. (ed.) ETVC 2008. LNCS, vol. 5416, pp. 268–283. Springer, Heidelberg (2009)
4. Asano, T.: Constant-Working-Space Image Scan with a Given Angle. In: Proc. 24th European Workshop on Computational Geometry, Nancy, France, March 18-20, pp. 165–168 (2008)
5. Asano, T.: Constant-Working Space Algorithm for Image Processing. In: Proc. of the First AAAC Annual meeting, Hong Kong, April 26-27, p. 3 (2008)
6. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer, Heidelberg (2008)
7. Chan, T.M., Chen, E.Y.: Multi-Pass Geometric Algorithms. Discrete & Computational Geometry 37(1), 79–102 (2007)
8. Chazelle, B.: Triangulating a simple polygon in linear time. Discrete & Computational Geometry 6(1), 485–524 (1991)
9. Munro, J.I., Raman, V.: Selection from read-only memory and sorting with minimum data movement. Theoretical Computer Science 165, 311–323 (1996)
10. Reingold, O.: Undirected connectivity in log-space. J. ACM 55, 24, Article #17 (2008)

Approximation Algorithms for Art Gallery Problems in Polygons and Terrains

Subir Kumar Ghosh

School of Technology & Computer Science
Tata Institute of Fundamental Research, Mumbai 400005, India
ghosh@tifr.res.in
http://www.tcs.tifr.res.in/∼ghosh

Abstract. In this survey paper, we present an overview of approximation algorithms that are designed for art gallery problems in polygons and terrains.

1 Problems and Results

The art gallery problem is to determine the number of guards that are sufficient to *cover* or *see* every internal point of an art gallery (Figure 2(a)). An art gallery can be viewed as an n-sided polygon P (with or without holes) and guards as points inside P. A *polygon* P is defined as a closed region in the plane bounded by a finite set of line segments (called *edges* of P) such that there exists a path between any two points of P which does not intersect any edge of P (see Figure 1). If the boundary of P consists of two or more cycles, then P is called a *polygon with holes*. Otherwise, P is called a *simple polygon* or a *polygon without holes* [31].

Any point $z \in P$ is said to be *visible* from a guard g if the line segment zg does not intersect the exterior of P (see Figure 1). In general, guards may be placed anywhere inside P. If the guards are allowed to be placed only on vertices of P, they are called *vertex guards*. If there is no such restriction, guards are called

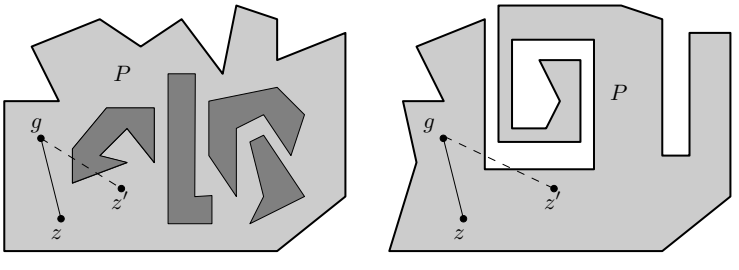


Fig. 1. Polygons with or without holes

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 21–34, 2010.

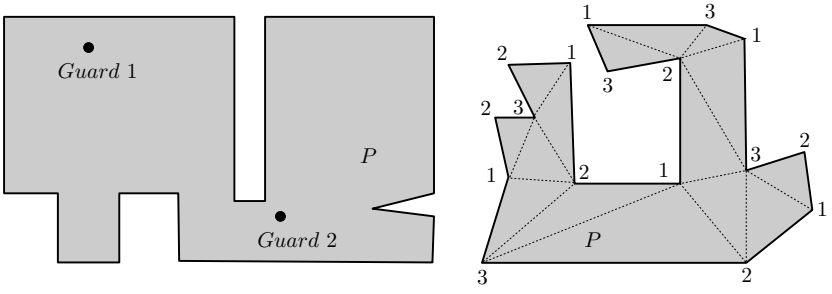


Fig. 2. (a) Two point guards are necessary and sufficient to see the entire polygon P. (b) The vertices of P are colored with three colors {1,2,3}.

point guards (see Figure 2(a)). Point and vertex guards are also called *stationary guards*. If guards are allowed to patrol along a segment inside P, they are called *mobile guards*. If they are allowed to patrol only edges of P, they are called *edge guards*.

The art gallery problem for stationary guards was first posed by Victor Klee in a conference (see [38]). Chavatal [13] showed that a simple polygon P needs at most $\lfloor n/3 \rfloor$ stationary guards. Fisk [28] later presented a simple proof of this result as follows. The vertices of P are colored with three colors (say, {1,2,3}) such that two vertices joined by an edge of P or by a diagonal in the triangulation of P receive different colors (see Figure 2(b)). Choose the color (say, 1) that has been assigned to the smallest number of vertices of P. Assign guards to vertices that received the color 1. Figure 3 (a) shows that $\lfloor n/3 \rfloor$ guards are sometimes necessary. Based on this proof, Avis and Toussaint [5] gave an $O(n \log n)$ time algorithm for placing guards in P. Kooshesh and Moret [45] later showed that the guards can be placed in $O(n)$ time.

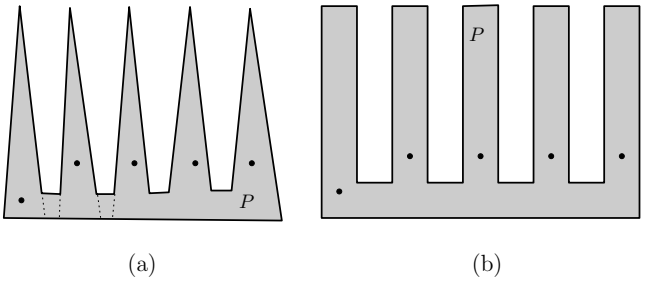


Fig. 3. (a) For any simple polygon P, $\lfloor n/3 \rfloor$ guards are sometimes necessary. (b) For any simple orthogonal polygon P, $\lfloor n/4 \rfloor$ guards are sufficient and sometimes necessary.

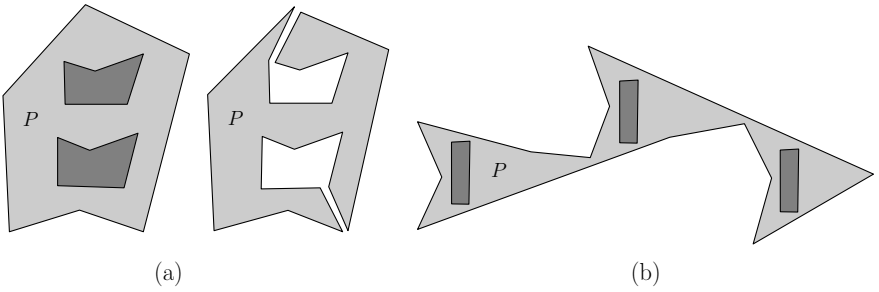


Fig. 4. (a) By adding 2 vertices for every hole, P is converted into a simple polygon of $n + 2h$ vertices. (b) A polygon requires $\lfloor \frac{n+h}{3} \rfloor$ guards [57].

For a simple orthogonal polygon P whose edges are horizontal or vertical, Kahn et al. [40] and O'Rourke [50] showed that P needs at most $\lfloor n/4 \rfloor$ stationary guards (see Figure 3(b)). These proofs partitions P into convex quadrilaterals before $\lfloor n/4 \rfloor$ guards are placed in P. A convex quadrilaterization of P can be constructed by algorithms of Edelsbrunner, O'Rourke and Welzl [22], Lubiw [48], Sack [53], and Sack and Toussaint [54].

For a polygon P with h holes, O'Rourke [52] showed that P can be converted into a simple polygon of $n + 2h$ vertices (see Figure 4(a)), and therefore, P needs at most $\lfloor \frac{n+2h}{3} \rfloor$ vertex guards. Shermer conjectured that P can always be guarded with $\lfloor \frac{n+h}{3} \rfloor$ vertex guards [52] and proved the conjecture for $h = 1$. For $h > 1$, the conjecture is still open. On the other hand, if the placement of guards is not restricted to vertices, Hoffmann et al. [36] and Bjorling-Sachs and Souvaine [8] proved independently that P can always be guarded with $\lceil \frac{n+h}{3} \rceil$ point guards (see Figure 4(b)). Bjorling-Sachs and Souvaine also presented an $O(n^2)$ time algorithm for placing the point guards.

If P is an orthogonal polygon with holes, then P requires at most $\lfloor \frac{n}{4} \rfloor$ point guards [35] or $\lfloor \frac{n}{3} \rfloor$ vertex guards [37] (see Figure 5(a)). On the other hand, a

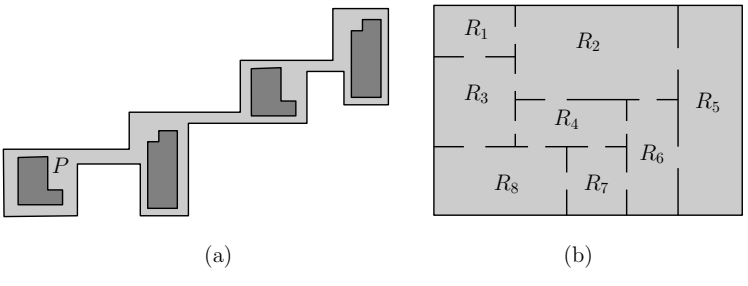


Fig. 5. (a) A polygon of 44 vertices with 4 holes requires 12 vertex guards [57]. (b) A rectilinear art gallery needs one guard for two adjacent rooms.

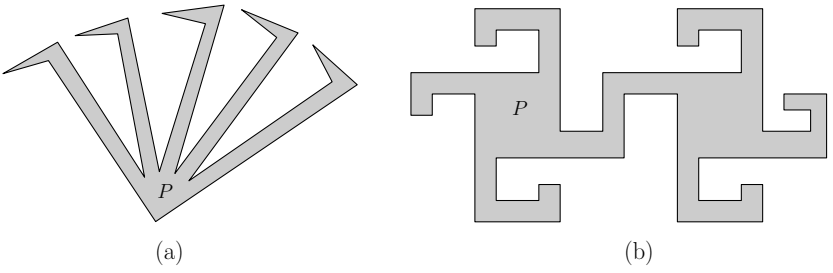


Fig. 6. (a) A polygon requires $\lfloor n/4 \rfloor$ mobile guards. (b) A rectilinear polygon requires $\lfloor \frac{3n+4}{16} \rfloor$ mobile guards [57].

rectangular art gallery with n rooms can be guarded with exactly $\lceil \frac{n}{2} \rceil$ guards [19] (see Figure 5(b)).

For placing mobile guards, O'Rourke [51,52] showed that a simple polygon P needs at most $\lfloor n/4 \rfloor$ mobile guards (see Figure 6(a)). For placing edge guards in P, $\lfloor n/4 \rfloor$ edge guards seem to be sufficient, except for a few polygons (see [57]). Aggarwal [2] proved that P needs at most $\lfloor \frac{3n+4}{16} \rfloor$ mobile guards (see Figure 6(b)) and the same bound holds for edge guards as shown by Bjorling-Sachs [7]. For guarding an orthogonal polygon P with h holes, Györi et al. [34] established that $\lfloor \frac{3n+4h+4}{16} \rfloor$ mobile guards are always sufficient and sometimes necessary. For survey of art gallery theorems, see O'Rourke [52], Shermer [56] and Urrutia [57].

The minimum guard problem is to find the minimum number of stationary or mobile guards that can see every point inside a polygon. O'Rourke and Supowit [52] proved that the minimum point, vertex and edge guard problems in polygons with holes are NP-hard. Even for polygons without holes, Lee and Lin [47] proved that these three guarding problems are NP-hard. They are also NP-hard for simple orthogonal polygons as shown by Katz and Rpoisman [41] and Schuchardt and H.-D. Hecker [55].

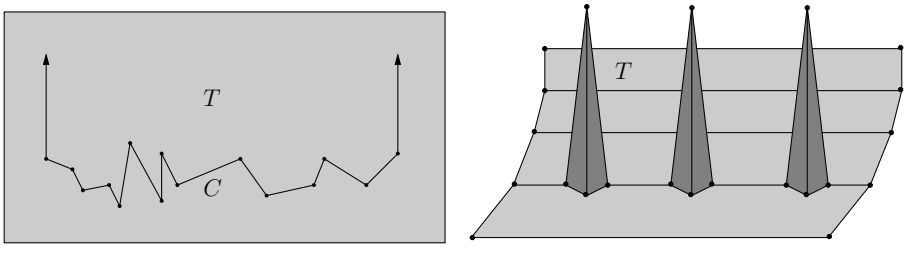
Ghosh [30,32] presented approximation algorithms for minimum vertex and edge guard problems for polygons with or without holes by transforming art gallery problems into set-cover problems after discretizing the entire polygon. For simple polygons P, approximation algorithms for both problems run in $O(n^4)$ time (after a recent improvement [32]) and yield solutions that can be at most $O(\log n)$ times the optimal solution. For polygons P with holes, approximation algorithms for both problems give the same approximation ratio of $O(\log n)$ but the algorithms take $O(n^5)$ time (after a recent improvement). Since 1986, this is the only known technique for transforming these four art gallery problems leading to efficient approximation algorithms in terms of worst case running times and approximation bounds. We present these algorithms in details in the next section.

For the minimum vertex guard problem in a simple polygon P, Efrat and Har-Peled [23] presented randomized approximation algorithms which run in $O(nc_{opt}^2 \log^4 n)$ expected time. The approximation ratio of the algorithm is $O(\log c_{opt})$, where c_{opt} is the number of vertex guards in the optimal solution and c_{opt} can be a fraction of n in the worst case. For a polygon P with h holes, their randomized approximation algorithm runs in $O(nhc_{opt}^3 polylog\ n)$ expected time, and the algorithm gives an approximation ratio of $O(\log n \log(c_{opt} \log n))$. It may be noted that their randomized approximation algorithms do not always guarantee solutions, and the quality of approximation is correct with high probability. For the same minimum vertex guard problem, Worman and Keil [59] gave a polynomial time exact algorithm for the special case of rectangle visibility in rectilinear polygons.

For the point guard problem, Nilsson [49] gave an $O(c_{opt})^2$-approximation algorithms for monotone polygons and simple orthogonal polygons, Deshpande et al. [21] gave a pseudo-polynomial time $O(\log n)$-approximation algorithm for simple polygons, and Efrat and Har-Peled [23] gave an exact algorithm for simple polygons that runs in $O((nc_{opt})^{3(c_{opt}+1)})$ time.

Recently, efforts are being made to design heuristics to solve stationary guard problems [3,16,17,18], where the efficiency of these heuristics are evaluated by experimentation. These heuristics essentially follow Ghosh's method of first discretizing the entire region of a polygon and then using the minimum set-cover solution. However, these heuristics use different criteria for discretization or in choosing candidate sets.

Let us consider the guarding problems in 1.5-dimensional and 2.5-dimensional terrains. Let C denote a polygonal chain which is monotone with respect to X-axis. The region of the plane T lying above C is called *1.5-dimensional terrain* (see Figure 7(a)). Two points of C are said to be mutually *visible* if the line segment joining then lies entirely inside T. Chen et al. [12] gave a proof showing that the minimum vertex guard problem for T is NP-hard but there seems to be some gap in the proof of hardness. Recently, King and Krohn [44] have proved that the problem is indeed NP-hard. Chen et al. [12] also considered the problem of placing the minimum number of guards for guarding every point of C from its left, and presented a linear time algorithm for this problem.



(a) (b)

Fig. 7. (a) T is a 1.5-dimensional terrain. (b) T is a 2.5-dimensional terrain.

For the general vertex guard problem in 1.5-dimensional terrain, Ben-Moshe et al. [6] designed an $O(1)$-approximation algorithm exploiting the geometric structure of C, and their algorithm runs in $O(n^4)$ time. Note that the value of the approximation ratio seems to be at least 6. Clarkson and Varadarajan [14] suggested another approximation algorithm with the constant factor approximation ratio using ϵ-nets, and showed that their algorithm runs in $O(n^2 \log n)$ time. King [42] presented an improved approximation algorithm for this problem which runs in $O(n^2)$ time and gives an approximation ratio of 4. It turned out latter that the approximation ratio of the algorithm is 5 and not 4. Using linear programming relaxation method, Elbassioni et al. [26] presented an approximation algorithm for this problem that runs in $O(n \log n)$ time giving an approximation ratio of 4. We present this algorithm in details in Section 3. For guarding 1.5-dimensional rectilinear terrains, Katz and Roisman [41] gave an approximation algorithm that runs in $O(n^2)$ time and gives an approximation ratio of 2.

Recently, Gibson et al. [33] have designed a polynomial time approximation scheme for the 1.5-dimensional terrain guarding problem using a local search in a planar graph that appropriately relates the local and global optimum. So, their polynomial time algorithm returns a guard cover whose cardinality is at most $(1 + \epsilon)$ time optimal for any $\epsilon > 0$. Complexity issues in guarding terrains have also been studied recently [43,44].

Let us consider the problem of guarding 2.5-dimensional terrains. Let T denote a polyhedral surface such that any vertical line intersects T exactly at one point. Then, T is called a *2.5-dimensional terrain* (see Figure 7(b)). Two points of T are said to be mutually *visible* if the line segment joining then lies entirely on or above T. Visibility problems on 2.5-dimensional terrains were studied initially in the context of geographical information system [20].

For the vertex and edge guards problems on T of n vertices, Bose et al. [11] proved that $\lfloor \frac{n}{2} \rfloor$ vertex guards are both necessary and sufficient, and Everett and Rivera-Campo [27] proved $\lfloor \frac{n}{3} \rfloor$ edge guards are always sufficient. Bose et al. [11] showed that $\lfloor \frac{4n-4}{13} \rfloor$ edge guards are sometimes necessary. They show gave linear time algorithms for placing $\lfloor \frac{3n}{5} \rfloor$ vertex guards and $\lfloor \frac{2n}{5} \rfloor$ edge guards on T. Using the technique of maximum matching in a bridgeless cubic graph, Bose et al. [9] gave $O(n^{3/2})$ time algorithm for placing $\lfloor \frac{n}{2} \rfloor$ vertex guards and $\lfloor \frac{n}{3} \rfloor$ edge guards on T.

For the minimum guard problems on T, Cole and Sharir [15] showed that the minimum point guard problem for T is NP-hard. It is not known whether the minimum vertex and edge guard problems on T are also NP-hard. For the minimum vertex guard problem on T, Eidenbenz [24] gave an approximation algorithm using the same method given by Ghosh [30]. The approximation algorithm runs in $O(n^8)$ time and gives an approximation ratio of $O(\log n)$. In Section 3, we present this algorithm in details. There is no approximation algorithms known for the minimum point and edge guard problems on T. Using the same method, Eidenbenz [24] also gave $O(\log n)$ approximation algorithms for related guarding problems on terrains with triangle restrictions.

2 Approximation Algorithms in Polygons

In this section, we present approximation algorithms of Ghosh [30,32] for vertex and edge guard problems in a polygon P with or without holes. Assume that vertices of P are labeled $v_1, v_2, \ldots, v_n$. Let F_i denote the set of all points of P that are visible from a vertex v_i [31]. So, the vertex guard problem of P can be viewed as a polygon decomposition problem in which decomposed pieces are fans.

It may appear that if the entire boundary of P is visible from vertex guards, then all internal points of P are also visible from them. However, this is not true as shown in Figure 8(a). This establishes that vertex guards must be chosen in such a way that the entire polygon P is visible from the chosen guards. A convex region $c \subset P$ is said to be a *convex component* of P such that c cannot be divided further by a line segment passing through two vertices of P (see Figure 8(b)). We have the following lemma.

Lemma 1. *Every convex component of P is either totally visible or totally not visible from a vertex of P.*

Proof: If there exists a convex component c partially visible from a vertex v_i of P, then there exists a vertex v_j in P such that the line drawn from v_i through v_j intersects c. So, c is not a convex component, a contradiction.

Corollary 1. *For every vertex v_i of P, the fan F_i is the union of convex components of P.*

If we consider convex components as elements of sets and every fan as a set consisting of these elements, then the problem of finding the minimum number of fans is the same as the minimum set-covering problem. Given a finite

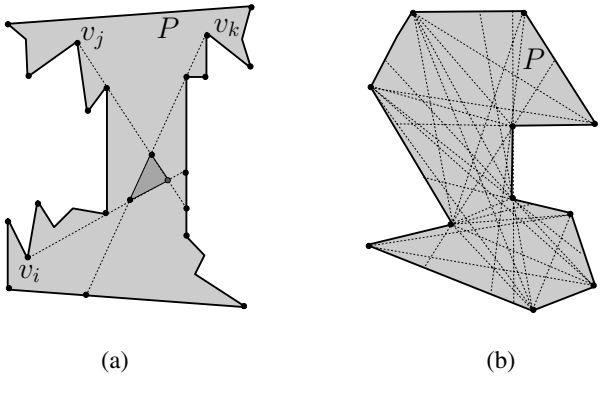


Fig. 8. (a) The entire boundary of P is visible from v_i, v_j and v_k but the shaded region in the middle is not visible from any of them. (b) The entire region of P is decomposed into convex components.

family of sets $S_1, \ldots, S_n$, the problem is to determine the minimum cardinality subset A such that $\bigcup_{i \in A} S_i = \bigcup_{j=1}^{n} S_j$ [29,58]. In the following, we present an approximation algorithm for the vertex guard problem in P.

Step 1. Compute all convex components $c_1, c_2, \ldots, c_m$ by drawing lines through every pair of vertices of P. Let $C = (c_1, c_2, \ldots, c_m)$, $N = (1, 2, \ldots, n)$ and $Q = \emptyset$.
Step 2. *For* $1 \leq j \leq n$, construct F_j by adding convex components of P totally visible from v_j.
Step 3. Find $i \in N$ such that $|F_i| \geq |F_j|$ *for all* $j \in N$ and $i \neq j$.
Step 4. Add i to Q and delete i from N.
Step 5. *For all* $j \in N$, $F_j := F_j - F_i$, and $C := C - F_i$.
Step 6. *If* $|C| \neq \emptyset$ *then goto* Step 3.
Step 7. Output the set Q and Stop.

Let us analyze the time complexity of the algorithm. Step 1 requires $O(n^4)$ time as $O(n^2)$ lines are drawn in P to compute convex components. For every convex component c_k of C, take a point $z_k \in c_k$ and identify the vertices of P that are visible from c_z, and then add c_z to the fans of these visible vertices. If P is a simple polygon, vertices visible from c_k can be identified in $O(n)$ time [31,46]. If P contains holes, vertices visible from c_k can be identified in $O(n)$ query time after spending $O(n^2)$ preprocessing time [4,31]. Therefore, constructing $F_1, F_2, \ldots, F_n$ in Step 2 can be done in $O(mn)$ time. Step 3 requires $O(n^2)$ time. Since the fans containing any convex component c_l is known, removing c_l from all fans containing c_l in Step 5 can be done in $O(n)$ time. Therefore, Step 5 takes $O(n^5)$ time. Hence, the overall time complexity of the approximation algorithm is $O(n^5)$.

It has been shown by Bose, Lubiw and Munro [10] that convex components can also be computed by the intersections of visibility polygons of P from v_i for all i, and the number of convex components m becomes $O(n^3)$ if P does not contain holes. Using this method for computing convex components, the running time of the approximation algorithm reduces to $O(n^4)$ for polygons without holes. We have the following theorem.

Theorem 1. *For a polygon P of n vertices, an approximate solution of the minimum vertex guard problem in P can be computed (i) in $O(n^4)$ time for P without holes and (ii) in $O(n^5)$ time for P with holes, and (iii) the size of the solution can be at most $O(\log n)$ times the optimal.*

We now consider the edge guard problem in a polygon P with or without holes. Assume that edges of P are labeled $e_1, e_2, \ldots, e_n$. A point $z \in P$ is said to be *weakly visible* from e_i if there exists a point $w \in e_i$ such that the segment zw lies inside P [31]. The set of all points of P weakly visible from e_i is referred as the *weak visibility polygon* of P from e_i. Since the region of P that can be seen by an edge guard is a weak visibility polygon of P, it can be seen that the edge guard problem of P is a polygon decomposition problem in which decomposed pieces are weak visibility polygons. Again, draw lines through every pair of vertices of P, and decompose P into convex components. We have the following lemma.

Lemma 2. *Every convex component of P is either totally visible or totally not visible from an edge of P.*

Corollary 2. *For every edge e_i of P, the weak visibility polygon of P from e_i is the union of some convex components of P.*

As before, the weak visibility polygon (denoted as E_i) from every edge e_i can be viewed as a set, and convex components as elements of sets. Hence, edge guards in P can be located from $E_1, E_2, \ldots, E_n$ following the method in the vertex guard approximation algorithm. We have the following theorem.

Theorem 2. *For a polygon P of n vertices, an approximate solution of the minimum edge guard problem in P can be computed (i) in $O(n^4)$ time for P without holes and (ii) in $O(n^5)$ time for P with holes, and (iii) the size of the solution can be at most $O(\log n)$ times the optimal.*

In the minimum set-covering problem, any subset of elements can form a set, whereas any subset of convex components may not form a polygonal region corresponding to the visibility polygon of P from a vertex or an edge. So, this geometric restriction on the input sets may not allow the approximation ratio to reach the upper bound of $O(\log n)$. Ghosh conjectured in 1986 that these approximation algorithms yield solutions within a constant factor of the optimal.

Regarding the lower bound on the approximation ratio, Eidenbenz, Stamm and Widmayer [25] showed that the problems of minimum vertex, point and edge guards in simple polygons are APX-hard. This means that there exists a constant $\epsilon > 0$ for each of these problems such that an approximation ratio of $1 + \epsilon$ cannot be guaranteed by any polynomial time approximation algorithm unless $P = NP$. Though there may be approximation algorithms for these problems whose approximation ratios are not small constants, these problems cannot be approximated for polygons with holes by a polynomial time algorithm with ratio $((1-\epsilon)/12)(ln\ n)$ for any $(\epsilon > 0)$, unless $NP \subseteq TIME(n^{O(loglogn)})$. These complexity results are obtained using gap-preserving reductions from the SET COVER problem. Hence, the open problem is to design approximation algorithms for vertex, edge and point guards problems for simple polygons whose approximation ratios are constants.

3 Approximation Algorithms on Terrains

In this section, we start by presenting the approximation algorithm given by Elbassioni et al. [26] for the vertex guard problem on 1.5-dimensional terrain. Assume that vertices of the polygonal chain C of T are numbered $v_0, v_1, \ldots, v_n$ from left to right. Let v_i and v_j be two mutually visible vertices such that $i < j$ and a guard g is placed on v_i to guard v_j. Then g is called a *left vertex guard* of v_j. A *right vertex guard* of v_j is defined analogously.

Consider the problem of guarding entire chain C using only left vertex guards (refereed as *the left guarding problem*). Compute the Euclidean shortest paths

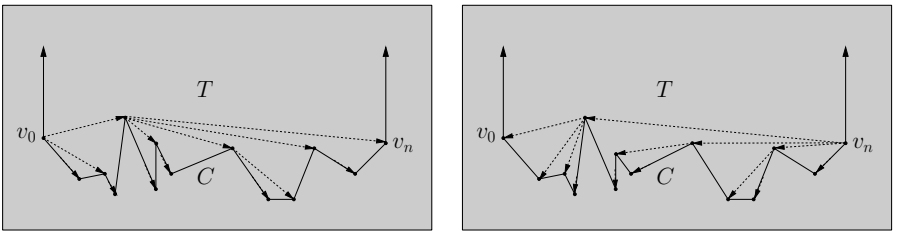


Fig. 9. The Euclidean shortest path trees rooted at v_0 and v_n

from v_0 to all vertices of C [31]. It can be seen that the union of these shortest paths gives the shortest path tree rooted at v_0 (denoted as $SPT(v_0)$) (see Figure 9(a)). Place guards at all vertices of $SPT(v_0)$ which are not leaves of $SPT(v_0)$. Let G_l denote the set of all these guards. We have the following lemma.

Lemma 3. *The guard set G_l is an optimal solution for the left guarding problem of T.*

Proof: Let v_i be the parent of a vertex v_j in $SPT(v_0)$ such that v_iv_j is an edge in the polygonal chain C. Since v_j is visible only from v_i for $0 \leq i < j$, the left vertex guard of v_j must be place on v_i. Since for every intermediate vertex v_i of $SPT(v_0)$, there exists such a vertex v_j, guards must be placed on every intermediate vertex of $SPT(v_0)$ which forms G_l. Note that all vertices that are on leaves of $SPT(v_0)$ also become visible from guards in G_l. Hence, G_l is an optimal solution for the left guarding problem of T.

For the corresponding right guarding problem, the optimal solution (denoted as G_r) can be obtained by placing guards at all vertices of $STP(v_n)$ which are not leaves of $SPT(v_n)$ (see Figure 9(b)). It is straightforward to show that the algorithm runs in $O(n)$ time [31].

It can be seen that for every vertex v_j of C, v_j is guarded by a left vertex guard in G_l and a right vertex guard in G_r. In order to minimize the number of guards, a selection of guards has to be made from $G_l \cup G_r$ such that they together see the entire polygonal chain. So, vertices of C are partitioned into two sets L and R such that if a vertex v_j belongs to L (or R) then its left (respectively, right) vertex guard from G_l (respectively, G_r) is added to the selected list of guards (say, G) provided v_j is not visible from any guard already in G. Note that vertices of L and R are considered from v_0 to v_n and from v_n to v_0 respectively for adding their guards to G.

In the following, we show how vertices of C can be partitioned in L and R such that $|G|$ is at most four times the optimal solution for the vertex guard problem in T. Let x_i denote the decision variable of a vertex v_i for all i. The problem can be formulated as integer linear programming problem:

$$\text{Minimize} \sum_{i=0}^{n} x_i$$

subject to

$$\sum_{v_i sees\ v_j} x_i \geq 1 \qquad \text{for all } v_j \in C$$

$$x_i \in \{0, 1\}$$

The variable $x_i = 1$ if a guard is placed on v_i. Otherwise, $x_i = 0$. This constraint can be relaxed to $1 \geq x_i \geq 0$. A vertex v_j is placed in L if

$$\sum_{v_i sees\ v_j} x_i \geq \frac{1}{2}$$

Otherwise, v_j is placed in R. Let LP solution be x_i^* for $0 \leq i \leq n$ giving rise to LP relaxation. Then $2x_i^*$ is a feasible solution for both L and R. Therefore, the solution for L is at most twice the LP solution which is bounded by twice the integer optimal solution. Similar bound holds for the solution for R. We have the following theorem.

Theorem 3. *For a 1.5-dimensional terrain T of n vertices, an approximate solution of the minimum vertex guard problem in T can be computed in polynomial time and the size of the solution can be at most four times the optimal.*

Let us now present the approximation algorithm of Eidenbenz [24] for the minimum vertex guard problem on a 2.5-dimensional terrain T. Convex components defined in the previous section can be generalize for T as follows. Triangulate all faces of T. Let v_i, v_j and v_k be vertices of T such that v_iv_j is an edge of a triangle of T. Construct a plane containing v_i, v_j and v_j and intersect T by the plane. Similarly, for every such three vertices of T, T is further partitioned by the corresponding plane define by the three vertices. These planes partition the space into three-dimensional *cells* and they have two-dimensional faces bounded by intersection points of three planes. Since there are $O(n)$ edges of T, the number of planes that are used for intersecting T can be $O(n^2)$. Since this arrangement of cells consists of $O(n^6)$ points, faces and cells, the arrangement of T can be computed in $O(n^6)$ time [1]. We have the following lemma.

Lemma 4. *Every face in the arrangement of T is either totally visible or totally not visible from a vertex of T.*

For each vertex v_j of T, construct a set F_j consisting of those faces in the arrangement of T that are totally visible, which can be computed in $O(n^7)$ time. So, $F_1, F_2, \ldots, F_n$ can be constructed in $O(n^8)$ time. Using the Johnson's approximation algorithm [39] for the minimum set-cover problem, an approximation solution can be computed for the vertex guard problem on T. We state the result in the following theorem.

Theorem 4. *For a 2.5-dimensional terrain T of n vertices, an approximate solution of the minimum vertex guard problem in T can be computed in $O(n^8)$ time and the size of the solution can be at most $O(\log n)$ times the optimal.*

References

1. Agarwal, P., Sharir, M.: Arrangements and their applications. In: Sack, J.R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 49–119. North-Holland, Amsterdam (2000)
2. Aggarwal, A.: The art gallery theorem: its variations, applications, and algorithmic aspects. Ph. D. Thesis, Johns Hopkins University (1984)
3. Amit, Y., Mitchell, J.S.B., Packer, E.: Locating guards for visibility coverage of polygons. In: Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX 2007), pp. 120–134. SIAM, Philadelphia (2007)
4. Asano, T., Asano, T., Guibas, L.J., Hershberger, J., Imai, H.: Visibility of disjoint polygons. Algorithmica 1, 49–63 (1986)
5. Avis, D., Toussaint, G.T.: An efficient algorithm for decomposing a polygon into star-shaped polygons. Pattern Recognition 13, 395–398 (1981)
6. Ben-Moshe, B., Katz, M., Mitchell, J.: A constant-factor approximation algorithm for optimal terrain guarding. SIAM Journal on Computing 36, 1631–1647 (2007)
7. Bjorling-Sachs, I.: Edge guards in rectilinear polygons. Computational Geometry: Theory and Applications 11, 111–123 (1998)
8. Bjorling-Sachs, I., Souvaine, D.L.: An efficient algorithm for guard placement in polygons with holes. Discrete & Computational Geometry 13, 77–109 (1995)
9. Bose, P., Kirkpatrick, D.G., Li, Z.: Efficient algorithms for guarding or illuminating the surface of a polyhedral terrain. In: Proceedings of the 8th Canadian Conference on Computational Geometry, pp. 217–222 (1996)
10. Bose, P., Lubiw, A., Munro, J.: Efficient visibility queries in simple polygons. Computational Geometry: Theory and Applications 23, 313–335 (2002)
11. Bose, P., Shermer, T., Toussaint, G.T., Zhu, B.: Guarding polyhedral terrains. Computational Geometry: Theory and Applications 7, 173–185 (1997)
12. Chen, D., Estivill-Castro, V., Urrutia, J.: Optimal guarding of polygons and monotone chains. In: Proceedings of the 7th Canadian Conference on Computational Geometry, pp. 133–138 (1995)
13. Chvatal, V.: A combinatorial theorem in plane geometry. Journal of Combinatorial Theory, Series B 18, 39–41 (1975)
14. Clarkson, K.L., Varadarajan, K.R.: Improved approximation algorithms for geometric set cover. Discrete & Computational Geometry 37, 43–58 (2007)
15. Cole, R., Sharir, M.: Visibility problems for polyhedral terrains. Journal of Symbolic Computation 7, 11–30 (1989)
16. Couto, M.C., de Rezende, P.J., de Souza, C.C.: An exact and efficient algorithm for the orthogonal art gallery problem. In: Proceedings of the 20th Brazilian Symposium on Computer Graphics and Image Processing, pp. 87–94 (2007)
17. Couto, M.C., de Rezende, P.J., de Souza, C.C.: Experimental evaluation of an exact algorithm for the orthogonal art gallery problem. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 101–113. Springer, Heidelberg (2008)
18. Couto, M.C., de Rezende, P.J., de Souza, C.C.: An IP solution to the art gallery problem. In: Proceedings of the 25th Annual ACM Symposium on Computational Geometry, pp. 88–89 (2009)
19. Czyzowicz, J., Rivera-Campo, E., Santoro, N., Urrutia, J., Zaks, J.: Guarding rectangular art galleries. Discrete Applied Mathematics 50, 149–157 (1994)
20. de Floriani, L., Magillo, P., Puppo, E.: Applications to computational geometry to geographic information systems. In: Sack, J.-R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 333–388. North-Holland, Amsterdam (2000)

21. Deshpande, A., Kim, T., Demaine, E.D., Sarma, S.E.: A pseudopolynomial time O(log n)-approximation algorithm for art gallery problems. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 163–174. Springer, Heidelberg (2007)
22. Edelsbrunner, H., O'Rourke, J., Welzl, E.: Stationing guards in rectilinear art galleries. Computer Vision, Graphics, Image Processing 27, 167–176 (1984)
23. Efrat, A., Har-Peled, S.: Guarding galleries and terrains. Information Processing Letters 100, 238–245 (2006)
24. Eidenbenz, S.: Approximation algorithms for terrain guarding. Information Processing Letters 82, 99–105 (2002)
25. Eidenbenz, S., Stamm, C., Widmayer, P.: Inapproximability results for guarding polygons and terrains. Algorithmica 31, 79–113 (2000)
26. Elbassioni, K., Krohn, E., Matijevi, D., Mestre, J., Severdija, D.: Improved approximations for guarding 1.5-dimensional terrains. Algorithmica (to appear, 2009)
27. Everett, H., Rivera-Campo, E.: Edge guarding polyhedral terrains. Computational Geometry: Theory and Applications 7 (1997)
28. Fisk, S.: A short proof of Chvatal's watchman theorem. Journal of Combinatorial Theory, Series B 24, 374 (1978)
29. Garey, M., Johnson, D.: Computer and Intractability: A guide to the theory of NP-completeness. W.H. Freeman and Company, New York (1979)
30. Ghosh, S.K.: Approximation algorithms for art gallery problems. In: Proceedings of Canadian Information Processing Society Congress, pp. 429–434 (1987)
31. Ghosh, S.K.: Visibility Algorithms in the Plane. Cambridge University Press, Cambridge (2007)
32. Ghosh, S.K.: Approximation algorithms for art gallery problems in polygons. Discrete Applied Mathematics (to appear, 2010)
33. Gibson, M., Kanade, G., Krohn, E., Varadarajan, K.: An approximation scheme for terrain guarding. In: Dinur, I., et al. (eds.) APPROX and RANDOM 2009. LNCS, vol. 5687, pp. 140–148. Springer, Heidelberg (2009)
34. Györi, E., Hoffmann, F., Kriegel, K., Shermer, T.: Generalized guarding and partitioning for rectilinear polygons. Computational Geometry: Theory and Applications 6, 21–44 (1996)
35. Hoffmann, F.: On the rectilinear art gallery problem. In: Paterson, M. (ed.) ICALP 1990. LNCS, vol. 443, pp. 717–728. Springer, Heidelberg (1990)
36. Hoffmann, F., Kaufmann, M., Kriegel, K.: The art gallery theorem for polygons with holes. In: Proceedings of the 32nd IEEE Symposium on the Foundation of Computer Science, pp. 39–48 (1991)
37. Hoffmann, F., Kriegel, K.: A graph-coloring result and its consequences for polygon-guarding problems. SIAM Journal on Discrete Mathematics 9, 210–224 (1996)
38. Honsberger, R.: Mathematical games II. Mathematical Associations for America (1979)
39. Johnson, D.S.: Approximation algorithms for combinatorial problems. Journal of Computer and System Sciences 9, 256–278 (1974)
40. Kahn, J., Klawe, M., Kleitman, D.: Traditional galleries require fewer watchmen. SIAM Journal of Algebraic and Discrete Methods 4, 194–206 (1983)
41. Katz, M., Roisman, G.: On guarding the vertices of rectilinear domains. Computational Geometry: Theory and Applications 39, 219–228 (2008)
42. King, J.: A 4-approximation algorithm for guarding 1.5-dimensional terrains. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 629–640. Springer, Heidelberg (2006)

43. King, J.: VC-dimension of visibility on terrains. In: Proceedings of the 20th Canadian Conference on Computational Geometry, pp. 27–30 (2008)
44. King, J., Krohn, E.: The complexity of guarding terrains. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (to appear, 2010)
45. Kooshesh, A.A., Moret, B.: Three-coloring the vertices of a triangulated simple polygon. Pattern Recognition 25, 443 (1992)
46. Lee, D.T.: Visibility of a simple polygon. Computer Vision, Graphics, and Image Processing 22, 207–221 (1983)
47. Lee, D.T., Lin, A.K.: Computational complexity of art gallery problems. IEEE Transactions on Information Theory IT-32, 276–282 (1986)
48. Lubiw, A.: Decomposing polygons into convex quadrilaterals. In: Proceedings of the 1st ACM Symposium on Computational Geometry, pp. 97–106 (1985)
49. Nilsson, B.J.: Approximate guarding of monotone and rectilinear polygons. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1362–1373. Springer, Heidelberg (2005)
50. O'Rourke, J.: An alternative proof of the rectilinear art gallery theorem. Journal of Geometry 211, 118–130 (1983)
51. O'Rourke, J.: Galleries need fewer mobile guards: A variation on Chvatal's theorem. Geometricae Dedicata 4, 273–283 (1983)
52. O'Rourke, J.: Art Gallery Theorems and Algorithms. Oxford University Press, New York (1987)
53. Sack, J.: An O(nlogn) algorithm for decomposing simple rectilinear polygons into quadrilaterals. In: Proceedings of the 20th Allerton Conference, pp. 64–75 (1982)
54. Sack, J., Toussaint, G.T.: Guard placement in rectilinear polygons. In: Toussaint, G.T. (ed.) Computational Morphology, pp. 153–175. North-Holland, Amsterdam (1988)
55. Schuchardt, D., Hecker, H.D.: Two NP-hard art-gallery problems for ortho-polygons. Mathematical Logic Quarterly 41, 261–267 (1995)
56. Shermer, T.: Recent results in art galleries. Proceedings of the IEEE 80, 1384–1399 (1992)
57. Urrutia, J.: Art gallery and illumination problems. In: Sack, J.-R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 973–1023. North-Holland, Amsterdam (2000)
58. Vazirani, V.: Approximation Algorithms. Springer, New York (2001)
59. Worman, C., Keil, J.M.: Polygon decomposition and the orthogonal art gallery problem. International Journal of Computational Geometry and Applications 17, 105–138 (2007)

The Hamiltonian Augmentation Problem and Its Applications to Graph Drawing

Emilio Di Giacomo and Giuseppe Liotta

Dip. di Ingegneria Elettronica e dell'Informazione, Università degli Studi di Perugia
{digiacomo,liotta}@diei.unipg.it

Abstract. In this talk we digress about the strict interplay between the graph-theoretic problem of computing a Hamiltonian augmentation of a planar graph G and the graph drawing problem of embedding G onto a given set of points. We review different Hamiltonian augmentation techniques and their impact on different variants of the corresponding graph drawing problem. We also look at universal point sets, simultaneous graph embeddings, and radial graph drawings.

1 Introduction

Let G be a graph with no self-loops and no multiple edges. A *Hamiltonian cycle* of G is a simple cycle that contains all vertices of G. A graph G that admits a Hamiltonian cycle is said to be *Hamiltonian*. A planar graph G is *sub-Hamiltonian* if either G is Hamiltonian or G can be augmented with dummy edges (but not with dummy vertices) to a graph $\mathsf{aug}(G)$ that is Hamiltonian and planar. A *subdivision* of a graph G is a graph obtained from G by replacing each edge by a path with at least one edge. Internal vertices on such a path are called *division vertices*. Let G be a planar graph and let $\mathsf{sub}(G)$ be a sub-Hamiltonian subdivision of G (it is easy to see that any planar graph always admits a subdivision that is sub-Hamiltonian). The graph $\mathsf{aug}(\mathsf{sub}(G))$ is called a *Hamiltonian augmentation of* G and will be denoted as $\mathsf{Ham}(G)$.

Hamiltonicity and Hamiltonian augmentation techniques have turned out to be useful tools in solving some graph drawing problems where the output must satisfy a set of semantic rules such as having the vertices drawn on a a given set of points, or on a given curve, or on a given line (see, e.g., [34] for a list of semantic rules in graph drawing). Namely, different Hamiltonian augmentation techniques can lead to drawing algorithms with different performances in terms of fundamental aesthetic requirements such as area and number of bends per edge in the output.

This paper shortly surveys different Hamiltonian augmentation techniques and their applications to graph drawing. We shall discuss the impact of Hamiltonicity in the following graph drawing problems: (i) computing a crossing-free drawing of a planar graph where the vertices are mapped to a given set of points; (ii) computing a simultaneous embedding of two planar graphs; and (iii) computing a radially layered drawing of a planar layered graph. For reasons of space, the applications of Hamiltonian augmentation techniques to upward point-set embeddability (see, e.g., [18,19,30,31]) are omitted from this paper.

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 35–46, 2010.

2 Hamiltonian Augmentations and Point-Set Embeddings

Let G be a planar graph with n vertices and let S be a set of n distinct points in the plane. A *point-set embedding of G on S* is a planar polyline drawing of G such that each vertex of G is mapped to a distinct point of S. The maximum number of bends along an edge, also called *curve complexity*, is a measure of the quality of the drawing.

The problem of computing a point-set embedding of a planar graph G has been studied both under the assumption that the mapping between each vertex of G and its corresponding point of S is given as part of the input and in the case that the drawing algorithm can choose this mapping. In the first case, we talk about *point-set embedding with mapping*; in the second case, we talk about *point-set embedding without mapping*.

2.1 Point-Set Embeddings with Mapping

Halton [21] proves that every planar graph admits a point-set embedding on any set of n points and for any given mapping; however, he does not address the problem of optimizing the curve complexity of the computed drawing. Pach and Wenger [32] revisit the question and show a construction that guarantees $O(n)$ curve complexity.

The drawing algorithm by Pach and Wenger to compute a point-set embedding with mapping of G on S relies on a Hamiltonian augmentation of a planar graph. They first prove the following lemma for Hamiltonian graphs.

Lemma 1. [32] *Let G be a Hamiltonian planar graph with n vertices, let S be any set of n distinct points in the plane, and let any mapping from the vertices of G to the points of S be given. There exists an $O(n^2)$ time algorithm that computes a point-set embedding of G on S with the given mapping and such that the curve complexity is at most $8n + 9$.*

In order to extend the lemma above to general planar graphs, Pach and Wenger describe an elegant Hamiltonian augmentation technique that introduces at most 2 division vertices per edge. The idea behind the method is based on computing a spanning tree of a planar embedded graph and on executing a Eulerian visit of this tree. Each time the Eulerian visit crosses an edge not in the tree, a division vertex is added. By this Eulerian tour the Hamiltonian cycle in the augmented graph can be easily derived.

Theorem 1. [32] *Every planar graph G with n vertices admits a Hamiltonian augmentation $\mathsf{Ham}(G)$ with at most 2 division vertices per edge and at most $5n - 10$ vertices in total. Also, $\mathsf{Ham}(G)$ can be computed in $O(n)$ time.*

According to Theorem 1 one can compute a point-set embedding of G on any set of points S and any given mapping, by computing a point-set embedding of $\mathsf{Ham}(G)$ on any superset of S (with a mapping that "implies" the original one) and then removing the division vertices. The number of bends along an edge of G depends on the number of division vertices of that edge. Namely, if and edge e is subdivided with k division vertices, then it is replaced by a path with $k + 1$ edges. Each of these edges has at most $8n' + 9$ bends, where n' denotes the number of vertices in $\mathsf{Ham}(G)$; also, every division vertex d can produce an additional bend if the two segments of e incident to d

are drawn with different slopes. In summary, if an edge is subdivided with k division vertices, then it can have up to $(k+1)(8n'+9)+k$ bends in the point-set embedding of G on S. Since by Theorem 1 every planar graph has a Hamiltonian augmentation with at most $k = 2$ division vertices per edge and at most $n' = 5n - 10$ vertices in total, the following theorem holds.

Theorem 2. [32] *Let G be a planar graph with n vertices, let S be any set of n distinct points in the plane, and let any mapping from the vertices of G to the points of S be given. There exists an $O(n^2)$ time algorithm that computes a point-set embedding of G on S with the given mapping and such that the curve complexity is at most $120n - 211$.*

Pach and Wenger also established a $\Omega(n)$ lower bound on the curve complexity of the point-set embeddability problem with mapping even for very simple classes of graphs such as paths or matchings.

Theorem 3. [32] *Let G be a planar graph with n vertices and m pairwise independent edges, let S be a set of n points in convex position, and assume a random mapping of the vertices of G to the points of S be given. Then, as n tends to infinity, in every point-set embedding of G on S with the given mapping there are almost surely at least $m/20$ edges each having at least $m/40^3$ bends.*

2.2 Point-Set Embeddings without Mapping

Kaufmann and Wiese [28] study point-set embeddings without mapping. Given a planar graph G with n vertices and any set S of n distinct points in the plane they show how to compute a point-set embedding of G on S without mapping such that the curve complexity is at most two, which is proved to be worst-case optimal. Similar to the work by Pach and Wenger described in the previous section, also the paper by Kaufmann and Wiese concentrates first on Hamiltonian graphs.

Lemma 2. [28] *Let G be a Hamiltonian planar graph with n vertices and let S be any set of n distinct points in the plane. There exists an $O(n)$ time algorithm that computes a point-set embedding of G on S without mapping and such that the curve complexity is at most 1.*

Also in this case, the technique of Lemma 2 is extended to general planar graphs by means of a Hamiltonian augmentation technique. As discussed in Section 2.1 we have that the number of bends of a point-set embedding of a planar graph G on a given set of points S is at most $2k + 1$ where k is the number of division vertices per edge in $\mathsf{Ham}(G)$. Namely, an edge of G is split into at most $k + 1$ edges in $\mathsf{Ham}(G)$; each of these edges has at most one bend by Lemma 2; and each division vertex can give rise to a new bend when removed. Thus, Lemma 2 and Theorem 1 imply that every planar graph has a point-set embedding without mapping on any set of points with curve complexity at most 5.

Kaufmann and Wiese show a different Hamiltonian augmentation technique which leads to optimal curve complexity. To this aim,they present a Hamiltonian augmentation technique that introduces at most one division vertex per edge. This technique is based

on combining two algorithms by Chiba and Nishizeki [5,6]. The first algorithm is used to find the separating triangles in G (which is assumed to be triangulated) [5]. For each separating triangle, one of the edges of the triangle is split with a division vertex which is suitably connected to existing vertices so that the triangle is no longer separating. This procedure is applied repeatedly until no separating triangle remains in the graph. The graph obtained by this procedure is four-connected and another algorithm by Chiba and Nishizeki [6] can be used to find a Hamiltonian cycle in the four-connected graph.

Theorem 4. [28] *Every planar graph with n vertices admits a Hamiltonian augmentation with at most 1 division vertices per edge. This Hamiltonian augmentation can be computed in $O(n)$ time.*

Theorem 4 implies that every planar graph G admits a point-set embedding without mapping on any set of points with at most 3 bends per edge. As pointed out above, one of the three bends of an edge e corresponds to the division vertex d of e and it exists if the two segments incident to d have different slopes. Kaufmann and Wiese describe a procedure to avoid this bend. This procedure consists in rotating the two segments incident to d so that they become both vertical.

Theorem 5. [28] *Let G be a planar graph with n vertices and let S be any set of n distinct points in the plane. There exists an $O(n^2)$ time algorithm that computes a point-set embedding Γ of G on S without mapping and such that the curve complexity is at most 2. Also, there exists a graph G and a set of collinear points S such that every point-set embedding of G on S has curve complexity at least 2.*

The result by Kaufmann and Wiese suggests two interesting research directions.

1. The construction that computes point-set embeddings with curve complexity two requires exponential area for some of the cases. One may wonder whether optimal curve complexity and polynomial area can be simultaneously achieved. It may be worth remarking that the technique by Kaufmann and Wiese computes drawings of polynomial area at the expenses of sub-optimal curve complexity (namely curve complexity 3).
2. The proof of the lower bound on the curve complexity uses a set of collinear points. It is natural to ask whether the same lower bound holds for sets of points in general position (no three points are collinear). While curve complexity zero can only be achieved for the family of outerplanar graphs [20], one may wonder whether all planar graphs admit point-set embedding without mapping with curve complexity one on every set of points in general position.

In the next sections we will present an alternative Hamiltonian augmentation technique based on the notion of *monotone topological book embedding*. We will show how to use monotone topological book embedding to answer the first question. We will also use monotone topological book embeddings to partially answer the second question. In addition, monotone topological book embeddings can be applied to extend and unify the results of Theorems 2 and 5 in the framework of colored point-set embeddings. Finally, applications of monotone topological book embeddings to radial layouts and simultaneous embeddings are also described.

3 Point-Set Embedding without Mapping: Optimal Curve Complexity and Polynomial Area

3.1 Flat Division Vertices

The algorithm by Kaufmann and Wiese first computes a drawing with curve complexity 3 and area $O(W^3)$ where W is the *size* of S, i.e., the length of the side of the smallest axis parallel square containing S. The rotation technique used to reduce the number of bends per edge from 3 to 2 may cause an exponential growth of the area of the drawing. In order to obtain optimal cure complexity and polynomial area we would like to avoid rotations.

We recall that the rotation involves two edges sharing a same division vertex; we also observe that this rotation is not necessary if the division vertex satisfies a special topological property that we call *flatness*. To define the notion of flatness we start by observing that a division vertex d of $\mathsf{Ham}(G)$ has degree at most four. Namely, it has degree two in $\mathsf{sub}(G)$ and it may be necessary to add at most two edges in order to guarantee that the Hamiltonian cycle passes through d[1]. Consider a planar embedding of $\mathsf{Ham}(G)$. Let $\mathcal{C}$ be the Hamiltonian cycle of $\mathsf{Ham}(G)$, let e be an edge of $\mathcal{C}$ and let $\mathcal{P} = \mathcal{C} \setminus e$ the Hamiltonian path obtained by removing e from $\mathcal{C}$. A division vertex d of $\mathsf{Ham}(G)$ is called a *flat* division vertex if:

- d is an internal vertex of $\mathcal{P}$ and has degree four in $\mathsf{Ham}(G)$;
- the two edges of $\mathcal{P}$ incident to d are not consecutive in the circular order around d defined by the planar embedding of $\mathsf{Ham}(G)$;
- letting (u, d) and (d, v) be the two edges incident to d that are not in $\mathcal{P}$, d is encountered after u and before v when walking along $\mathcal{P}$.

The following lemma relates the flatness of the division vertices with the curve complexity of a point-set embedding.

Lemma 3. [3] *Let G be a planar graph. If G has a Hamiltonian augmentation such that each edge has at most k division vertices k_f of which are flat, then G admits a point-set embedding without mapping on any set of points with curve complexity at most $2k + 1 - k_f$.*

In the next section we show an alternative Hamiltonian augmentation technique where all division vertices are flat and there is at most one division vertex per edge. By Lemma 3 this implies that optimal curve complexity two is achievable by only flat division vertices. Hence no rotation is needed and point-set embeddings without mapping with optimal curve complexity and polynomial area can be computed.

3.2 Monotone Topological Book Embeddings

Let G be a graph. A *book embedding* of G consists of a total order $<_\sigma$ of $V(G)$ and a partition of $E(G)$ into disjoint sets, called *pages*, such that there are no edges (v, w)

[1] We assume that the edges added to $\mathsf{sub}(G)$ to obtain $\mathsf{Ham}(G)$ all belong to the Hamiltonian cycle $\mathcal{C}$ of $\mathsf{Ham}(G)$. This assumption is not restrictive because a dummy edge that were not in $\mathcal{C}$ could be removed without altering the Hamiltonicity and the planarity of $\mathsf{Ham}(G)$.

and (x, y) in a single page with $v <_\sigma x <_\sigma w <_\sigma y$. A *topological book embedding* of G is a book embedding of a subdivision of G. A (topological) book embedding with h pages is also called a *h-page (topological) book embedding*. As observed in [15], every planar graph admits a 2-page topological book embedding.

A 2-page (topological) book embedding of a planar graph G can also be regarded as a planar drawing of (a subdivision of) G such that all vertices lie along a horizontal straight line, called *spine*, ordered according to $<_\sigma$ and each edge is completely drawn in one of the two half-planes defined by the spine. According to this equivalent definition of (topological) book embedding, a division vertex of G in a topological book embedding of G is also called a *spine crossing*.

Starting from a 2-page topological book embedding of G it is immediate to define a Hamiltonian augmentation of G. Namely, let $\mathsf{sub}(G)$ be the subdivision of G that defines the topological book embedding and let $v_1, v_2, \ldots, v_{n'}$ be the vertices of $\mathsf{sub}(G)$ in the order they appear along the spine. For every pair of non-adjacent vertices v_i, v_{i+1} we add a dummy edge (v_i, v_{i+1}). We also add edge $(v_0, v_{n'})$ if this edge does not exist in G. The cycle $\mathcal{C} = \langle v_0, v_1, \ldots, v_{n'} \rangle$ is a Hamiltonian cycle of the graph $\mathsf{aug}(\mathsf{sub}(G))$ consisting of $\mathsf{sub}(G)$ plus the dummy edges, and therefore $\mathsf{aug}(\mathsf{sub}(G))$ is a Hamiltonian augmentation of G. Also the number of division vertices in $\mathsf{Ham}(G)$ is equal to the number of spine crossings in the 2-page topological book embedding of G.

Di Giacomo et al. [11] introduce and study a special type of topological book embedding. A *monotone topological book embedding* of a planar graph G is a 2-page topological book embedding such that:

1. each edge $e = (u, v)$ of G has at most one spine crossing d;
2. d is encountered between u and v when walking along the spine;
3. edge (u, d) is below the spine, while edge (d, v) is above the spine.

Starting from a monotone topological book embedding γ of G we obtain a Hamiltonian augmentation of G with at most one division vertex per edge and such that all division vertices are flat. Consider the Hamiltonian path $\mathcal{P} = \mathcal{C} \setminus (v_0, v_{n'})$. Each edge e of G has at most one division vertex d in $\mathsf{Ham}(G)$ because e has at most one spine crossing in γ (Property 1 of the definition of monotone topological book embedding). The two edges of $\mathcal{P}$ incident to d are not consecutive in the circular order around d defined by the planar embedding of $\mathsf{Ham}(G)$ by Property 3 of the definition of monotone topological book embedding. Finally, d is encountered after u and before v when walking along $\mathcal{P}$ by Property 2 of the definition of monotone topological book embedding.

Theorem 6. [11] *Every planar graph with n vertices admits a monotone topological book embedding that can be computed in $O(n)$ time.*

Corollary 1. [11] *Every planar graph with n vertices admits a Hamiltonian augmentation with at most 1 division vertices per edge such that all division vertices are flat. This Hamiltonian augmentation can be computed in $O(n)$ time.*

By Corollary 1 and Theorem 3 we can answer the first question at the end of Section 2.2.

Theorem 7. *Let G be a planar graph with n vertices and let S be any set of n distinct points in the plane. There exists an $O(n \log n)$ time algorithm that computes a point-set embedding Γ of G on S without mapping and such that the curve complexity is at most 2. Also, the area of the drawing is $O(W^3)$, where W is the length of the side of the smallest axis parallel square containing S.*

4 Point-Set Embeddings with Curve Complexity 1

Everett et al. [17] describe set of points that make it possible to point-set embed every planar graph with curve complexity at most one. More precisely, a set S of m points is *h-bend universal* for a family of planar graphs with n vertices ($n \leq m$) if each graph in the family admits a point-set embedding on a subset of S that has curve complexity at most h.

Gritzman, Mohar, Pach and Pollack [20] proved that every set of n distinct points in general position in the plane is 0-bend universal for the class of outerplanar graphs with n vertices. De Fraysseix, Pach, and Pollack [8] and independently Schnyder [33] proved that a grid with $O(n^2)$ points is 0-bend universal for all planar graphs with n vertices. De Fraysseix et al. [8] also showed that a 0-bend universal set of points for all planar graphs having n vertices cannot have $n + o(\sqrt{n})$ points. This last lower bound was improved by Chrobak and Karloff [7] and later by Kurowski [29] who showed that linearly many extra points are necessary for a 0-bend universal set of points for all planar graphs having n vertices. On the other hand, if two bends along each edge are allowed, a tight bound on the size of the point-set is implied by Theorem 5. Finally, if one bend along each edge is allowed, Di Giacomo et al. [11] proved a related result that every planar graph can be drawn with its vertices on any given convex curve; however, the positions of the points on the curve depend on the planar graph.

Everett et al. [17] prove the following theorem.

Theorem 8. [17] *Let $\mathcal{F}_n$ be the family of all planar graphs with n vertices. There exists a set of n distinct points in the plane in general position that is 1-bend universal for $\mathcal{F}_n$.*

The proof of Theorem 8 is constructive. A set S of n points is defined and a point-set embedding of a planar graph G on this set of points is constructed by exploiting the Hamiltonian augmentation technique of Corollary 1. Namely, the points are chosen to be in convex position and the Hamiltonian cycle $\mathcal{C}$ of $\mathsf{Ham}(G)$ is drawn as the convex hull CH of S suitably enriched with extra points that represent the division vertices. The edges of $\mathsf{Ham}(G)$ that are not in $\mathcal{C}$ are either inside $\mathcal{C}$ or outside it. Those inside are drawn as chords inside CH, the others are drawn with one bend outside CH. The choice of points and the flatness of the division vertices guarantee that no additional bend is required when the division vertices are removed.

5 Colored Hamiltonicity and Colored Point-Set Embeddability

In this section we present an application of Hamiltonian augmentation techniques to a problem that generalizes and encompasses those described in Sections 2.1 and 2.2.

Namely, we revisit both the point-set embeddability problem with mapping and the one without mapping in the framework of colored point-set embeddability. This new framework is studied by investigating a novel notion of Hamiltonicity, called *colored Hamiltonicity*, and the corresponding colored Hamiltonian augmentation problem. Colored Hamiltonicity and colored point-set embeddings have been first introduced by Di Giacomo et al. [13] and further studied in [3, 10, 12].

5.1 Colored Hamiltonicity

Let G be a planar graph with n vertices, and with a partition of the vertex set into subsets $V_0, \ldots, V_{k-1}$ for some positive integer $1 \leq k \leq n$. We say that each index i is a *color*, G is a *k-colored planar graph*. Note that vertices of the same color may be adjacent.

Let k and n be two positive integers. A *k-colored sequence* σ is a linear sequence of (possibly repeated) colors $c_0, c_1, \ldots, c_{n-1}$ such that $0 \leq c_j \leq k-1$ $(0 \leq j \leq n-1)$. We say that σ is *compatible with* a k-colored planar graph G if, for every $0 \leq i \leq k-1$, color i occurs $|V_i|$ times in σ.

Let $\mathcal{C}$ be the Hamiltonian cycle of a Hamiltonian augmentation $\mathsf{Ham}(G)$ of G. Let e be an edge of $\mathcal{C}$, let $\mathcal{P} = \mathcal{C} \setminus e$ be a Hamiltonian path obtained by removing an edge from $\mathcal{C}$, and let $v_0, v_1, \ldots, v_n$ be the vertices of G in the order they appear along $\mathcal{P}$. $\mathcal{P}$ is a *k-colored Hamiltonian path consistent with* σ if $\mathrm{col}(v_i) = c_i$ $(0 \leq i \leq n'-1)$. $\mathcal{C}$ is a *k-colored Hamiltonian cycle consistent with* σ if there exists an edge $e \in \mathcal{C}$ such that $\mathcal{P} = \mathcal{C} \setminus e$ is a k-colored Hamiltonian path consistent with σ. $\mathsf{Ham}(G)$ is called a *k-colored Hamiltonian augmentation of G consistent with σ*.

Theorem 9. [3] *Let G be a k-colored planar graph with n vertices and let σ be a k-colored sequence compatible with G. G admits a k-colored Hamiltonian augmentation consistent with σ with at most $3n-1$ division vertices per edge, $3n-3$ of which are flat. This Hamiltonian augmentation can be computed in $O(n^2 \log n)$ time.*

The idea behind the proof of Theorem 9 can be shortly summarized as follows. In order to to find a k-colored Hamiltonian path consistent with the sequence σ, start from a monotone topological book embedding of G and then transform it into a different topological book embedding such that the order of the vertices along the spine is consistent with σ. This topological book embedding is then used to compute a k-colored Hamiltonian augmentation of G consistent with σ. The number of spine crossing in this topological book embedding is such that the obtained Hamiltonian augmentation of G has at most $3n-1$ division vertices per edge, $3n-3$ of which are flat.

5.2 Colored Point-Set Embeddings

Let G be a k-colored planar graph with n vertices such that each color i has $|V_i|$ vertices of that color. Let S be a set of n distinct points in the plane with a partition into subsets $S_0, \ldots, S_{k-1}$ with $|V_i| = |S_i|$ $(0 \leq i \leq k-1)$. S is a *k-colored set of points compatible with G*. A *k-colored point-set embedding of G on S* is a planar polyline drawing of G such that each vertex of V_i is mapped to a distinct point of S_i.

Observe that a k-colored point-set embedding is a point-set embedding with mapping if $k = n$ and that it is a point-set embedding without mapping if $k = 1$. Some of the literature about red and blue point sets can also be revisited within the general framework of k-colored point-set embeddings with $k = 2$ (see, e.g., [1,2,26,22,23,24,27,25]).

Badent et al. [3] study the curve complexity of k-colored point-set embeddings and prove both upper and lower bounds.

The proof of the upper bound is constructive. The idea is to exploit the result about colored Hamiltonian augmentation of Theorem 9 to extend the technique of Kaufamnn and Wiese [28] for 1-colored point-set embeddings to the more general case of k colors ($k > 1$). In a nutshell, the drawing algorithm by Badent et al. can be sketched as follows.

Let $p_0, p_1, \ldots, p_{n-1}$ be the points of S ordered according to their x-coordinates. The left-to-right sequence of the colors of the points defines a k-colored sequence σ compatible with the k-colored graph G. By using Theorem 9, a k-colored Hamiltonian augmentation of G is computed having a k-colored Hamiltonian path $v_0, v_1, \ldots, v_n$ consistent with σ. Each vertex v_i is mapped to point p_i ($0 \leq i \leq n-1$); dummy points representing the division vertices are suitably added to the point set. By drawing one extra edge, we obtain that a k-colored Hamiltonian cycle consistent with σ is represented with all edges as straight-line segments except one that has one bend. All other edges of the k-colored Hamiltonian augmentation are now drawn either inside or outside the cycle with one bend each. The result follows from considerations analogous to those of Lemma 3.

Theorem 10. [3] *Let G be a k-colored planar graph with n vertices and let S be any k-colored set of points compatible with G. There exists an $O(n^2 \log n)$ time algorithm that computes a k-colored point-set embedding of G on S with curve complexity at most $3n+2$. Also, for every $n \geq 16$ and for every $k \geq 2$, there exists a k-colored plana graph G with n vertices and k-colored set of n points S compatible with G such that every k-colored point-set embedding of G on S has curve complexity $\Omega(n)$.*

An implication of the upper bound in Theorem 10 is that the curve complexity of n-colored point-set embeddings is improved from $120n - 211$ proved by Pach and Wenger [32] to $3n + 2$. Also, the lower bound extends the one of Pach and Wenger [32] to the less constrained case of two or more colors and has larger constant factors. We observe however that the graph used in the lower bound of [3] is biconnected, while the lower bound by Pach and Wenger [32] holds even for matchings and paths.

5.3 More Points Than Vertices

In order to obtain k-colored point-set embeddings whose curve complexity does not depend on n, Di Giacomo et al. [12] used colored Hamiltonicity to study variants and extensions of the result by Badent et al [3]. They first considered a specific configuration of points, i.e., *ordered* set of points. A k-colored set of n points is *ordered* if for each color all points of that color are consecutive along the x-direction.

Theorem 11. [12] *Let G be an n-vertex k-colored planar graph ($1 \leq k \leq n$). Let S be an ordered k-colored set of points compatible with G. There exists an $O(n \log n + k\, n)$-time algorithm that computes a k-colored point-set embedding of G on S having curve complexity at most $3k + 7$.*

Di Giacomo et al. [12] then used the result of Theorem 11 to study the case when the k-colored set of points S is such that the number of points of color i is larger that the number of vertices of color i, i.e., $|S_i| \geq |V_i|$ for every $i = 0, 1, \ldots, k-1$. They prove that $O(kn)$ points for each color are sufficient to guarantee a k-colored point-set embedding with curve complexity $O(k)$.

Theorem 12. [12] *Let $G = (\bigcup_{i=0}^{k-1} V_i, E)$ be an n-vertex k-colored planar graph ($1 \leq k \leq n$) with $|V_i| = n_i$ ($i = 0, 1, \ldots, k-1$); let $S = \bigcup_{i=0}^{k-1} S_i$ be any k-colored set of points such that $|S_i| = k(n_i - 1) + 1$. There exists an $O(n \log n + k\, n)$-time algorithm that computes a k-colored point-set embedding of G on a subset of S having curve complexity at most $3k + 7$.*

6 Other Applications of Hamiltonicity to Graph Drawing

In this section we briefly recall applications of Hamiltonian augmentation techniques to other graph drawing problems.

6.1 Simultaneous Embeddings

Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be two planar graphs with the same set of vertices. A *simultaneous embedding* of G_1 and G_2 is a pair of drawings $\langle \Gamma_1, \Gamma_2 \rangle$ of G_1 and G_2, respectively, such that both Γ_1 and Γ_2 are planar and each vertex $v \in V$ has the same coordinates in Γ_1 and in Γ_2.

One can think to solve the simultaneous embedding problem as a point-set embeddability problem. Namely, one can compute a planar drawing of G_1 and then a point-set embedding with mapping of G_2 on the points representing the vertices of G_1. In this case, however, the lower bounds on the curve complexity of a n-colored point-set embedding of Theorems 3 and 10 imply that a curve complexity of $\Omega(n)$ can be required.

Erten and Kobourov [16] observe that a simultaneous embedding of two planar graphs with at most three bends per edge can be constructed in polynomial area as follows. Let $\mathsf{Ham}(G_1)$ and $\mathsf{Ham}(G_2)$ be two Hamiltonian augmentations of G_1 and G_2, respectively. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two Hamiltonian paths of $\mathsf{Ham}(G_1)$ and $\mathsf{Ham}(G_2)$, respectively. Compute a straight-line simultaneous embedding of $\mathcal{P}_1$ and $\mathcal{P}_2$ with the technique of Brass et al. [4]; apply the technique by Kaufmann and Wiese [28] to complete the simultaneous embedding. Analogously to the case of point-set embeddings, polynomial area and curve complexity two can be obtained by using Hamiltonian augmentations with only flat division vertices. Based on this observation, Di Giacomo and Liotta [14] improve the result by Erten and Kobourov [16] by exploiting Hamiltonian augmentation computed by using monotone topological book embeddings.

Theorem 13. [14] *Every pair of planar graphs G_1 and G_2 admit a simultaneous embedding of size $O(n^2) \times O(n^2)$ with at most two bends per edge.*

6.2 Radially Layered Drawings

A *layered graph* $G = (V, E, \phi)$, consists of a set of vertices V, a set of edges E and a function $\phi : V \to \{0, 1, \ldots, k-1\}$ that maps each vertex to an integer between 0 and $k-1$, which represents its *centrality*. A *radially layered drawing* of $G = (V, E, \phi)$ on a set of k concentric circles $\mathcal{C} = \{C_0, \ldots, C_{k-1}\}$ is such that each vertex $v \in V$ is drawn as a point of circle $C_{\phi(v)}$.

In [9] it is studied how to compute radially layered drawings of graphs by taking into account additional geometric constraints which correspond to typical aesthetic and semantic requirements for the visualization. Namely, the following requirements are considered: edge crossings, curve complexity, and radial distribution of the vertices (i.e., a measure of how uniformly the vertices are distributed on a polar grid). Trade-offs among these requirements are discussed and different linear-time drawing algorithms are presented. In particular, Hamiltonian augmentation techniques are used to prove the following.

Theorem 14. [9] *Every planar layered graph has a planar radially layered drawing with optimal radial distribution and whose curve complexity is 3. Also, this drawing can be computed in $O(n)$ time, where n is the number of vertices of the graph.*

References

1. Abellanas, M., Garcia-Lopez, J., Hernández-Peñver, G., Noy, M., Ramos, P.A.: Bipartite embeddings of trees in the plane. Discrete Applied Mathematics 93(2-3), 141–148 (1999)
2. Akiyama, J., Urrutia, J.: Simple alternating path problem. Discrete Mathematics 84, 101–103 (1990)
3. Badent, M., Di Giacomo, E., Liotta, G.: Drawing colored graphs on colored points. Theoretical Computer Science 408(2-3), 129–142 (2008)
4. Braß, P., Cenek, E., Duncan, C.A., Efrat, A., Erten, C., Ismailescu, D., Kobourov, S.G., Lubiw, A., Mitchell, J.S.B.: On simultaneous planar graph embeddings. Comput. Geom. 36(2), 117–130 (2007)
5. Chiba, N., Nishizeki, T.: Arboricity and subgraph listing algorithms. SIAM Journal on Computing 14, 210–223 (1985)
6. Chiba, N., Nishizeki, T.: The hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs. Journal of Algorithms 10, 189–211 (1989)
7. Chrobak, M., Karloff, H.: A lower bound on the size of universal sets for planar graphs. SIGACT News 20(4), 83–86 (1989)
8. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10, 41–51 (1990)
9. Di Giacomo, E., Didimo, W., Liotta, G.: Radial drawings of graphs: Geometric constraints and trade-offs. Journal of Discrete Algorithms 6(1), 109–124 (2008)
10. Di Giacomo, E., Didimo, W., Liotta, G., Meijer, H., Trotta, F., Wismath, S.K.: k-colored point-set embeddability of outerplanar graphs. Journal of Graph Algorithms and Applications 12(1), 29–49 (2008)
11. Di Giacomo, E., Didimo, W., Liotta, G., Wismath, S.K.: Curve-constrained drawings of planar graphs. Computational Geometry 30, 1–23 (2005)
12. Di Giacomo, E., Liotta, G., Trotta, F.: Drawing colored graphs with constrained vertex positions and few bends per edge. Algorithmica (to appear)
13. Di Giacomo, E., Liotta, G., Trotta, F.: On embedding a graph on two sets of points. IJFCS, Special Issue on Graph Drawing 17(5), 1071–1094 (2006)

14. Di Giacomo, E., Liotta, G.: Simultaneous embedding of outerplanar graphs, paths, and cycles. International Journal of Computational Geometry and Applications 17(2), 139–160 (2007)
15. Enomoto, H., Miyauchi, M.S.: Embedding graphs into a three page book with $O(m \log n)$ crossings of edges over the spine. SIAM J. Discrete Math. 12(3), 337–341 (1999)
16. Erten, C., Kobourov, S.G.: Simultaneous embedding of planar graphs with few bends. Journal of Graph Algorithms and Applications 9(3), 347–364 (2005)
17. Everett, H., Lazard, S., Liotta, G., Wismath, S.K.: Universal sets of n points for 1-bend drawings of planar graphs with n vertices. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 345–351. Springer, Heidelberg (2008)
18. 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)
19. Giordano, F., Liotta, G., Whitesides, S.: Embeddability problems for upward planar digraphs. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 242–253. Springer, Heidelberg (2009)
20. Gritzmann, P., Mohar, B., Pach, J., Pollack, R.: Embedding a planar triangulation with vertices at specified points. Amer. Math. Monthly 98(2), 165–166 (1991)
21. Halton, J.H.: On the thickness of graphs of given degree. Information Sciences 54, 219–238 (1991)
22. Kaneko, A., Kano, M.: Straight line embeddings of rooted star forests in the plane. Discrete Applied Mathematics 101, 167–175 (2000)
23. Kaneko, A., Kano, M.: Discrete geometry on red and blue points in the plane - a survey. In: Aronov, B., Basu, S., Pach, J., Sharir, M. (eds.) Discrete & Computational Geometry. Algorithms and Combinatories, vol. 25, pp. 551–570. Springer, Heidelberg (2003)
24. Kaneko, A., Kano, M., Suzuki, K.: Path coverings of two sets of points in the plane. In: Pach, J. (ed.) Towards a Theory of Geometric Graphs. Contemporary Mathematics, vol. 342. American Mathematical Society (2004)
25. Kaneko, A., Kano, M., Yoshimoto, K.: Alternating hamilton cycles with minimum number of crossing in the plane. International Journal of Computational Geometry & Application 10, 73–78 (2000)
26. Kaneko, A., Kano, M.: Straight-line embeddings of two rooted trees in the plane. Discrete & Computational Geometry 21(4), 603–613 (1999)
27. Kaneko, A., Kano, M., Tokunaga, S.: Straight-line embeddings of three rooted trees in the plane. In: Canadian Conference on Computational Geometry, CCCG 1998 (1998)
28. Kaufmann, M., Wiese, R.: Embedding vertices at points: Few bends suffice for planar graphs. Journal of Graph Algorithms and Applications 6(1), 115–129 (2002)
29. Kurowski, M.: A 1.235 lower bound on the number of points needed to draw all n-vertex planar graphs. Inf. Process. Lett. 92(2), 95–98 (2004)
30. Mchedlidze, T., Symvonis, A.: Crossing-optimal acyclic hamiltonian path completion and its application to upward topological book embeddings. In: Das, S., Uehara, R. (eds.) WALCOM 2009. LNCS, vol. 5431, pp. 250–261. Springer, Heidelberg (2009)
31. Mchedlidze, T., Symvonis, A.: Crossing-optimal acyclic hp-completion for outerplanar t-digraphs. In: Ngo, H.Q. (ed.) COCOON 2009. LNCS, vol. 5609, pp. 76–85. Springer, Heidelberg (2009)
32. Pach, J., Wenger, R.: Embedding planar graphs at fixed vertex locations. Graph and Combinatorics 17, 717–728 (2001)
33. Schnyder, W.: Embedding planar graphs on the grid. In: Proc. 1st ACM-SIAM Sympos. Discrete Algorithms (SODA 1990), pp. 138–148 (1990)
34. Sugiyama, K.: Graph Drawing and Applications. World Scientific, Singapore (2002)

Small Grid Drawings of Planar Graphs with Balanced Bipartition*

Xiao Zhou, Takashi Hikino, and Takao Nishizeki

Graduate School of Information Sciences, Tohoku University,
Sendai 980-8579, Japan
zhou@ecei.tohoku.ac.jp, hikino@nishizeki.ecei.tohoku.ac.jp,
nishi@ecei.tohoku.ac.jp

Abstract. In a grid drawing of a planar graph, every vertex is located at a grid point, and every edge is drawn as a straight-line segment without any edge-intersection. It has been known that every planar graph G of n vertices has a grid drawing on an $(n-2) \times (n-2)$ integer grid and such a drawing can be found in linear time. In this paper we show that if a planar graph G has a balanced bipartition then G has a grid drawing with small grid area. More precisely, if a separation pair bipartitions G into two edge-disjoint subgraphs G_1 and G_2, then G has a grid drawing on a $W \times H$ grid such that both the width W and height H are smaller than the larger number of vertices in G_1 and in G_2. In particular, we show that every series-parallel graph G has a grid drawing on a $(2n/3) \times (2n/3)$ grid and such a drawing can be found in linear time.

1 Introduction

In a *straight line drawing* of a planar graph, all edges are drawn as straight line segments without any edge-intersection. A *grid drawing* is a straight line drawing in which all vertices are put on grid points of integer coordinates [1]. This paper deals with a *grid drawing* of a planar graph G in a variable embedding setting, in which one can choose any plane embedding of G. Every plane graph G (with a fixed embedding) has a grid drawing on an $(n-2) \times (n-2)$ grid [2,3] and hence the area upper bound is $(n-2)^2 = O(n^2)$, where n is the number of vertices in G. A plane graph of nested triangles needs a $\lfloor 2(n-1)/3 \rfloor \times \lfloor 2(n-1)/3 \rfloor$ area [4], while the graph can be drawn in $2n^2/9 + O(n)$ area and needs $2n^2/9 + \Omega(n)$ area when one can choose the outer face [5]. Thus the area upper bound $O(n^2)$ above is optimal within a coefficient. However, it is still unknown whether every planar graph G has a grid drawing with a quadratic area whose coefficient of n^2 is less than one. It is conjectured that every plane graph G has a $\lceil 2n/3 \rceil \times \lceil 2n/3 \rceil$ grid drawing. The conjecture holds for some restricted classes of planar graphs. For example, every 4-connected plane graph G with at least four vertices on the outer face has a $(\lceil n/2 \rceil - 1) \times \lfloor n/2 \rfloor$ grid drawing [6]. Every tree has a grid

* This work is supported in part by a Grant-in-Aid for Scientific Research (C) 19500001 from Japan Society for the Promotion of Science (JSPS).

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 47–57, 2010.

drawing in area $O(n \log n)$ [7]. Every outerplanar graph has a grid drawing with area $O(n^{1.48})$ in a variable embedding setting [8].

In this paper we show that if a planar graph G has a balanced bipartition then G has a grid drawing with small area. More precisely, if a separation pair bipartitions G into two edge-disjoint subgraphs G_1 and G_2 as illustrated in Fig. 1(a), then G has a grid drawing on a $W \times H$ grid such that $W, H < \max\{n(G_1), n(G_2)\}$, where $n(G_1)$ and $n(G_2)$ are the numbers of vertices in G_1 and G_2, respectively. The plane embedding of G in the drawing may be different from that of a given plane graph. The outer face boundary of G is drawn as a parallelogram as illustrated in Fig. 1(c), while it is drawn as a triangle by most of the known algorithms [1,3,9,10]. In particular, we show that every series-parallel graph has a balanced bipartition and has a $\lfloor 2n/3 \rfloor \times (\lfloor 2n/3 \rfloor - 1)$ grid drawing as illustrated in Fig. 2, and hence the conjecture above holds for series-parallel graphs in a variable embedding setting. We also show that such a drawing can be found in linear time.

2 Planar Graph

In this section, we show that if a planar graph G has a balanced bipartition then G has a small grid drawing.

We deal with an undirected simple graph G. We denote the set of vertices of G by $V(G)$ and the set of edges by $E(G)$. We denote the number of vertices in G by $n(G)$ or simply by n. An edge joining vertices u and v is denoted by (u, v).

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 in the plane. A plane graph G divides the plane into connected regions called *faces*. The unbounded region is called the *outer* face of G.

A $W \times H$ *integer grid* consists of $W+1$ vertical grid lines and $H+1$ horizontal grid lines, and has a rectangular contour. We call W and H the *width* and *height* of the integer grid, respectively. We denote by $W(D)$ the width of the minimum integer grid enclosing a grid drawing D of a graph, and by $H(D)$ the height of D.

We call a pair of distinct vertices $\{s, t\}$ in a graph G a *separation pair* of G if G has two subgraphs G_1 and G_2 such that

(a) $V(G) = V(G_1) \cup V(G_2)$, $V(G_1) \cap V(G_2) = \{s, t\}$; and
(b) $E(G) = E(G_1) \cup E(G_2)$, $E(G_1) \cap E(G_2) = \emptyset$.

(See Fig. 1(a).) Such a pair of subgraphs $\{G_1, G_2\}$ is called a *bipartition* of G. For example, the pair of ends of any edge in G is a separation pair. It should be noted that G_1 or G_2 may not be connected and that s or t may be a cut-vertex of G.

Chrobak and Kant gave a linear-time algorithm to find a (convex) grid drawing D of a 3-connected plane graph G such that $W(D) = H(D) = n - 2$ and all the face boundaries are drawn as convex polygons [2]. If G is a maximal planar graph, then the outer face boundary of G is drawn as a triangle such that

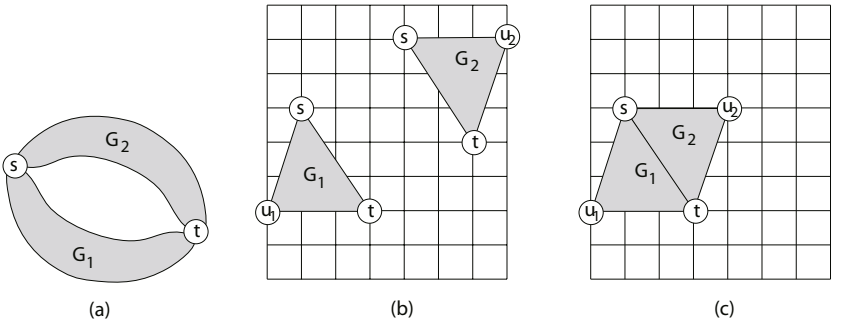


Fig. 1. (a) Bipartition $\{G_1, G_2\}$ of G, (b) grid drawings of G_1 and G_2, and (c) grid drawing of G

(a) one of the three sides is horizontal, and the length is $n-2$;
(b) the height of the triangle is $n-2$; and
(c) the x-coordinate of the top vertex of the triangle is equal to the x-coordinate of the left end of the horizontal side plus 1, and hence the two oblique sides have slope $n-2$ and $-(n-2)/(n-3)$, respectively.

(See Figs. 1(b) and 2(c).) Using their method, we show that if G has a bipartition $\{G_1, G_2\}$ of balanced size then G has a grid drawing with small area, as follows.

Theorem 1. *Let G be a planar graph, and let $\{G_1, G_2\}$ be an arbitrary bipartition of G. Then G has a grid drawing D such that*

$$W(D) \leq \max\{n(G_1), n(G_2)\} - 1$$

and

$$H(D) \leq \max\{n(G_1), n(G_2)\} - 2.$$

Furthermore such a drawing D can be found in linear time.

Proof. Clearly Theorem 1 holds if $n(G) = 2$. One may thus assume that $n(G) \geq 3$ and $n(G_1) \geq n(G_2)$.

We first consider the case where $n(G_2) = 2$. In this case, $n(G) = n(G_1) \geq 3$ and one may assume that $G = G_1$. Add dummy edges to G, if necessary, so that the resulting graph is maximal planar. Find a grid drawing of the graph by the algorithm in [2]. Erase the dummy edges from the drawing to obtain a drawing D of G. Then $W(D) = n(G) - 2 < \max\{n(G_1), n(G_2)\} - 1$ and $H(D) = n(G) - 2 = \max\{n(G_1), n(G_2)\} - 2$, and hence the theorem holds true.

We then consider the other case where $n(G_2) \geq 3$. One may assume that a separation pair $\{s, t\}$ bipartitions G to G_1 and G_2, as illustrated in Figs. 1(a) and 2(a). Add dummy edges to G_1, if necessary, so that the resulting graph is maximal planar and has an edge (s, t), as illustrated in Figs. 1(b) and 2(b) where

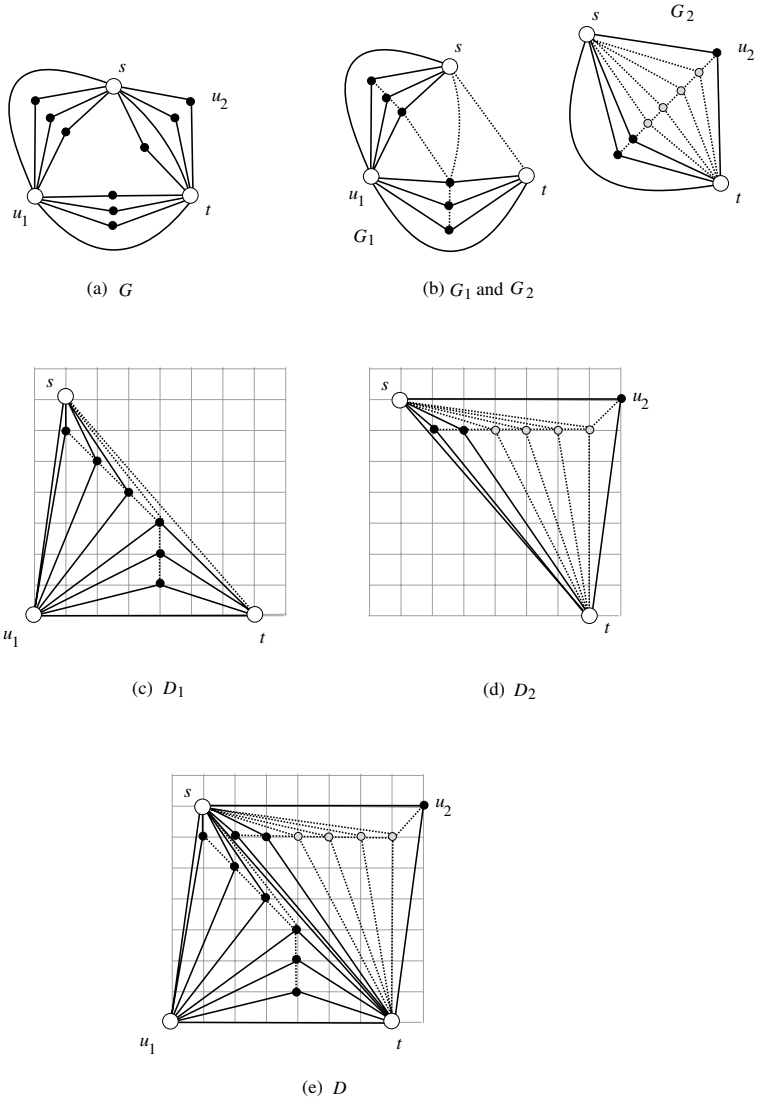


Fig. 2. (a) A series-parallel graph G, (b) G_1 and G_2, (c) drawing D_1 of G_1, (d) drawing D_2 of G_2, and (e) drawing D of G

dummy edges are drawn by dotted lines. The resulting graph is also denoted by G_1. Similarly, add to G_2 dummy edges and $(n(G_1) - n(G_2))$ dummy vertices so that the resulting graph is maximal planar and has an edge (s, t) and exactly $n(G_1)$ vertices, as illustrated in Figs. 1(b) and 2(b). The resulting graph is also denoted by G_2. We embed G_i, $i = 1, 2$, so that the edge (s, t) is on the outer face of G_i. Since G_i is a maximal plane graph, there are exactly three vertices on the

outer face. Let u_i be the vertex on the outer face other than s and t. Using the algorithm in [2], we obtain a grid drawing D_1 of G_1 such that $W(D_1) = n(G_1)-2$, $H(D_1) = n(G_1)-2$, edge (u_1, t) is horizontal, and edge (s, t) has slope $-(n(G_1)-2)/(n(G_1)-3)$, as illustrated in Figs. 1(b) and 2(c). Similarly, we obtain a grid drawing D_2 of G_2 such that $W(D_2) = n(G_1)-2$, $H(D_2) = n(G_1)-2$, edge (s, u_2) is horizontal, and edge (s, t) has slope $-(n(G_1)-2)/(n(G_1)-3)$, as illustrated in Figs. 1(b) and 2(d). The edge (s, t) in D_1 has the same length and slope as the edge (s, t) in D_2. Combining the two drawings D_1 and D_2 and erasing all the dummy vertices and edges, we obtain a grid drawing D of G, as illustrated in Figs. 1(c) and 2(e). Since G has no multiple edges, either the edge (s, t) in G_1 or the edge (s, t) in G_2 is dummy. Therefore, the combination above does not produce any edge-crossing. Since D_1 and D_2 are obtained by the algorithm in [2], we have $x(s) = x(u_1) + 1$ and $x(t) = x(u_2) - 1$ in D, where $x(v)$ denotes the x-coordinate of a vertex v. We thus have

$$W(D) = W(D_1) + 1 = n(G_1) - 1 = \max\{n(G_1), n(G_2)\} - 1,$$

and

$$H(D) = H(D_1) = n(G_1) - 2 = \max\{n(G_1), n(G_2)\} - 2.$$

Furthermore, D can be found in linear time, because D_1 and D_2 can be found in linear time by the algorithm in [2]. □

Figure 2 illustrates the drawing process of the plane graph in Fig. 2(a). The outer face boundary is drawn as a parallelogram, as illustrated in Fig. 2(e). The embedding of the obtained drawing in Fig. 2(e) is different from that in Fig. 2(a); compare edge (s, t) in the two embeddings.

Using the shift method in [9], one can find a grid drawing D of a maximal plane graph G such that $W(D) = H(D) = n - 1$ and the outer face boundary is drawn as an isosceles right triangle having a horizontal side and a vertical side. Using the method in place of the algorithm in [2], one can show that if G has a bipartition $\{G_1, G_2\}$ then G has a grid drawing D such that its outer face boundary is drawn as a square and $W(D) = W(H) = \max\{n(G_1), n(G_2)\} - 1$.

3 Series-Parallel Graph

In this section we give a linear algorithm to find a small grid drawing of a series-parallel graph.

A *series-parallel graph* (with *terminals* s and t) is a simple graph recursively defined as follows [11]:

(a) A graph G of a single edge is a series-parallel graph. The ends s and t of the edge are called the terminals of G. (See Fig. 3(a).)
(b) Let G_1 be a series-parallel graph with terminals s_1 and t_1, and let G_2 be a series-parallel graph with terminals s_2 and t_2.

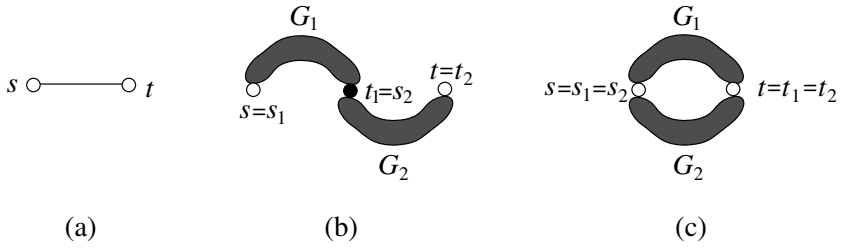


Fig. 3. (a) K_2, (b) series, and (c) parallel connections

(i) A graph G obtained from G_1 and G_2 by identifying vertex t_1 with vertex s_2 is a series-parallel graph, whose terminals are $s = s_1$ and $t = t_2$. Such a connection is called a *series connection*. (See Fig. 3(b).)

(ii) A graph G obtained from G_1 and G_2 by identifying s_1 and t_1 with s_2 and t_2, respectively, is a series-parallel graph, whose terminals are $s = s_1 = s_2$ and $t = t_1 = t_2$. Such a connection is called a *parallel connection*. (See Fig. 3(c).)

For example, the graph in Fig. 2(a) is series-parallel.

Every series-parallel graph has a balanced bipartition, as in the following Lemma 1. It is well known that one can decompose any tree into two edge-disjoint forests of balanced size by deleting a vertex [12, Theorem 9.1]. Applying the result to a "decomposition tree" of a series-parallel graph, one can prove Lemma 1. However, we give a direct simple proof without using the result so that the proof would be self-contained.

Lemma 1. *Every series-parallel graph G of n vertices has a bipartition $\{G_1, G_2\}$ such that $n(G_1), n(G_2) \leq \lfloor 2n/3 \rfloor + 1$. Furthermore such a bipartition can be found in linear time.*

Proof. (a) *Existence of a desired bipartition*

Suppose for a contradiction that a series-parallel graph G does not have a desired bipartition. Then $n \geq 4$; otherwise, $n \leq \lfloor 2n/3 \rfloor + 1$ and hence the pair of ends of any edge in G would produce a desired bipartition. One may further assume that G is 2-connected; otherwise, add to G a dummy edge joining the terminals of G, and let G be the resulting 2-connected series-parallel graph.

Let $\{s, t\}$ be a separation pair of G such that $\max\{n(G_1), n(G_2)\}$ is minimum among all bipartitions of G. One may assume without loss of generality that $n(G_1) \geq n(G_2)$ and that $(s, t) \notin E(G_1)$ and $(s, t) \in E(G_2)$ if $(s, t) \in E(G)$. Since G is 2-connected, both G_1 and G_2 are connected and series-parallel. G is a parallel connection of G_1 and G_2. Since G has no desired bipartition and $n(G_1) \geq n(G_2)$, we have

$$n(G_1) = \max\{n(G_1), n(G_2)\} \geq \lfloor 2n/3 \rfloor + 2 \tag{1}$$

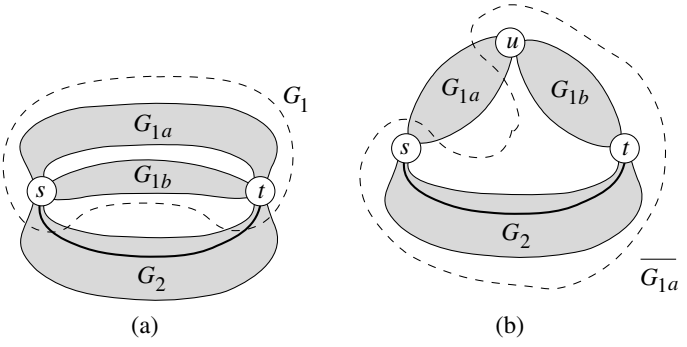


Fig. 4. (a) Bipartition $\{G_1, G_2\}$ with respect to $\{s, t\}$, and (b) bipartition $\{G_{1a}, \overline{G_{1a}}\}$ with respect to $\{s, u\}$

and hence

$$n(G_2) = n - n(G_1) + 2 \leq \lceil n/3 \rceil. \tag{2}$$

Substituting $n \geq 4$ to Eq. (1), we have $n(G_1) \geq 4$, and hence the series-parallel graph G_1 is obtained from two subgraphs G_{1a} and G_{1b} by a series or parallel connection, as illustrated in Fig. 4. We shall thus consider the following two cases.

Case 1: G_1 is a parallel connection of G_{1a} and G_{1b} (see Fig. 4(a)).

One may assume without loss of generality that $n(G_{1a}) \geq n(G_{1b})$. We then have

$$n(G_1) = n(G_{1a}) + n(G_{1b}) - 2 \geq 2n(G_{1b}) - 2. \tag{3}$$

We now consider another bipartition $\{G_{1a}, \overline{G_{1a}}\}$ with respect to the same separation pair $\{s, t\}$, where $\overline{G_{1a}}$ is a parallel connection of G_{1b} and G_2. Since $\max\{n(G_1), n(G_2)\}$ is minimum among all bipartitions of G, we have

$$\max\{n(G_{1a}), n(\overline{G_{1a}})\} \geq \max\{n(G_1), n(G_2)\} = n(G_1). \tag{4}$$

Since G_1 has no edge (s, t), we have $n(G_{1b}) \geq 3$. Therefore

$$n(G_{1a}) = n(G_1) - n(G_{1b}) + 2 < n(G_1). \tag{5}$$

By Eqs. (4) and (5) we have

$$n(G_1) \leq n(\overline{G_{1a}}) = n(G_2) + n(G_{1b}) - 2. \tag{6}$$

From Eqs. (3) and (6) one can easily derive

$$n(G_1) \leq 2n(G_2) - 2. \tag{7}$$

On the other hand, Eq. (1) implies

$$n(G_1) \geq \lfloor 2n/3 \rfloor + 2 \geq 2\lceil n/3 \rceil. \tag{8}$$

Substituting Eq. (2) to Eq. (8), we have $n(G_1) \geq 2n(G_2)$, contrary to Eq. (7).

Case 2: G_1 is a series connection of G_{1a} and G_{1b} (see Fig. 4(b)).

One may assume that $n(G_{1a}) \geq n(G_{1b})$. Let u be the common terminal of G_{1a} and G_{1b}. We now choose $\{s, u\}$ as a new separation pair of G, which bipartitions G into G_{1a} and the remaining graph $\overline{G_{1a}}$ as illustrated in Fig. 4(b). $\overline{G_{1a}}$ is a series connection of G_2 and G_{1b}. Since $\max\{n(G_1), n(G_2)\}$ is minimum among all bipartitions of G, we have

$$\max\{n(G_{1a}), n(\overline{G_{1a}})\} \geq n(G_1). \tag{9}$$

Since $n(G_{1a}) < n(G_1)$, by Eq. (9) we have $n(\overline{G_{1a}}) \geq n(G_1)$. Therefore, by Eq. (1) we have

$$n(\overline{G_{1a}}) \geq \lfloor 2n/3 \rfloor + 2. \tag{10}$$

On the other hand, since $n(G_{1a}) \geq n(G_{1b})$, we have

$$n(G_1) = n(G_{1a}) + n(G_{1b}) - 1 \leq 2n(G_{1a}) - 1$$

and hence

$$n(G_{1a}) \geq n(G_1)/2 + 1/2. \tag{11}$$

Substituting Eq. (1) to Eq. (11), we have

$$n(G_{1a}) \geq \lfloor 2n/3 \rfloor/2 + 3/2. \tag{12}$$

Substituting Eqs. (10) and (12) to $n = n(G_{1a}) + n(\overline{G_{1a}}) - 2$, we have

$$n \geq (3/2)\lfloor 2n/3 \rfloor + 3/2 \geq n + 1/2,$$

a contradiction.

(b) *Algorithm*

A series-parallel graph G can be represented by a decomposition tree T, which is a rooted binary tree and can be obtained in linear time [11]. Figure 5 depicts a decomposition tree T of the series-parallel graph G in Fig. 2(a). Each internal node of T labeled s means a series connection, while each internal node labeled p means a parallel connection. Each leaf of T corresponds to an edge of G. Each internal node u of T corresponds to a subgraph G_u of G induced by all edges that are descendants of u in T. The number of vertices in G_u is attached to an internal node u of T. Thus the root of T corresponds to G. One can easily observe that a desired bipartition of G can be found in linear time by the following procedure Bipartition(G, T).

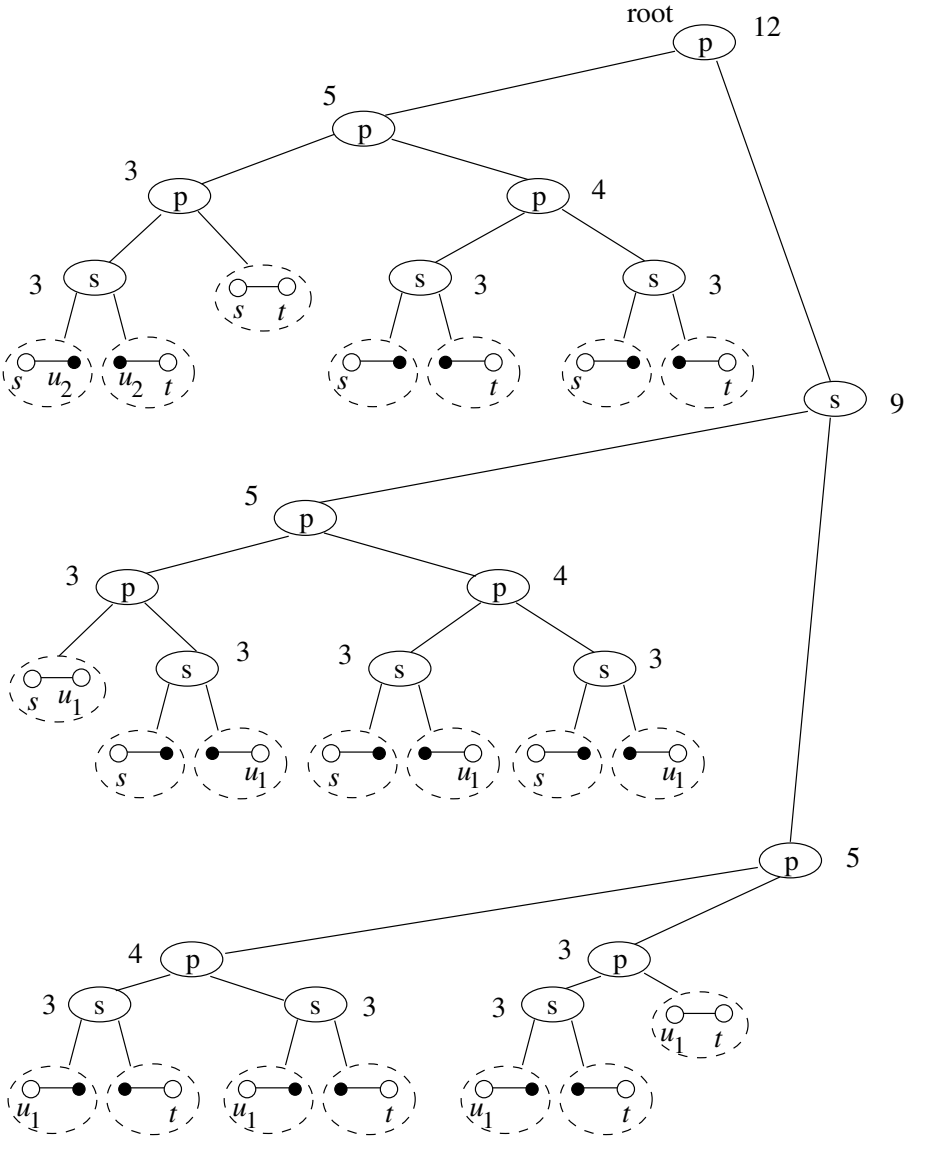


Fig. 5. A decomposition tree of the series-parallel graph in Fig. 2(a)

Bipartition(G, T);
{ G is a series-parallel graph of n vertices with a decomposition tree T. }
begin
let u be the root of T;
while $n(G_u) \geq \lfloor 2n/3 \rfloor + 2$ **do**
begin
let v_1 and v_2 be the two children of u in T;
if $n(G_{v_1}) \geq n(G_{v_2})$
then let $u := v_1$
else let $u := v_2$;
end;
let $G_1 := G_u$ and let G_2 be the remaining graph $\overline{G_1}$;
return a bipartition $\{G_1, G_2\}$;
end;

□

By Theorem 1 and Lemma 1 we immediately have the following theorem.

Theorem 2. *Every series-parallel graph of n vertices has a grid drawing D such that $W(D) \leq \lfloor 2n/3 \rfloor$ and $H(D) \leq \lfloor 2n/3 \rfloor - 1$. Furthermore such a drawing D can be found in linear time.*

One can observe from the example in Figs. 2(a) and (b) that the bound $\lfloor 2n/3 \rfloor + 1$ in Lemma 1 is best possible. Recently, Biedl showed that every series-parallel graph has a visibility representation of area $O(n^{3/2})$, which can be converted to a poly-line drawing of asymptotically the same area [13].

4 Conclusions

In this paper we showed that if a planar graph G has a balanced bipartition then G has a small grid drawing. In particular, we showed that every series-parallel graph has a balanced bipartition and has a $\lfloor 2n/3 \rfloor \times (\lfloor 2n/3 \rfloor - 1)$ grid drawing. We also showed that such a drawing can be found in linear time. There exist series-parallel graphs requiring $\Omega(n \log n)$ area in any grid drawing [14]. Thus, there is a big gap between this $\Omega(n \log n)$ lower bound and our $4n^2/9$ upper bound, and it is still open whether every series-parallel graph has a grid drawing of $o(n^2)$ area [13,15]. Adding dummy edges, one can extend a partial 2-tree to a 2-tree, which is series-parallel. Therefore, every partial 2-tree has a $\lfloor 2n/3 \rfloor \times (\lfloor 2n/3 \rfloor - 1)$ grid drawing. It is still open whether every planar graph has a $\lceil 2n/3 \rceil \times \lceil 2n/3 \rceil$ grid drawing.

References

1. Nishizeki, T., Rahman, M.S.: Planar Graph Drawing. World Scientific, Singapore (2004)
2. Chrobak, M., Kant, G.: Convex grid drawings of 3-connected planar graphs. International Journal of Computational Geometry and Applications 7, 211–223 (1997)

3. Schnyder, W.: Embedding planar graphs on the grid. In: Proc. of First ACM-SIAM Symposium on Discrete Algorithms, pp. 138–148 (1990)
4. Dolev, D., Leighton, F.T., Trickey, H.: Planar embedding of planar graphs. Advances in Computing Research 2, 147–161 (1984)
5. Frati, F., Patrignami, M.: A note on minimum-area straight-line drawings of planar graphs. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 339–344. Springer, Heidelberg (2008)
6. Miura, K., Nakano, S., Nishizeki, T.: Grid drawings of 4-connected plane graphs. Discrete & Computational Geometry 26, 73–87 (2001)
7. Shiloach, Y.: Arrangements of Planar Graphs on the Planar Lattice. PhD thesis, Weizmann Institute of Science (1976)
8. Battista, G.D., Frati, F.: Small area drawings of outerplanar graphs. Algorithmica 54, 25–53 (2009)
9. Chrobak, M., Payne, T.H.: A linear-time algorithm for drawing a planar graph on a grid. Information Processing Letters 54, 241–246 (1995)
10. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10, 41–51 (1990)
11. Takamizawa, K., Nishizeki, T., Saito, N.: Linear-time computability of combinatorial problems on series-parallel graphs. J. ACM 29, 623–641 (1982)
12. Nishizeki, T., Chiba, N.: Planar Graphs: Theory and Algorithms. Dover Publications, New York (2008)
13. Biedl, T.: On small drawings of series-parallel graphs and other subclasses of planar graphs. In: Proc. of GD 2009 (to appear)
14. Frati, F.: A lower bound on the area requirements of series-parallel graphs. In: Broersma, H., Erlebach, T., Friedetzky, T., Paulusma, D. (eds.) WG 2008. LNCS, vol. 5344, pp. 159–170. Springer, Heidelberg (2008)
15. Biedl, T.: Drawing outer-planar 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)

Switch-Regular Upward Planar Embeddings of Trees*

Carla Binucci[1], Emilio Di Giacomo[1], Walter Didimo[1], and Aimal Rextin[2]

[1] University of Perugia, Italy
{binucci,digiacomo,didimo}@dei.unipg.it
[2] University of Limerick, Ireland
aimal.tariq@ul.ie

Abstract. Upward planar drawings of digraphs are crossing free drawings where all edges flow in the upward direction. The problem of deciding whether a digraph admits an upward planar drawing is called the *upward planarity testing problem*, and it has been widely studied in the literature. In this paper we investigate a new version of this problem: Deciding whether a digraph admits a *switch-regular* upward planar drawing, i.e., an upward planar drawing with specific topological properties. Switch-regular upward planar drawings have applications in the design of efficient checkers and in the design of effective compaction heuristics. We provide characterizations for the class of directed trees that admit a switch-regular upward planar drawing. Based on these characterizations we describe an optimal linear-time testing and embedding algorithm.

1 Introduction

An *upward drawing* of a digraph is a drawing such that each vertex is mapped to a distinct point of the plane and all edges are drawn as simple Jordan curves monotonically increasing in the vertical direction. Upward drawings have a long tradition in Graph Drawing and they are commonly adopted for the visual representation of acyclic digraphs that model hierarchical structures, like PERT diagrams or class inheritance diagrams. A digraph is said to be *upward planar* if it admits an *upward planar drawing*, i.e., a crossing free upward drawing.

Although it is immediate to see that every acyclic digraph has an upward drawing, it is well known that not all acyclic planar digraphs are upward planar. Since edge crossings negatively affect the readability of a drawing, a very rich body of research has been devoted so far to the so called *upward planarity testing problem* (i.e., the problem of deciding whether or not a planar digraph admits an upward planar drawing), and many combinatorial and algorithmic results have been described (see, e.g., [8]). In particular, Bertolazzi *et al.* proved that given an embedded planar digraph G with n vertices, it is possible to decide in $O(n^2)$ time if G admits an embedding preserving upward planar drawing [2]. Conversely, Garg and Tamassia proved that the upward planarity testing problem in the variable embedding setting is NP-complete [12]. In the middle of these two results, polynomial-time upward planarity testing algorithms have been described for specific sub-families of planar digraphs [3,11,14,15] and more general exponential-time algorithms can be found in [1,6,13].

* The problem studied in this paper was posed during the first Bertinoro Workshop on Graph Drawing (http://www.diei.unipg.it/~bwgd/).

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 58–69, 2010.

Concerning the topological properties of upward planar drawings, Di Battista and Liotta discovered and characterized a meaningful sub-family of upward planar drawings whose embedding has some special properties of "regularity" [9]. Namely, an upward planar drawing has a *switch-regular* embedding if: (i) the boundary of each internal face contains at most one maximal subsequence of "small" angles (i.e., angles smaller than π) of length greater than one. (ii) the external boundary does not contain two consecutive "small" angles.

From a practical point of view, finding switch-regular upward planar embeddings is relevant for two main applications:

(i) Design of Efficient Checkers. A *checker* is an algorithm that efficiently checks the correctness of the output produced by another algorithm (see, e.g., [7]). Di Battista and Liotta showed that it is possible to design an optimal linear-time checker that verifies the correctness of a computed upward planar drawing, provided that its embedding is switch-regular. Namely, suppose we are given an algorithm that takes as input a planar digraph G and that computes, if any, an upward planar drawing Γ of G; if the embedding of Γ is switch-regular, the checker described by Di Battista and Liotta efficiently verifies the correctness of Γ in terms of upward planarity.

(ii) Effective Drawing Compaction. Area and aspect ratio are considered two of the most important aesthetic requirements for the readability of a drawing. There are works that experimentally show that, starting from a switch-regular embedding of a digraph or augmenting a non-switch regular embedding to a switch-regular one, it is possible to design heuristics that compute drawings with compact area and that dramatically improve aspect ratio with respect to previous drawing approaches [10]. We also remark that similar heuristics, based on a analogous concept of "regularity", were previously adopted and successfully experimented for the computation of orthogonal drawings [5].

These applications naturally motivate the following new upward planarity testing problem: *"Given a planar digraph G, is it possible to test in polynomial time whether G admits a switch-regular upward planar embedding?"*. In this paper we solve the problem for digraphs whose underlying undirected graph is a tree (we call such a digraph a *directed tree* for short). We remark that a directed tree always admits an upward planar embedding, but it may not admit a switch-regular one. Also, since an embedding of a tree has only one face (i.e., the external one) the switch-regularity reduces to the requirement that there are no two consecutive "small" angles along the face boundary. For example, Fig. 2(c) shows a tree that does not admit a switch-regular embedding. Our results are as follows:

– We provide three equivalent characterizations of switch-regular directed trees, i.e., directed trees that admit a switch-regular upward planar embedding.

– Exploiting the above characterizations, we describe an optimal linear-time algorithm that tests whether a directed tree is switch-regular and that computes a switch-regular upward planar embedding of the tree in the affirmative case.

We observe that, besides their practical relevance, our techniques make use of new theoretical ingredients that are interesting in their own right. The outline of the paper is as follows. Section 2 recalls the formal definition of switch-regular upward planar

embeddings. In Section 3 we define and study two main ingredients of our results: (i) We prove that switch-regular upward planar embeddings of a directed tree cannot contain subdivisions of a digraph that we call a *3-hook*. (ii) We define a new kind of decomposition for a directed tree, called *red-blue decomposition*, and we study the interplay between 3-hooks and the properties of red-blue decompositions. In Section 4 we study the relationship between red-blue decompositions and switch-regularity. Section 5 gives characterizations and a linear-time testing and embedding algorithm based on the results of Sections 3 and 4. Conclusions and open problems can be found in Section 6.

In the remainder of the paper many proofs are sketched or omitted for reasons of space. Complete proofs can be found in [4][1].

2 Basic Definitions

We assume familiarity with the basic concepts of graph drawing and graph planarity [8]. Let G be an embedded planar digraph. A vertex v of G is *bimodal* if all incoming edges of v (and hence all outgoing edges of v) are consecutive in the circular clockwise order around v. G is called *bimodal* if all its vertices are bimodal. Acyclicity and bimodality are necessary conditions for the upward planar drawability of an embedded planar digraph, but they are not sufficient conditions [2].

Let f be a face of G and suppose that the boundary of f is traversed counterclockwise. If $e_1 = (u_1, v)$ and $e_2 = (v, u_2)$ are two edges encountered in this order along the boundary of f, the triplet $s = (e_1, v, e_2)$ is called an *angle of* f. Note that, e_1 and e_2 may coincide if G is not biconnected. Angle s is called a *switch of* f if e_1 and e_2 are both incoming edges or both outgoing edges of v: in the first case s is also called a *sink-switch of* f, while in the second case it is a *source-switch of* f. Observe that the number of source-switches of f equals the number of sink-switches of f. We denote by $2n_f$ the total number of switches of f. The *capacity of* f is denoted by c_f and it is defined by $c_f = n_f - 1$ if f is an internal face and by $c_f = n_f + 1$ if f is the external face. If G is bimodal an assignment of the sources and the sinks of G to the faces of G is called an *upward consistent assignment* if: (i) Each source and each sink is assigned to exactly one of its incident face; (ii) for each face f, the total number of sources and sinks assigned to f is equal to c_f . The following theorem characterizes the embedded planar digraphs that admit an upward planar drawing.

Theorem 1. [2] *An acyclic embedded planar bimodal digraph is upward planar if and only if it admits an upward consistent assignment.*

The *upward planar embedding* of G corresponding to an upward consistent assignment of G is a planar embedding of G with labels at the switches of every face. Namely, a switch $s = (e_1, v, e_2)$ of f is labeled L when v is a source or a sink assigned to f and s is labeled S otherwise. If f is a face of an upward planar embedding, the circular list of labels of f is denoted by σ_f. Also, S_{σ_f} and L_{σ_f} denote the number of S and L labels of f, respectively. An upward planar drawing of a digraph G can be constructed

[1] Observe that in [4] red-blue decompositions are called red-black decompositions. We renamed them in order to avoid confusion with the well-known red-black tree data structure.

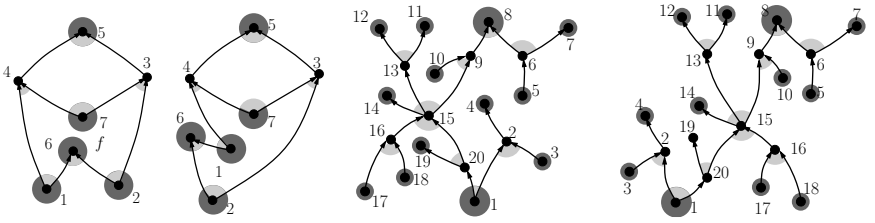


Fig. 1. (a) A non switch-regular embedding and (b) a switch-regular embedding of the same digraph. (c) A non switch-regular embedding and (d) a switch-regular embedding of the same directed tree. Dark angles are switches labeled L, light angles are switches labeled S.

for a given upward planar embedding of G; this drawing is such that each switch angle labeled L forms a geometric angle larger than π, and each switch angle labeled S forms a geometric angle smaller than π. An internal face f of an upward planar embedding is called *switch-regular* if σ_f does not contain two distinct maximal subsequences σ_1 and σ_2 of S labels such that $S_{\sigma_1} > 1$ and $S_{\sigma_2} > 1$. The external face f is *switch-regular* if σ_f does not contain two consecutive S labels.

An upward planar embedding is *switch-regular* if all its faces are switch-regular. We say that a digraph G is *switch-regular* if it admits a switch-regular upward planar embedding. Examples of switch-regular and non switch-regular upward planar embeddings are depicted in Fig. 1. Intuitively speaking, a switch-regular embedding does not have a face containing two L labeled angles that are "opposite" each other (like vertices 6 and 7 in face f of Fig. 1(a) or vertices 4 and 5 in Fig. 1(c)). The remainder of the paper concentrates on directed trees. We remark that a directed tree always admits an upward planar embedding, but it may not admit a switch-regular one.

If $e = (u, v)$ is a directed edge of a digraph G, a *subdivision* of e is a path of directed edges $(u, w_1), (w_1, w_2), \ldots, (w_k, v)$ that replaces e $(k > 0)$. A digraph obtained from G by subdividing some edges (possible none) of G is called a *subdivision* of G.

3 3-Hooks and Red-Blue Decompositions

A *hook* is a digraph H whose underlying undirected graph is a path consisting of three vertices such that the middle vertex is either a source or a sink of H. A 3-*hook* is a directed tree consisting of three hooks sharing an endvertex. A subdivision of a hook is called a *hook subdivision* and a subdivision of a 3-hook is called a 3-*hook subdivision*. Fig. 2(a) and 2(b) show two different 3-hook subdivisions. The *center* of a 3-hook subdivision is the vertex shared by the three hook-subdivisions; the *middle vertices* of a 3-hook subdivision are the vertices corresponding to the middle vertices of the 3-hook.

Lemma 1. *A switch-regular directed tree does not contain* 3*-hook subdivisions.*

Sketch of Proof: Assume by contradiction that T contains a 3-hook subdivision T'. We will show that T is not switch-regular in this case. Let ψ be an upward planar embedding of T, and let v be the center of T'. Assume that H is any of the three

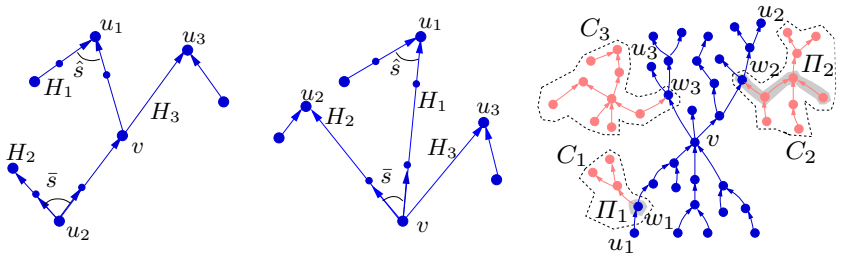


Fig. 2. (a)-(b) Illustration of Lemma 1: (a) **Case 1**; (b) **Case 2**. (c) A red-blue decomposition with respect to v; C_1, C_2, and C_3 are red components of $RB(T, v)$ and w_1, w_2, and w_3 are their corresponding attaching vertices. C_1 is strongly regular, C_2 is weakly regular, and C_3 is not regular. Π_1 and Π_2 are backbones of C_1 and C_2, respectively. The directed blue path from u_1 to v is an incoming attaching path of $RB(T, v)$, while the directed paths from v to u_2 and from v to u_3 are two outgoing attaching paths of $RB(T, v)$.

subgraphs of T' that are hook subdivisions, and let u denote the middle vertex of H. We say that H is an *incoming hook* (*outgoing hook*) if there is a directed path from u to v (v to u). Let ψ' be the upward planar embedding of T' induced by ψ; one of the two switches at u is labeled S in ψ' while the other is labeled L. We say that H is a *left hook* if walking counterclockwise around T', starting from v, the switch labeled L incident to u is encountered before the switch labeled S, while H is a *right hook* otherwise.

One of the two following cases always holds for ψ': **Case 1.** There is an incoming and an outgoing hook such that one is a right hook and the other one is a left hook. **Case 2.** There are two incoming or two outgoing hooks, such that both are either right hooks or left hooks. In both cases there are two consecutive switches $\hat{s}$ and $\bar{s}$ labeled S. In **Case 1** they are at the middle vertices of two different hooks (see Fig. 2(a)), while in **Case 2** they are one at the middle vertex of a hook and the other one is at the center of T' (see Fig. 2(b)).

Both **Case 1** and **Case 2** have four subcases. Let $\Pi = H_1 \cup H_2$ in Fig. 2(a) and let $\Pi = H_1$ in Fig. 2(b). **Sub-Case 1.** No switch is encountered when walking counterclockwise around T from $\hat{s}$ to $\bar{s}$. In this case $\hat{s}$ and $\bar{s}$ form a sequence of two consecutive S labels and therefore T is not switch-regular. **Sub-Case 2.** Walking counterclockwise from $\hat{s}$ to $\bar{s}$ the first switch $s_1 = (e_1, w, e'_1)$ encountered after $\hat{s}$ is such that e_1 is an edge of Π, w is a vertex of Π, and e'_1 leaves w. The switches $\hat{s}$ and s_1 form a sequence of two consecutive S labels because s_1 is also labeled S. **Sub-Case 3.** Walking counterclockwise from $\hat{s}$ to $\bar{s}$ the last switch $s_1 = (e'_1, w, e_1)$ encountered before $\bar{s}$ is such that e_1 is an edge of Π, w is a vertex of Π, and edge e'_1 enters w. The switches s_1 and $\bar{s}$ form a sequence of two consecutive S labels because s_1 is also labeled S. **Sub-Case 4.** Walking counterclockwise from $\hat{s}$ to $\bar{s}$ there are two switches $s_1 = (e'_1, w_1, e_1)$ and $s_2 = (e_2, w_2, e'_2)$ such that e_i is an edge of Π ($i = 1, 2$), w_i is a vertex of Π ($i = 1, 2$), e'_1 is an edge entering w_1 and e'_2 is an edge leaving w_2. Both s_1 and s_2 are labeled S. If they are consecutive they form a sequence of two consecutive S labels, otherwise walking counterclockwise from

s_1 to s_2 there are two consecutive switches with the same properties as s_1 and s_2. These switches form a sequence of two consecutive S labels. □

From now on we concentrate on trees with at least one vertex of degree larger than two, otherwise T is a path, which admits a switch-regular upward planar embedding.

An *hourglass tree* T is a directed tree with a vertex v such that either there is a directed path from v to every other vertex of T or there is a directed path from every other vertex of T to v. The vertex v is called the *center of the hourglass*. It easy to see that every upward planar embedding of an hourglass tree is switch-regular.

Let T be a directed tree and let v be a vertex of T with $deg(v) \geq 3$. A *red-blue decomposition* of T with respect to v is a coloring of the vertices and edges of T such that: (i) a vertex u of T is colored *blue* if there exists a directed path either from u to v or from v to u, and u is colored *red* otherwise; (ii) an edge e of T is colored *blue* if both its endvertices are blue, and e is colored *red* otherwise (see Fig. 2(c)). We denote by $RB(T, v)$ such a decomposition. If e is a red edge of $RB(T, v)$, then either both end-vertices of e are red or one is red and the other is blue. By definition the subgraph consisting of all blue vertices is an hourglass tree.

Let u and w be a red and a blue vertex of $RB(T, v)$, respectively. We say that u is *attached* to w if there exists a (non-directed) path from u to w whose vertices are all red vertices except w. We also say that w is the *attaching vertex* of u. Let $C' = (V_{C'}, E_{C'})$ be any connected component obtained by removing the blue vertices. Note that all vertices of C' have the same attaching vertex w. Let e be the (unique) edge of T that connects w to C'. The subtree $C = (V_{C'} \cup \{w\}, E_{C'} \cup \{e\})$ is called a *red component of* $RB(T, v)$ and vertex w is called the *attaching vertex* of the red component C. We say that C is *regular* if it contains a (non-directed) path Π having w as an endvertex and such that C minus the edges of Π is a forest of hourglass trees whose centers belong to Π. If C is regular, we call Π a *backbone* of C. We say that C is *strongly regular* if the path consisting of the only vertex w is a backbone of C (in this case no edge is removed from C). In other words, C is strongly regular if either all vertices of C are reachable with a directed path from w, or w is reachable with a directed path from all vertices of C. If C is regular but not strongly regular, it is said to be *weakly regular*. Fig. 2(c) shows examples of regular and non-regular components in a red-blue decomposition of a directed tree.

An *outgoing (incoming) attaching path of* $RB(T, v)$ is a directed blue path Π from v to a leaf of T (from a leaf of T to v) such that at least one vertex of Π is an attaching vertex (see Fig. 2(c)). Given an attaching path Π we call *last attaching vertex* of Π the attaching vertex that has maximum distance from v. Let Π_1 and Π_2 be two attaching paths. We say that Π_1 and Π_2 are *distinct* if their last attaching vertices are distinct. Clearly, an outgoing and an incoming attaching paths are always distinct. We say that Π_1 and Π_2 are *commonly oriented* if they are both incoming or both outgoing. Let u be a vertex of an attaching path Π of $RB(T, v)$; we say that u is: (i) a k*-regular* vertex ($k > 0$), if it is the attaching vertex of at least k regular red components; a k-regular vertex is also called *regular*. (ii) A k*-weak-regular* vertex ($k > 0$), if it is the attaching vertex of at least k weakly regular red components; a k-weak-regular vertex is also called *weak-regular*. (iii) A *branch* vertex, if it has two incident blue edges such that they are both entering/leaving u and one of them belongs to Π. (iv) A *branch attaching*

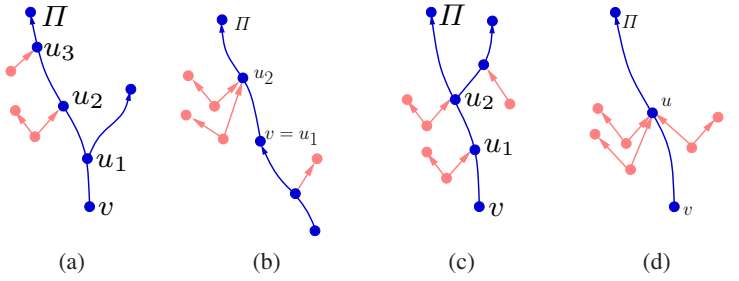


Fig. 3. Some examples of forbidden configurations. (a) **FC1**: u_1 is branch, u_2 is weak-regular and u_3 is regular. (b) **FC2**: u_1 is internal attaching, u_2 is 2-weak-regular. (c) **FC3**: u_1 is weak-regular, u_2 is weak-regular and branch attaching. (d) **FC4**: u is 3-weak-regular.

vertex, if it is a branch shared by two distinct attaching paths. Note that a k-weak-regular vertex is also k-regular and that a branch attaching vertex is also a branch vertex. We say that v is *internal attaching* if it is shared by an incoming and an outgoing attaching path. Fig. 3 shows some examples of these concepts.

Let $RB(T, v)$ be a red-blue decomposition of T with respect to v, such that all red components of $RB(T, v)$ are regular and let Π be an attaching path of $RB(T, v)$. We have a *forbidden configuration* if, walking along Π starting from v, we encounter:

FC1. Three not necessarily consecutive vertices u_1, u_2, u_3 (u_1 may coincide with v) such that: u_1 is either weak-regular or branch or internal attaching, u_2 is weak-regular or branch attaching, and u_3 is regular or branch attaching (see, e.g., Fig. 3(a)).

FC2. Two not necessarily consecutive vertices u_1, u_2 (u_1 may coincide with v) such that u_1 is either weak-regular or branch or internal attaching, and u_2 is either 2-weak-regular or weak-regular and branch attaching at the same time (see, e.g., Fig. 3(b)).

FC3. Two not necessarily consecutive vertices u_1, u_2 (u_1 may coincide with v) such that u_1 is either 2-weak-regular or weak-regular and branch attaching at the same time, and u_2 is either regular or branch attaching (see, e.g., Fig. 3(c)).

FC4. One vertex that is either 3-weak-regular or 2-weak-regular and branch attaching at the same time (see, e.g., Fig. 3(d)).

Let T be a directed tree and let v be a vertex of T with $deg(v) \geq 3$. $RB(T, v)$ is said to be *regular* if the following conditions hold: **RB1.** $RB(T, v)$ has at most two distinct attaching paths; **RB2.** Every red component of $RB(T, v)$ is regular; **RB3.** $RB(T, v)$ has no forbidden configuration.

Lemma 2. *Let T be a directed tree. If T does not contain 3-hook subdivisions, then for every vertex v with $deg(v) \geq 3$ $RB(T, v)$ is regular.*

Sketch of Proof: Assume by contradiction that there exists a vertex v with $deg(v) \geq 3$ such that $RB(T, v)$ is not regular. We show that T contains a 3-hook subdivision. Since $RB(T, v)$ is not regular then at least one of **RB1**-**RB3** does not hold. Here, we only consider the case when **RB1** does not hold, the other cases can be proven with similar arguments. Since **RB1** does not hold, there exist at least three distinct attaching paths

Π_1, Π_2, and Π_3. Let u_i be an attaching vertex of Π_i not shared with another attaching path Π_j and let e_i be a red edge incident to u_i ($1 \leq i \neq j \leq 3$). Let w_i be the last vertex (i.e., the farthest from v) of Π_i shared with another attaching path Π_j and let Π_i' be the portion of Π_i from w_i to u_i ($1 \leq i \neq j \leq 3$). We have the following two cases. **Case 1:** $w_1 = w_2 = w_3$**.** Notice that $w_1 = w_2 = w_3$ may coincide with v. In this case $\Pi_i' \cup e_i$ is a hook subdivision with w_i as an endvertex ($i = 1, 2, 3$). Hence, there is a 3-hook subdivision with center w_1. **Case 2:** $w_1 = w_2$**.** Notice that $w_1 = w_2$ does not coincide with v and therefore Π_1 and Π_2 are commonly oriented. In this case $\Pi_i' \cup e_i$ is a hook subdivision with w_i as an endvertex ($i = 1, 2$). Let Π^* be the path from w_1 to w_3. If Π_3 is commonly oriented with Π_1 and Π_2, let e be the edge of Π_3' incident to w_3. Then $\Pi^* \cup e$ is a hook subdivision with w_1 as an endvertex and therefore we have a 3-hook subdivision. If Π_3 is not commonly oriented with Π_1 and Π_2, then $\Pi^* \cup e_3$ is a hook subdivision with w_1 as an endvertex and therefore we have a 3-hook subdivision. □

4 Red-Blue Decompositions and Switch-Regularity

Lemmas 1 and 2 imply that if a directed tree T with a vertex v with $deg(v) \geq 3$ is switch-regular then $RB(T, v)$ is regular. In this section we prove that the converse is also true, i.e., that if $RB(T, v)$ is regular then T is switch-regular. To this aim we describe an algorithm that computes a switch-regular upward planar embedding of T. The algorithm first computes an upward planar embedding of the blue subtree T_b of $RB(T, v)$ (**Phase 1**). Then it adds the weakly regular red components (**Phase 2**) and finally it adds the strongly regular red components (**Phase 3**). For reasons of space, we describe each phase of the algorithm and omit the proofs of their correctness (Lemmas 5, 6 and 7). The next two lemmas will be used in the description of the phases.

Lemma 3. *Let T be a directed tree and let v be a vertex of T with $deg(v) \geq 3$ and such that $RB(T, v)$ is regular. If $RB(T, v)$ has only one attaching path, then it has at most two weak-regular vertices. If $RB(T, v)$ has two distinct attaching paths, then each of them can have at most one weak-regular vertex. Moreover, there cannot be a weak-regular vertex shared by two distinct attaching paths.*

Lemma 4. *Let T be a directed tree with a vertex v such that $deg(v) \geq 3$. A weakly regular red component of $RB(T, v)$ admits a switch-regular upward planar embedding.*

Phase 1 - Computing an upward planar embedding of T_b. Tree T_b is an hourglass tree, hence every upward planar embedding of T_b is switch-regular. We choose an embedding of T_b that allows red components to be added while maintaining switch-regularity. Let Π be an outgoing (incoming) attaching path of $RB(T, v)$. Let u_1, w, u_2 be three vertices encountered consecutively in this order when walking along Π starting from v. We say that Π is *left externally embedded* if: (i) the edge e of Π incident to v is the last outgoing (incoming) edge in the counterclockwise order around v; (ii) for every pair of edges $e_1 = (u_1, w)$ and $e_2 = (w, u_2)$ on Π, the triplet (e_2, w, e_1) is an angle of T_b. Angle (e_2, w, e_1) is said to be the *external angle* of w; also, (e_2, w, e_1) is said to be a left angle. We say that Π is *right externally embedded* if: (i) the edge e of Π

incident to v is the first outgoing (incoming) edge in the counterclockwise order around v; (ii) for every pair of consecutive edges $e_1 = (u_1, w)$ and $e_2 = (w, u_2)$ on Π, the triplet (e_1, w, e_2) is an angle of T_b. Angle (e_1, w, e_2) is said to be the *external angle* of w; also, (e_1, w, e_2) is said to be a right angle. When an attaching path Π is externally embedded, each attaching vertex of Π has at least one external angle. An attaching vertex w may have both a left angle and a right angle if all paths leaving (entering) v have a common subpath Π_s and $w \in \Pi_s$.

We describe how to construct an embedding of T_b by distinguishing the following cases: **Case 1:** $RB(T, v)$ has one attaching path. We choose an upward planar embedding of T_b where the attaching path is externally embedded. **Case 2:** $RB(T, v)$ has two distinct commonly oriented attaching paths. We choose an upward planar embedding where both attaching paths are externally embedded. **Case 3:** $RB(T, v)$ has one incoming and one outgoing attaching path. We choose an upward planar embedding where the attaching paths are either both right externally embedded or both left externally embedded. **Case 4:** $RB(T, v)$ has two distinct attaching paths that share a subpath. It can be proved that there is no branch vertex before the branch attaching vertex because otherwise there would be a forbidden configuration **FC1**. Hence, we can choose an upward planar embedding where the two attaching paths are both externally embedded.

Let s be an angle of a tree T, *prev*(s) and *next*(s) denote the switches that precede and follow s in the counterclockwise order around T, respectively.

Lemma 5. *Let T be a directed tree with a vertex v such that $deg(v) \geq 3$ and $RB(T, v)$ is regular. The upward planar embedding of T_b computed by Phase 1 is switch-regular and for each external angle s,* prev(s) *and* next(s) *are labeled L.*

Phase 2 - Adding weakly regular red components. By Lemma 3 there are at most two weakly regular red components. We assume that there is at least one weakly regular red component because otherwise this phase is not executed. By Lemma 4 each weakly regular red component C has a switch-regular upward planar embedding. Let s^* be the only switch of C at the attaching vertex of C. A switch-regular upward planar embedding of C is a *left embedding* if *prev*(s^*) is labeled L, and a *right embedding* otherwise. Note that, in a right embedding *next*(s^*) is labeled L. Given a switch-regular upward planar embedding of C, it is possible to make this embedding a left or a right embedding. When we add C to T_b we say that C is *left (right) embedded* to mean that C is added inside the left (right) angle of its attaching vertex and a left (right) embedding is chosen for C.

The weakly regular red components are added to the embedding of T_b as follows. If the attaching vertex w of C has only one external angle s, then C will be left or right embedded depending on the fact that s is a left or a right angle. If w has two external angles, C is left or right embedded according to the following cases. **Case 1:** There is only one attaching path Π. Assume that Π is left externally embedded, the other case is symmetric. If there is only one weakly regular red component, then it is right embedded. If there are two weakly regular red components attached to distinct attaching vertices, then the first attaching vertex w_1 on Π has two external angles (otherwise there is a forbidden configuration of type **FC1**), while attaching vertex w_2 has either one or two external angles. If w_2 has only one external angle then this is a left angle because Π is left externally embedded. The weakly regular red component attached to w_1 will

be right embedded, the other one will be left embedded. If $w_1 = w_2$, then w_1 has two external angles; in this case one of the two component is left embedded and the other one is right embedded. **Case 2:** There are one incoming and one outgoing attaching paths. Assume the two attaching paths are left externally embedded, the other case is symmetric. If there is only one weakly regular red component, then it is left embedded. If there are two weakly regular red components, then they are attached to distinct attaching paths by Lemma 3. By hypothesis, one of the two attaching vertices has two external angles, while the other one, call it w, can have one or two external angles. If w has only one external angle, then this is a left angle because the two attaching paths are left externally embedded. In both cases both weakly regular red components can be left embedded. **Case 3:** There are two commonly oriented attaching paths. An attaching vertex w with two external angles can exist in this case only if the two attaching paths Π_1 and Π_2 share a subpath and w belongs to this subpath. However, by Lemma 3, w cannot be the attaching vertex of a weakly regular red component.

Lemma 6. *Let T be a directed tree with a vertex v such that $deg(v) \geq 3$ and $RB(T, v)$ is regular. Let T_W be the subtree of T induced by T_b and the weakly regular red components of $RB(T, v)$. The upward planar embedding of T_W computed by Phase 2 is switch-regular.*

Phase 3 - Adding the strongly regular red components. Let T_W be the tree induced by T_b plus the weakly regular red components. Let $C_1, C_2, \ldots, C_k$ be the strongly regular red components, and let w_i be the attaching vertex of C_i, where $i = 1, 2, \ldots, k$. Since different strongly regular red components may have the same attaching vertex, some of the vertices denoted as w_i, $1 \leq i \leq k$, may coincide. We consider the components C_i ($1 \leq i \leq k$) one per time. Since C_i is an hourglass, every upward planar embedding of C_i is switch-regular. Hence, we choose an arbitrary upward planar embedding for C_i and then we add C_i to the embedding by placing C_i inside an *insertion angle* s_i identified as follows. If, when C_i is considered, there are no red components (weakly or strongly) with attaching vertex w_i already added to the embedding, then s_i is the external angle of w_i; otherwise s_i is an angle formed by a blue edge of the attaching path containing w_i and a red edge incident to w_i. This can be done while maintaining switch-regularity.

Lemma 7. *Let T be a directed tree with a vertex v, such that $deg(v) \geq 3$ and $RB(T, v)$ is regular. Then T is switch-regular.*

5 Characterization and Test

The next theorem immediately follows from Lemmas 1, 2, and 7. From Theorem 2 and Lemma 2 we can prove Lemma 8, which shows that if a directed tree T is switch-regular then $RB(T, u)$ is regular for each vertex u of T with $deg(u) \geq 3$.

Theorem 2. *Let T be a directed tree with at least one vertex of degree larger than two. The following three statements are equivalent: (a) T is switch-regular. (b) T does not contain 3-hook subdivisions. (c) There exists a vertex v with $deg(v) \geq 3$ such that $RB(T, v)$ is regular.*

Lemma 8. *Let T be a directed tree and let v_1 and v_2 be any two vertices of T such that $deg(v_1) \geq 3$ and $deg(v_2) \geq 3$. If $RB(T, v_1)$ is regular, then $RB(T, v_2)$ is regular.*

Proof. Since $RB(T, v_1)$ is regular, then by Theorem 2 T does not contain 3-hook subdivisions. By Lemma 2, $RB(T, v)$ is regular for each v with $deg(v) \geq 3$, hence $RB(T, v_2)$ is regular. □

We now give a short description of the testing algorithm. Based on Theorem 2 the testing algorithm checks that a red-blue decomposition of T is regular. By Lemma 8 it can test a red-blue decomposition of T with respect to an arbitrarily chosen vertex v with $deg(v) \geq 3$. $RB(T, v)$ can be easily computed by performing two visits of T starting from v. During the first (second) visit only the edges oriented away from (towards) the root are considered. The blue vertices/edges are those reached/traversed by one of the two visits. Once $RB(T, v)$ is computed the algorithm verifies that conditions **RB1**-**RB3** hold. In the following, we assume that T is rooted at v (note that, this does not imply that all edges are commonly oriented towards or away from v). Hence, each red component is rooted at its attaching vertex.

– RB1 can be tested by counting the number of distinct attaching paths whose last attaching vertex is a descendant of v. This is done by a recursive visit of the blue subtree T_b of $RB(T, v)$. At the generic blue vertex u, if the sum of the values returned by the recursive invocations of the algorithm on the children of u is larger than 0, this number is returned; otherwise the value returned is 0 if u is not an attaching vertex and 1 if it is. **RB1** holds if the invocation of the algorithm on v returns at most 2.

– RB2 can be tested by performing a recursive visit of each red component. Let u be a vertex of a red component C distinct from the attaching vertex w of C and let u' be the parent of u; we denote by T_u the subtree of C rooted at u and by T'_u the subtree $T_u \cup \{(u, u')\}$. The algorithm is invoked with a red vertex u of a red component C as a parameter and returns 0 if T'_u is a strongly regular red component, 1 if it is a weakly one, and a value larger than 1 if it is not regular. If u is a leaf the algorithm returns 0. If u is not a leaf, let sum denote the sum of the values returned by the recursive invocations of the algorithm on the children of u. If $sum = 0$, for each child u_i of u, T'_{u_i} is a strongly regular red component with backbone $\{u\}$. Then, if edges (u, u_i) are entering u and edge (u, u') is leaving u (or viceversa), the algorithm returns 0. If there exists at least one edge (u, u_i) such that (u, u') and (u, u_i) are both leaving (or both entering) u, then T'_u of C is a weakly regular red component with backbone $\{u', u\}$ and the algorithm returns 1. If $sum = 1$, there exists exactly one subtree T'_{u_i} that is a weakly regular red component with backbone $\{u, u_i, \ldots, x\}$. In this case $T_u \setminus T'_{u_i}$ is an hourglass tree with center u and the path $\{u', u, u_i, \ldots, x\}$ is a backbone of T'_u: the algorithm returns 1. If $sum > 1$ either there exists one subtree T'_{u_i} that is not regular, in which case also T'_u is not regular, or there exist at least two subtrees T'_{u_i} and T'_{u_j} that are weakly regular red components with backbones $\{u, u_i, \ldots, x\}$ and $\{u, u_j, \ldots, y\}$, respectively. Also in this case T'_u is not regular. In both cases the algorithm returns a value greater than 1.

– RB3 can be checked by scanning the vertices of each attaching path and using information previously stored, so to verify that there is no forbidden configuration **FC1**-**FC4**.

If the testing algorithm on a directed tree T returns true, a switch-regular upward planar embedding of T can be computed according to the techniques illustrated in Section 4. It can be proved that the time complexity of the embedding algorithm is $O(n)$.

Theorem 3. *Let T be a directed tree with n vertices. There exists an $O(n)$ time algorithm to test whether T is switch-regular and, in the affirmative case, to compute a switch-regular upward planar embedding of T.*

6 Conclusions and Open Problems

We addressed the new problem of deciding whether an acyclic digraph admits a special kind of upward planar embedding, called *switch-regular*. Besides the theoretical interest of this problem, our research is motivated by the relevance of switch-regular upward planar embeddings in different graph drawing applications. We described different characterizations and a linear-time testing and embedding algorithm for the class of directed trees. It remains open the question whether the switch-regular upward planarity testing problem for general digraphs is NP-hard or not. We believe that it can also be interesting to extend our results to other sub-families of planar digraphs.

References

1. Bertolazzi, P., Di Battista, G., Didimo, W.: Quasi-upward planarity. Algorithmica 32(3), 474–506 (2002)
2. Bertolazzi, P., Di Battista, G., Liotta, G., Mannino, C.: Upward drawings of triconnected digraphs. Algorithmica 6(12), 476–497 (1994)
3. Bertolazzi, P., Di Battista, G., Mannino, C., Tamassia, R.: Optimal upward planarity testing of single-source digraphs. SIAM Journal on Computing 27, 132–169 (1998)
4. Binucci, C., Giacomo, E.D., Didimo, W., Rextin, A.: Switch-regular upward planarity testing of directed trees. Technical Report RT-001-09, DIEI, Univ. Perugia (2009), http://www.diei.unipg.it/rt/rt_frames.htm
5. Bridgeman, S.S., Battista, G.D., Didimo, W., Liotta, G., Tamassia, R., Vismara, L.: Turn-regularity and optimal area drawings of orthogonal representations. Comput. Geom. 16(1), 53–93 (2000)
6. Chan, H.: A parameterized algorithm for upward planarity testing. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 157–168. Springer, Heidelberg (2004)
7. Devillers, O., Liotta, G., Preparata, F.P., Tamassia, R.: Checking the convexity of polytopes and the planarity of subdivisions. Comput. Geom. 11(3-4), 187–208 (1998)
8. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing. Prentice-Hall, Upper Saddle River (1999)
9. Di Battista, G., Liotta, G.: Upward planarity checking: "faces are more than polygon". In: Whitesides, S.H. (ed.) GD 1998. LNCS, vol. 1547, pp. 72–86. Springer, Heidelberg (1999)
10. Didimo, W.: Upward planar drawings and switch-regularity heuristics. Journal of Graph Algorithms and Applications 10(2), 259–285 (2006)
11. Didimo, W., Giordano, F., Liotta, G.: Upward spirality and upward planarity testing. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 117–128. Springer, Heidelberg (2006)
12. Garg, A., Tamassia, R.: On the computational complexity of upward and rectilinear planarity testing. SIAM Journal on Computing 31(2), 601–625 (2001)
13. Healy, P., Lynch, K.: Fixed-parameter tractable algorithms for testing upward planarity. International Journal of Foundations of Computer Science 17(5), 1095–1114 (2006)
14. Hutton, M.D., Lubiw, A.: Upward planarity testing of single-source acyclic digraphs. SIAM Journal on Computing 25(2), 291–311 (1996)
15. Papakostas, A.: Upward planarity testing of outerplanar dags. In: Tamassia, R., Tollis, I.G. (eds.) GD 1994. LNCS, vol. 894, pp. 298–306. Springer, Heidelberg (1995)

A Global k-Level Crossing Reduction Algorithm*

Christian Bachmaier, Franz J. Brandenburg,
Wolfgang Brunner, and Ferdinand Hübner

University of Passau, Germany
{bachmaier,brandenb,brunner,huebnerf}@fim.uni-passau.de

Abstract. Directed graphs are commonly drawn by the Sugiyama algorithm, where crossing reduction is a crucial phase. It is done by repeated one-sided 2-level crossing minimizations, which are still $\mathcal{NP}$-hard.

We introduce a global crossing reduction, which at any particular time captures all crossings, especially for long edges. Our approach is based on the sifting technique and improves the level-by-level heuristics in the hierarchic framework by a further reduction of the number of crossings by 5 – 10%. In addition it avoids type 2 conflicts which help to straighten the edges, and has a running time which is quadratic in the size of the input graph independently of dummy vertices. Finally, the approach can directly be extended to cyclic, radial, and clustered level graphs where it achieves similar improvements over the previous algorithms.

1 Introduction

The Sugiyama framework [12] is the standard drawing algorithm for directed graphs. It displays them in a hierarchical manner and operates in four phases: cycle removal (reverse appropriate edges to eliminate cycles), leveling (assign vertices to levels which define the y-coordinates and introduce dummy vertices on long edges), crossing reduction (permute the vertices on the levels), and coordinate assignment (assign x-coordinates to the vertices under some aesthetic criteria). Typical applications are schedules, UML diagrams, and flow charts.

In this paper we focus on the crossing reduction phase, where the vertices on each level are permuted to minimize the total number of crossings. The common solution for *k-level crossing minimization* is a reduction to the *one-sided 2-level crossing minimization problem*, which is solved repeatedly in some up and down sweeps [12,9]. In the down sweep, the vertices V_{i-1} on the upper level are fixed and the vertices V_i of the lower level are reordered reducing the local number of edge crossings. In the up sweep the roles are switched. Even the one-sided 2-level crossing minimization problem is $\mathcal{NP}$-hard [6]. There are many heuristics for this problem [9]. Bastert and Matuszewski claim [9] that the results of this level-by-level sweep are far from optimum. "One can expect better results by considering all levels simultaneously, but k-level crossing minimization is a very hard problem" [9, page 102]. Our approach addresses this gap. Note that existing approximation ratios of 2-level algorithms do not translate to k-level graphs.

* Supported by the Deutsche Forschungsgemeinschaft (DFG), grant BR835/15-1.

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 70–81, 2010.

An important feature of such algorithms is the guarantee of no *type 2 conflicts* which are crossings of two edges between dummy vertices. Among others, the standard fourth phase algorithm [4] by Brandes and Köpf assumes the absence of type 2 conflicts. Then it aligns long edges vertically and so achieves a crucial aesthetic criterion [9] for pleasing hierarchical drawings.

Common 2-level crossing reductions are the *barycenter* and *median heuristics* [9]. They place each vertex $v \in V_i$ in the barycenter or median position of its predecessors in V_{i-1}. After that V_i is sorted by these values. The idea is that on these positions the edges get short and, thus, generate few crossings. These techniques are simple, fast, and avoid type 2 conflicts, but leave many crossings.

Although such 2-level algorithms reduce the crossings between V_{i-1} and V_i, the number of crossings between V_i and V_{i+1} (and thus even the overall number of crossings) can increase while permuting V_i. These heuristics push the crossings downwards or upwards until they are resolved at level k or 1, respectively. An extension is *centered 3-level crossing reduction*, i. e., treating three consecutive levels V_{i-1}, V_i, V_{i+1} and permuting V_i while the orders of V_{i-1} and V_{i+1} are fixed s. t. the crossings between the three levels are reduced. However, this generates type 2 conflicts. For reaching a global optimum, all these algorithms are restricted to a local view. Thus, they may tend to get stuck in local optima.

Sifting was first used for vertex minimization in ordered binary decision diagrams [11] and later adapted to the one-sided 2-level crossing reduction [10]. The idea is to keep track of the number of crossings while in a *sifting step* a vertex $v \in V_i$ is moved along a fixed ordering of the vertices in V_i. Finally v is placed at its locally optimal position. The method is an extension of the greedy-switch heuristic [5], where v is swapped iteratively with its current successor. We call a single swap a *sifting swap* and the execution of a sifting step for every vertex in V_i a *sifting round*. Sifting leaves fewer crossings than the simple heuristics in general at the expense of a higher running time and potential type 2 conflicts [9].

Matuszewski et al. [10] have extended sifting towards a global view, which we call *ordered k-level sifting*. There the vertices are sorted by their degree and are sifted first in increasing order and then in decreasing order. All neighbors of the vertices to swap, i. e., on both neighboring levels, are considered. The heuristic does not sweep level-by-level but is still limited to a local view as long edges are not treated as a whole. Our *centered 3-level sifting* does the same level-by-level instead of ordered by degree. Both algorithms produce similar results. Jünger et al. [8] presented an exact ILP approach for the $\mathcal{NP}$-hard k-level crossing minimization, which can be used in practice for small graphs. Moreover, metaheuristics have been proposed in the literature, such as genetic algorithms, tabu search, or windows optimization.

In this paper we propose a new and global crossing reduction technique. The algorithm yields better results than traditional heuristics. It is easily extendable to more general crossing reduction problems, avoids type 2 conflicts, and runs in quadratic time in the size of the graph. Most 2-level approaches extensively use dummy vertices, whose number is up to $\mathcal{O}(k \cdot |E|) \subseteq \mathcal{O}(|V|^3)$ and do not make use of the edge bundling techniques of [7], which cannot be used for sifting.

2 Preliminaries

We suppose that a directed graph without self-loops has passed through the cycle removal and leveling phases. The outcome is a *k-level graph* $G = (V, E, \phi)$, where $\phi\colon V \to \{1, 2, \ldots, k\}$ is a surjective level assignment of the vertices with $\phi(u) < \phi(v)$ for each edge $(u, v) \in E$. For an edge $e = (u, v) \in E$ we define $\text{span}(e) := \phi(v) - \phi(u)$. An edge e is *short* if $\text{span}(e) = 1$ and *long* otherwise. A graph is *proper* if all edges are short. Each level graph can be made proper by adding $\text{span}(e) - 1$ dummy vertices for each edge e which split e in $\text{span}(e)$ many short edges. Let $G' = (V', E', \phi')$ denote the proper version of G. As in [4], short edges are called *segments* of e. The first and the last segments are the *outer* and the others the *inner segments*. Inner segments connect two dummy vertices.

For a vertex v we denote the set of neighbors from incoming and outgoing segments by $N^-(v) := \{\, u \in V' \mid (u, v) \in E' \,\}$ and $N^+(v) := \{\, w \in V' \mid (v, w) \in E' \,\}$, respectively. In an *ordered* proper level graph the vertices on each level as well as the sets $N^-(\cdot)$ and $N^+(\cdot)$ are ordered from left to right. Each proper level graph can be made ordered by choosing an arbitrary ordering for each level and sorting the sets $N^-(\cdot)$ and $N^+(\cdot)$ accordingly. In an ordered level graph there are two *conflicting* segments if they cross or share a vertex. Conflicts are of *type 0, 1* or *2*, if they are induced by 0, 1, or 2 inner segments, respectively.

Next we define blocks, which prevent dealing with dummy vertices and so keep the running time independent of them. A *block* is a single vertex of V or a maximum connected subgraph of dummy vertices, i. e., the inner segments of a long edge. The blocks represent the vertices of a graph related to G', where the edges are the outer segments. For a block A define $x = \text{upper}(A)$ ($y = \text{lower}(A)$) to be the unique vertex x in A (y in A) with no incoming (outgoing) segments in A. x and y always exist but may coincide. We define $N^-(A) := N^-(\text{upper}(A))$, $N^+(A) := N^+(\text{lower}(A))$, $\deg(A) := |N^-(A)| + |N^+(A)|$, and the set of all level numbers on which A has (dummy) vertices as $\text{levels}(A)$. With $\text{block}(v)$ we denote the block of the vertex $v \in V'$. Let $\mathcal{B}$ be any ordered list of all blocks and let $\pi\colon \mathcal{B} \to \{0, \ldots, |\mathcal{B}| - 1\}$ assign each block its current *position* in this ordering.

3 Global Sifting

A major drawback of the established crossing reduction algorithms is their local view. We present a new approach using ideas from [4] and [7][1] and avoiding type 2 conflicts. We treat all dummy vertices of an edge (and each original vertex) as one block and try to find the best position for the entire block in one step. This eliminates the problems of classic 2-level approaches which lack this global view on crossings of long edges. As an initialization the list of blocks $\mathcal{B}$ is sorted arbitrarily and each block A gets $\pi(A)$ as its position in $\mathcal{B}$ (line 1 in Algorithm 1). At any time during the execution of the algorithm interpreting $\pi(A)$ for each

[1] The authors of [7] use a data structure similar to our blocks and avoid type 2 conflicts. However, for crossing reduction they proceed level-by-level in the traditional fashion. Thus, only the running time but not the quality of the result is improved.

Algorithm 1. GLOBAL-SIFTING

Input: Proper k-level graph $G' = (V', E', \phi')$, number ρ of sifting rounds
Output: Graph G' with vertices ordered by values $\pi(v)$ for each $v \in V'$

```
1 create list B of all blocks in G'
2 for 1 ≤ i ≤ ρ do
3 |   foreach A ∈ B do
4 |   |   B ← SIFTING-STEP(G', B, A)
5 foreach v ∈ V' do π(v) ← π(block(v))
6 return G'
```

block A as an x-coordinate for each vertex v in A and $\phi(v)$ as its y-coordinate results in a drawing respecting the current ordering of $\mathcal{B}$. All vertices of a block get the same x-coordinate and, thus, the ordering is type 2 conflict free. These are important invariants of Algorithm 1.

The main part of the algorithm is the sifting step (line 4). There all positions for a block A are tested and A is moved to that position where it has the fewest crossings. This is done for each block $A \in \mathcal{B}$ (line 3) and repeated a certain number of times ρ (line 2). In practice, ten rounds suffice. Finally, each vertex is set to the position of its block (line 5) and the graph is returned (line 6).

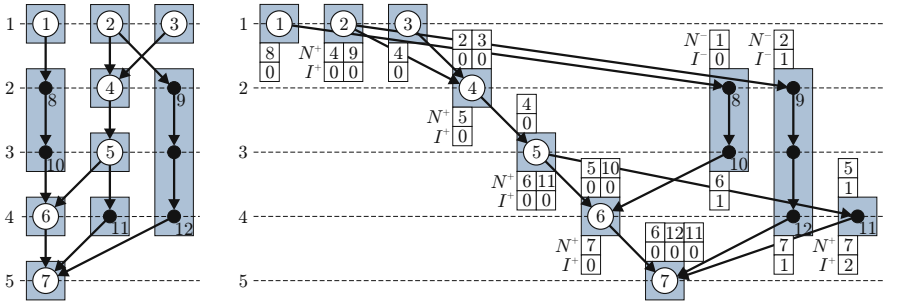
3.1 Building the Block List

The graph is partitioned into blocks. Each block A gets an arbitrary but unique position $\pi(A)$ in the block list $\mathcal{B}$. As an example consider Fig. 1(a). The input graph with 7 vertices gets 6 dummy vertices drawn as black circles. The dummy vertices are combined into 3 blocks and each original vertex forms its own block. The 10 resulting blocks are shown in Fig. 1(b) with an arbitrary ordering π.

If a given ordering should only be improved in a postprocessing step, a straightforward initialization strategy is to topologically sort the blocks according to the orderings on the levels from left to right in $\mathcal{O}(|E'|)$. Our experiments showed, that a good initial ordering of the blocks leads to better results. However, these can also be achieved by one or two additional sifting rounds.

3.2 Initialization of a Sifting Step

To improve the performance of one sifting step [3] it is necessary to keep the adjacency lists $N^-(A)$ and $N^+(A)$ of each block $A \in \mathcal{B}$ sorted according to ascending positions of the neighboring blocks in $\mathcal{B}$. We store them as arrays for random access. Additionally, we store two index arrays $I^-(A) = I^-(\text{upper}(A))$ and $I^+(A) = I^+(\text{lower}(A))$ of lengths $|I^-(A)| := |N^-(A)|$ and $|I^+(A)| := |N^+(A)|$, respectively. $I^-(A)$ stores the indices where upper(A) is stored in each adjacent block B's adjacency $N^+(B)$. More precisely, let $b = N^-(A)[i]$ be a neighbor of upper(A) with $B = \text{block}(b)$. Then $I^-(A)[i]$ holds the index at which upper(A)



(a) A level graph with ten blocks

(b) Separate π-positions, ordered adjacency lists N^- and N^+, and index arrays I^- and I^+ for each block

Fig. 1. Blocks as sifting objects

is stored in $N^+(B) = N^+(b)$. Symmetrically, $I^+(A)$ stores the indices at which lower(A) is stored in the adjacency $N^-(B)$ of each adjacent block B. See Fig. 1(b) for an example. The creation of the four arrays for each block (line 2 of Algorithm 3) can be done in $\mathcal{O}(|E|)$ time as Algorithm 2 shows: Traverse the blocks A in the current order of $\mathcal{B}$ and add upper(A) (lower(A)) to the next free position j of the cleared adjacency array $N^+(\text{lower}(B))$ ($N^-(\text{upper}(B))$) of each incoming (outgoing) neighbor B. Both values for $I^+(B)$ and $I^-(A)$ ($I^-(B)$ and $I^+(A)$) and their positions are only known after the second traversal of a segment s. Thus, we cache the first array position j as an attribute p of s. Benchmarks have shown that there is a considerable speed-up if only those adjacencies are updated that are no longer sorted after a sifting step. The theoretical running time is unaffected by this improvement.

Algorithm 2. SORT-ADJACENCIES

Input: Proper k-level graph $G' = (V', E', \phi')$, ordered list $\mathcal{B}$ of blocks in G'
Output: Ordered sets $N^{\cdot}(A)$ and $I^{\cdot}(A)$ for each block $A \in \mathcal{B}$

```
1  for i ← 0 to |B| − 1 do π(B[i]) ← i; clear arrays N·(B[i]) and I·(B[i])
2  foreach A ∈ B do
3      foreach s ∈ { (u, v) ∈ E' | v = upper(A) } do
4          add v to the next free position j of N+(u)
5          if π(A) < π(block(u)) then p[s] ← j                  // first traversal of s
6          else I+(u)[j] ← p[s]; I−(v)[p[s]] ← j                // second traversal of s
7      foreach s ∈ { (w, x) ∈ E' | w = lower(A) } do
8          add w to the next free position j of N−(x)
9          if π(A) < π(block(x)) then p[s] ← j                  // first traversal of s
10         else I−(x)[j] ← p[s]; I+(w)[p[s]] ← j                // second traversal of s
```

3.3 Sifting Step

In a sifting step (Algorithm 3) all positions p in $\mathcal{B}$ are tested for a block $A \in \mathcal{B}$ (lines 5–8) and then A is moved to the position p^* which has caused the least number of crossings. Note that it is not necessary to count the crossings for each position of A. As in [3] and contrary to classic sifting which always maintains the absolute number of crossings, we treat the number of crossings of A when put to the first position as $\chi = 0$. Then, we only compute the change in the number of crossings when iteratively swapping A with its right neighbor (line 6).

Algorithm 3. SIFTING-STEP

Input: Proper k-level graph $G' = (V', E', \phi')$, ordered list $\mathcal{B}$ of blocks in G', block $A \in \mathcal{B}$ to sift
Output: Updated ordering of $\mathcal{B}$

```
1 B' ← A ≺ B[0] ≺ ··· ≺ B[|B| − 1]          // new ordering B' with A put to front
2 SORT-ADJACENCIES(G', B')
3 χ ← 0; χ* ← 0                              // current and best number of crossings
4 p* ← 0                                     // best block position
5 for p ← 1 to |B'| − 1 do
6 |   χ ← χ + SIFTING-SWAP(A, B'[p])
7 |   if χ < χ* then
8 |   |   χ* ← χ; p* ← p
9 return B'[1] ≺ ··· ≺ B'[p*] ≺ A ≺ B'[p* + 1] ≺ ··· ≺ B'[|B'| − 1]
```

3.4 Sifting Swap

The sifting swap is the actual computation of the change in the number of crossings when a block A is swapped with its right neighbor B. In contrast to one-sided crossing reduction, our global approach takes the whole neighborhood of both blocks into account when the change in the number of crossings is computed. Lemma 1 states which segments are involved.

Lemma 1. *Let $\mathcal{B}$ be the block list in the current ordering. Let $B \in \mathcal{B}$ be the successor of $A \in \mathcal{B}$. If swapping A and B changes the crossings between any two segments, then one of them is an incident outer segment of A or B. The other segment is an incident outer segment of the same kind (incoming or outgoing) of the other block or an inner segment of the other block.*

Proof. Note that only segments between the same levels can cross. As no type 2 conflicts occur at least one of the segments of a crossing has to be an outer segment. Let (a, b) and (c, d) be two segments between the same levels with $a \neq c$ and $b \neq d$. If the two segments cross after swapping A and B but did not cross before (or vice versa) either a and c or b and d were swapped. Therefore, one of the segments is adjacent to A or is a part of A and the other is adjacent to B or

is a part of B. If b and d were swapped and thus a and c were not, $\phi(b) = \phi(d)$ is the upper level of A or B and thus one of the crossing segments is an incoming outer segment of A or B. The other segment is either an incoming outer segment or an inner segment of the other block. Note that it cannot be an outgoing outer segment of this block because then neither a and c nor b and d would have been swapped. The other case of swapping a and c instead of b and d is symmetric. □

Proposition 1. *Let $\mathcal{B}$ be the block list in the current ordering. Let $B \in \mathcal{B}$ be the successor of $A \in \mathcal{B}$. Let i and j be the two levels framing the incoming outer segments of A, the other three cases are symmetric. If there is a segment (u, v) between i and j which is either an incoming outer segment of B or an inner segment of B, then the incoming segments of A starting at a block left of* $\text{block}(u)$ *cross (u, v) after the swap of A and B only, the segments starting at* $\text{block}(u)$ *never cross (u, v), and the segments starting right of* $\text{block}(u)$ *cross (u, v) before the swap only. There are no other changes of crossings due to Lemma 7.*

Algorithm 4 shows the details of a sifting swap. First, the levels at which (significant) swaps occur and the direction of the segments changing their crossings are found (lines 2–6). For each entry (l, d) of the set $\mathcal{L}$ the two vertices a and b of A and B on level l are retrieved. Note that when swapping A and B only a and b are swapped on their level and that in the level of their neighbors $N^d(a)$ and $N^d(b)$ no order changes. Thus, the computation of the change in the number of crossings can be done as in [3] (lines 14–24): The neighbors are traversed from left to right. If a neighbor of a is found (lines 19, 20) its segment will cross all remaining $s - j$ incident segments of b after the swap. If a neighbor of b is found (lines 21, 22) its segment has crossed all remaining $r - i$ incident segments of a before the swap. Common neighbors present both cases at the same time (line 23). An update of the adjacency after a swap (line 10) is only necessary if a and b have common neighbors. Algorithm 5 shows how this can be done in overall $\mathcal{O}(\deg(A) + \deg(B))$ time similarly to the crossing counting function `uswap`.

3.5 Time Complexity

Lemma 2. *Let $G = (V, E, \phi)$ be a level graph. Then $\sum_{B \in \mathcal{B}} \deg(B) \leq 4 \cdot |E|$.*

Proof. Each edge $e \in E$ contains at most two outer segments. Each outer segment increases the degree of its two incident blocks by one each. □

Theorem 1. *One round of global sifting (Algorithm 1) has a time complexity of $\mathcal{O}(|E|^2)$ for a non-necessarily proper level graph $G = (V, E, \phi)$.*

Proof. Let $\mathcal{B}$ be the blocks of G. Swapping two blocks $A, B \in \mathcal{B}$ needs $\mathcal{O}(\deg(A) + \deg(B))$ time. Initializing a sifting step takes $\mathcal{O}(\sum_{B \in \mathcal{B}} \deg(B)) = \mathcal{O}(|E|)$ time. A sifting step of a block A needs $\mathcal{O}(\sum_{B \in \mathcal{B} \setminus \{A\}} (\deg(A) + \deg(B))) = \mathcal{O}(|E| \cdot \deg(A))$ time. Thus, a sifting round positioning each block $A \in \mathcal{B}$ has time complexity $\mathcal{O}(\sum_{A \in \mathcal{B}} (|E| \cdot \deg(A)) = \mathcal{O}(|E|^2)$. Since $|V'| \leq k \cdot |E| \in \mathcal{O}(|E|^2)$ (no empty levels), traversing all (dummy) vertices in pre- and postprocessing has no effect on the worst case time complexity. □

Algorithm 4. SIFTING-SWAP

Input: Consecutive blocks A, B
Output: Change in crossing count

begin
  $\mathcal{L} \leftarrow \emptyset$; $\Delta \leftarrow 0$
  if $\phi(\text{upper}(A)) \in \text{levels}(B)$ **then** $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\phi(\text{upper}(A), -)\}$
  if $\phi(\text{lower}(A)) \in \text{levels}(B)$ **then** $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\phi(\text{lower}(A), +)\}$
  if $\phi(\text{upper}(B)) \in \text{levels}(A)$ **then** $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\phi(\text{upper}(B), -)\}$
  if $\phi(\text{lower}(B)) \in \text{levels}(A)$ **then** $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\phi(\text{lower}(B), +)\}$
  foreach $(l, d) \in \mathcal{L}$ **do**
    let a in A and b in B be the vertices with $\phi(a) = \phi(b) = l$
    $\Delta \leftarrow \Delta + \mathsf{uswap}(a, b, N^d(a), N^d(b))$
    UPDATE-ADJACENCY$(a, b, N^d(a), I^d(a), N^d(b), I^d(b))$
  swap positions of A and B in $\mathcal{B}$; $\pi(A) \leftarrow \pi(A) + 1$; $\pi(B) \leftarrow \pi(B) - 1$
  return Δ
end

function *uswap*$(a, b, N^d(a), N^d(b))$: *integer*
  let $x_0 \prec \cdots \prec x_{r-1} \in N^d(a)$ be the neighbors of a in direction d
  let $y_0 \prec \cdots \prec y_{s-1} \in N^d(b)$ be the neighbors of b in direction d
  $c \leftarrow 0; i \leftarrow 0; j \leftarrow 0$
  while $i < r$ *and* $j < s$ **do**
    if $\pi(\text{block}(x_i)) < \pi(\text{block}(y_j))$ **then**
      $c \leftarrow c + (s - j)$; $i \leftarrow i + 1$
    else if $\pi(\text{block}(x_i)) > \pi(\text{block}(y_j))$ **then**
      $c \leftarrow c - (r - i)$; $j \leftarrow j + 1$
    else $c \leftarrow c + (s - j) - (r - i)$; $i \leftarrow i + 1$; $j \leftarrow j + 1$
  return c

Algorithm 5. UPDATE-ADJACENCIES

Input: Vertices $a, b \in V'$, $N^d(a), I^d(a), N^d(b), I^d(b)$
Output: Updated adjacencies of a and b and all common neighbors

let $x_0 \prec \cdots \prec x_{r-1} \in N^d(a)$ be the neighbors of a in direction d
let $y_0 \prec \cdots \prec y_{s-1} \in N^d(b)$ be the neighbors of b in direction d
$i \leftarrow 0; j \leftarrow 0$
while $i < r$ *and* $j < s$ **do**
  if $\pi(\text{block}(x_i)) < \pi(\text{block}(y_j))$ **then** $i \leftarrow i + 1$
  else if $\pi(\text{block}(x_i)) > \pi(\text{block}(y_j))$ **then** $j \leftarrow j + 1$
  else
    $z \leftarrow x_i$ // $= y_j$
    swap entries at positions $I^d(a)[i]$ and $I^d(b)[j]$ in $N^{-d}(z)$ and in $I^{-d}(z)$
    $I^d(a)[i] \leftarrow I^d(a)[i] + 1$; $I^d(b)[j] \leftarrow I^d(b)[j] - 1$
    $i \leftarrow i + 1; j \leftarrow j + 1$

4 Simple Global Crossing Reductions

We have extended the barycenter and median crossing reduction strategies towards blocks as well: We iteratively take the π-positions of the blocks in $\mathcal{B}$ and compute the barycenter or median for each block, respectively, and sort $\mathcal{B}$ according to these values. Our benchmarks show that both are very fast, however, are not competitive with global sifting in the number of crossings.

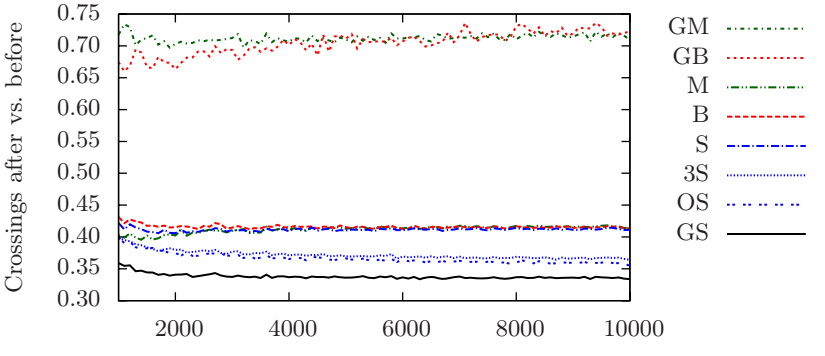
Theorem 2. *One round of global barycenter or global median has time complexity* $\mathcal{O}(|E| \log |E|)$ *or* $\mathcal{O}(|E|)$*, respectively.*

Proof. Computing the barycenters or medians for the $\mathcal{O}(|E|)$ blocks can be done in $\mathcal{O}(|E|)$ time due to Lemma 2. Sorting the barycenters takes $\mathcal{O}(|E| \log |E|)$ time. The medians can be sorted in $\mathcal{O}(|E|)$ time using bucket sort. □

5 Experimental Results

We have compared the iterative one-sided 2-level barycenter (B), median (M), and sifting (S), iterative centered 3-level sifting (3S), ordered k-level sifting (OS), and our new global barycenter (GB), global median (GM), and global sifting (GS) algorithms.

In a nutshell, classic sifting is fast, leaves few type 2 conflicts, but many crossings. Centered 3-level sifting is fast, leaves few crossings, but many type 2 conflicts. Global sifting leaves even less crossings (Fig. 2) without any type 2 conflicts within a still feasible running time in practice (Fig. 3). Further measurements reflect that the running time of global sifting is independent of the number of dummy vertices. This parallels the advanced algorithm in [7].



Graph size $|V'|$ (75% dummy vertices and $|E'| = 2 \cdot |V'|$, i. e., $|E| = 5 \cdot |V|$)

Fig. 2. Benchmark: number of crossings after vs. before applying the crossing reduction

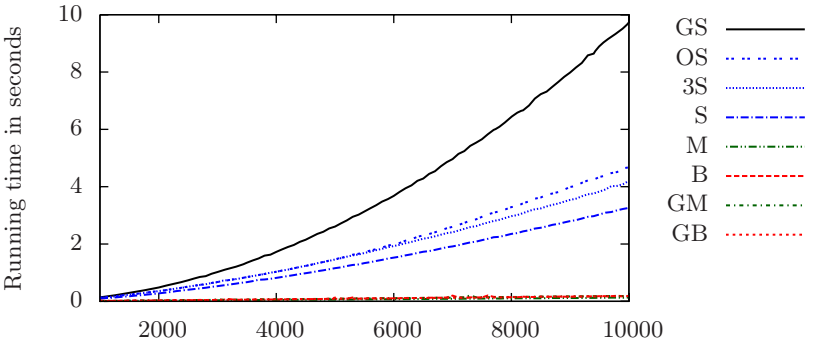


Fig. 3. Benchmark: running times

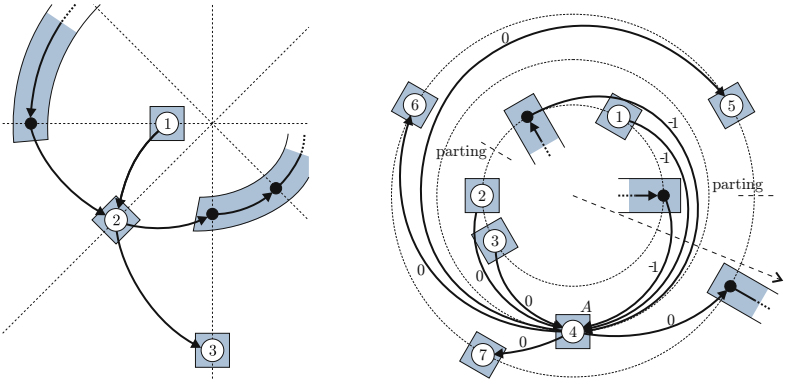
6 Applications of the Global Crossing Reduction

The idea of using blocks for long edges can be used in several other algorithms to improve their performance in a straightforward way. Further, this advances the drawability of their results as type 2 conflicts are avoided.

Optimal Crossing Reduction Using an ILP. Jünger et al. [8] gave an ILP formulation for the exact crossing minimization of k-level graphs. Using pairs of overlapping blocks, i. e., on non-disjoint levels, as variables gives a direct formulation which naturally excludes type 2 conflicts and uses fewer variables.

Clustered Crossing Reduction. In a clustered level graph vertices are combined to subgraphs in a hierarchical way. The crossing reduction has to ensure that all (dummy) vertices of a subgraph on the same level are consecutive and that all subgraphs spanning several levels have a matching ordering on each level to avoid crossings of subgraphs. This is rather complicated using a 2-level crossing reduction approach. Using global sifting this is quite simple: Instead of swapping a vertex with its right neighbor in a sifting swap we swap all blocks of a subgraph with its right neighbor (which itself is either a block or a subgraph) and determine the change in the number of crossings. The time complexity stays the same as in the normal global sifting. If the layout of the subgraphs themselves is not fixed, then global sifting can be applied to the subgraphs as well, e. g., performing a sifting round for each hierarchical layer.

Cyclic and Radial Level Graphs. Level graphs can be extended to cyclic or radial level graphs. In cyclic level graphs the set of levels is ordered in a cyclic way, i. e., the first level follows the last one. In radial level graphs each level itself is ordered in a cyclic way, i. e., the first vertex on each level is the right neighbor of the last one. See Fig. 4 for clippings of drawings. For both, global sifting is the first crossing reduction to guarantee the needed absence of type 2 conflicts.



(a) Cyclic drawing

(b) Partings of the block A in a radial drawing

Fig. 4. Clippings of cyclic and radial drawings

Cyclic levels are normally drawn forming a star in 2D (see Fig. 4(a)). These drawings explicitly visualize cycles in graphs [2], which is often required in bioinformatics. Our global sifting algorithm can be used for cyclic level graphs without any changes within the same time complexity. Note that one-sided 2-level algorithms cannot be applied here, since each of them pushes most of the crossings to the next level only. Even the absence of type 2 conflicts cannot be guaranteed then, because the sweep has to stop at some level.

In a radial level graph the levels are concentric circles (see Fig. 4(b)). These drawings visualize distances or importance and are the traditional drawings of social networks. They map structural centrality of the graph to geometric centrality. Our global sifting approach guarantees radially aligned long edges and can be used with minor modifications: Each block of the block list $\mathcal{B}$ has its own angle. The ordering of $\mathcal{B}$ starts at an arbitrary block. Similar to [1], we define an *offset* $\psi : E \to \mathbb{Z}$ for each outer segment. The absolute value $|\psi(e)|$ counts the crossings of segment e with an imaginary *ray* splitting up the levels with a straight halfline from the concentric center to infinity. If $\psi(e) < 0$ ($\psi(e) > 0$), e has clockwise (counter-clockwise) direction read from source to target. When sifting a block $A \in \mathcal{B}$, we have to update the *partings*, which are the two borders between the counterclockwise and clockwise segments on the levels above and below A, see Fig. 4(b). Since we can do this independently of each other and add the results of the change in crossings to Δ in Algorithm 4, we use the same technique as in [1]. We sift a block from its current position in counterclockwise direction. Thus, for few crossings the partings have to follow in this direction on their levels. The test during the swap whether changing the orientations of some of the first of the (ordered) incident segments of A by incrementing their offsets, and thus putting them last, leads to less crossings and counting the difference

raises the overall running time to $\mathcal{O}(|E|^3)$. The radial coordinate assignment phase in [1] relies on the obtained absence of type 2 conflicts.

7 Summary

We have presented an algorithm for the global crossing reduction problem of k-level graphs. It produces high quality results with fewer crossings than common approaches at the expense of a quadratic running time, which is still feasible in practice. This was an open problem since the introduction of the hierarchical framework [12] in 1981. For cyclic and radial level crossing reduction we presented the first algorithms which guarantee the absence of type 2 conflicts. Our approach can easily be used to simplify and improve several other algorithms concerning level planarity or crossing reduction.

References

1. Bachmaier, C.: A radial adaption of the sugiyama framework for visualizing hierarchical information. IEEE Trans. Vis. Comput. Graphics 13(3), 583–594 (2007)
2. Bachmaier, C., Brunner, W.: Linear time planarity testing and embedding of strongly connected cyclic level graphs. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 136–147. Springer, Heidelberg (2008)
3. Baur, M., Brandes, U.: Crossing reduction in circular layouts. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 332–343. Springer, Heidelberg (2004)
4. Brandes, U., Köpf, B.: Fast and simple horizontal coordinate assignment. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 31–44. Springer, Heidelberg (2002)
5. Eades, P., Kelly, D.: Heuristics for reducing crossings in 2-layered networks. Ars Combinatorica 21(A), 89–98 (1986)
6. Eades, P., Wormald, N.C.: Edge crossings in drawings of bipartite graphs. Algorithmica 11(1), 379–403 (1994)
7. Eiglsperger, M., Siebenhaller, M., Kaufmann, M.: An efficient implementation of sugiyama's algorithm for layered graph drawing. J. Graph Alg. App. 9(3), 305–325 (2005)
8. Jünger, M., Lee, E.K., Mutzel, P., Odenthal, T.: A polyhedral approach to the multi-layer crossing minimization problem. In: Di Battista, G. (ed.) GD 1997. LNCS, vol. 1353, pp. 13–24. Springer, Heidelberg (1997)
9. Kaufmann, M., Wagner, D. (eds.): Drawing Graphs. LNCS, vol. 2025. Springer, Heidelberg (2001)
10. Matuszewski, C., Schönfeld, R., Molitor, P.: Using sifting for k-layer straightline crossing minimization. In: Kratochvíl, J. (ed.) GD 1999. LNCS, vol. 1731, pp. 217–224. Springer, Heidelberg (1999)
11. Rudell, R.: Dynamic variable ordering for ordered binary decision diagrams. In: Proc. IEEE/ACM International Conference on Computer Aided Design, ICCAD 1993, pp. 42–47. IEEE Computer Society Press, Los Alamitos (1993)
12. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Trans. Syst., Man, Cybern. 11(2), 109–125 (1981)

Computation of Non-dominated Points Using Compact Voronoi Diagrams

Binay Bhattacharya[1], Arijit Bishnu[2], Otfried Cheong[3], Sandip Das[2], Arindam Karmakar[2], and Jack Snoeyink[4]

[1] School of Computing Science, Simon Fraser University, Canada
[2] Advanced Computing and Microelectronics Unit, Indian Statistical Institute, 203, B.T. Road, Kolkata, India - 700108
[3] Department of Computer Science, Korea Advanced Institute of Science and Technology, Gwahangno 335, Yuseong-gu, Daejeon 305-701, Korea
[4] Department of Computer Science, University of North Carolina at Chapel Hill, USA

Abstract. We discuss in this paper a method of finding skyline or non-dominated points in a set P of n points with respect to a set S of m sites. A point $p_i \in P$ is non-dominated if and only if for each $p_j \in P$, $j \neq i$, there exists at least one point $s \in S$ that is closer to p_i than p_j. We reduce this problem of determining non-dominated points to the problem of finding sites that have non-empty cells in an additively weighted Voronoi diagram under convex distance function. The weights of the said Voronoi diagram are derived from the co-ordinates of the points of P and the convex distance function is derived from S. In the 2-dimensional plane, this reduction gives a $O((m+n)\log m + n\log n)$-time randomized incremental algorithm to find the non-dominated points.

1 Introduction

Consider a trip to a conference in a new city! A set P of n hotels (located at fixed locations) has already been identified from a travel guide. On reaching the city, the scientist identifies a set S of m sites to visit, say for example the conference venue, museum, restaurant, garden, beach, etc. The scientist wants to visit all sites in S but prefers a hotel that has at least one site in S that is closer to it than any other hotel. Now, which are the most interesting hotels in the set P with respect to the sites of S in terms of distance? A hotel is interesting if it has at least one site closer to it than any other hotel. This problem gives rise to the *spatial skyline queries* [11]. A point $p_i \in P$ is a *skyline point* if it has at least one site in S that is closer to p_i than to any other point in P.

There can be applications of this problem in other areas like identifying a set of buildings for quick evacuation in case of multiple fires. Here, the set of buildings is P and the set of multiple fires is S. The set of skyline points is the buildings among P that are to be evacuated ahead of the other buildings. Sharifzadeh and Shahabi [11] identify some other applications as well.

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 82–93, 2010.

1.1 Formal Definition

Let $P = \{p_1, \ldots, p_n\}$ be a set of n points and $S = \{s_1, \ldots, s_m\}$ be a set of m sites in $\mathbb{R}^2$. Let $d(x, y)$ be the usual L_2 distance in $\mathbb{R}^d$. Each point $p_i \in P$ has m spatial attributes determined by all the m distances $\{d(p_i, s) \mid s \in S\}$, of the points in S to p_i. Next, we define *domination*.

Definition 1. *(Domination) [11] Given a set P of n points and a query set S of m sites in the plane, $p_i \in P$ dominates $p_j \in P$ $(j \neq i)$ with respect to S if and only if $d(p_i, s) < d(p_j, s)$, $\forall s \in S$.*

In this setting, the distances to the sites of S can be considered *feature vectors* that describe the sites p_i and p_j. The feature vector for p_i dominates the vector for p_j if and only if it dominates on all coordinates. If p_i dominates p_j, then p_j is a non-interesting point (hotel) vis-a-vis p_i with respect to S. Note that p_i is not dominated by p_j if it has at least one point in S that is closer to it than p_j.

We define *skyline points* as the set of those points in P which are *not dominated* by any other point in P with respect to S. We denote the set of skyline points as $\mathcal{SP}$. We will use the terms *skyline points* and *non-dominated points* interchangeably.

Definition 2. *(Skyline point) $p_i \in P$ is a skyline point if and only if we have the following:*

$$\text{for each } p_j \in P,\ i \neq j,\ \exists s \in S \text{ such that } \ d(p_i, s) < (p_j, s) \tag{1}$$

Our problem is to extract skyline points of P with respect to S. Consider a brute force approach. Let $h(p_i, p_j)$ be the half-plane of p_i with respect to p_j. For each $p_i \in P$, determine if there is at least a point $s \in S$ which lies in $h(p_i, p_j)$ for all $j \neq i$. If at least one such s is found for every p_j, then p_i is a skyline point. This takes $O(nm)$ time for each p_i. With the assumption that $n = m$, the total time complexity is $O(n^3)$.

1.2 Prior Work

Given two points $p_i = (p_i^1, p_i^2, \ldots, p_i^d)$ and $p_j = (p_j^1, p_j^2, \ldots, p_j^d)$ in $\mathbb{R}^d$, p_i dominates p_j if and only if $p_i^m \leq p_j^m$ for $1 \leq m \leq d$ and $p_i^m < p_j^m$ for some $1 \leq m \leq d$. For a point set P in $\mathbb{R}^d$ the *skyline query* finds those points in P which are not dominated by any other point. *Skyline operator* was introduced by Börzsönyi et al. [1]. They implemented *skyline query* to update an existing (relational, object-oriented or object-relational) database system with a new logical operator that they refer to as the *skyline operator*. Börzsönyi et al. used divide-and-conquer techniques and index structures to solve the problem in $O(n \log^{d-2} n + n \log n)$ time where d is the dimensionality of the points. Since the introduction of skyline query and skyline operators by Börzsönyi et al. [1], there have been several works using nearest neighbor search [7], sorting [4] and index structures [9,13]. These works mostly try to show an experimental improvement over the results of Börzsönyi et al. [1].

The problem of *Spatial skyline query*, as introduced in this paper, was first addressed by Sharifzadeh et al. [11]. The basic difference between the work of Börzsönyi et al. [1] and Sharifzadeh et al. [11] lies in the definition of *domination*. Börzsönyi et al. define *domination* between two points based on their respective coordinates, whereas Sharifzadeh et al. define *domination* between two points with respect to a set of points as given in Section 1.1. Note that the method of Börzsönyi et al. can be applied to the problem of *spatial skyline query*, but then the time complexity would be $O(n \log^{m-2} n + n \log n)$ if all the $O(nm)$ distances are already computed. Sharifzadeh et al. [11] propose an $O(m^2|\mathcal{SP}| + \sqrt{n})$ algorithm for the above problem where $|\mathcal{SP}|$ denotes the cardinality of the solution set. They solve this problem using *Voronoi diagram*, *convex hull* and *Delaunay graph*. Observe that if we put $m = O(1)$ in the time complexity derived by Sharifzadeh et al. [11], their worst-case time complexity becomes $O(n)$ which is highly unlikely. Moreover, Son et al. [12] have shown that the algorithm and the time complexity analysis of Sharifzadeh et al. [11] is incorrect. They also proposed a solution whose time complexity is $O(n|\mathcal{SP}| \log |\mathcal{CH}(S)| + n \log n)$ where $|\mathcal{SP}|$ and $|\mathcal{CH}(S)|$ denote the cardinality of non-dominated points and convex hull of the set S respectively. If we look at the worst case complexity of the algorithm devised by Son et al. [12], it turns out to be $O(n^2 \log m + n \log n)$.

1.3 Our Work

As discussed earlier in Section 1.1, the set of skyline points in P is the non-dominated subset of P with respect to S. In Section 2, we show using lifting techniques [2] that the set of non-dominated points has a correspondence with a lower envelope of cones where each point $p \in P$ has a corresponding cone in 3-D. In Section 2.1, we show that the non-dominated points of P correspond to the apices of the lower envelope of the said cones. Computing the lower envelope of cones is also costly. So, we show in Section 2.2 that the lower envelope of these cones corresponds to an additively weighted Voronoi diagram for a convex distance function. The skyline points of P with respect to S are those with non-empty Voronoi cells under this convex distance function determined by S with additive weights determined by P.

After having shown the relation of skyline points to the non-empty cells of the said Voronoi diagram, we proceed in Section 3 along the lines of McAllister et al. [8], where compact diagrams that avoid the high combinatorial complexity of Voronoi diagrams under convex distance function are used for solving certain problems. We show that computing a compact diagram that can be used to find non-empty Voronoi cells under a convex distance function determined by S with additive weights determined by P takes $O((m + n) \log m + n \log n)$ time for a randomized incremental construction.

2 Reduction to a Voronoi Diagram

In this section we relate point domination to additively weighted Voronoi diagrams of a convex distance function in $\mathbb{R}^2$. We will define these terms as we go,

culminating in Theorem 7 at the end of this Section. We state Theorem 7 as Result 1 here.

Result 1. *The non-dominated points of set P with respect to sites S are those with non-empty Voronoi cells under a convex distance function determined by S with additive weights determined by P.*

A brief sketch relating non-domination and lower envelope of cones to non-empty Voronoi cells under a convex distance function with additive weights is as follows. Choose an origin in the convex hull of S. The cones corresponding to P are obtained as follows. Consider an unit paraboloid with its apex at the origin. Fix a point $p \in P$. Let its lifted version on the unit paraboloid be p'. Now, for all sites $s \in \mathcal{CH}(S)$, consider the discs centered at s and passing through p. Each such disc, if lifted onto the unit paraboloid, forms a plane passing through p'. Now, the lifted versions of all such discs form a cone with apex at p'. So, we will have n such cones. The apices of the lower envelope of cones correspond to the non-dominated or skyline points.

This scheme can be alternatively interpreted in a Voronoi diagram model as follows. The lifted coordinates of the points in P are taken as their additive weights. With the same origin and the unit paraboloid as before, lift each site of S to planes tangent to the unit paraboloid. Translate these planes to include the origin and intersect their halfspaces to define a cone. The cross section of this cone at a unit distance vertically above the origin defines a convex polygon. We take this convex polygon to define a convex distance function. Thus, the convex distance function is determined by S. The lower envelope of cones with apices at the lifted points of P bounds the non-dominated (additively weighted) points, and can be interpreted as an additively weighted Voronoi diagram for the said convex distance function. We elucidiate further.

2.1 Dominated Points and the Cone

Let $C(x, y)$ denote a disc with center x and radius equal to $d(x, y)$. For each point $p_i \in P$, consider disc $C(s_k, p_i)$ centered at each $s_k \in S$. Obviously, for any point $p \in \mathbb{R}^2$ inside the disc $C(s_k, p_i)$ s_k is closer to p than to p_i. Therefore, if a point p_j dominates p_i, then $p_j \in \cap_{k=1}^{m} C(s_k, p_i)$. Let $D_i = \cap_{k=1}^{m} C(s_k, p_i)$. We term D_i as the *dominator region* of p_i. The significance of such a *dominator region* is that any point belonging to D_i dominates p_i with respect to S. Now we have the following observations.

Observation 2. *For any point $p_i \in P$:*

(i) *D_i is nonempty and may overlap with another D_j where $j \neq i$*
(ii) *D_i is a convex region bounded by circular arcs.*
(iii) *p_i is a skyline point if its dominator region D_i does not contain any point $p_j \in P, j \neq i$ in its interior.*
(iv) *if a point $p_j \in P$ lies inside the region $D_i, i \neq j$, then the dominator region D_j of p_j is a subset of D_i, i.e. $D_j \subset D_i$*

Proof. (i), (ii) and (iii) are trivial. We prove (iv) by contradiction. If p_j lies inside D_i then all s_ks' are nearer to p_j than p_i; so p_j dominates p_i with respect to S. Assume that D_i does not contain D_j. This implies that there is a point $z \in D_j$ and $z \notin D_i$, such that z dominates p_j but not p_i. But p_j dominates p_i. As the domination relation is transitive there is a contradiction. Hence, we have the observation. □

Note that, for any dominator region D_i of a point p_i, the boundary of D_i is determined by at most m circular arcs. So the total complexity of this configuration of dominator regions for all the points in P can be $O(mn) \sim O(n^2)$ under the assumption that $m = n$.

Let $\mathcal{CH}(S)$ denote the convex hull of S. Assume that the origin for the sets S and P lies inside $\mathcal{CH}(S)$. Now, lift each point of S and P to points on a unit paraboloid Ψ, where $\{\Psi = (x, y, z)|z = x^2+y^2\}$ [2]. So, any point $p = (x, y)$ in the plane is lifted to a point $p' = (x, y, x^2 + y^2)$ on Ψ. We will refer to this geometric transformation as *paraboloid* or *lifting transformation*. For any s_k and p_i in the plane, let s'_k and p'_i respectively denote the lifted image on the unit paraboloid Ψ. Now, a paraboloid transformation of a circle $C = \{(x - c_1)^2 + (y - c_2)^2 = r^2\}$ in the two dimensional xy-plane is a curve $C' = \{z-2c_1x-2c_2y+c_1^2+c_2^2-r^2 = 0\}$ on Ψ [10]. The equation of C' depicts a plane in 3-dimensional space. Observe that the disc $C(s_k, p_i)$ is a plane in 3-d; we denote this plane as $C'(s_k, p_i)$. Moreover, each $C'(s_k, p_i)$ $(k = 1, \ldots, m)$ passes through p'_i and is parallel to the tangent plane of the unit paraboloid Ψ at s'_k. Now we have an observation linking the dominator region D_i of p_i with a cone having its apex at p'_i.

Observation 3. *For any point p_i, the dominator region D_i in the 2D plane is mapped to a cone Ω_i in 3-space with its apex at p'_i under paraboloid transformation.*

Proof. From Observation 2, we know that $D_i = \bigcap_{k=1}^{m} C(s_k, p_i)$. Now each $C'(s_k, p_i)$ is a plane passing through p'_i where $C'(s_k, p_i)$ and p'_i are the lifted images of $C(s_k, p_i)$ and p_i respectively by paraboloid transformation. The intersections of the upper half spaces of $\{C'(s_k, p_i)|\ \forall k\}$'s define a cone Ω_i with apex at p'_i in 3-space. This cone in 3-space corresponds to the dominator region in the plane. □

Let p_j dominate p_i. We already know from Observation 2 that $D_j \subset D_i$. Next, we explore the relation between Ω_i and Ω_j where Ω_j is the cone corresponding to the dominator region D_j of p_j.

Lemma 4. *The cone Ω_j corresponding to the dominator region D_j of point p_j contains the cone Ω_i corresponding to the dominator region D_i of point p_i, if p_j dominates p_i.*

Proof. Consider a point s'_k on the unit paraboloid Ψ corresponding to $s_k \in S$. τ_k is the tangent plane of Ψ at s'_k. Translate the plane τ_k to include p'_i and we denote that plane as τ_i^k. The projection of the intersection of τ_i^k and Ψ on

the plane is $C(s_k, p_i)$. As p_j dominates p_i, during translation τ_k will include p'_j before p'_i. This is true for all $\tau_k (k = 1, \ldots, m)$. Since Ω_i is defined by intersection of the upper half spaces of $\{\tau_i^k | \ \forall k = 1, 2, \ldots, m\}$, Ω_j will contain Ω_i. □

From Lemma 4, we can conclude that a cone Ω_i in 3-space corresponding to a dominated point p_i will be contained in at least one cone Ω_j. The following corollary points out that the lower envelope of cones gives the non-dominated or skyline points.

Corollary 5. *The apices corresponding to the lower envelope of the cones* $\{\Omega_1, \ldots, \Omega_n\}$ *are the nondominated points, i.e the skyline points of the point set* P.

Proof. Follows from Observations 2 and 3 and Lemma 4. □

2.2 Relation of Lower Envelope of Cones to Additively Weighted Voronoi Diagrams of a Convex Distance Function

As deduced in Corollary 5, the set of skyline points is nothing but the points of P corresponding to the apices of the lower envelope of the cones Ω_i. Constructing the entire lower envelope of cones is costly however. So, we explore a relation between additively weighted Voronoi diagram under a convex distance function with lower envelope of such cones. To give the details, we need to define convex distance functions and additively-weighted Voronoi diagrams.

Definition 3. *(Convex distance function) Minkowski showed that any convex set* M *whose interior contains the origin defines a convex distance function* $d_M(p, q)$*, where the distance from point* p *to* q *with respect to* M *is the amount that* M *must be scaled to include* $q - p$.

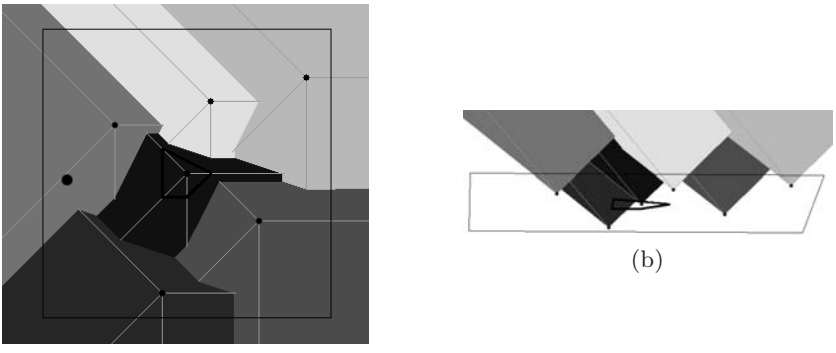
$$d_M(p, q) = \inf\{\lambda \geq 0 : q - p \in \lambda M\}$$

If M is closed, then the infimum operation can be replaced by the minimum operation. A distance function may not be a metric, since d_M is symmetric if and only if M is centrally symmetric. (If we denote the reflection of the set M through the origin by $M^{\ominus}$, then $d_M(p, q) = d_{M^{\ominus}}(q, p)$.) The distance function d_M does always satisfy the triangle inequality for points [3]: $d_M(p, q) + d_M(q, r) \geq d_M(p, r)$. Note that the boundary of M serves as the unit ball for this distance function. For a fixed p, the graph of $d_M(p, q)$ as a function of q is a cone with apex at p.

Definition 4. *(Additively weighted Voronoi diagram) Given a finite set of points* $P = p_1, p_2, \ldots, p_n \subset \mathbb{R}^d$*, with additive weights* $\omega_1, \omega_2, \ldots, \omega_n$*, and any distance function,* $d(p, q)$*, we can define a generalized Voronoi diagram by labeling each site* $q \in \mathbb{R}^d$ *with its set of closest points.*

$$label(q) = \arg\min_{\mathbf{i} \in [1,n]} d(p_i, q) + \omega_i,$$

and partitioning the plane into maximally connected regions having the same labels. Voronoi cells are regions with a single closest neighbor and vertices have degree $d + 1$ *(or more, in degenerate configurations).*



(a) (b)

Fig. 1. A Voronoi diagram of 6 sites in the plane using a convex quadrilateral as distance function; its view as a lower envelope of cones.

Figure 1(a) illustrates a simple example for 6 distinct sites, all having weight zero. The distance function is the black convex quadrilateral around the point at the origin, and each cell is drawn in a different shade. Figure 1(b) shows the cones for which the Voronoi diagram is the lower envelope. When all weights are zero, each distinct site has a non-empty cell. The following observation, which we state without proof, is central to our idea of relating the lower envelope of cones with additively weighted Voronoi diagrams of a convex distance function.

Observation 6. *The cell for a site shrinks if we increase the weight of the site; we essentially translate the cone upwards until the lifted site disappears from the lower envelope.*

2.3 Reduction of Lower Envelope of Cones to Additively Weighted Voronoi Diagrams of a Convex Distance Function

We can do the reduction by simply giving a different interpretation of the inequalities defining dominance. Recall that, as per Definition 1, a point p dominates q $(p, q \in P)$ with respect to the set of sites S if and only if $\forall s \in S$, $d(p, s) < d(q, s)$.

Without loss of generality, choose the origin to be some point inside $\mathcal{CH}(S)$, then assign each point $p \in P$ a weight $\omega_p = p.x^2 + p.y^2$, which we can use as the lifting coordinate. If we square both sides of the dominance inequality, we obtain a linear expression in site coordinates and weights:

$$(p - s) \cdot (p - s) < (q - s) \cdot (q - s) \tag{2}$$

$$\text{if and only if} \qquad \omega_p < \omega_q - 2s \cdot (q - p). \tag{3}$$

Note that for each $s \in S$, the above inequality gives rise to a (hyper)plane passing through $p' = (p.x, p.y, \omega_p)$, which is the lifted version of p. The intersection of the halfspaces defined by the points of S gives a cone with apex p' that contains all points dominated by p (see Lemma 4). We now show that the collection of all

cones for the points P corresponds to an additively weighted Voronoi diagram for a convex distance function defined by S.

We claim that if we intersect this cone with a (hyper)plane $\omega = \omega_p + 1$, then we obtain a convex polytope that defines a distance function containing $(p.x, p.y, \omega_p + 1)$ as its origin. Let M be the projection of this convex polytope onto the (hyper)plane $\omega = 0$. Note that, moving from this origin $(p.x, p.y, \omega_p + 1)$ by any vector $(v.x, v.y, 0)$ must leave the polytope, since any v can be expressed as a convex combination $v = -2 \sum_{1 \le i \le m} \alpha_i s_i$ where reals α_i are not all equal to zero because the sites $S = \{s_1, s_2, \ldots, s_m\}$ contain the origin in their convex hull. Thus, the Voronoi diagram of distance d_M for points $p \in P$ with weights ω_p generates the same lower envelope of cones. The sites that are not dominated with respect to S are those with non-empty cells. From the above discussion, and Lemma 4 and Observation 6, we get the final result as in Theorem 7.

Theorem 7. *The skyline or non-dominated points of set P with respect to sites S are those with non-empty Voronoi cells under a convex distance function determined by S with additive weights determined by P.*

3 Computing Non-dominance in the Plane

Theorem 7 tells us points in P that have non-empty Voronoi cells in an additively weighted Voronoi diagram of P under a convex distance function determined by S correspond to the skyline points. We have also described the method to obtain this convex set M from S at the beginning of Section 2. Note that $|M| = O(|S|)$. So, our problem of finding skyline points is now transformed into computing non-empty Voronoi cells in an additively weighted Voronoi diagram under a convex distance function M. Actually, we want to avoid computing all the edges because, as seen in Figure 1(a), they are fairly complicated, even when all weights are zero. The bisector between two points can consist of $\Theta(|M|)$ line segments, so the generalized Voronoi diagram of points P can then have complexity $\Theta(|P| \cdot |M|)$. We can see this from the cone view. So, we use the *compact piece-wise linear Voronoi diagram* concept of McAllister et al. [8]. A *compact Voronoi diagram* is an approximated version of an *abstract Voronoi diagram*(AVD) [6] defined using a convex distance function. An AVD is defined only in terms of bisectors of pairs of points and are computed using the ordering of the two points along a bisector and the ordering of these bisectors that pass through a common point. So, we would compute an additively weighted *compact Voronoi diagram* under a convex distance function instead of the additively weighted Voronoi diagram under a convex distance function. We will show that we can locate empty (or non-empty) cells in the compact representation also. McAllister et al. [8] showed that one could compute a compact Voronoi diagram of the point set P having $\Theta(|P|)$ complexity in $O(|P|(\log |P| + \log |M|))$ time. In this diagram, the closest neighbor of each query point is not a unique candidate but one of two candidates. To develop the concept of the compact diagram under additive weights, we need the following preliminaries.

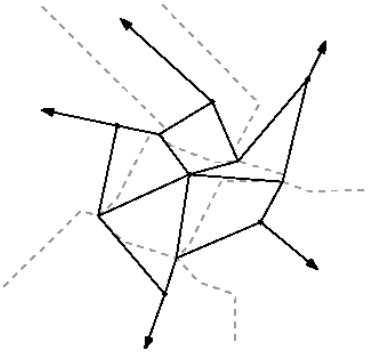


Fig. 2. The spoke diagram (solid lines) depends only on the number of sites, and not on the complexity of the distance function. The dashed lines indicate the Voronoi diagram and the solid lines indicate the compact diagram.

3.1 Geometric Preliminaries

We can give a geometric interpretation of the convex distance function between two points p, $a \in I\!R^2$ using a convex set M. Let M_p^a denote the convex set M scaled by $d_M(p,a)$ and translated to p; i.e. $M_p^a = d_M(p,a)M + p$.

Definition 5. *spoke*(p,a) *is the line segment* $\overline{pa}$ *such that* a *lies on* M_p^a.

Definition 6. *A set* $X \in I\!R^2$ *is star shaped with respect to* a *if* $a \subseteq X$ *and every spoke*(p,a) *with* $p \in X$*, is contained in* X.

Any point k on the bisector between any two sites p_i, $p_j \in P$ in an additively weighted Voronoi diagram satisfies $d_M(p_i,k)+\omega_i = d_M(p_j,k)+\omega_j$ where ω_i and ω_j are the weights of the points p_i and p_j respectively. Now, proceeding along the lines of McAllister et al. [8], we can show that the bisector between any two points is a continuous curve and it separates the plane into two regions - one star shaped with respect to p_i and the other star shaped with respect to p_j. This in turn leads to the fact that a Voronoi cell of p_i is star shaped with respect to p_i. The boundary of the Voronoi cell of p_i consists of portions of bisectors with other points. A *finite Voronoi vertex* is formed by the intersection of two adjacent bisectors at a point that is equidistant from p_i and the other two points defining the bisectors under additive weights. Two adjacent bisectors that may not intersect at a finite point is said to be *Voronoi vertex at infinity*. Again, as in Corollary 2.6 of McAllister et al. [8], we can show that by introducing spokes from the finite and infinite Voronoi vertices around the boundary of the Voronoi cell of p_i in Voronoi diagram of P, the cell of p_i is decomposed into regions bounded by portions of a single p_ip_j-bisector. This follows from the fact that Voronoi cells are star shaped and the spokes lie in the corresponding Voronoi cell. Next, we define a compact Voronoi diagram using $O(|P|)$ line segments (independent of $|M|$), and this Voronoi diagram breaks the plane into *spoke regions* instead of Voronoi regions. In this compact Voronoi diagram, the closest

neighbor of a query point cannot be uniquely determined, but two candidates can be determined out of which one will be the closest. Notice that each spoke region is a quadrilateral whose four corners are two Voronoi vertices and two points. Each quadrilateral lies in the union of the Voronoi cells for the two defining sites. The compact Voronoi diagram is thus defined in terms of the Voronoi vertices and the spoke regions induced by the spokes. It follows from McAllister et al. [8], that the number of Voronoi vertices and spokes would be $O(|P|)$.

Lemma 8. *The compact Voronoi diagram of $|P|$ sites under a convex distance function induced by a convex $|M|$-gon with additive weights has $O(|P|)$ Voronoi vertices and $O(|P|)$ spokes.*

3.2 Algorithm

For the case with all weights zero, McAllister et al. [8] showed that one could compute a compact Voronoi diagram of $\Theta(|P|)$ complexity in $O(|P|(\log|P| + \log|M|))$ time such that the closest neighbor of each point was one of two candidates. The idea is simple if one can locate the vertices efficiently. We need to simply draw the *spokes* to each Voronoi vertex from each defining site. Figure 2 shows such a diagram and implicitly includes a vertex at infinity as well. Notice that the spoke diagram is composed of quadrilaterals whose four corners are two Voronoi vertices and two points; each quadrilateral lies in the union of the Voronoi cells for the two defining sites. We now show that this diagram can also be computed with additive weights.

Two important primitives are needed for our algorithm.

(1) Finding the distance $d_M(p, q)$ given weighted points p and q.
(2) Finding the Voronoi vertex for three weighted sites.

Lemma 9. *The distance $d_M(p, q)$ given weighted points p and q can be found in $O(\log|M|)$ time.*

Proof. A binary search on the vertices of the convex polygon M finds $d_M(p, q)$ and hence it requires $O(\log|M|)$ time. □

Next, we show the method of finding the Voronoi vertex.

Lemma 10. *A Voronoi vertex for three weighted points can be computed in $O(\log|M|)$ time under a convex distance function d_M.*

Proof. Let P, Q and R be the points whose corresponding weights are ω_P, ω_Q and ω_R respectively. For finding the vertex of the weighted voronoi diagram for a set of points, we consider a set of circles whose centers are the set of points and radii are the weights corresponding to the points. Assume that the sites P, Q and R are ordered in clockwise direction and P_c, R_c and Q_c be the respective circles. We want to compute a vertex v such that the smallest homothet of M centered at v contains the circles P_c, Q_c and R_c. We draw the common inner tangents in clockwise direction from P_c to Q_c and Q_c to R_c. If there is an arc of

Q_c lying between these two tangents, the vertex of these sites will be at infinity. Otherwise, there will be a finite vertex. Now we only deal with those portions of M that touch the circles P_c and Q_c when the smallest homothet of M centered at v contain these circles. Typically, this contact point will be an edge e of M. Compute the outer tangents to P_c (respectively Q_c) that are parallel to e's neighboring edges. The clockwise circular arc between the tangential points are the probable portion of P_c that touches the smallest homothet of M containing P_c, Q_c and R_c. Using the tentative prune and search technique of Kirkpatrick and Snoeyink [5], we can compute the fixed point and the voronoi vertex in $O(\log |M|)$ time. □

Our randomized incremental construction of the compact Voronoi diagram follows the same randomized technique as of McAllister et al. [8]. It maintains a conflict history DAG where the nodes of the DAG correspond to the spoke regions. We initialize with the Voronoi diagram of three points. To insert a new point p into the spoke diagram for $k \leq |M|$ points, we locate the quadrilateral shaped spoke region containing p. This corresponds to two points p_i and p_j any one of which may be the closest point. We measure the distances $d_M(p_i, p) + \omega_i$ and $d_M(p_j, p) + \omega_j$ to the two existing point that define the quadrilateral, and find out whether p has a non-empty Voronoi cell. If the cell is non-empty, then the Voronoi cell for new point p will carve out a tree from the existing Voronoi diagram; the tree topology still shows which Voronoi vertices are together in quadrilaterals of the spoke diagram. We can explore this tree using a number of distance and Voronoi vertex computations that is proportional to the number of spokes that are added or deleted.

If we randomly order the points and then perform incremental construction, the algorithm locates each point and constructs an expected $O(|P|)$ spokes. Thus, the total expected time is $O(|P|(\log |P| + \log |M|))$. Coupled with the initial computation of convex hull of S, and the fact that $|P| = n$ and $|S| = m$, we have the final result.

Theorem 11. *The set of non-dominated or skyline points of a set P of n points with respect to a set S of m sites can be found in $O((m+n)\log m + n \log n)$ expected time.*

4 Conclusions

In this paper, we proposed an algorithm for finding the non-dominated points among a point set P with respect to a set of sites S in $\mathbb{R}^2$. This problem was initially proposed by Sharifzadeh and Shahabi [11] and termed as *spatial skyline query* problem. We give some geometric insights into this problem to design an efficient algorithm. It would be worthwhile to extend the algorithm to higher dimensions. We intend to work on the dynamic version of the problem where the set of *skyline points* changes dynamically due to insertion and deletion of sites and data points. Finding non-dominated points under different domination relations between points will also be interesting to investigate.

References

1. Borzsonyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of the 17th International Conference on Data Engineering, Washington, DC, USA, pp. 421–430. IEEE Computer Society, Los Alamitos (2001)
2. Brown, K.Q.: Geometric transforms for fast geometric algorithms. Ph.D. thesis, Dept. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, Report CMU-CS-80-101 (1980)
3. Cassels, J.: An Introduction to the Geometry of Numbers. Springer, Heidelberg (1959)
4. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with presorting. In: Proceedings of the 17th International Conference on Data Engineering, Washington, DC, USA, pp. 717–816. IEEE Computer Society, Los Alamitos (2003)
5. Kirkpatrick, D., Snoeyink, J.: Tentative prune-and-search for computing fixed-points with applications to geometric computation. Fundam. Inform. 22, 353–370 (1995)
6. Klein, R.: Concrete and Abstract Voronoi Diagrams. LNCS, vol. 400. Springer, Heidelberg (1989)
7. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: An online algorithm for skyline queries. In: Proceedings of VLDB, pp. 275–286 (2002)
8. McAllister, M., Kirkpatrick, D., Snoeyink, J.: A compact piecewise-linear Voronoi diagram for convex sites in the plane. Discrete Comput. Geom. 15, 73–105 (1996)
9. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM Transaction on Database System 30(1), 41–82 (2005)
10. Sack, J.-R., Urrutia, J.: Handbook of computational geometry. North-Holland Publishing Co., Amsterdam (2000)
11. Sharifzadeh, M., Shahabi, C.: The spatial skyline queries. In: VLDB 2006: Proceedings of the 32nd international conference on Very large data bases, pp. 751–762. VLDB Endowment (2006)
12. Son, W., Lee, M.-W., Ahn, H.-K., Hwang, S.w.: Spatial skyline queries: An efficient geometric algorithm. CoRR, abs/0903.3072 (2009)
13. Tan, K.-L., Eng, P.-K., Ooi, B.C.: Efficient progressive skyline computation. In: VLDB 2001: Proceedings of the 27th International Conference on Very Large Data Bases, pp. 301–310. Morgan Kaufmann Publishers Inc., San Francisco (2001)

Cutting a Convex Polyhedron Out of a Sphere
(Extended Abstract)

Syed Ishtiaque Ahmed, Masud Hasan, and Md. Ariful Islam

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka-1000, Bangladesh
ishtiaque@csebuet.org, masudhasan@cse.buet.ac.bd, arifulislam@csebuet.org
http://www.buet.ac.bd/cse

Abstract. Given a convex polyhedron P of n vertices inside a sphere Q, we give an $O(n^3)$-time algorithm that cuts P out of Q by using guillotine cuts and has cutting cost $O(\log^2 n)$ times the optimal.

Keywords: Approximation algorithm, guillotine cut, polyhedra cutting.

1 Introduction

The problem of cutting a convex polygon P out of a piece of planar material Q (P is already drawn on Q) with minimum total cutting length is a well studied problem in computational geometry. The problem was first introduced by Overmars and Welzl in 1985 [12] but has been extensively studied in the last eight years [1–4, 7, 8, 10, 12–14] with several variations, such as P and Q are convex or non-convex polygons, Q is a circle, and the cuts are line cuts or ray cuts.

This type of cutting problems have many industrial applications such as in metal sheet cutting, paper cutting, furniture manufacturing, ceramic industries, fabrication, ornaments, and leather industries. Some of their variations also fall under *stock cutting problems* [3].

If Q is another convex polygon with m edges, this problem with line cuts has been approached in various ways [2–5, 8, 9, 12, 13]. If the cuts are allowed only along the edges of P, Overmars and Welzl [12] proposed an $O(n^3 + m)$-time algorithm for this problem with optimal cutting length, where n is the number of edges of P. The problem is more difficult if the cuts are more general, i.e., they are not restricted to touch only the edges of P. In that case Bhadury and Chandrasekaran showed that the problem has optimal solutions that lie in the algebraic extension of the input data field [3] and due to this algebraic nature of this problem, an approximation scheme is the best that one can achieve [3]. They also gave an approximation scheme with pseudo-polynomial running time [3].

After the indication of Bhadury and Chandrasekaran [3] to the hardness of the problem, many have given polynomial time approximation algorithms. Dumitrescu proposed an $O(\log n)$-approximation algorithm with $O(mn + n \log n)$ running time [8, 9]. Then Daescu and Luo [5] gave the first constant factor approximation algorithm with ratio $2.5 + ||Q||/||P||$, where $||P||$ and $||Q||$ are the

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 94–101, 2010.

perimeters of P and the minimum area bounding rectangle of P respectively. Their algorithm has a running time of $O(n^3 + (n+m)\log(n+m))$. The best known constant factor approximation algorithm is due to Tan [13] with an approximation ratio of 7.9 and running time of $O(n^3+m)$. In the same paper [13], the author also proposed an $O(\log n)$-approximation algorithm with improved running time of $O(n+m)$. As the best known result so far, very recently, Bereg, Daescu and Jiang [2] gave a polynomial time approximation scheme (PTAS) for this problem with running time $O(m + \frac{n^6}{\epsilon^{12}})$.

For ray cuts, Demaine, Demaine and Kaplan [7] gave a linear time algorithm to decide whether a given polygon P is *ray-cuttable* or not. For optimally cutting P out of Q by ray cuts, if Q is convex and P is non-convex but ray-cuttable, then Daescu and Luo [5] gave an almost linear time $O(\log^2 n)$-approximation algorithm. If P is convex, then they gave a linear time 18-approximation algorithm. Tan [13] improved the approximation ratio for both cases as $O(\log n)$ and 6, respectively, but with much higher running time of $O(n^3+m)$. See Table 1 for a summary of these results.

Our results. The generalization of this problem in 3D is very little known. To the best of our knowledge, the only result is to decide whether a polyhedral object can be cut out form a larger block using continuous hot wire cuts [10]. In this paper we attempt to generalize the problem in 3D. We consider the problem of cutting a convex polyhedron P which is fixed inside a sphere Q by using only guillotine cuts with minimum total cutting cost. A *guillotine cut*, or simply a *cut*, is a plane that does not pass through P and partitions Q into two smaller convex pieces. After a cut is applied, Q is updated to the piece that contains P. The *cutting cost* of a guillotine cut is the area of the newly created face of Q. We give an $O(n^3)$-time algorithm that cuts P out of Q by using only guillotine cuts and has cutting cost no more than $O(\log^2 n)$ times the optimal cutting cost. Also see Table 1.

Table 1. Comparison of the results

Dim.	Cut Type	Q	P	Approx. Ratio	Running Time	Reference
2D	Ray	-	Non-convex	Ray-cuttable?	$O(n)$	[7]
		Convex	Convex	18	$O(n)$	[5]
		Convex	Non-convex	$O(\log^2 n)$	$O(n)$	[5]
		Convex	Convex	6	$O(n^3+m)$	[13]
		Convex	Non-convex	$O(\log n)$	$O(n^3+m)$	[13]
	Line	Convex	Convex	$O(\log n)$	$O(mn + n\log n)$	[8, 9]
		Convex	Convex	$2.5 + \|\|Q\|\|/\|\|P\|\|$	$O(n^3 + (n+m)\log(n+m))$	[5]
		Convex	Convex	7.9	$O(n^3+m)$	[13]
		Convex	Convex	$(1+\epsilon)$	$O(m + \frac{n^6}{\epsilon^{12}})$	[2]
		Circle	Cornered con.	$O(\log n)$	$O(n)$	[1]
		Circle	Cornered con.	6.48	$O(n^3)$	[1]
3D	Hot-wire	-	Non-convex	Cuttable?	$O(n^5)$	[10]
	Guillotine	**Sphere**	**Convex**	$\boldsymbol{O(\log^2 n)}$	$\boldsymbol{O(n^3)}$	**This paper**

2 The Algorithm

The overall idea is as follows. Let C^* be the optimal cutting cost. We shall have two phases in our algorithm: *box cutting phase* and *carving phase*. In the box cutting phase, we shall cut a minimum volume rectangular box B containing P out of Q with cutting cost no more than a constant factor of C^*. Then in the carving phase we shall cut P out of B with cutting cost bounded by $O(\log^2 n)$ times of C^*.

A cut is *vertex/edge/face cut* if it is tangent to P at a single vertex/a single edge/a face respectively. We call P to be *cornered* if it does not contain the center o of Q, otherwise it is called *centered*. For cornered P, the *D-separation* of P is the minimum-cost (single) cut that separates P from o. A point p of P is *visible* from o if the line segment $\overline{op}$ does not intersect any other point of P.

2.1 Box Cutting Phase

If P is cornered, we first apply a D-separation to Q.

Lemma 1. *The D-separation must be either a vertex, an edge or a face cut. Moreover, if $\overline{oo'}$ is the line segment perpendicular to the D-separation at o', then o' must be the corresponding vertex or a point of the corresponding edge or face.*

Proof. Let x be the closest point of P from o. Clearly, x is visible from o. A D-separation must be the plane that can separate o from x and is furthest from o. This plane is none but the plane perpendicular to $\overline{ox}$ at x. This plane is also tangent to P, since otherwise x would not be closest to o. □

Observe that since P is convex, the D-separation is unique.

Lemma 2. *The D-separation can be found in $O(n)$ time.*

Proof. By Lemma 1 we need to find the closest point x. To check whether x is a vertex of P, for each vertex v we draw a plane π_v perpendicular to $\overline{ov}$ at v. If π_v is tangent to P, then π_v is the D-separation. Checking π_v to be a tangent of P can be done in $O(d_v)$, where d_v is the degree of v. Over all v, it is $O(n)$.

To check whether x is a point of an edge e (similarly, a face f) of P, for each edge e (face f) we draw the line segment $\overline{oo'}$ perpendicular to the line l_e passing through e (to the supporting plane π_f of f). If o' is a point of l_e (π_f), then x is o'. □

For cornered P, after the D-separation is applied, Q is a spherical segment and let r be the radius of its base circle.

Lemma 3. *For cornered P, cost of the D-separation, which is πr^2, is at most C^*.*

Proof. [Sketch only] The proof depends upon the fact that the cuts in an optimal cutting sequence must be tangents to P. Overmars and Welzl [12] proved this fact for 2D, whose 3D generalization also holds. The idea is that if c is the first cut that does not touch P, then the cost of c and the subsequent cuts behaves, while moving c parallelly, as a concave function in the distance of c from P. Therefore, the minimum cost is achieved when it touches P or is infinitely away from P. With the above fact, the authors in [1] proved in 2D that to separate P from o an optimal cutting sequence must use the D-separation. The 3D generalization of this fact also holds. The main idea is that, to separate o from P if a single cut is used that is not a D-separation, then it must have cost more than the D-separation, since D-separation is the minimum such cut. If more than one cut are used, then their total cost would be even higher. □

A similar lemma for centered P is the following.

Lemma 4. *For centered P, $C^* \geq \pi R^2$, where R is the radius of Q.*

Proof. [Sketch only] Since P contains the center o of Q, any cutting sequence, starting from the boundary of Q, must wrap P and finally get out of Q by different location in the boundary of Q. That means the wrapping must enclose the center o. In the best case when P is the center o, the sequence must traverse at least $\frac{1}{2}\pi R^2$ area to reach P and need to traverse another $\frac{1}{2}\pi R^2$ area to finish the cutting. In the worst case when P is almost the sphere Q, the sequence must traverse the whole area of Q, which is $4\pi R^2$. □

We next find a minimum volume rectangular bounding box B of P in $O(n^3)$ time by the algorithm of O'Rourke [11]. Then we cut out this box from Q by applying six cuts along the six faces of B.

Lemma 5. *Cost of cutting B out of Q is at most $3C^*$ for cornered P and at most $4C^*$ for centered P.*

Proof. Let S be the surface of Q. For cornered P, area $|S| \leq 3\pi r^2 \leq 3C^*$ (by Lemma 3) and for centered P, $|S| = 4\pi R^2 \leq 4C^*$ (by Lemma 4). While cutting along the faces of B, for each cut c let Q' be the portion of Q that does not contain P. Let q' be the portion of the surface of Q' that is "inherited" from S, i.e., that was a part of the surface of S. One important observation is that the cost of c is no more than the area of q'. Moreover, over all six cuts, sum of these inherited surface area is $|S|$. Therefore, the lemma holds. □

Once the minimum area bounding box B has been cut, the lower bound on cutting cost can be given in terms of the area of B.

Lemma 6. *$C^* \geq \frac{1}{6}|B|$, where $|B|$ is the area of B.*

Proof. Let h be a maximum area face of B. Project P orthogonally from the direction perpendicular to h. P projects to a convex polygon X. In this projection, h is the minimum area bounding rectangle of X, since otherwise we could rotate

the four faces of B that are not perpendicular to h and would get a bounding rectangle smaller than h, which in turn would give a bounding box smaller than B, but that is a contradiction that B is the smallest bounding box. It implies that the area of X is at least $\frac{1}{2}|h|$. Now, C^* is at least twice the area of X, and $|B| \leq 6|h|$. Therefore, $C^* \geq 2|X| \geq 2 \cdot \frac{1}{2}|h| \geq \frac{1}{6}|B|$. □

2.2 Carving Phase

Let $T = B - P$ be the portion of B that is "trapped" between the boundaries of P and B. T is a polyhedral object, convex or non-convex and possibly disconnected. The *inner* (*outer*) surface of T is the surface that touches (does not touch) the faces of P. Our idea is to apply an edge cut through each edge of P, and we shall do that in two types of rounds: *face rounds* and *edge rounds*. Face rounds will find polygonal chains that will partition the faces of P into smaller connected components and edge rounds will apply edge cuts through the edges of those polygonal chains. There will be $O(\log n)$ face rounds. Within each face round there will be a number of edge rounds but their total cost will be $O(C^* \log n)$. Once we have applied edge cuts through all the edges of P, each face f of P will have a small "cap"-like portion of T over it, which we shall cut at a cost of the area of f to get P, giving a cost of $O(C^*)$ for all faces.

Face Rounds. Let F be a *connected face set* of l faces of P. At the very first face round $i = 0$, F consists of all the faces of P. We find a chain of edges P' that will partition F into two smaller connected face sets F_1 and F_2 by the following lemma.

Lemma 7. *It is always possible to find in $O(l \log l)$ time an orthogonal projection of P which is non-degenerate w.r.t the faces of F such that the sets of visible and invisible faces of F contain at least $\lfloor \frac{l}{2} \rfloor$ faces each.*

Proof. For this proof we shall move on to the surface of an origin-centered sphere s. For each face $f \in F$, its outward normal is uniquely represented by a point of s, which we call the *normal point* of f. Each point of s also represents an orthogonal projection direction of P. So, an orthogonal projection of P which is non-degenerate w.r.t the faces of F is represented by a great circle of s that does not pass through the normal points of the faces of F. We need one such great circle satisfying an additional criterion that its two hemispheres contain at least $\lfloor \frac{l}{2} \rfloor$ normal points each. There exists infinitely many such great circles and one of them can be found in $O(l \log l)$ time as follows. Take any two antipodal points that are normal points as poles. Take a great circle g through these two poles and rotate it around these poles until the number of normal points in its two hemisphere differ by at most one. If it happens that some normal points fall on the great circle, then slightly change the poles and the great circle to distribute those normal points into two hemispheres as necessary. For running time, all we need to do is to sort the normal points according to their angular distance with the plane of g at the origin. □

We call the projection direction to achieve g by the above lemma the *zone direction* of of F. P' is the chain of edges in the boundary of the above projection whose each edge has both adjacent faces (one is visible and another is invisible) in F. We call P' a *separating chain* of F. We shall apply edge cuts through the edges of P' by the edge rounds as described in the next paragraph. In the next face round $i+1$, we shall apply Lemma 7 for each of F_1 and F_2 and shall thus get two separating chains and four connected face sets. We shall repeat the same procedure for each of these four face sets. We shall continue like this until each face set has only one face. Clearly, we need $O(\log n)$ face rounds.

Edge Rounds. Let $P' = e_1, e_2, \ldots, e_k$ with its two ends from e_1 and e_k touching the outer surface of T. We shall apply edge cuts through the edges of P' such that all of them are parallel to a particular direction. Such a direction can be the corresponding projection direction. We call this set of k edge cuts a *zone* of cuts and its direction the *zone cut direction*. We shall apply these cuts in $\log k$ edge rounds. At the very first edge round $j = 0$, we apply an edge cut through $e_{k/2}$ in the zone cut direction. This cut will partition the edges of P' into two subchains of size at most $\lfloor \frac{k}{2} \rfloor$. In the next round, we apply two edge cuts through the two middle edges of these two subchains, which will result into four subchains. Then in the next round we apply four similar cuts to the four subchains. We continue like this until each subchain has only one edge. Clearly, we need $O(\log k)$ edge rounds for P'.

Lemma 8. *After all the face rounds and the corresponding edge rounds are completed, all edges of P get an edge cut.*

Proof. Let e be an edge that does not get an edge cut. Then the two adjacent faces of e are in the same face set. But that is a contradiction that each face set has only one face. □

Analysis. We define the *box area* of a face set F as follows. When F contains all faces of P, its box area is B —the whole surface area of B. Zone of cuts through the separating chain of F partitions F into F_1 and F_2 and T into two components, say T_1 and T_2, respectively. Then the *box area* of F_1 (F_2) is the outer surface area of T_1 (T_2), which we denote by by B_1 (B_2). Observe that $|B_1| + |B_2| \leq |B|$. Box area of any subsequent face set is similarly defined. Moreover, two face sets from the same face round have their box areas disjoint and in any face round sum of all box area is at most $|B|$.

Lemma 9. *Let P'_m be the separating chain with k edges of an arbitrary face set F_m to which we apply $O(\log k)$ edge rounds. Let B_m be the box area of F_m. At each edge round j, total cost of 2^j cuts is $O(|B_m|)$. Over all $\log k$ edge rounds, total cost is $O(|B_m| \log n)$.*

Proof. This proof is similar to that of Lemma 5. Consider a particular edge round j. For each cut c the cost of c is no more than the portion of B_m that is

thrown away by c. Moreover, these cuts are pairwise disjoint. Indeed, they can at best intersect the cut which is in between them and was applied in $(j-1)$-th round. It implies that the total cost of 2^j cuts is at most $|B_m|$. Since $k \leq n$, the second part of the lemma follows. □

Lemma 10. *Let F be the face set consisting of all faces of P to which we shall apply $O(\log n)$ face rounds. At each face round i, total cost of 2^i zones of cuts is $O(|B| \log n)$. Over all $O(\log n)$ face rounds, the total cost is $O(C^* \log^2 n)$.*

Proof. At each face round i, we apply 2^i zones of cuts to 2^i face sets. By the previous lemma, for a particular face set F_m, $0 \leq m \leq 2^i$, cost of the zone of cuts applied to it is at most $O(|B_m| \log n)$. Since $\sum_1^{2^i} |B_m| \leq |B|$, cost of all zone cuts is $\sum_1^{2^i} O(|B_m| \log n) = O(|B| \log n)$. Over all $O(\log n)$ face rounds, the total cost is $O(|B| \log^2 n)$, which by Lemma 6 is $O(C^* \log^2 n)$. □

Running time in face round i involves finding 2^i separating chains, each of size $\frac{n}{2^i}$, plus applying a zone of cuts to each of them. Each separating chain can be found in $O(\frac{n}{2^i} \log \frac{n}{2^i})$ time by Lemma 7. Each cut needs to update Q, which "can be done" in $O(n)$ time assuming that Q is represented by suitable data structure [6]. It gives that a zone of cuts needs $O(\frac{n^2}{2^i})$ time. So, in round i total time is $O(2^i(\frac{n^2}{2^i} + \frac{n}{2^i} \log \frac{n}{2^i})) = O(n^2)$. Over all $O(\log n)$ rounds, it becomes $O(n^2 \log n)$.

Theorem 1. *Given a convex polyhedron P fixed inside a sphere Q, P can be cut out of Q by using only guillotine cuts in $O(n^3)$ time with cutting cost $O(\log^2 n)$ times the optimal, where n is the number of vertices of P.*

3 Conclusion

In this paper, we have given an $O(n^3)$-time algorithm that cuts a convex polyhedron P with n vertices from a sphere Q, where P is fixed inside Q, by using guillotine cuts with cutting cost $O(\log^2 n)$ times the optimal.

This problem is well studied in 2D, where the series of results include several $O(\log n)$ and constant factor approximation algorithms and a PTAS. The key ingredients of the 2D algorithms involve three major steps: (1) take some approximate vertex cuts through the vertices of P, (2) use dynamic programming to find an optimal cutting sequence among the edge cuts and the vertex cuts taken in step (1), and (3) show that the cutting cost of the sequence obtained in step (2) is within the desired factor of the optimal. Using the idea of 2D algorithms may be a way to improve the approximation ratio of our algorithm. Among the above three steps, it may not be difficult to generalize steps (1) and (3) for 3D, but the most difficult part we find is the applying a dynamic programming.

References

1. Ahmed, S.I., Hasan, M., Islam, M.A.: Cutting a cornered convex polygon out of a circle. Journal of Computers (to appear), http://203.208.166.84/masudhasan/cut.pdf
2. Bereg, S., Daescu, O., Jiang, M.: A PTAS for cutting out polygons with lines. In: Chen, D.Z., Lee, D.T. (eds.) COCOON 2006. LNCS, vol. 4112, pp. 176–185. Springer, Heidelberg (2006)
3. Bhadury, J., Chandrasekaran, R.: Stock cutting to minimize cutting length. Euro. J. Oper. Res. 88, 69–87 (1996)
4. Chandrasekaran, R., Daescu, O., Luo, J.: Cutting out polygons. In: CCCG 2005, pp. 183–186 (2005)
5. Daescu, O., Luo, J.: Cutting out polygons with lines and rays. International Journal of Computational Geometry and Applications 16, 227–248 (2006)
6. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications, 2nd edn. Springer, Heidelberg (2000)
7. Demaine, E.D., Demaine, M.L., Kaplan, C.S.: Polygons cuttable by a circular saw. Computational Geometry: Theory and Algorithms 20, 69–84 (2001)
8. Dumitrescu, A.: An approximation algorithm for cutting out convex polygons. Computational Geometry: Theory and Algorithms 29, 223–231 (2004)
9. Dumitrescu, A.: The cost of cutting out convex n-gons. Discrete Applied Mathematics 143, 353–358 (2004)
10. Jaromczyk, J.W., Kowaluk, M.: Sets of lines and cutting out polyhedral objects. Computational Geometry: Theory and Algorithms 25, 67–95 (2003)
11. O'Rourke, J.: Finding minimal enclosing boxes. International Journal of Computer and Information Sciences 14(3), 183–199 (1985)
12. Overmars, M.H., Welzl, E.: The complexity of cutting paper. In: SoCG 1985, pp. 316–321 (1985)
13. Tan, X.: Approximation algorithms for cutting out polygons with lines and rays. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 534–543. Springer, Heidelberg (2005)
14. Toussaint, G.: Solving geometric problems with the rotating calipers. In: MELECON 1983, Athens, Greece (1983)

A Simple Algorithm for Approximate Partial Point Set Pattern Matching under Rigid Motion

Arijit Bishnu, Sandip Das, Subhas C. Nandy, and Bhargab B. Bhattacharya

Advanced Computing and Microelectronics Unit, Indian Statistical Institute,
203, B.T. Road, Kolkata, India - 700108

Abstract. This paper deals with the problem of *approximate point set pattern matching* in 2D. Given a set P of n points, called *sample set*, and a *query set* Q of k points ($k \leq n$), the problem is to find a match of Q with a subset of P under rigid motion (rotation and/or translation) transformation such that each point in Q lies in the ϵ-neighborhood of a point in P. The ϵ-neighborhood region of a point $p_i \in P$ is an axis-parallel square having each side of length ϵ and p_i at its centroid. We assume that the point set is well-seperated in the sense that for a given $\epsilon > 0$, each pair of points $p, p' \in P$ satisfy at least one of the following two conditions (i) $|x(p) - x(p')| \geq \epsilon$, and (ii) $|y(p) - y(p')| \geq 3\epsilon$, and we propose an algorithm for the approximate matching that can find a match (if it exists) under rigid motion in $O(n^2k^2(k\log k + \log n))$ time. If only translation is considered then the existence of a match can be tested in $O(nk^2 \log n)$ time. The salient feature of our algorithm for the rigid motion and translation is that it avoids the use of intersection of high degree curves.

1 Introduction

A fundamental problem in pattern matching is to design efficient algorithms for testing how closely a *query set* Q of k points resembles a *sample set* P of n points, where $k \leq n$ [11, 17]. In computer vision, remote sensing, finger print and related applications, point sets represent some spatial features like spots, corners, lines, curves in the images [16, 17]. In many problems of pattern recognition, such as registration and identification of an object, a suitably chosen set of points may efficiently preserve desired attributes of the object. In all such cases, the problem can be transformed to matching point sets with templates.

It is well-known that in $\mathbb{R}^d$, the fastest known algorithm for exact matching of a query set of k points in a sample set of n points runs in $O(kn^d)$ time [18], and the problem is NP-complete when d is unbounded [1]. Recently, it is shown that the problem is $W[1]$-hard [6], and no fixed parameter tractable algorithm for getting the optimum solution can be designed.

We focus our attention on points in $\mathbb{R}^2$. The problem has several variants [3] based on (i) class of allowable transformations; (ii) exact or approximate matching; (iii) equal cardinality, subset, or largest common point set matching; (iv) one-to-one or one-to-many matching. The most simple kind of transformation is translation. If rotation is allowed along with translation, it is called rigid motion

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 102–112, 2010.

or congruence. Other transformations include reflection and scaling. The latter refers to magnifying (or reducing) the object by a certain factor π. Combination of rotation, translation and scaling is called similarity. Under these transformations, the problem can be classified into three groups [3]: (a) *exact matching*, (b) *partial matching* and (c) *approximate matching*. The exact (partial) pattern matching problem with $k = n$ (with $k \leq n$) is to decide if the patterns exactly (partially) match under some transformations. The subset or partial matching is a difficult problem compared to the equal cardinality matching as there can be $\binom{n}{k}$ possibilities and also no condition can be fixed about the centroid of the points. The problems of finding the exact and partial matches have received wide attention [1–3, 5, 18]. The problem of approximate matching of point sets in the one-to-one case, in which we are interested in this paper, in its most general setting was solved in the seminal paper by Alt et al. [2] and in some restricted setting by Arkin et al. [4]. The algorithms in [2, 4] involve intersection of curves of high degree leading to numerical instability in computation and also the time complexities of these methods are high. These reasons have given rise to a considerable body of work dealing with different types of matchings and mostly fast approximation algorithms [7–9, 11–15, 17].

1.1 Background and Motivation

The decision version of the approximate matching of point sets in the one-to-one case is to test, given two point sets P and Q ($|P| = |Q|$), if there is a bijection $\ell : Q \to P$ and a congruence T, such that $T(q) \in U_\epsilon(\ell(q))$, for all $q \in Q$, where $U_\epsilon(p)$ denotes the closed ϵ-neighborhood of a point $p \in P$. Both the papers [2, 4] address the problem for equal cardinality, i.e., $k = n$. The solutions are elegant and use some geometric facts and bipartite graph matching. For a nice summary of the work in [2], see [3, 12]. But, the algorithms are difficult to implement and numerically unstable as they involve computing the intersection of complex algebraic curves [3, 11, 14]. For finding the said match, the algorithms in [2, 4] try to find some alignments. They do this with the help of a geometric argument that in effect is also an *if and only if condition* for the match. The geometric fact used is as follows. If there exists a valid matching of Q with P, then there is one matching where two points $q_i, q_j \in Q$ are matched exactly to the boundaries of the ϵ-neighborhoods $U_\epsilon(p_\alpha), U_\epsilon(p_\beta)$ of two points $p_\alpha, p_\beta \in P$. Now, the anchoring of $q_i, q_j \in Q$ with $p_\alpha, p_\beta \in P$ implies that $q_i, q_j \in Q$ are constrained to belong to the ϵ-neighborhoods $U_\epsilon(p_\alpha)$, $U_\epsilon(p_\beta)$. With this constrained movement of q_i and q_j, a third point $q_\ell \in Q \setminus \{q_i, q_j\}$ traces out specifically a 6 degree algebraic curve. Now, intersection of this curve is found with the ϵ-neighborhood of the other points in P. Finding this intersection gives rise to the numeric instability in computation.

The algorithm of Alt et al. [2] is indeed a general algorithm in that it can work for overlapping and non-overlapping ϵ-circles and ϵ-boxes both taking $O(n^8)$ time. For ϵ-boxes, the algebraic curve will be of degree four. A more interesting observation was made later by Heffernan and Schirra [12] in which they show that this $O(n^8)$ algorithm is indeed optimal for ϵ-circles. We show that if instead

of overlapping ϵ-circles, non-overlapping ϵ-boxes are used, the time complexity is reduced considerably.

1.2 Our Work

The problem that we are concerned in this paper is whether we can avoid finding the intersection of higher degree algebraic curves and still attain a comparable time complexity for the approximate point set pattern matching problem as in [2, 4]. We assume that, the ϵ-neighborhood region of each point $p \in P$ is an axis-parallel square having each side of length ϵ (called ϵ-*box*) with point p at its centroid. We show that if the point set is well-separated, i.e., each pair of points $p, p' \in P$ satisfy either $|x(p) - x(p')| \geq \epsilon$ or $|y(p) - y(p')| \geq 3\epsilon$ or both, then a simple algorithm for the approximate matching under rigid motion (i.e., with translation and rotation) can be designed that runs in $O(n^2k^2(k\log k + \log n))$ time in worst case. Here the ϵ-boxes are non-overlapping (see Figure 1), and using an anchoring scheme of a point in Q with a point in P, we can avoid the intersection of higher degree algebraic curves. If instead of rigid motion, only translation is permitted for a match, then the existence of a match can be tested in $O(nk^2 \log n)$ time.

It needs to be mentioned that, in [4] the equal cardinality case with non-overlapping ϵ-boxes was solved in $O(n^4 \log n)$ time using intersection of higher degree algebraic curves. Our work is more general in the sense that it works for any k ($\leq n$), but its limitation is that it works for the well-separated point set as defined earlier.

2 Preliminaries

The equal cardinality approximate matching in [2] rests on the use of a simple geometric fact that if there exists a valid matching of Q with a k-subset of P, then there is one where two points $q_i, q_j \in Q$ are matched exactly to the boundaries of the ϵ-neighborhoods $U_\epsilon(p_\alpha)$, $U_\epsilon(p_\beta)$ of two points $p_\alpha, p_\beta \in P$. This fact allows for finding an alignment which may lead to a match, if it exists. In line with the above, we have the following simple lemma.

Lemma 1. *If there exists a transformation $T(Q)$ for the said match, then there exists another transformation $T'(Q)$, such that one point of Q lies on the left boundary of the ϵ-box of a point in P, and one point of Q lies on the top boundary of the ϵ-box of a point in P.*

Proof. Take the transformation $T(Q)$; obtain $T'(Q)$ by pushing all the points towards left (by the same amount) until a point hits the left boundary of an ϵ-box (say for a point p_α), and then push all the points above until a point hits the top boundary of an ϵ-box (say for a point p_β). Note that p_α and p_β may or may not be distinct. □

Note that, Lemma 1 is a necessary and sufficient condition for the existence of a match. It needs to be mentioned that there exists a degenerate case where both

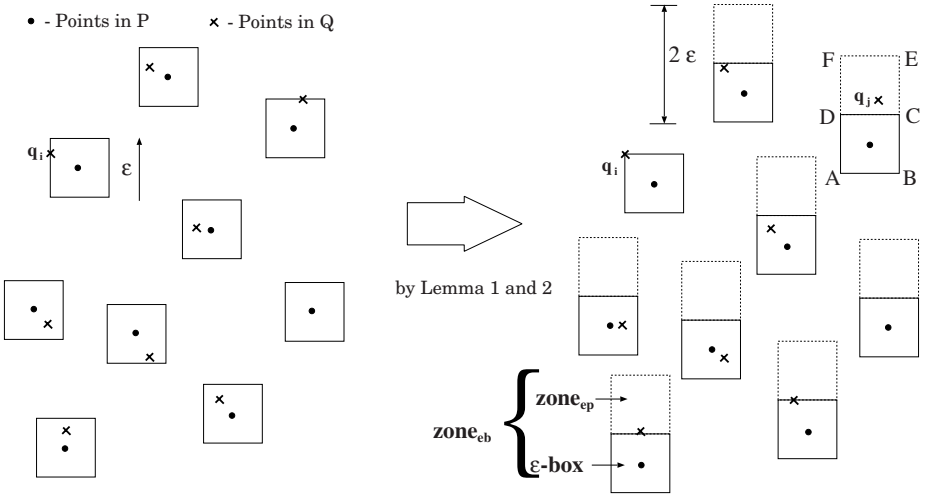


Fig. 1. Pushing scheme of Lemma 1 and Lemma 2

the points appearing on the left and top boundaries may be the same. Here, a point in Q lies on the top-left corner of the ϵ-box of a point in P, and each of the other points in Q lies properly inside some other ϵ-box. For this special case finding a match is easier and hence, we are disregarding this special case.

Let Q be the query set satisfying Lemma 1. If we push the points such that the point lying on the left-boundary of an ϵ-box reaches the top-left corner of that box, the one which lies on the top-boundary of some other ϵ-box, may go outside of that box. Many other points may also go out of its corresponding ϵ-box due to this upward shift. But, using the concept of *extended box* (as stated below), the next lemma must be true for the existence of a match.

Definition 1. *Consider the ϵ-box $ABCD$ around a point p. The* extended ϵ-box *of p is a $\epsilon \times 2\epsilon$ box which is formed by attaching another $\epsilon \times \epsilon$ square $CDFE$ above the ϵ-box $ABCD$ as shown in Figure 1. The portion $CDFE$ is called the extended portion of the ϵ-box.*

Lemma 2. *If there exists a transformation $T(Q)$ for the said match, there exists another transformation $T'(Q)$, such that (i) one point, say $q \in Q$, lies at the top-left corner of the ϵ-box of a point in P, (ii) at least one point lies in the extended portion of the ϵ-box (a region like $CDFE$ shown in Figure 1) of a point in P, (iii) each of the remaining members in Q lie in the extended ϵ-box (a region like $AFEB$ shown in Figure 1) of some point in P.*

It needs to be mentioned that, Lemma 2 is only a necessary condition for the existence of a match, but is not a sufficient condition. The key idea behind Lemma 2 is based on a relative motion between the ϵ-boxes and the points from

the query set. Here, instead of allowing the two points lying on the vertical and horizontal boundary to move on their respective boundaries and making a third point trace an algebraic curve, we move the ϵ-boxes to form the extended ϵ-boxes. That allows us to anchor points from the query set with specified points from the ϵ-box. We first consider the allowed transformation of Q to be only translation. Next, we extend it for searching a match under rigid motion of Q.

3 Translation in 2D

Here, a point $q \in Q$ is anchored with a point $p \in P$. This anchoring fixes the unknown translational parameters of the transformation. The anchoring is done as follows: *position q at the top-left corner of the ϵ-box of p, and check whether each point in $Q \setminus \{q\}$ lies inside the extended box of some point of P.* If the aforesaid checking returns *false*, then no match exists with q at the top-left corner of the ϵ-box of p. But, if it returns *true*, then a match may or may not exist. Here, two cases may arise: (i) each member in $Q \setminus \{q\}$ lies inside the ϵ-box of some point of P, and (ii) the points in $Q \setminus \{q\}$ can be partitioned into two subsets Q_1 and Q_2, where each member in Q_1 lies in the ϵ-box of some point in P, and each member in Q_2 lies in the extended portion of ϵ-box of some member in P. In case (i), a matching is already established. For case (ii), we need to perform the following procedure to compute the necessary translation (if any) for the existence of a match.

For each point in Q_2, compute the amount of downward shift that is at least required to pull down it inside the corresponding ϵ-box, and observe the maximum among them (say Δ_1). Similarly for each point in Q_1, compute the allowable downward shift to ensure that the point is in its corresponding ϵ-box, and compute the minimum among them (say Δ_2). If $\Delta_1 < \Delta_2$, a match exists.

We maintain a planar straight line graph (PSLG) data structure with the ϵ-boxes corresponding to the members in P. While anchoring a point q with a point $p \in P$, one needs to perform k point location queries in the PSLG. This needs $O(k \log n)$ time. The computation of Δ_1 and Δ_2 needs another $O(k)$ time. As we may need to consider anchoring of each point in Q with each point in P (see Lemma 1) in the worst case, the overall time complexity result is as follows:

Theorem 1. *The worst case time complexity of the approximate matching of Q with a k-subset of P in 2D when only translation is considered is $O(nk^2 \log n)$.*

It needs to be noted that because of Lemma 1, the *translation only* case can be solved with only the point location queries, and no intersection of high degree curves is needed.

4 Rigid Motion in 2D

The rigid motion transformation $T(Q)$ under which we are to check is $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$, where t_x, t_y and θ are unknown. By vertically pushing Q to the corner by an amount d ($< \epsilon$), we have altered $T(Q)$ to $T'(Q)$ as $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} t_x \\ t_y - d \end{bmatrix} + \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$.

For a given query set Q, we first try to find out $T'(Q)$ satisfying the condition stated in Lemma 2. Next, we use $T'(Q)$ to find out $T(Q)$ for a match as stated in Lemma 1, if such a match at all exists. We anchor a query point $q_i \in Q$ at the corner of the ϵ-box corresponding to a point $p \in P$ and search for a transformation $T'(Q)$ (as stated in Lemma 2) by applying rotation with an appropriate angle. If such an angle of rotation exists, we can find another point $q_j \in Q$ such that the circle $\mathcal{C}_{ij}$ centered at q_i and with radius $\overline{q_i q_j}$ intersects the *extended portion of the ϵ-box* corresponding to some member in P. Note that, more than one ϵ-boxes may exist, which are cut by the aforesaid circle $\mathcal{C}_{ij}$. For each such intersection, we need to inspect for a match. Consider one such event where C_{ij} intersects with the extended box corresponding to p' (in Figure 2 see the arc GH in the extended box $DCEF$). Let the lines $\overline{q_i G}$ and $\overline{q_i H}$ make angles θ_1 and θ_2 with the x-axis, respectively. Thus, the arc GH indicates an interval $\mathcal{I} = [\theta_1, \theta_2] \in [0, 2\pi]$ of the angle of rotation of Q centering q_i at the top-left corner of the ϵ-box of p. We find a sub-interval of angle $\mathcal{I}^* = [\theta_1^*, \theta_2^*] \subseteq \mathcal{I}$ such that Lemma 2 is satisfied for a rotation of the point set Q by an amount $\theta \in \mathcal{I}^*$. In other words, each point in $Q \setminus \{q_i, q_j\}$ lies in the extended ϵ-box of some point

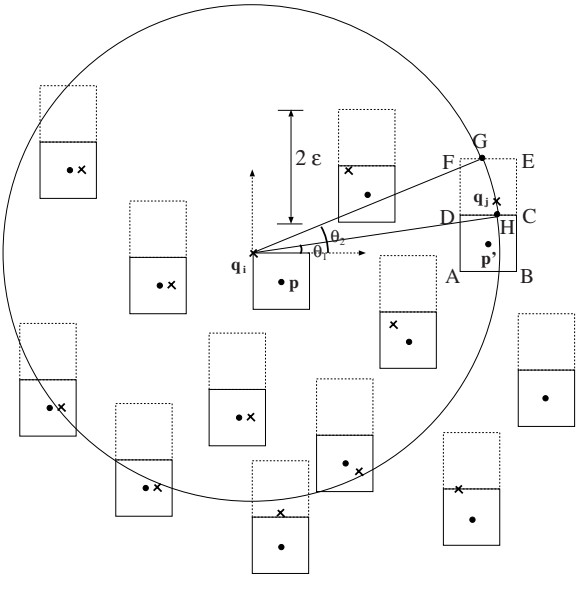


Fig. 2. Finding a suitable rotation for a match

of P. Next, we need to check whether a shift of Q ($\leq \epsilon$) vertically downward exists so that Lemma 1 is also satisfied for the rotation angle θ. In other words, q_i remains on the left boundary of the ϵ-box of p, q_j reaches the top-boundary of the ϵ-box of p', and each member in $Q \setminus \{q_i, q_j\}$ enters in the ϵ-box of some point in P.

In order to determine $\mathcal{I}^*$, we rotate each point $q_k \in Q \setminus \{q_i, q_j\}$ around q_i, and observe the extended ϵ-squares which the circular arc intersects while the angle of rotation θ is in the interval $\mathcal{I}$ ($= [\theta_1, \theta_2]$). Note that, each such q_k may intersect $O(n)$ ϵ-boxes in the worst case. Thus, finding $\mathcal{I}^*$ may require very high running time complexity in the worst case. Again, any member in $Q \setminus \{q_i\}$ may appear on the top-boundary of any ϵ-box in a successful matching. In order to reduce the time complexity of determining the feasible angles of rotation of Q, we rotate Q around q_i, and globally observe the movement of each point in Q.

Consider the $k-1$ concentric circles $\mathcal{C}_{ij}$ for all $q_j \in Q \setminus \{q_i\}$. Each circle $\mathcal{C}_{ij}$ intersects some extended ϵ-boxes. Since the point set P is well-separated, the extended boxes are non-overlapping, and these intersections will contribute a set of non-overlapping arcs. The arcs corresponding to all the points in $Q \setminus \{q_i\}$ will define a circular arc graph G [10]. A clique of size $k-1$ in G implies that the corresponding arcs are on $k-1$ different circles. This, in turn says that each point in $Q \setminus \{q_i\}$ can be placed inside an extended ϵ-box. Thus, for each clique of size $k-1$ we need to test for a matching as stated below. As the number of nodes in G is $O(nk)$ in the worst case, the number of such cliques may also be $O(nk)$.

Let χ be a $k-1$ clique, which corresponds to the rotation angle $\mathcal{I}^* = [\theta_1^*, \theta_2^*]$. We partition the arcs (corresponding to the points in $Q \setminus \{q_i\}$) into two subsets Q_1 and Q_2 such that the arcs in Q_1 are all inside the ϵ-boxes, and those in Q_2 are all inside the extended portion of the ϵ-boxes. If an arc overlaps both the ϵ-box and the *extended portion of the ϵ-box*, then we break it up into two parts, one corresponds to the ϵ-box and the other one corresponds to the *extended portion of the ϵ-box*. If $Q_2 = \phi$, we have already obtained a match with q_i at the top-left position of the ϵ-box of p. If $Q_1 = \phi$, then for any arbitrary rotation angle $\theta \in \mathcal{I}^*$, a match surely exists under necessary ($\leq \epsilon$) downward translation. If both $Q_1, Q_2 \neq \phi$, a match may or may not exist. Below we introduce the notion of *critical angle*. The existence of such an angle implies a match.

Definition 2. *If one can find a match (under Lemma 1) with the application of a rotation of Q by the angle $\theta^* \in \mathcal{I}^*$ and a suitable vertical downward shift, then θ^* is said to be a critical angle.*

Thus, after applying rotation of Q with a critical angle θ^* and then applying a vertical downward translation, one can bring all the members in Q_2 inside its corresponding ϵ-box, and all the points in Q_1 will not leave its corresponding ϵ-box. We first split the angular interval corresponding to a clique into homogeneous intervals as defined below. Next, we consider each homogeneous interval, and compute a *critical angle* (if any).

4.1 Homogeneous Splitting of $\mathcal{I}^*$

Let $\mathcal{I}^* = [\theta_1, \theta_2]$ be a $k-1$ clique, and consider a rotation angle $\theta \in \mathcal{I}^*$. For each element $q \in Q_1$, we use a function $f_q^1(\theta)$ to denote the distance of q from the bottom-boundary of the corresponding ϵ-box. Similarly, for each element $q \in Q_2$, a function $f_q^2(\theta)$ denotes the distance of q from the top-boundary of the corresponding ϵ-box. The functions $f_q^i(\theta)$ $i = 1, 2$ are of the form $\overline{q_a q} \sin(\alpha + \theta) - c$; if $q_a \in Q_1$, c is the y-coordinate of the bottom-boundary of the corresponding ϵ-box, and if $q_a \in Q_2$, c is the y-coordinate of the top-boundary of the corresponding ϵ-box (see Figure 3(a)).

Observation 1. *For a given $q \in Q_i$, $i = 1$ or 2, the univariate function $f_q^i(\theta)$ is continuous, and unimodular.*

Next, let us define two more functions, namely $\mathcal{L}(\theta)$ and $\mathcal{U}(\theta)$ for $\theta \in \mathcal{I}^*$, where $\mathcal{L}(\theta)$ denotes the lower envelope of $|Q_1|$ functions, namely $f_q^1(\theta)$, $q \in Q_1$, and $\mathcal{U}(\theta)$ denotes the upper envelope of $|Q_2|$ functions, namely $f_q^2(\theta)$, $q \in Q_2$.

At a rotation angle $\theta \in \mathcal{I}^*$, the minimum amount of downward translation required to place the points in Q_2 in the corresponding ϵ-box is $max_{q \in Q_2} f_q^2(\theta) = \mathcal{U}(\theta)$. Similarly, the maximum amount of downward translation that may retain all the points in Q_1 in its corresponding ϵ-box is $min_{q \in Q_1} f_q^1(\theta) = \mathcal{L}(\theta)$. Now, for each of the two functions $\mathcal{L}$ and $\mathcal{U}$, we define *break-points* as follows:

Definition 3. *A rotation angle θ is said to be a break-point in the function $\mathcal{L}$ if $\mathcal{L}(\theta) = f_{q_a}^1(\theta) = f_{q_b}^1(\theta)$ for two distinct points $q_a, q_b \in Q_1$. Similarly, the break-points of the $\mathcal{U}$ function is defined.*

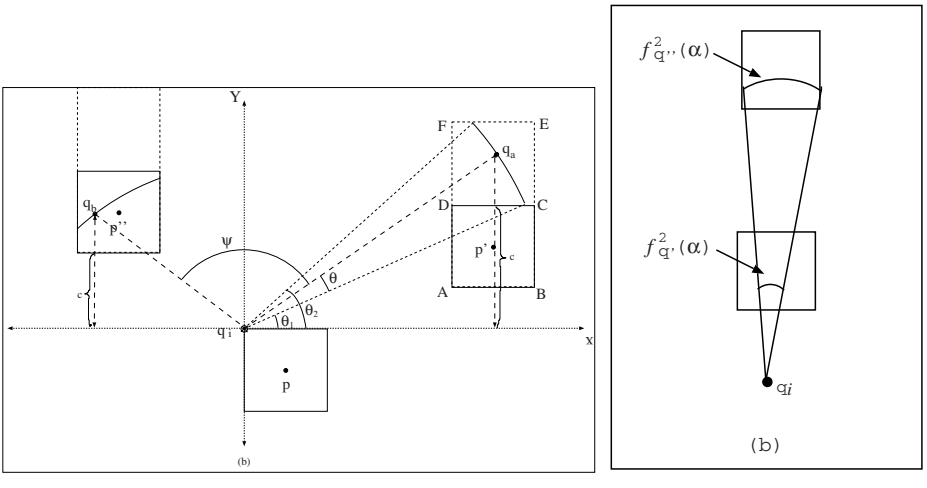


Fig. 3. (a) Finding a suitable translation for a match, (b) proof of Lemma 3

Lemma 3. *If we plot a pair of functions $f^1_{q'}(\theta)$ and $f^1_{q''}(\theta)$ (corresponding to two different points $q', q'' \in Q_1$) with respect to $\theta \in \mathcal{I}^*$, they may intersect in at most two points. The same result is true for a pair of functions $f^2_{q'}(\theta)$ and $f^2_{q''}(\theta)$ for a pair of points $q', q'' \in Q_2$.*

Proof. Follows from Observation 1. See, also Figure 3(b). □

Lemma 3 implies that, for a pair of points $q', q'' \in Q_2$, it may happen that $f^2_{q'}(\theta) < f^2_{q''}(\theta)$ for both $\theta = \theta^*_1$ and θ^*_2 of an interval of rotation angle $\mathcal{I}^*$, but there exists a sub-interval $[\phi_1, \phi_2] \in \mathcal{I}^*$, such that $f^2_{q'}(\theta) < f^2_{q''}(\theta)$ for $\theta \in [\phi_1, \phi_2]$.

Observation 2. *The collection of functions $\{f^1_q(\theta), q \in Q_1\}$ follows a $(k, 2)$-Davenport-Schinzel sequence. The same result is true for the collection of functions $\{f^2_q(\theta), q \in Q_2\}$. See [19] for details.*

Lemma 4. *The maximum number of break-points in the function $\mathcal{L}(\theta)$ is $\lambda_2(|Q_1|) = 2|Q_1| - 1$, and it can be computed in $O(|Q_1| \log |Q_1|)$ time. Similarly, the maximum number of break-points in the function $\mathcal{U}(\theta)$ is $\lambda_2(|Q_2|) = 2|Q_2| - 1$, and it can be computed in $O(|Q_2| \log |Q_2|)$ time.*

Proof. Follows from Observation 2 [19]. □

Thus, the entire interval $\mathcal{I}^* = [\theta^*_1, \theta^*_2]$ is split into sub-intervals defined by the break-points of $\mathcal{L}(\theta)$ and $\mathcal{U}(\theta)$, and the number of such sub-intervals is $O(k)$ in the worst case.

4.2 Computation of Critical-Angle

For a rotation angle θ in a sub-interval $[\theta_1, \theta_2]$, the vertical downward shift will be determined by two points, say $q_\alpha, q_\beta \in Q$, where q_α and q_β are such that $f^1_{q_\alpha}(\theta) = \mathcal{L}(\theta)$ and $f^2_{q_\beta}(\theta) = \mathcal{U}(\theta)$. It is also observed that, q_α lies in the extended ϵ-box of p', q_β lies in the ϵ-box of p'', and the angle $\angle q_a q_i q_b$ ($= \psi$ say) does not change due to the rigid motion. We use $\delta(a, b)$ to denote the Euclidean distance between a pair of points a and b. Now consider the following quantities:

$\Delta_1 =$ Minimum amount of downward shift required to bring q_α inside the ϵ-box of p'. Thus, $\Delta_1 = \delta(q_i, q_\alpha) sin(\theta + \theta_1) - (y(p') + \frac{\epsilon}{2})$.

$\Delta_2 =$ Maximum amount of permissible downward shift keeping q_β inside the ϵ-box of p''. Thus, $\Delta_2 = \delta(q_i, q_\beta) sin(\theta + \theta_1 + \psi) - (y(p'') - \frac{\epsilon}{2})$.

Thus, a feasible solution θ (if it exists) must satisfy $\Delta_1 \leq \Delta_2$. This, on simplification, gives $\delta(q_i, q_\alpha) sin(\theta + \theta_1) - \delta(q_i, q_\beta)(sin(\theta + \theta_1) cos(\psi) + cos(\theta + \theta_1) sin(\psi)) \leq y(p') - y(p'') + \epsilon$.

Let A and B be two constants such that $(\delta(q_i, q_\alpha) - \delta(q_i, q_\beta) \cos(\psi)) = A \cos(B)$ and $\delta(q_i, q_\beta) \sin(\psi) = A \sin(B)$.

Thus, we have $A \sin(\theta + \theta_1 - B) \leq y(p') - y(p'') + \epsilon$.

This implies, $\theta \leq \sin^{-1}(\frac{y(p') - y(p'') + \epsilon}{A}) + B - \theta_1$.

If $\sin^{-1}(\frac{y(p')-y(p'')+\epsilon}{A}) + B - \theta_1 \leq \theta_1$, the critical angle in the sub-interval $[\theta_1, \theta_2]$ does not exist. Otherwise, for any rotation angle in the interval

$$[\theta_1, min(\sin^{-1}(\frac{y(p')-y(p'')+\epsilon}{A}) + B - \theta_1, \theta_2)],$$

a match can be obtained with a Δ_1 unit of downward translation.

4.3 Complexity Analysis

Theorem 2. *The time complexity of the proposed algorithm for ϵ-approximate matching of Q with a subset of P where the neighbourhood around a point (in P) is defined as an ϵ-box, is $O(n^2k^2(\log n + k\log k))$.*

Proof. Each point of Q needs to be anchored at the top-left corner of the ϵ-box of each point in P. The nodes of the circular arc graph G are obtained in $O(nk)$ time, and the cliques of graph G can be obtained after getting the circular order of the nodes; this takes $O(nk\log n)$ time.

While processing a clique, the computation of the functions $\mathcal{U}$ and $\mathcal{L}$ needs $O(k\log k)$ time, and number of elements in $\mathcal{U} \cup \mathcal{L}$ is $O(k)$ in the worst case [19]. The algebraic computation for processing each element in $\mathcal{U} \cup \mathcal{L}$ needs $O(1)$ time. Thus, the overall time complexity result follows. □

5 Conclusion

An easy to implement and numerically robust algorithm for the approximate matching of point set is described in this paper, where the ϵ-neighborhood of the sample points are axis-parallel squares instead of circles. Our algorithm works for well-separated points, where each pair of points $p, p' \in P$ satisfy either $|x(p) - x(p')| \geq \epsilon$ or $|y(p) - y(p')| \geq 3\epsilon$ or both. Our algorithm does not use the computation of the intersection of high degree algebraic curves; it only needs to compute intersections of circles and squares. The worst case time complexity of the algorithm is shown to be $O(n^2k^2(k\log k + \log n))$. Though the time complexity of the algorithm is still high, it terminates much faster in all practical instances. As a measure of comparison we would like to mention that, the same problem was solved earlier in $O(n^4\log n)$ for the equal cardinality case ($k = n$) using intersection of high degree algebraic curves [4]. The problem for labeled cases [2] can be solved in $O(k^2)$ time using our scheme. For other types of ϵ-regions [2, 4], our scheme can be applied with a suitable modification to the shapes of the extended boxes.

References

1. Akutsu, T.: On determining the congruence of point sets in d dimensions. Computational Geometry: Theory and Applications 9, 247–256 (1998)
2. Alt, H., Mehlhorn, K., Wagener, H., Welzl, E.: Congruence, similarity and symmetries of geometric objects. Discrete Computational Geometry 3, 237–256 (1988)

3. Alt, H., Guibas, L.: Discrete geometric shapes: Matching, interpolation, and approximation. In: Handbook of Computational Geometry, pp. 121–153. Elsevier Science Publishers B.V. North-Holland, Amsterdam (1999)
4. Arkin, E.M., Kedem, K., Mitchell, J.S.B., Sprinzak, J., Werman, M.: Matching points into pairwise-disjoint noise regions: combinatorial bounds and algorithms. ORSA Journal on Computing 4, 375–386 (1992)
5. Brass, P., Knauer, C.: Testing the congruence of d-dimensional point sets. Int. J. Computational Geometry and Applications 12, 115–124 (2002)
6. Cabello, S., Giannopoulos, P., Knauer, C.: On the parameterized complexity of d-dimensional point set pattern matching. Information Processing Letters 105, 73–77 (2008)
7. Chew, L.P., Goodrich, M.T., Huttenlocher, D.P., Kedem, K., Kleinberg, J.M., Kravets, D.: Geometric pattern matching under euclidean motion. Computational Geometry: Theory and Applications 7, 113–124 (1997)
8. Efrat, A., Itai, A.: Improvements on bottleneck matching and related problems using geometry. In: Proc. 12th ACM Symposium on Computational Geometry, pp. 301–310. ACM, New York (1996)
9. Gavrilov, M., Indyk, P., Motwani, R., Venkatasubramanian, S.: Geometric pattern matching: a performance study. In: Proc. 15th ACM Symposium on Computational Geometry, pp. 79–85. ACM, New York (1999)
10. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, NY (1980)
11. Goodrich, M.T., Mitchell, J.S.B., Orletsky, M.W.: Approximate geometric pattern matching under rigid motions. IEEE Trans. PAMI 21(4), 371–379 (1999)
12. Heffernan, P.J., Schirra, S.: Approximate decision algorithms for point set congruence. Computational Geometry: Theory and Applications 4(3), 137–156 (1994)
13. Imai, K., Sumino, S., Imai, H.: Minimax geometric fitting of two corresponding sets of points. In: Proc. 5th ACM Symposium on Computational Geometry, pp. 266–275. ACM, New York (1989)
14. Indyk, P., Motwani, R., Venkatasubramanian, S.: Geometric matching under noise: combinatorial bounds and algorithms. In: Proc. 10th SIAM-ACM Symposium on Discrete Algorithms, pp. 457–465. ACM-SIAM, New York (1999)
15. Irani, S., Raghavan, P.: Combinatorial and experimental results for randomized point matching algorithms. In: Proc. 12th ACM Symposium on Computational Geometry, pp. 68–77. ACM, New York (1996)
16. Maltoni, D., Maio, D., Jain, A.K., Prabhakar, S.: Handbook of Fingerprint Recognition. Springer, NY (2003)
17. Mount, D.M., Netanyahu, N.S., Moigne, J.L.: Efficient algorithms for robust feature matching. Pattern Recognition 32, 17–38 (1999)
18. Rezende, P.J., Lee, D.T.: Point set pattern matching in d-dimensions. Algorithmica 13, 387–404 (1995)
19. Sharir, M., Agarwal, P.K.: Davenport-Schinzel Sequences and Their Geometric Applications. Cambridge University Press, NY (1995)

Acyclically 3-Colorable Planar Graphs

Patrizio Angelini and Fabrizio Frati

Dipartimento di Informatica e Automazione – Roma Tre University
{angelini,frati}@dia.uniroma3.it

Abstract. In this paper we study the planar graphs that admit an acyclic 3-coloring. We show that testing acyclic 3-colorability is $\mathcal{NP}$-hard for planar graphs of maximum degree 4 and we show that there exist infinite classes of cubic planar graphs that are not acyclically 3-colorable. Further, we show that every planar graph has a subdivision with one vertex per edge that is acyclically 3-colorable. Finally, we characterize the series-parallel graphs such that every 3-coloring is acyclic and we provide a linear-time recognition algorithm for such graphs.

1 Introduction

A *coloring* of a graph is an assignment of *colors* to vertices such that no two adjacent vertices have the same color. A *k-coloring* is a coloring using k colors. Planar graph colorings have been widely studied from both a combinatorial and an algorithmic point of view. The existence of a 4-coloring for every planar graph, proved by Appel and Haken [4,5], is one of the most famous results in Graph Theory. A quadratic-time algorithm is known to compute a 4-coloring of any planar graph [15].

An *acyclic coloring* is a coloring with *no bichromatic cycle*. An *acyclic k-coloring* is an acyclic coloring using k colors. Acyclic colorings have been deeply investigated in the literature. From an algorithmic point of view, Kostochka proved in [12] that deciding whether a graph admits an acyclic 3-coloring is $\mathcal{NP}$-hard. From a combinatorial point of view, the most interesting result is perhaps the one proved by Alon *et al.* in [2], namely that every graph with degree Δ can be acyclically colored with $O(\Delta^{4/3})$ colors, while there exist graphs requiring $\Omega(\Delta^{4/3}/\sqrt[3]{\log \Delta})$ colors in any acyclic coloring.

Acyclic colorings of planar graphs have been first considered in 1973 by Grünbaum, who proved in [10] that there exist planar graphs requiring 5 colors in any acyclic coloring. The same lower bound holds even for bipartite planar graphs [13]. Grünbaum conjectured that such a bound is tight and proved that 9 colors suffice for constructing such a coloring. The Grünbaum upper bound was improved to 8 [14], to 7 [1], to 6 [11], and finally to 5 by Borodin [6].

Since there exist planar graphs requiring 5 colors in any acyclic coloring, it is natural to study which planar graphs can be acyclically 3- or 4-colored. In this paper we study the acyclically 3-colorable planar graphs, from both an algorithmic and a combinatorial perspective. We show the following results.

– In Sect. 3 we prove that deciding whether a planar graph of maximum degree 4 has an acyclic 3-coloring is an $\mathcal{NP}$-complete problem. An $\mathcal{NP}$-hardness proof for deciding acyclic 3-colorability was known for bipartite planar graphs of degeneracy

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 113–124, 2010.

2 [12]. The $\mathcal{NP}$-hardness result is not surprising, since an analogous result is known for deciding (possibly non-acyclic) 3-colorability of planar graphs of degree 4 [9]. However, we show an interesting difference between the class of 3-colorable planar graphs and the class of acyclically 3-colorable planar graphs, by exhibiting an infinite number of cubic planar graphs not admitting any acyclic 3-coloring (while K_4 is the only cubic graph that can not be 3-colored [8]). We remark that it is known how to construct acyclic 4-colorings of every cubic (even non-planar) graph [16].

– In Sect. 4 we prove that every planar graph has a subdivision with one vertex per edge that is acyclically 3-colorable. Acyclic colorings of graph subdivisions have been already considered by Wood in [18], where the author observed that every graph has a subdivision with two vertices per edge that is acyclically 3-colorable.
– In Sect. 5 we consider the problem of determining the planar graphs such that every 3-coloring is acyclic. Such a problem has been introduced by Grünbaum [10], who showed that every 3-coloring of a maximal outerplanar graph is acyclic. We improve his result by characterizing the series-parallel graphs such that every 3-coloring is acyclic and by providing a linear-time recognition algorithm. As a side result, we show a simple algorithm for obtaining an acyclic 3-coloring of any series-parallel graph.

In Sect. 6 we conclude and we present some open problems. Some proofs are omitted because of space limitations and can be found in the full version of the paper [3].

2 Preliminaries

A graph G is *k-connected* if removing any k-1 vertices leaves G connected; 3-connected and 2-connected graphs are called *triconnected* and *biconnected* graphs, respectively. The *degree of a vertex* is the number of incident edges. The *degree of a graph* is the maximum degree of the vertices of the graph. In a *cubic* graph (resp. a *subcubic* graph) each vertex has degree exactly 3 (resp. at most 3). A *subdivision* of a graph G is obtained by replacing each edge of G with a path. A *k-subdivision* of G is such that any path replacing an edge of G has at most k internal vertices. The internal (extremal) vertices of the paths replacing the edges of G are called *subdivision vertices* (resp. *main vertices*).

A *planar graph* is a graph with no K_5-minor and no $K_{3,3}$-minor. A planar graph is *maximal* if all its faces are delimited by 3-cycles. An *outerplanar graph* is a graph admitting a planar drawing with all the vertices on the outer face. Combinatorially, an outerplanar graph is a graph with no K_4-minor and no $K_{2,3}$-minor. An outerplanar graph is *maximal* if all its internal faces are delimited by 3-cycles. A *series-parallel graph (SP-graph)* is a graph with no K_4-minor. SP-graphs are inductively defined as follows. An edge (u, v) is an SP-graph with *poles* u and v. Denote by u_i and v_i the poles of an SP-graph graph G_i. A *series composition* of SP-graphs $G_0, \ldots, G_k$, with $k \geq 1$, is an SP-graph with poles u=u_0 and v=v_k, containing graphs G_i as subgraphs, and such that v_i=u_{i+1}, for each i=$0, 1, \ldots, k-1$. A *parallel composition* of SP-graphs $G_0, \ldots, G_k$, with $k \geq 1$, is an SP-graph with poles u=u_0=u_1=$\ldots$=u_k and v=v_0=v_1=$\ldots$=v_k and containing graphs G_i as subgraphs. The SPQ-tree $\mathcal{T}$ of an SP-graph G is the tree, rooted at any node, representing the series and parallel compositions of G.

3 Deciding the Acyclic 3-Colorability of Planar Graphs

In this section we study the problem of deciding whether a given planar graph admits an acyclic 3-coloring. First, we present a very simple proof that Planar Graph Acyclic 3-Colorability is $\mathcal{NP}$-hard. We remark that a proof of $\mathcal{NP}$-hardness for Planar Graph Acyclic 3-Colorability has been already presented by Kostochka in [12]. We later prove an analogous complexity result for planar graphs of maximum degree 4.

Theorem 1. *Planar Graph Acyclic 3-Colorability is $\mathcal{NP}$-complete.*

The membership in $\mathcal{NP}$ is trivial. To show the $\mathcal{NP}$-hardness, we sketch a simple reduction from Planar Graph 3-Colorability that uses the graph G_9 shown in Fig. 1.a as a gadget. It is easy to see that G_9 has only one acyclic 3-coloring (up to a switch of the color classes), which satisfies the following properties: (P1) u_1 and u_2 have different colors; (P2) every path connecting u_1 and u_2 contains vertices of all the three colors.

The reduction works as follows. Let G be an instance of Planar Graph 3-Colorability (see Fig. 1.b). Replace each edge (u, v) of G with a copy of G_9 by identifying vertices u and v with u_1 and u_2, respectively (see Fig. 1.c). Let G' be the resulting planar graph. We argue that G admits a 3-coloring if and only if G' admits an acyclic 3-coloring.

First, suppose that G admits a 3-coloring. For each edge (u, v) of G, color the corresponding graph G_9 in G' by assigning the color of u to u_1, the color of v to u_2, and by then completing the unique acyclic 3-coloring of G_9. The resulting coloring of G' is acyclic. Namely, assume, for a contradiction, that G' contains a bichromatic cycle $\mathcal{C}$. Such a cycle is not entirely contained inside a graph G_9 replacing an edge of G in G' (in fact, the 3-coloring of each graph G_9 is acyclic). Hence, $\mathcal{C}$ contains vertices of more than one graph G_9. This implies that $\mathcal{C}$ contains as a subgraph a simple path p connecting vertices u_1 and u_2 of a graph G_9. However, by property P2 of the G_9's coloring, p contains vertices of all the three colors, a contradiction.

Second, if G' admits an acyclic 3-coloring, a coloring of G is obtained from the acyclic 3-coloring of G' by assigning to each vertex of G the color of the corresponding vertex of G'. By property P1, each edge of G connects vertices of distinct colors.

Next, we show that testing whether a planar graph has an acyclic 3-coloring remains an $\mathcal{NP}$-hard problem even when restricted to planar graphs of degree 4.

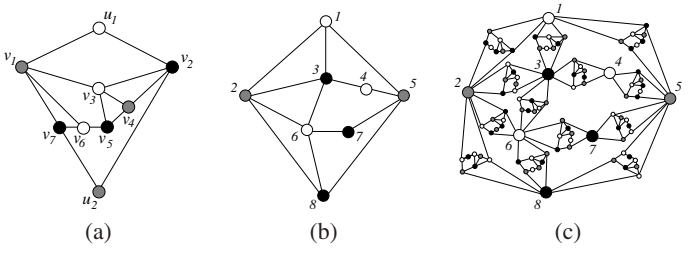


Fig. 1. (a) Graph G_9 and its unique acyclic 3-coloring. (b) A planar graph G. (c) The planar graph G' obtained by replacing each edge of G with a copy of G_9.

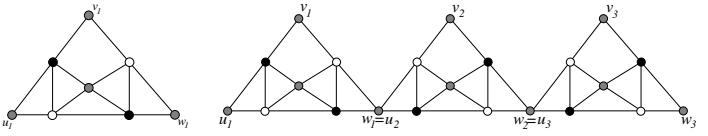


Fig. 2. (a) Graph H_1. (b) Graph H_3.

Theorem 2. *Degree-4 Planar Graph Acyclic 3-Colorability is $\mathcal{NP}$-complete.*

The membership in $\mathcal{NP}$ is trivial. To show the $\mathcal{NP}$-hardness, we sketch a simple reduction from Planar Graph Acyclic 3-Colorability. Consider the family of graphs H_i defined as follows. H_1 is shown in Fig. 2.a. H_i is obtained from a copy of H_{i-1} and a copy of H_1 by renaming vertices u_1, v_1, and w_1 of H_1 with labels u_i, v_i, and w_i, respectively, and by identifying vertex w_{i-1} of H_{i-1} and vertex u_i of H_1. H_3 is shown in Fig. 2.b. Vertices u_j, v_j, and w_j of H_i, for $1 \leq j \leq i$, are the *outlets* of H_i. The family of graphs H_i has been defined in [9] to perform a reduction from *Planar Graph Colorability* to *Degree-4 Planar Graph Colorability*. Here we use the same graph class to reduce Planar Graph Acyclic 3-Colorability to Degree-4 Planar Graph Acyclic 3-Colorability. It is easy to see that H_i satisfies the following properties: (P0) H_i admits an acyclic 3-coloring; (P1) in any acyclic 3-coloring of H_i, the outlets have the same color c_0; (P2) in any acyclic 3-coloring of H_i, for any two outlets x_j and y_k of H_i, there exist two bichromatic paths with colors c_0 and c_1, and with colors c_0 and c_2, respectively, where $x, y \in \{u, v, w\}$ and $j, k \in \{1, 2, \ldots, i\}$.

We reduce Planar Graph Acyclic 3-Colorability to Degree-4 Planar Graph Acyclic 3-Colorability. Let G be any instance of Planar Graph Acyclic 3-Colorability (Fig. 3.a). For each vertex z of G with d neighbors $z_1, z_2, \ldots, z_d$, delete z and its incident edges from G, introduce a copy $H(z)$ of H_d, and add an edge between outlet v_j of $H(z)$ and z_j, for each j=1, 2, ..., d (Fig. 3.b). We argue that the resulting planar graph G' of degree 4 admits an acyclic 3-coloring if and only if G admits an acyclic 3-coloring.

Suppose that G admits an acyclic 3-coloring. Color the outlets z_j corresponding to each vertex z of G with the color of z. By properties P0 and P1, the coloring of each $H(z)$ can be completed to an acyclic 3-coloring. Any cycle C' of G' either is entirely

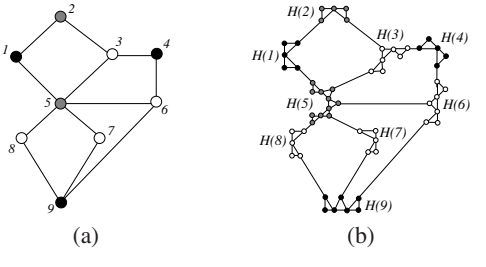


Fig. 3. (a) A planar graph G. (b) Graph G' obtained by replacing each degree-d vertex z of G with a copy $H(z)$ of H_d. For each graph $H(z)$, only its outlets are shown.

contained in a graph $H(z)$ (hence $\mathcal{C}'$ is not bichromatic), or contains vertices of several graphs $H(z)$. In the latter case suppose, for a contradiction, that $\mathcal{C}'$ is bichromatic. Consider the (possibly non-simple) cycle $\mathcal{C}$ of G containing a vertex z if $\mathcal{C}'$ passes through vertices of $H(z)$ and containing an edge (z_1, z_2) if $\mathcal{C}'$ contains an edge between a vertex of $H(z_1)$ and a vertex of $H(z_2)$. Since the outlets of $H(z)$ have the same color of z, the colors of the vertices of $\mathcal{C}$ are a subset of the colors of the vertices of $\mathcal{C}'$; since $\mathcal{C}'$ is bichromatic, $\mathcal{C}$ is bichromatic, as well, contradicting the assumption that the coloring of G is acyclic.

Suppose that G' admits an acyclic 3-coloring. Color G by assigning to each vertex z the color of the outlets of $H(z)$ (by Property P1, all such outlets have the same color). Suppose that G contains a bichromatic cycle $\mathcal{C}$ with colors c_0 and c_1. A bichromatic cycle $\mathcal{C}'$ in G' is found by replacing each vertex z_1 of $\mathcal{C}$ with a path with colors c_0 and c_1 connecting the outlets of $H(z_1)$ adjacent to the outlets of $H(z_2)$ and $H(z_3)$, where z_2 and z_3 are the neighbors of z_1 in $\mathcal{C}$. Such a path exists by Property P2. Then, $\mathcal{C}'$ is a bichromatic cycle in G', contradicting the assumption that the coloring of G' is acyclic.

Now we show infinite classes of cubic planar graphs not admitting any acyclic 3-coloring. Such a result is based on the following lemmata. Denote by $K_{2,3}$ the complete bipartite graph whose vertex sets $V^A_{2,3}$ and $V^B_{2,3}$ have two and three vertices, respectively. Denote by $K_{1,1,2}$ the complete tripartite graph whose vertex sets $V^A_{1,1,2}$, $V^B_{1,1,2}$, and $V^C_{1,1,2}$ have one, one, and two vertices, respectively.

Lemma 1. *Let G be a graph having a vertex z of degree 2 adjacent to two vertices u and v. Let G' be the graph obtained by substituting z with a copy of $K_{2,3}$, where a vertex $u^B_{2,3}$ of $V^B_{2,3}$ is connected to u and a vertex $v^B_{2,3} \neq u^B_{2,3}$ of $V^B_{2,3}$ is connected to v (see Fig. 4 a and Fig. 4 b). Then, G' has an acyclic 3-coloring if and only if G has an acyclic 3-coloring.*

Proof: Suppose that G has an acyclic 3-coloring. Color each vertex of G' not in $K_{2,3}$ as in G, the vertices in $V^B_{2,3}$ with the color c_z of z, and the vertices in $V^A_{2,3}$ with the two colors different from c_z. Every cycle $\mathcal{C}'$ in G' either does not pass through vertices of $K_{2,3}$ (hence it is also a cycle in G and it is not bichromatic), or it is a subgraph of $K_{2,3}$ (hence it is not bichromatic), or it passes through vertices of $K_{2,3}$ and contains a path $\mathcal{P}'$ from $u^B_{2,3}$ to $v^B_{2,3}$ whose vertices do not belong to $K_{2,3}$ (except for $u^B_{2,3}$ and $v^B_{2,3}$). However, $\mathcal{P}'$ is a cycle in G (where $u^B_{2,3}$ and $v^B_{2,3}$ are identified to be the same vertex z), hence it is not bichromatic.

Suppose that G' has an acyclic 3-coloring. In any acyclic coloring of $K_{2,3}$, the vertices in $V^B_{2,3}$ have the same color c_z. Color each vertex of G different from z as in G' and color z with c_z. Every cycle $\mathcal{C}$ in G either does not pass through z (hence it is also a cycle in G' and it is not bichromatic), or passes through z. In the latter case, if $\mathcal{C}$ is bichromatic then each of its vertices has either the color of z or the one of u. However, one vertex in $V^A_{2,3}$, say $x^A_{2,3}$, has the color of u, hence the cycle $\mathcal{C}'$ of G' obtained from $\mathcal{C}$ by replacing (u, z, v) with $(u, u^B_{2,3}, x^A_{2,3}, v^B_{2,3}, v)$ is bichromatic, a contradiction. □

Lemma 2. *Let G be a graph having a vertex z of degree 2 adjacent to two vertices u and v. Let G' be the graph obtained by substituting z with a copy of $K_{1,1,2}$, where a vertex $u^C_{1,1,2}$ of $V^C_{1,1,2}$ is connected to u and a vertex $v^C_{1,1,2} \neq u^C_{1,1,2}$ of $V^C_{1,1,2}$ is connected*

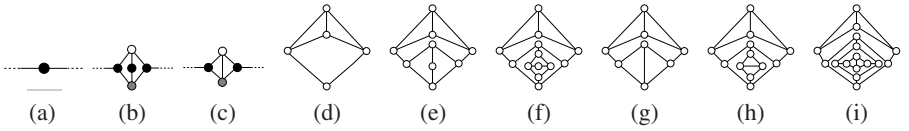


Fig. 4. (a) and (b) Replacement of a degree-2 vertex with a $K_{2,3}$. (a) and (c) Replacement of a degree-2 vertex with a $K_{1,1,2}$. (d) G_5. (e) G_9. (f) G_{13}. (g) G_5^+. (h) G_9^+. (i) G_{13}^+.

to v (see Fig. 4.a and Fig. 4.c). Then, G' has an acyclic 3-coloring if and only if G has an acyclic 3-coloring.

Graph G_5 (Fig. 4.d) has no acyclic 3-coloring and has a degree-2 vertex. For $i > 0$, replace the degree-2 vertex of G_{4i+1} with a copy of $K_{2,3}$, obtaining a graph G_{4i+5} that has a degree-2 vertex and, by Lemma 1, is not acyclically 3-colorable. Figs. 4.e–f show G_9 and G_{13}. Replacing the degree-2 vertex of G_{4i+1} with a copy of $K_{1,1,2}$ yields a graph G_{4i+1}^+ that, by Lemma 2, is not acyclically 3-colorable. Figs. 4.g–i show G_5^+, G_9^+, G_{13}^+. Graphs G_{4i+1}^+ are cubic, for every $i > 0$.

4 Acyclic 3-Colorings of Planar Graph Subdivisions

In this section we prove the following theorem.

Theorem 3. *Every planar graph has a 1-subdivision that admits an acyclic 3-coloring.*

Proof: It suffices to prove the statement for maximal planar graphs. In fact, suppose that the statement holds for maximal planar graphs. Let G be a planar graph. Augment G to a maximal planar graph G' by adding dummy edges. Then G' has a 1-subdivision G'_s that has an acyclic 3-coloring c. Remove the edges of G'_s corresponding to subdivided dummy edges of G', obtaining a planar graph G_s that is a subdivision of G. Since every cycle of G_s is also a cycle of G'_s, c is an acyclic 3-coloring of G_s.

Consider a planar drawing of any maximal planar graph G. Let G_s be the planar graph obtained by subdividing each edge of G with one subdivision vertex. Partition the vertices of G into disjoint sets $V^0, V^1, \ldots, V^k$ as follows. Let G^0=G; while there are vertices in G^i, denote by V^i the main vertices incident to the outer face of G^i; remove the vertices in V^i and their incident edges from G^i obtaining a graph G^{i+1}. Each edge of G is either incident to two vertices in the same set V^i or to two vertices in sets V^i and V^{i+1}, for some $i \in \{0, 1, \ldots, k-1\}$.

Color the main vertices in V^i with color $c_{j(i)}$, where $j(i) \in \{0, 1, 2\}$ and $j(i) \equiv i \bmod 3$. Color each subdivision vertex adjacent to a vertex in V^i and to a vertex in V^{i+1} with color $c_{j(i+2)}$. See Fig. 5.a. It remains to color each subdivision vertex adjacent to two vertices belonging to the same V^i. Consider the outerplanar subgraph O^i of G induced by the vertices in V^i. Augment O^i to maximal by adding dummy edges. See Fig. 5.b. Let O^i_s be the graph obtained by subdividing each edge of O^i with one subdivision vertex. Each subdivision vertex of G_s adjacent to two vertices belonging to the same V^i, for some $i \in \{1, 2, \ldots, k\}$, is also a subdivision vertex of O^i_s. Hence, a coloring of the subdivision vertices of O^i_s determines a coloring of the subdivision

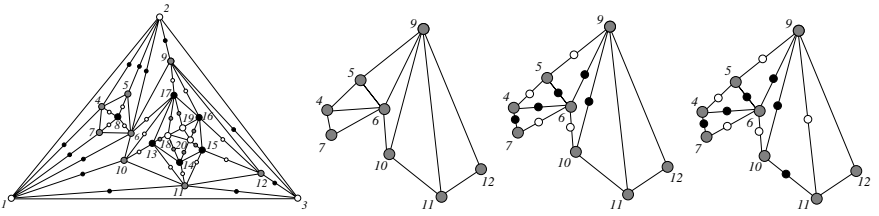


Fig. 5. (a) Coloring the main vertices and the subdivision vertices of G_s adjacent to a vertex in V^i and to a vertex in V^{i+1}. Thick edges connect vertices of G in the same V^i. (b) Subgraph O^2 of G augmented to maximal. (c)–(d) Coloring O^2_s at steps x and $x+1$ of the algorithm. Not yet colored subdivision vertices of O^2_s are not shown.

vertices of G_s adjacent to two vertices in the same V^i. We show how to color the subdivision vertices of O^i_s. The algorithm already chose to color all the main vertices of O^i_s with color $c_{j(i)}$. Since O^i is maximal, every internal face of O^i_s has three subdivision vertices. The coloring algorithm consists of several steps. At the first step, consider any internal face f^* of O^i_s. Color two of its subdivision vertices with $c_{j(i+1)}$ and the third one with $c_{j(i+2)}$. At the x-th step, with $x \geq 2$, suppose that the subgraph $O^{i,x}_s$ of O^i_s induced by the colored subdivision vertices and by their neighbors is biconnected. See Fig. 5.c. Consider any internal face of O^i_s of which one subdivision vertex has already been colored. Color the other two subdivision vertices incident to the face, one with $c_{j(i+1)}$ and the other one with $c_{j(i+2)}$. See Fig. 5.d.

We show that the resulting coloring of G_s is acyclic. Consider any simple cycle $\mathcal{C}$. If $\mathcal{C}$ contains main vertices in V^i and V^{i+1}, then $\mathcal{C}$ contains two edges (v_p, v_s) and (v_s, v_q), where v_p and v_q are main vertices in V^i and V^{i+1}, respectively, and v_s is a subdivision vertex. However, v_p, v_q, and v_s have color $c_{j(i)}$, $c_{j(i+1)}$, and $c_{j(i+2)}$, respectively, hence $\mathcal{C}$ is not bichromatic. Otherwise, $\mathcal{C}$ only contains main vertices in the same V^i. Then, $\mathcal{C}$ is also a cycle of O^i_s. We show by induction that the described coloring of O^i_s is acyclic. The coloring of f^* is acyclic. Suppose that, after a certain step of the coloring algorithm for the vertices of O^i_s, the subgraph $O^{i,x}_s$ of O^i_s induced by the colored subdivision vertices and by their neighbors is acyclic. When a new face is considered and two subdivision vertices v_1 and v_2 are colored with colors $c_{j(i+1)}$ and $c_{j(i+2)}$, respectively, every cycle either entirely belongs to $O^{i,x}_s$, hence by induction it is not bichromatic, or passes through v_1, v_2, and their common neighbor, hence it is not bichromatic. □

5 Acyclic 3-Colorings of Series-Parallel Graphs

In this section we consider the problem of determining which are the SP-graphs such that every 3-coloring is acyclic. First, we show a simple algorithm to construct an acyclic 3-coloring of any SP-graph. Let $c(x)$ denote the color assigned to vertex x.

Theorem 4. *Every SP-graph G with poles u and v admits an acyclic 3-coloring such that $c(u)\neq c(v)$ and every path connecting u and v, except for edge (u,v), contains a vertex w with $c(w)\neq c(u), c(v)$.*

Proof: We prove the statement by induction on the number n of vertices. Case n=2 is trivial. If $n > 2$, distinguish two cases: (Case 1) G is a series composition of SP-graphs $G_0, \cdots, G_k$, such that G_i has poles u_i and v_i, with u_0=u, v_i=u_{i+1}, and v_k=v; (Case 2) G is a parallel composition of SP-graphs $G_0, \cdots, G_k$ with poles u and v.

In Case 1, apply induction to construct an acyclic 3-coloring of G_i with colors c_0, c_1, and c_2 such that $c(u_i)$=$c_{j(i)}$ and $c(v_i)$=$c_{j(i+1)}$, for each i=$0, 1, \ldots, k-1$, where $j(i) \in \{0,1,2\}$ and $j(i) \equiv i \bmod 3$. Apply induction to construct an acyclic 3-coloring of G_k with colors c_0, c_1, and c_2 such that $c(u_k)$=$c_{j(k)}$, and such that $c(v_k)$=c_1, if $c(u_k)$=c_0 or $c(u_k)$=c_2, and $c(v_k)$=c_2, if $c(u_k)$=c_1. By construction, $c(u_0$=$u)$=c_0, $c(u_1)$=c_1, $c(u_2)$=c_2. Every path connecting u and v passes through u_0, u_1, and u_2, hence it is not bichromatic. Further, any simple cycle in G is also a cycle in a component G_i. Hence, by induction, the coloring of G is acyclic.

In Case 2, apply induction to construct an acyclic 3-coloring of G_i, for i=$0, 1, \cdots, k$, with colors c_0, c_1, and c_2 such that $c(u)$=c_0, $c(v)$=c_1, and every path connecting u and v in G_i, except for edge (u,v), contains a vertex w with $c(w)$=c_2. By construction, $c(u)$=c_0 and $c(v)$=c_1. Further, every path connecting u and v is also a path in a component G_i which, by induction, contains a vertex with color c_2, unless it is edge (u,v). Let $\mathcal{C}$ be any simple cycle in G. If all the vertices of $\mathcal{C}$ belong to a graph G_i, then $\mathcal{C}$ is not bichromatic by induction. Otherwise, $\mathcal{C}$ contains vertices u and v, hence it consists of two paths $\mathcal{P}_1$ and $\mathcal{P}_2$ connecting u and v and belonging to two distinct components G_i and G_j. At most one of $\mathcal{P}_1$ and $\mathcal{P}_2$, say $\mathcal{P}_1$, coincides with edge (u,v). By induction, $\mathcal{P}_2$ contains a vertex of color c_2. □

Second, we characterize the SP-graphs that have a 3-coloring in which the poles have distinct colors and the SP-graphs that have a 3-coloring in which the poles have the same color.

Corollary 1. *Every SP-graph with poles u and v admits a 3-coloring with $c(u)\neq c(v)$.*

Lemma 3. *Every SP-graph G with poles u and v admits a 3-coloring with $c(u)$=$c(v)$ if and only if G does not contain edge (u,v).*

Proof: The necessity is trivial. We inductively prove the sufficiency. Suppose that G is a parallel composition of SP-graphs $G_0, G_1, \ldots, G_k$ and that G does not contain edge (u,v). Then, no component G_i contains (u,v), hence it admits a 3-coloring in which $c(u)$=$c(v)$ by induction. Suppose that G is a series composition of graphs $G_0, G_1, \ldots, G_k$. Color G_0 so that $c(u)$=c_0 and the other pole of G_0 has color c_1. Such a coloring exists by Corollary 1. For $1 \leq i \leq k-1$, assume that the color of the pole that G_i shares with G_{i-1} has been already determined to be either c_1 or c_2. Color the pole that G_i shares with G_{i+1} with color c_2 or c_1, respectively, and color G_i so that its poles have colors c_1 and c_2 (such a coloring exists by Corollary 1). Complete the coloring of G by setting $c(v)$=c_0 and by coloring G_k so that its poles have colors c_0 and either c_1 or c_2. Again, such a coloring exists by Corollary 1. □

Third, we characterize the SP-graphs that have a 3-coloring in which there exists a bichromatic path between the poles.

Lemma 4. *Let G be an SP-graph with poles u and v. Suppose that G is a parallel composition of SP-graphs $G_0, G_1, \ldots, G_k$. Then, G admits a 3-coloring with $c(u)\neq c(v)$ and with a bichromatic path between u and v if and only if there exists a component that admits a 3-coloring with $c(u)\neq c(v)$ and with a bichromatic path between u and v.*

Proof: The necessity comes from the observation that every bichromatic path between u and v in G is internal to a component G_i. We prove the sufficiency. There exists a G_i admitting a 3-coloring with $c(u)\neq c(v)$ and with a bichromatic path between u and v. By Corollary 1, all other components can be colored with $c(u)\neq c(v)$, thus completing a 3-coloring of G with the required properties. □

Lemma 5. *Let G be an SP-graph with poles u and v. Suppose that G is a series composition of SP-graphs $G_0, G_1, \ldots, G_k$. Then, G admits a 3-coloring with $c(u)\neq c(v)$ and with a bichromatic path between u and v if and only if the following two conditions are satisfied: (1) Each component admits a 3-coloring with a bichromatic path between its poles and (2) there exists a component G_i with poles u_i and v_i that admits a 3-coloring with $c(u_i)=c(v_i)$ and with a bichromatic path between u_i and v_i, and a 3-coloring with $c(u_i)\neq c(v_i)$ and with a bichromatic path between u_i and v_i, or there exists an odd number of components that admit a 3-coloring in which the poles have different colors and are connected by a bichromatic path.*

Proof: We prove the necessity of (1). Suppose that there exists a G_i that admits no 3-coloring with a bichromatic path between its poles. Every path connecting u and v contains a path between G_i's poles, hence it is not bichromatic. We prove the necessity of (2). Suppose, for a contradiction, that (2) does not hold. Then, in every 3-coloring of G with a bichromatic path between u and v, there is an even number of components G_i such that $c(u_i)\neq c(v_i)$, hence $c(u)=c(v)$. We prove the sufficiency. Suppose that each component G_i admits a 3-coloring with a bichromatic path between its poles. First, suppose that there exists a component G_i with poles u_i and v_i that admits a 3-coloring with $c(u_i)=c(v_i)$ and with a bichromatic path between u_i and v_i, and a 3-coloring with $c(u_i)\neq c(v_i)$ and with a bichromatic path between u_i and v_i. Set $c(u_0)=c_0$. For $0 \leq j \leq i-1$, assume that $c(u_j)$ has already been determined to be either c_0 or c_1; color G_j so that there exists a bichromatic path between u_j and v_j and so that $c(v_j)$ is either c_0 or c_1. Analogously, set $c(v_k)=c_1$. For $k \geq j \geq i+1$, assume that $c(v_j)$ has been determined to be either c_0 or c_1; color G_j so that there exists a bichromatic path between u_j and v_j and so that $c(u_j)$ is either c_0 or c_1. Color G_i so that there exists a bichromatic path between u_i and v_i; this can be done both if $c(u_i)=c(v_i)$ and if $c(u_i)\neq c(v_i)$. Second, suppose that there exists an odd number of components that admit a 3-coloring in which the poles have different colors and are connected by a bichromatic path. Each component has either a 3-coloring with a bichromatic path between its poles and the poles have the same color, or a 3-coloring with a bichromatic path between its poles and the poles have distinct colors. Color each component with such a coloring, so that its poles have colors in $\{c_0, c_1\}$. Since an odd number of components have poles with different colors, $c(u)\neq c(v)$. □

Lemma 6. *Let G be an SP-graph with poles u and v. Suppose that G is a parallel composition of SP-graphs $G_0, G_1, \ldots, G_k$. Then, G admits a 3-coloring with $c(u)$=$c(v)$ and with a bichromatic path between u and v if and only if G does not contain edge (u, v) and there exists a component admitting a 3-coloring with $c(u)$=$c(v)$ and with a bichromatic path between u and v.*

Lemma 7. *Let G be an SP-graph with poles u and v. Suppose that G is a series composition of SP-graphs $G_0, G_1, \ldots, G_k$. Then, G admits a 3-coloring with $c(u)$=$c(v)$ and with a bichromatic path between u and v if and only if the following two conditions are satisfied: (1) Each component admits a 3-coloring with a bichromatic path between its poles and (2) there exists a component G_i with poles u_i and v_i admitting a 3-coloring with $c(u_i)$=$c(v_i)$ and with a bichromatic path between u_i and v_i, and a 3-coloring with $c(u_i)\neq c(v_i)$ and with a bichromatic path between u_i and v_i, or there exists an even number of components admitting a 3-coloring in which the poles have different colors and are connected by a bichromatic path.*

Fourth, we characterize the SP-graphs such that every 3-coloring in which the poles have distinct colors is acyclic and the SP-graphs such that every 3-coloring in which the poles have the same color is acyclic.

Lemma 8. *Let G be an SP-graph with poles u and v. Suppose that G is a parallel composition of SP-graphs $G_0, G_1, \ldots, G_k$. Then, every 3-coloring of G with $c(u)\neq c(v)$ is acyclic if and only if the following two conditions are satisfied: (1) For each component G_i, every 3-coloring with $c(u)\neq c(v)$ is acyclic; (2) there exist no two components admitting a 3-coloring with $c(u)\neq c(v)$ and with a bichromatic path between u and v.*

Lemma 9. *Let G be an SP-graph with poles u and v. Suppose that G is a series composition of SP-graphs $G_0, G_1, \ldots, G_k$. Then, every 3-coloring of G with $c(u)\neq c(v)$ is acyclic if and only if the following two conditions are satisfied: (1) For each component G_i with poles u_i and v_i, every 3-coloring with $c(u_i)\neq c(v_i)$ is acyclic; (2) for each component G_i with poles u_i and v_i, every 3-coloring with $c(u_i)$=$c(v_i)$ is acyclic.*

Lemma 10. *Let G be an SP-graph with poles u and v. Suppose that G is a parallel composition of SP-graphs $G_0, G_1, \ldots, G_k$. Then, every 3-coloring of G with $c(u)$=$c(v)$ is acyclic if and only if one of the following two conditions is satisfied: (1) There exists a component G_i not admitting any 3-coloring with $c(u_i)$=$c(v_i)$; or (2) for each component G_i, every 3-coloring with $c(u)$=$c(v)$ is acyclic and no two components exist admitting a 3-coloring with $c(u)$=$c(v)$ and with a bichromatic path between u and v.*

Lemma 11. *Let G be an SP-graph with poles u and v. Suppose that G is a series composition of SP-graphs $G_0, G_1, \ldots, G_k$. Then, every 3-coloring of G with $c(u)$=$c(v)$ is acyclic if and only if the following three conditions are satisfied: (1) For each component G_i with poles u_i and v_i, every 3-coloring with $c(u_i)\neq c(v_i)$ is acyclic; (2) if $k > 2$, for each component G_i with poles u_i and v_i, every 3-coloring with $c(u_i)$=$c(v_i)$ is acyclic; (3) if k=2, for each component G_i with poles u_i and v_i, every 3-coloring with $c(u_i)$=$c(v_i)$ is acyclic, or there exists a component not admitting any 3-coloring in which $c(u_i)$=$c(v_i)$.*

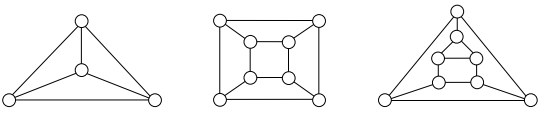


Fig. 6. Triconnected cubic planar graphs with no acyclic 3-coloring

Finally, we conclude by observing that an SP-graph with poles u and v is such that every 3-coloring is acyclic if and only if every 3-coloring in which $c(u)\neq c(v)$ is acyclic and every 3-coloring in which $c(u)=c(v)$ is acyclic. The above characterization gives rise to a linear-time recognition algorithm.

Theorem 5. *There exists a linear-time algorithm for deciding whether an SP-graph is such that every 3-coloring is acyclic.*

Proof: The SPQ-tree $\mathcal{T}$ of an SP-graph G can be computed in linear-time (see, e.g., [17]). Then, each node μ of $\mathcal{T}$ with poles u_μ and v_μ can be equipped with values indicating whether: (i) $G(\mu)$ admits a 3-coloring with $c(u_\mu)=c(v_\mu)$; (ii) $G(\mu)$ admits a 3-coloring with $c(u_\mu)\neq c(v_\mu)$ and with a bichromatic path between u_μ and v_μ, $G(\mu)$ admits a 3-coloring with $c(u_\mu)=c(v_\mu)$ and with a bichromatic path between u_μ and v_μ, and $G(\mu)$ admits a 3-coloring with a bichromatic path between u_μ and v_μ; and (iii) every 3-coloring of $G(\mu)$ in which $c(u_\mu)\neq c(v_\mu)$ is acyclic, every 3-coloring of $G(\mu)$ in which $c(u_\mu)=c(v_\mu)$ is acyclic, and every 3-coloring of $G(\mu)$ is acyclic. Due to Lemmata 3-11, the computation of such values for μ only requires simple checks on analogous values for the children of μ in $\mathcal{T}$. □

6 Conclusions

In this paper we have shown several results on the acyclic 3-colorability of planar graphs. We proved that recognizing acyclic 3-colorable planar graphs of degree 4 is $\mathcal{NP}$-hard. Further, we exhibited infinite classes of subcubic and cubic planar graphs with no acyclic 3-coloring, result contrasting with the fact that all cubic planar graphs have a 3-coloring, except for K_4 [8]. However, the following problem is still open.

What is the time complexity of testing whether a sub-cubic graph (resp. a cubic graph) admits an acyclic 3-coloring?

The problem is interesting even when restricted to *triconnected* cubic planar graphs. Moreover, we are aware of only three graphs that are cubic, triconnected, and not acyclic 3-colorable (see Fig. 6). The graphs depicted in Figs. 6.a and 6.b were already known to have no acyclic 3-coloring. On the other hand, the graph depicted in Fig. 6.c seems to have gone unnoticed in the literature.

Does an infinite number of triconnected, cubic, and not acyclic 3-colorable planar graphs exist? What is the time complexity of testing whether a triconnected cubic planar graph admits an acyclic 3-coloring?

We have shown that it is possible to test in linear time whether every 3-coloring of an SP-graph is acyclic. Testing and characterizing the same property for general planar graphs seems to be interesting and non-trivial.

Is it possible to test in polynomial time whether every 3-coloring of a given planar graph is acyclic?

Finally, we would like to remind a problem that has been already studied in the literature but that has not been tackled in this paper.

Which is the smallest k such that all planar graphs with girth at least k are acyclic 3-colorable?

The best known lower bound for k is 5 (the second graph of Fig. 6, proposed by Grünbaum, has girth 4 and is not acyclic 3-colorable [10]), while the best known upper bound for k is 7, as proved by Borodin, Kostochka, and Woodall [7].

References

1. Albertson, M.O., Berman, D.: Every planar graph has an acyclic 7-coloring. Israel J. Math. 14, 390–408 (1973)
2. Alon, N., McDiarmid, C., Reed, B.A.: Acyclic coloring of graphs. Random Struct. Algorithms 2(3), 277–288 (1991)
3. Angelini, P., Frati, F.: Acyclically 3-colorable planar graphs. Tech. Report RT-DIA-147-2009, Dept. of Computer Science and Automation, University of Roma Tre. (2009), http://web.dia.uniroma3.it/ricerca/rapporti/rt/2009-147.pdf
4. Appel, K., Haken, W.: Every planar map is 4-colorable. Part I. Discharging. Illinois J. Math. 21(3), 429–490 (1977)
5. Appel, K., Haken, W., Koch, J.: Every planar map is 4-colorable. Part II. Reducibility. Illinois J. Math. 21(3), 491–567 (1977)
6. Borodin, O.V.: On acyclic colourings of planar graphs. Discr. Math. 25, 211–236 (1979)
7. Borodin, O.V., Kostochka, A.V., Woodall, D.R.: Acyclic colourings of planar graphs with large girth. J. London Math. Soc. 60(2), 344–352 (1999)
8. Brooks, R.L.: On coloring the nodes of a network. Proc. Cambridge Philos. Soc. 37, 194–197 (1941)
9. Garey, M.R., Johnson, D.S., Stockmeyer, L.J.: Some simplified NP-complete graph problems. Theor. Comput. Sci. 1(3), 237–267 (1976)
10. Grünbaum, B.: Acyclic colorings of planar graphs. Israel J. Math. 14, 390–408 (1973)
11. Kostochka, A.V.: Acyclic 6-coloring of planar graphs. Metody Diskret. Anal. 28, 40–56 (1976)
12. Kostochka, A.V.: Upper Bounds of Chromatic Functions of Graphs. PhD thesis, University of Novosibirsk, in Russian (1978)
13. Kostochka, A.V., Melnikov, L.S.: To the paper of B. Grünbaum on acyclic colorings. Discrete Math. 14, 403–406 (1976)
14. Mitchem, J.: Every planar graph has an acyclic 8-coloring. Duke Math. J. 41, 177–181 (1974)
15. Robertson, N., Sanders, D.P., Seymour, P.D., Thomas, R.: Efficiently four-coloring planar graphs. In: STOC, pp. 571–575 (1996)
16. Skulrattanakulchai, S.: Acyclic colorings of subcubic graphs. Inf. Proc. Lett. 92(4), 161–167 (2004)
17. Valdes, J., Tarjan, R.E., Lawler, E.L.: The recognition of series parallel digraphs. SIAM J. Comput. 11(2), 298–313 (1982)
18. Wood, D.R.: Acyclic, star and oriented colourings of graph subdivisions. Discr. Math. Theor. Comp. Sc. 7(1), 37–50 (2005)

Reconstruction Algorithm for Permutation Graphs

Masashi Kiyomi, Toshiki Saitoh, and Ryuhei Uehara

School of Information Science, JAIST, Asahidai 1-1, Nomi, Ishikawa 923-1292, Japan
{mkiyomi,toshikis,uehara}@jaist.ac.jp

Abstract. PREIMAGE CONSTRUCTION problem by Kratsch and Hemaspaandra naturally arose from the famous graph reconstruction conjecture. It deals with the algorithmic aspects of the conjecture. We present an $O(n^8)$ time algorithm for PREIMAGE CONSTRUCTION on permutation graphs, where n is the number of graphs in the input. Since each graph of the input has $n-1$ vertices and $O(n^2)$ edges, the input size is $O(n^3)$. There are polynomial time isomorphism algorithms for permutation graphs. However the number of permutation graphs obtained by adding a vertex to a permutation graph is generally exponentially large. Thus exhaustive checking of these graphs does not achieve any polynomial time algorithm. Therefore reducing the number of preimage candidates is the key point.

Keywords: the graph reconstruction conjecture, permutation graphs, polynomial time algorithm.

1 Introduction

The graph reconstruction conjecture proposed by Ulam and Kelly[1] has been studied by many researchers intensively. We call the multi-set $\{G - v \mid v \in V\}$ the *deck* of a graph $G = (V, E)$, where $G - v$ is a graph obtained from G by removing v and incident edges. More precisely the graphs in a deck are vertex-unlabeled. Otherwise the argument below has no mean. A graph G is a *preimage* of a deck of a graph G' if G and G' has the same deck. We also say that a graph G is a preimage of the n graphs when the deck of G exactly consists of them. The graph reconstruction conjecture is that there is at most one preimage of given n graphs ($n \geq 3$). No one has given a positive nor a negative proof of this conjecture, while small graphs are checked positively [15]. Kelly showed the following lemma.

Lemma 1 (Kelley's Lemma [11]). *Let G be any preimage of the given deck, and let H be a graph whose number of vertices is smaller than that of G. Then we can uniquely determine the number of subgraphs in G isomorphic to H from the deck.*

[1] Determining the first person who proposed the graph reconstruction conjecture is difficult, actually. See [9] for the detail.

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 125–135, 2010.

Greenwell and Hemminger extended this lemma to a more general form [8]. We can know the degree sequence of a preimage from these lemmas. Kelly also showed that the conjecture is true on regular graphs, trees, and disconnected graphs. Tutte proved that the dichromatic rank and Tutte polynomials are reconstructible (i.e. looking at the deck, they are uniquely determined) [19]. Bollobás showed that almost all graphs are reconstructible from three well-chosen graphs in its deck [2]. About permutation graphs, Rimscha showed that permutation graphs are recognizable in the sense that looking at the deck of G one can decide whether or not G belongs to permutation graphs [20]. To be precise Rimscha showed in the paper that comparability graphs are recognizable. Even's result [6] directly gives a proof in the case of permutation graphs. Rimscha also showed in the same paper that many subclasses of perfect graphs including perfect graphs themselves are recognizable, and some of subclasses are reconstructible. There are a lot of papers about the conjecture, and many good surveys about this conjecture. See for example [3,9].

There are several kinds of algorithmic problems related to the graph reconstruction conjecture. We consider algorithmic problems proposed by Kratsch and Hemaspaandra [13] described below.

- Given a graph G and a multi-set of graphs D, check whether D is the deck of G (DECK CHECKING).
- Given a multi-set of graphs D, determine whether there is a graph whose deck is D (LEGITIMATE DECK).
- Given a multi-set of graphs D, construct a graph whose deck is D (PREIMAGE CONSTRUCTION).
- Given a multi-set of graphs D, compute the number of (pairwise nonisomorphic) graphs whose decks are D (PREIMAGE COUNTING).

Kratsch and Hemaspaandra showed that these problems are solvable in polynomial time for graphs of bounded degree, partial k-trees for any fixed k, and graphs of bounded genus, in particular for planar graphs [13]. In the same paper they proved many GI related complexity results. Hemaspaandra et al. extended the results [10]. The authors presented a polynomial time PREIMAGE CONSTRUCTION algorithm for interval graphs [12].

We present an $\mathrm{O}(n^8)$ time algorithm for PREIMAGE CONSTRUCTION on permutation graphs. It is easy to see that every graph in the deck of a permutation graph is a permutation graph. We propose PREIMAGE CONSTRUCTION algorithm for a deck consisting of permutation graphs. We state our main theorem below.

Theorem 1. *There is an $O(n^8)$ time PREIMAGE CONSTRUCTION algorithm for a deck D consisting of n permutation graphs.*

2 Preliminaries

2.1 Notations

All the graphs in this paper are simple unless stated otherwise. We denote a complement of graph G by $\overline{G}$.

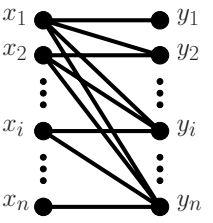


Fig. 1. Graph $H(2n)$

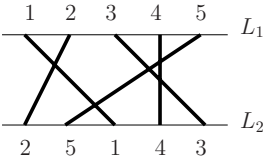


Fig. 2. The permutation diagram of permutation $(2, 5, 1, 4, 3)$

Let $G = (V, E)$ be a graph, and let $V' \subset V$ is a vertex subset of G. We denote by $G[V']$ the graph induced by V' from G.

We denote by $N_G(v)$ the neighbor set of vertex v, and by $N_G[v]$ the closed neighbor set of vertex v in graph G. "Closed" means that $N_G[v]$ contains v itself. Vertices u and v are called *strong twins* if $N_G[u]$ is equal to $N_G[v]$, and *weak twins* if $N_G(u)$ is equal to $N_G(v)$.

We denote by $u(G)$ the graph obtained by adding one universal vertex to the graph G such that the vertex connects to every vertex in G.

Let $G = (V, E)$ be a graph. For a vertex $v \in V$, we denote by $G - v$ the graph obtained by removing v and its incident edges from G. Let S be a set, and $s \in S$. We denote $S \setminus \{s\}$ by $S - s$.

We define graph $H(2n)$. $H(2n)$ is a bipartite graph (X, Y, E) such that $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$, and $\{x_i, y_j\} \in E$ iff $i \leq j$. See Fig. 1.

Let $\pi = (\pi_1, \ldots, \pi_n)$ be a permutation of $1, \ldots, n$. A permutation diagram of π is a set of n line segments $l_1, \ldots, l_n$ that connect two parallel lines L_1, L_2 on Euclidean plane such that end-points of $l_1, \ldots, l_n$ appear in this order on L_1, and appear in the order of $\pi_1, \ldots, \pi_n$ on L_2. A permutation diagram defines a permutation. See Fig. 2. We denote by π^{V} the permutation whose permutation diagram is obtained by reversing that of π vertically, by π^{H} the permutation whose permutation diagram is obtained by reversing that of π horizontally, and by π^{R} the permutation whose permutation diagram is obtained by reversing that of π both vertically and horizontally[2].

[2] $\pi^{\mathrm{V}} = \pi^{-1}$ holds. If we write $\overline{\pi} = (\pi_n, \ldots, \pi_1)$, $\pi^{\mathrm{H}} = \overline{\overline{\pi}^{-1}}^{-1}$, and $\pi^{\mathrm{R}} = \overline{\overline{\pi}^{-1}}$ hold.

2.2 Modular Decomposition

Modular decomposition is a strong tool for developing fast algorithms in many areas. Here we summarize it. For the detail see for example [4,18].

Let $G = (V, E)$ be a graph. The subset $M \subset V$ is a *module* in G, if for all vertices $u, v \in M$ and $w \in V \setminus M$, $\{u, w\} \in E$ if and only if $\{v, w\} \in E$. A module M in G is *trivial* if $M = V$, $M = \emptyset$, or $|M| = 1$. G is called a *prime* (with respect to modular decomposition) if G contains only trivial modules. A module M is *strong* if it does not overlap any other modules in G, i.e.

$$M \cap M' = \emptyset,\ M \subset M',\ \text{or } M' \subset M\ (\text{for } {}^{\forall}M' : \text{module in } G)$$

holds. We call a module that contains at least two vertices a *multi-vertex module*.

A *modular decomposition tree* of a graph G is a tree whose each node corresponds to each strong module of G such that for any two nodes N_1 and N_2 which correspond to modules M_1 and M_2 respectively, N_1 is an ancestor of N_2 if and only if M_1 contains M_2. We sometimes say that strong module M_1 is a parent of strong module M_2, and M_2 is a child of M_1, if the node corresponding to M_1 is the parent of the node corresponding to M_2 in the modular decomposition tree.

A strong multi-vertex module M in graph G whose child modules are disconnected to each other in $G[M]$ is a *parallel module*. A strong multi-vertex module M in graph G whose child modules are disconnect to each other in $\overline{G[M]}$ is a *series module*. Let M' be a strong multi-vertex module. If M' is not a parallel module, and M' is not a series module, then M' is called a *prime module*. A graph induced by a prime module is connected in both G and $\overline{G}$ [18]. We say a strong multi-vertex module M is *minimal* if every child of M is a module of one vertex. Note that every graph of the size more than one has at least one minimal strong multi-vertex module. We introduce a basic lemma.

Lemma 2 (Gallai [7]). *A minimal strong multi-vertex module that is a prime module induces a prime.*

Let $G = (V, E)$ be a prime. We say that G is *critical* if $G - v$ is not a prime for any $v \in V$. It is known that a critical graph $G = (V, E)$ is isomorphic to $H(|V|)$ or to $\overline{H(|V|)}$ [16]. Hence the number of vertices in a critical graph is always even.

2.3 Permutation Graphs

Let π be a permutation of the numbers $1, 2, \ldots, n$. $G(\pi) = (V, E)$ is a graph satisfying that

- $V = \{1, \ldots, n\}$, and
- $\{i, j\} \in E \Leftrightarrow (i - j)(\pi_i^{-1} - \pi_j^{-1}) < 0$.

A graph G is called a *permutation graph* if there exists a permutation π such that G is isomorphic to $G(\pi)$. Equivalently a graph G is a permutation graph if there exists a permutation π such that G is an intersection model of the permutation diagram of π.

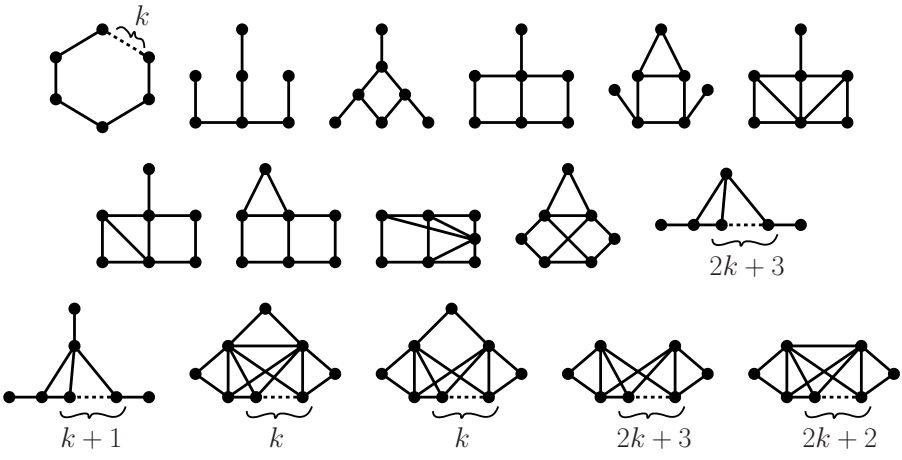


Fig. 3. Forbidden graphs of permutation graphs are these graphs, the complements of them, and odd-holes ($k \geq 0$)

A permutation graph is a comparability graph and is a co-comparability graph [6]. Thus if a graph G is a permutation graph, $\overline{G}$ is also a permutation graph.

For two permutation graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ satisfying $|V_1| = |V_2| = n$, there is an $O(n^2)$ time algorithm that determines if G_1 and G_2 are isomorphic [17].

It is known that a permutation graph G that is a prime with respect to modular decomposition has a unique representation [4,14]. Note that $G(\pi)$, $G(\pi^{\mathrm{V}})$, $G(\pi^{\mathrm{H}})$, and $G(\pi^{\mathrm{R}})$ are isomorphic. Thus the sentence "G has a unique representation" here means that there are at most four permutations whose representing graphs are isomorphic to G.

Gallai characterized comparability graphs with the forbidden subgraphs [7]. Since permutation graphs are equivalent to comparability and co-comparability graphs [6], the characterization of permutation graphs is easily obtained. A graph G is a permutation graph if and only if G is (C_{k+6}, T_2, X_2, X_3, X_{30}, X_{31}, X_{32}, X_{33}, X_{34}, X_{36}, XF_1^{2k+3}, XF_2^{k+1}, XF_3^k, XF_4^k, XF_5^{2k+3}, XF_6^{2k+2}, co-C_{k+6}, co-T_2, co-X_2, co-X_3, co-X_{30}, co-X_{31}, co-X_{32}, co-X_{33}, co-X_{34}, co-X_{36}, co-XF_1^{2k+3}, co-XF_2^{k+1}, co-XF_3^k, co-XF_4^k, co-XF_5^{2k+3}, co-XF_6^{2k+2}, and odd-hole)-free. See Fig. 3.

3 Polynomial Time Reconstruction Algorithm

Our algorithm outputs preimages that are permutation graphs. However it is possible that a non-permutation graph has a deck that consists of permutation graphs, though it is exceptional. Since considering this case all the time in the

main algorithm makes it complex, we attempt to get done with this special case in subsection 3.1.

Then we present a DECK CHECKING algorithm for permutation graphs. Since an $O(n^2)$ time isomorphism algorithm for permutation graphs [17] is known, developing a polynomial time DECK CHECKING algorithm for permutation graphs is not very difficult.

Next we present our main algorithm. Our main algorithm has two parts. One is for a preimage G that has a minimal strong multi-vertex module M such that $G[M]$ is not critical, and the other part is for otherwise. In both the parts, we construct polynomially many candidates of a preimage, and use DECK CHECKING algorithm to check whether each candidate is a preimage. Since we of course do not know the properties of a preimage when we are given a input deck, we execute both these two parts for the input deck.

3.1 Non-permutation Graph Preimage Case

Let D be a deck consisting of n graphs $G_1, G_2, \ldots, G_n$. It is clear that $G_1, G_2, \ldots, G_n$ have the same number of vertices $n-1$, and that the number of vertices in a preimage G is n. Since the number of the forbidden graphs of the size n is $O(1)$, we can check if one of them is a preimage of the input graphs in the polynomial time with DECK CHECKING algorithm which we will describe in the next subsection. The time complexity is $O(n^4)$, since the time complexity of the DECK CHECKING algorithm is $O(n^4)$.

Theorem 2. *If n permutation graphs $G_1, G_2, \ldots, G_n$ have a preimage G that is not a permutation graph, we can reconstruct G from $G_1, G_2, \ldots, G_n$ in $O(n^4)$ time.*

3.2 DECK CHECKING

Given a deck D that consists of permutation graphs, and given a preimage candidate $G = (V, E)$ whose deck consists of permutation graphs, we first prepare the deck $\hat{D}$ of G in $O(|V|(|V| + |E|))$ time. We then add a universal vertex to every graph in D and $\hat{D}$ in order to make each graph connected. Note that for any permutation graph G, $u(G)$ is also a permutation graph. Since the disjoint union of permutation graphs is clearly a permutation graph, we can check if D and $\hat{D}$ are isomorphic in $O((|V|(|V| + 1))^2)$=$O(|V|^4)$ time by applying the isomorphism algorithm for permutation graphs to the disjoint union of graphs in D and the disjoint union of graphs in $\hat{D}$. Now we obtain the theorem below.

Theorem 3. *There is $O(|V|^4)$ time DECK CHECKING algorithm for a deck that consists of permutation graphs, and a preimage candidate $G = (V, E)$ whose deck consists of permutation graphs.*

3.3 Non-critical Case

First we consider the case that a preimage $G = (V, E)$ has a minimal strong multi-vertex module M such that $|M| \geq 3$, and $G[M]$ is not critical. If M is a prime module, since $G[M]$ is a prime due to Lemma 2, $G[M]$ has a vertex v such that $G[M] - v$ is a prime, and hence $M - v$ is a minimal strong multi-vertex module of $G[M] - v$. If M is not a prime module, due to the definition of modular decomposition, $G[M]$ is a complete graph, or $G[M]$ consists of isolated vertices. And thus $G[M]$ also has a vertex v such that $M - v$ is a minimal strong multi-vertex module of $G[M] - v$.

We search for a preimage by adding a vertex v to every minimal strong multi-vertex module M' of every graph in the deck to check if M' is the desired $M - v$. For every candidate, we use the DECK CHECKING algorithm to check if it is a preimage.

If we can specify $N_G(v)$, we can construct a candidate of G. We can easily specify $N_G(v) \setminus M'$, since $M' \cup \{v\}$ should be a module in G, i.e. every vertex in M' and v should seem the same from the vertices in $V \setminus M'$. Thus the remaining task is to specify $N_G(v) \cap M'$.

Due to the definition of a modular decomposition, M' is one of a clique, a collection of isolated vertices, and a module that induces a prime. It is not difficult to construct the candidate of G if M' is a clique, or M' consists of isolated vertices, since we know the degree sequence of G,[3] that is, we know the degree $\deg_G(v)$ of v in G. To be concrete, we have to connect v to $\deg_G(v) - |N_G(v) \setminus M'|$ vertices in M'.

Next we consider the case that $G[M']$ is a prime. A permutation graph that is a prime with respect to modular decomposition has a unique representation [4,14]. Thus there are only $O(n^2)$ ways of connection of v and vertices in M'. Note that the number of permutation diagrams obtained by adding a line segment to a permutation diagram is clearly $O(n^2)$, since there are $O(n)$ choices for the end-point on L_1, and there are $O(n)$ choices for the end-point on L_2. Therefore by checking each of $O(n^2)$ candidates whether it is a preimage with the DECK CHECKING algorithm, we have a polynomial time algorithm. We show in Fig. 4 the whole algorithm for the case that a preimage has a module that does not induce a critical graph.

We now mention the time complexity of the algorithm in Fig. 4. There are n graphs in the deck. Each graph in the deck has $O(n)$ minimal strong multi-vertex modules. We can compute these modules in $O(n + m)$ time [5]. The time complexity of DECK CHECKING is $O(n^4)$. We can compute a permutation diagram of a permutation graph in $O(n+m)$ time. Therefore the time complexity of the algorithm is $O(n \cdot n((n+m)+n^2 \cdot n^4)) = O(n^8)$. Hence we have the theorem below.

Theorem 4. *If a preimage $G = (V, E)$ that is a permutation graph has a minimal strong multi-vertex module M such that $|M| \geq 3$, and $G[M]$ is not critical, we can reconstruct G in $O(n^8)$ time.*

[3] Kelly's lemma directly gives the degree sequence of a preimage. See [11].

```
for each graph G' in the deck {
    for each minimal strong multi-vertex module M' of G' {
        prepare a isolated vertex v;
        connect v to vertices in V \ M' suitably;
        if M' is a clique, or M' are isolated vertices {
            connect v to deg_G(v) − |N_G(v) \ M'| vertices in M';
            do DECK CHECKING;
        } else {
            create a unique permutation diagram of G[M'];
            for each way of adding v
                do DECK CHECKING;
        }
    }
}
```

Fig. 4. The algorithm for the case that a preimage has a module that does not induce a critical graph

3.4 Critical Case

Lastly we consider the case that for every minimal strong multi-vertex module M of a preimage $G = (V, E)$, $G[M]$ is critical, or every minimal strong multi-vertex module has the size two.

Assume that all the minimal strong multi-vertex modules of G have the size two. Since a module of the size two makes twins, the reconstruction of G is easy in this case. Any graph G' in the deck is obtained by removing a vertex that is one of twins from G. Thus G can be reconstructed by copying a vertex in G'. We make weak and strong twins of each vertex of every graph in the deck, and check whether the obtained graph is a preimage by the DECK CHECKING algorithm. This achieve a polynomial time algorithm.

Now we consider the case that some of minimal strong multi-vertex modules in G have the size more than two. Let M be a minimal strong multi-vertex module of G whose size is more than two. Then since $G[M]$ is a critical graph, $G[M]$ is isomorphic to $H(|M|)$ or $\overline{H(|M|)}$. x_1 and x_2 are almost twins in both the $H(|M|)$ and $\overline{H(|M|)}$. In fact $N_{H(|M|)}(x_1)$ and $N_{H(|M|)}(x_2)$ differ only in y_1, and $N_{\overline{H(|M|)}}[x_1]$ and $N_{\overline{H(|M|)}}[x_2]$ also differ only in y_1. We denote by v_1 and v_2 the vertices in M corresponding to x_1 and x_2 such that $|N_{G[M]}(v_1)| = |N_{G[M]}(v_2)| + 1$, or $|N_{G[M]}[v_1]| = |N_{G[M]}[v_2]| + 1$ holds. Since M is a module of G, $N_G(v_1)$ contains exactly one vertex in addition to the vertices in $N_G(v_2)$, or $N_G[v_1]$ contains exactly one vertex in addition to the vertices in $N_G[v_2]$.

Now we consider $G - v_2$. $G - v_2$ must be in the deck. Thus we check for every graph G' in the deck if it is $G - v_2$. If G' is $G - v_2$, we can reconstruct G from G' by copying a vertex in G' and removing an edge. We show the algorithm in Fig. 5.

```
for each graph G' in the deck {
    for each vertex v of G' {
        make weak twin v' of vertex v;
        do DECK CHECKING;
        for each edge e of N_G(v') {
            remove e;
            do DECK CHECKING;
            add e;
        }
        remove v';
        make strong twin v' of vertex v;
        do DECK CHECKING;
        for each edge e of N_G(v') {
            remove e;
            do DECK CHECHEN;
            add e;
        }
        remove v';
    }
}
```

Fig. 5. The algorithm for the case that a preimage has a module that induces a critical graph

We mention the time complexity. There are $O(n)$ graphs in the deck. The number of vertices in each graph is $O(n)$. We have to remove $O(n)$ edges in each iteration. The time complexity of DECK CHECKING is $O(n^4)$. Thus the total time complexity of the algorithm is $O(n \cdot n \cdot n \cdot n^4) = O(n^7)$. Thus we have the theorem below.

Theorem 5. *If every minimal strong multi-vertex module of a graph G induces a critical graph, or if every minimal strong multi-vertex module of a graph G has the size two, we can reconstruct G in $O(n^7)$ time.*

Combining Theorem 2,4, and 5, we have the Theorem 1.

4 Concluding Remarks

Since we can use PREIMAGE CONSTRUCTION algorithms for LEGITIMATE DECK and PREIMAGE COUNTING, we also have the LEGITIMATE DECK and PREIMAGE COUNTING algorithms running in the same time complexity for permutation graphs. These results do not help directly the proofs of the graph reconstruction conjecture on permutation graphs. The conjecture on permutation graphs still remains to be open.

We presented a polynomial time algorithm for PREIMAGE CONSTRUCTION on permutation graphs. PREIMAGE CONSTRUCTION on interval graphs is solvable in polynomial time [12]. Kratsch and Hemaspaandra showed that PREIMAGE CONSTRUCTION on graph class $\mathcal{C}$ is GI-hard if the graph isomorphism is GI-hard on $\mathcal{C}$ [13]. Remaining famous graph classes that we can find in [4] on which graph isomorphism are not GI-hard contain circular-arc graphs and distance-hereditary graphs (of course there are other non-GI-hard classes such as threshold graphs. However we mention here higher classes in the hierarchy of the inclusion relation). It is known that for a distance-hereditary graph G, there is a distance-hereditary graph obtained by removing a degree one vertex from G, or there is a distance-hereditary graph obtained by removing one of twins from G in the deck of G [1]. Hence we can easily develop a polynomial time algorithm for PREIMAGE CONSTRUCTION on distance-hereditary graphs (just add a degree one vertex to every vertex in the deck, and check the resulting graph is a preimage. And, copy every vertex in the deck, and check the resulting graph is a preimage). PREIMAGE CONSTRUCTION on circular-arc graphs may be a challenging problem. Ma and Spinrad showed that a circle graph G has a unique representation if G is a prime with respect to split decomposition [14]. Split decomposition is a generalization of modular decomposition. Therefore it may be possible that PREIMAGE CONSTRUCTION on circle graphs is solvable in polynomial time in a similar way described in this paper. Circle graphs contain permutation graphs and distance-hereditary graphs.

References

1. Bandelt, H., Mulder, H.M.: Distance-hereditary graphs. Journal of Combinatorial Theory, Series B 41, 182–208 (1986)
2. Bollobás, B.: Almost every graph has reconstruction number three. Journal of Graph Theory 14, 1–4 (1990)
3. Bondy, J.A.: A graph reconstructor's manual. In: Surveys in Combinatorics. London Mathematical Society Lecture Note Series, vol. 166, pp. 221–252 (1991)
4. Brandstädt, A., Spinrad, J.P.: Graph classes: a survey. SIAM, Philadelphia (1999)
5. Dahlhaus, E., Gustedt, J., McConnell, R.M.: Efficient and practical algorithms for sequential modular decomposition. Journal of Algorithms 41, 360–387 (2001)
6. Even, S.: Algorithmic Combinatorics. Macmillan, New York (1973)
7. Gallai, T.: Transitiv orientierbare graphen. Acta Mathematica Hungarica 18, 25–66 (1967)
8. Greenwell, D.L., Hemminger, R.L.: Reconstructing the n-connected components of a graph. Aequationes Mathematicae 9, 19–22 (1973)
9. Harary, F.: A survey of the reconstruction conjecture. In: Graphs and Combinatorics. Lecture Notes in Mathematics, vol. 406, pp. 18–28. Springer, Heidelberg (1974)
10. Hemaspaandra, E., Hemaspaandra, L., Radziszowski, S., Tripathi, R.: Complexity results in graph reconstruction. Discrete Applied Mathematics 152, 103–118 (2007)
11. Kelly, P.J.: A congruence theorem for trees. Pacific Journal of Mathematics 7, 961–968 (1957)

12. Kiyomi, M., Saitoh, T., Uehara, R.: Reconstruction of interval graphs. In: Ngo, H.Q. (ed.) COCOON 2009. LNCS, vol. 5609, pp. 106–115. Springer, Heidelberg (2009)
13. Kratsch, D., Hemaspaandra, L.A.: On the complexity of graph reconstruction. Mathematical Systems Theory 27, 257–273 (1994)
14. Ma, T.H., Spinrad, J.P.: An $O(n^2)$ algorithm for undirected split decomposition. Journal on Algorithms 16, 145–160 (1994)
15. McKay, B.D.: Small graphs are reconstructible. Australasian Journal of Combinatorics 15, 123–126 (1997)
16. Schmerl, J.H., Trotter, W.T.: Critically indecomposable partially ordered sets, graphs, tournaments and other binary relational structures. Discrete Mathematics 113, 191–205 (1993)
17. Spinrad, J., Valdes, J.: Recognition and isomorphism of two-dimensional partial orders. In: Díaz, J. (ed.) ICALP 1983. LNCS, vol. 154, pp. 676–686. Springer, Heidelberg (1983)
18. Spinrad, J.P.: Efficient Graph Representations. AMS (2003)
19. Tutte, W.T.: On dichromatic polynomials. Journal of Combinatorial Theory 2, 310–320 (1967)
20. von Rimscha, M.: Reconstructibility and perfect graphs. Discrete Mathematics 47, 283–291 (1983)

Harmonious Coloring on Subclasses of Colinear Graphs*

Kyriaki Ioannidou and Stavros D. Nikolopoulos

Department of Computer Science, University of Ioannina
GR-45110 Ioannina, Greece
{kioannid,stavros}@cs.uoi.gr

Abstract. Given a simple graph G, a harmonious coloring of G is a proper vertex coloring such that each pair of colors appears together on at most one edge. The harmonious chromatic number is the least integer k for which G admits a harmonious coloring with k colors. Extending previous NP-completeness results of the harmonious coloring problem on subclasses of chordal and co-chordal graphs, we prove that the problem remains NP-complete for split undirected path graphs; we also prove that the problem is NP-complete for colinear graphs by showing that split undirected path graphs form a subclass of colinear graphs. Moreover, we provide a polynomial solution for the harmonious coloring problem for the class of split strongly chordal graphs, the interest of which lies on the fact that the problem has been proved to be NP-complete on both split and strongly chordal graphs.

Keywords: Harmonious coloring, colinear coloring, colinear graphs, split graphs, undirected path graphs, strongly chordal graphs, complexity.

1 Introduction

A *harmonious coloring* of a simple graph G is a proper vertex coloring such that each pair of colors appears together on at most one edge, while the *harmonious chromatic number* $h(G)$ is the least integer k for which G admits a harmonious coloring with k colors [5].

Several NP-complete problems on arbitrary graphs admit polynomial solutions when restricted to the classes of strongly chordal graphs and undirected path graphs and, thus, interval graphs (see e.g. [12,18]). However, the harmonious coloring problem, which is NP-hard on arbitrary graphs [22], remains NP-complete even when restricted to graphs that are simultaneously interval and cographs [3]. More specifically, Bodlaender [3] provides a proof that establishes the NP-completeness of the harmonious coloring problem for disconnected interval graphs and cographs. Recently, we extended Bodlaender's results by showing that the problem remains NP-complete for connected interval graphs [1]. Note that the problem of determining the harmonious chromatic number of connected

* This research is co-financed by E.U.-European Social Fund (80%) and the Greek Ministry of Development-GSRT (20%).

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 136–148, 2010.

cographs is trivial, since in such a graph each vertex must receive a distinct color as it is at distance at most 2 from all other vertices [5]. Therefore, the harmonious coloring problem has been proved to be NP-complete on the class of interval graphs and, thus, on the classes of strongly chordal and undirected path graphs.

Additionally, the NP-completeness of the problem has been also proved for the classes of split graphs [1], trees and disconnected bipartite permutation graphs [9,10], connected bipartite permutation graphs [2], and disconnected quasi-threshold graphs [2]. Since the problem of determining the harmonious chromatic number of a connected cograph is trivial, the harmonious coloring problem is polynomially solvable on connected quasi-threshold graphs and threshold graphs.

In this paper we study the complexity status of the harmonious coloring problem on two subclasses of colinear graphs [17,16]. We first show that the harmonious coloring problem is NP-complete on split undirected path graphs and, then, we show that the class of split undirected path graphs forms a subclass of colinear graphs; thus, we obtain the NP-completeness of the harmonious coloring problem on colinear graphs as well. Moreover, we provide a polynomial solution for the harmonious coloring problem on split strongly chordal graphs, the interest of which lies on the fact that the problem is NP-complete on both split graphs and strongly chordal graphs [1,3]. However, the complexity status of the problem for the class of connected linear graphs still remains an open question; note that the harmonious coloring problem is NP-complete on disconnected linear graphs, since it is NP-complete on disconnected quasi-threshold graphs [2] and quasi-threshold graphs form a subclass of linear graphs [17,16].

2 Background Results

In this section we provide some basic graph theory definitions and give some background results on colinear coloring, colinear graphs, and linear graphs. For basic definitions in graph theory refer to [4,15], and for more details on colinear coloring, colinear and linear graphs refer to [17,16].

2.1 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. 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 or $G[A]$. 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. Also, by $\overline{G}$ we denote the complement graph of a graph 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 particular, 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 [15]. A hole is a chordless cycle C_n if $n \geq 5$; the complement of a hole is an antihole. *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 [7] 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 [15].

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 [15]. A chordal graph is an *undirected path graph* if it is the vertex intersection graph of undirected paths in a tree [14,20,21]. A graph is *strongly chordal* if it admits a strong elimination ordering. Strongly chordal graphs were introduced by Farber in [11] and are characterized completely as those chordal graphs which contain no k-sun as an induced subgraph (for the definition of a k-sun see Section 4).

2.2 Colinear Coloring and Colinear Graphs

Motivated by the definition of linear coloring on simplicial complexes associated to graphs, first introduced in the context of algebraic topology [8], we recently introduced the colinear coloring on graphs [17].

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 $i \in \{1, 2, \ldots, k\}$. 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 $\lambda(G)$.*

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 studying the relation between the chromatic number $\chi(G)$ and the colinear chromatic number $\lambda(\overline{G})$. In Figure 1 we depict a colinear coloring of the well

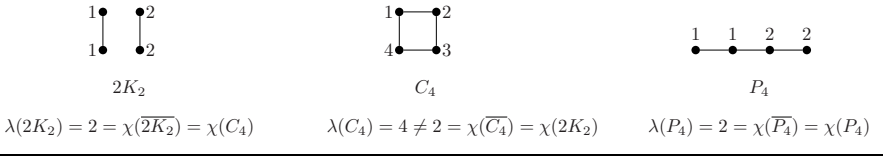


Fig. 1. Illustrating a colinear coloring of the graphs $2K_2$, C_4 and P_4 with the least possible colors

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 $\lambda(\overline{G})$.

In [17] we presented a polynomial time algorithm for colinear coloring which can be applied to any graph G and, also, we proved the following results.

Proposition 1. *([17]) For any graph G, $\lambda(\overline{G}) \geq \chi(G)$.*

Proposition 2. *([17]) 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 $\kappa(u) = \kappa(v)$.*

Motivated by these results and the Perfect Graph Theorem [15], we studied those graphs for which the equality $\chi(G) = \lambda(\overline{G})$ holds for every induced subgraph and characterized known graph classes in terms of the χ-colinear and the α-colinear properties [17]. Moreover, it was interesting to study those graphs which are characterized completely by the χ-colinear or the α-colinear property. The outcome of this study was to conclude that these graphs form two new classes of perfect graphs, which we call colinear and linear graphs, respectively [16].

Definition 3. *A graph G is called colinear if and only if $\chi(G_A) = \lambda(\overline{G}_A)$, $\forall A \subseteq V(G)$. A graph G is called linear if and only if $\alpha(G_A) = \lambda(G_A)$, $\forall A \subseteq V(G)$.*

We also showed inclusion relations between the classes of colinear and linear graphs and other subclasses of co-chordal and chordal graphs [16]. More specifically, the class of colinear graphs is a subclass of co-chordal graphs, a superclass of threshold graphs, and is distinguished from the class of split graphs. Additionally, linear graphs form a subclass of chordal graphs and a superclass of quasi-threshold graphs. We also proved that any P_6-free strongly chordal graph is a linear graph.

The inclusion relations among the classes of colinear graphs, linear graphs, and other subclasses of co-chordal and chordal graphs are depicted in Figure 2. Note that since any P_6-free strongly chordal graph is a linear graph, it follows that split strongly chordal graphs form a subclass of linear graphs. Then, we can easily obtain that any split strongly chordal graph is a colinear graph, since if a graph G is strongly chordal then $\overline{G}$ is also a strongly chordal graph.

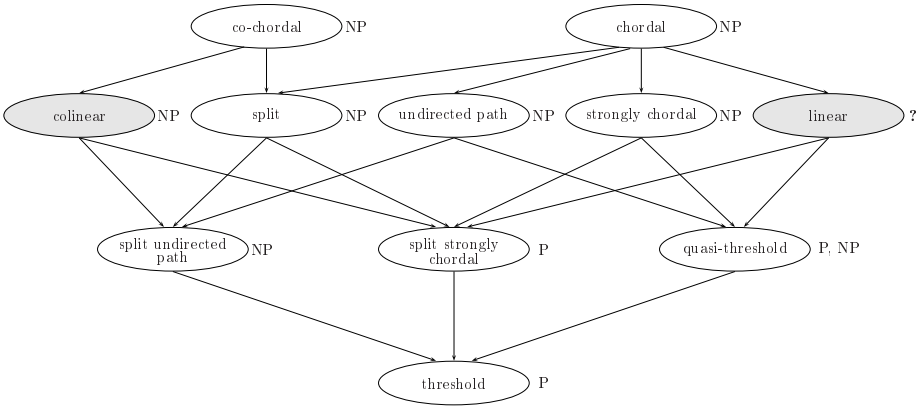


Fig. 2. Illustrating the complexity status of the harmonious coloring problem, and the inclusion relations, for the classes of colinear graphs, linear graphs, and other subclasses of co-chordal and chordal graphs

3 Harmonious Coloring on Colinear Graphs

The formulation of the harmonious coloring problem in [5] is equivalent to the following formulation.

Harmonious Coloring Problem
Instance: Graph G, positive integer $K \leq |V(G)|$.
Question: Is there a positive integer $k \leq K$ and a proper coloring using k colors such that each pair of colors appears together on at most one edge?

In this section we show that the harmonious coloring problem remains NP-complete when restricted to the class of colinear graphs, which is a subclass of co-chordal graphs and a superclass of threshold graphs. The problem is NP-complete on co-chordal graphs, since it is NP-complete on split graphs [1], and it has a polynomial solution on threshold graphs. Therefore, it is interesting to study the complexity of the problem on colinear graphs.

We first show that the problem remains NP-complete even when restricted to graphs which are simultaneously split graphs and undirected path graphs. Then, we show that every split undirected path graph is a colinear graph, thus, proving that the problem is NP-complete on colinear graphs.

The following characterization of undirected path graphs will be used for obtaining our results. Note that, $\mathcal{C}$ denotes the set of all maximal cliques of a graph G; recall that, $C(v)$ denotes the set of all maximal cliques containing v.

Theorem 1. *([14,20]) A graph G is an undirected path graph if and only if there exists a tree T whose set of vertices is $\mathcal{C}$, so that for every vertex $v \in V(G)$, the subgraph $T[C(v)]$ of T induced by the vertex set $C(v)$, is a path in T. Such a tree will be called characteristic tree of G.*

We next show that the harmonious coloring problem is NP-complete for split undirected path graphs by exhibiting a reduction from the chromatic number problem for general graphs, which is known to be NP-complete [13].

Let G be an arbitrary graph with n vertices $v_1, v_2, \ldots, v_n$ and m edges $e_1, e_2, \ldots, e_m$. We construct in polynomial time a split graph $\widehat{G}$, where $V(\widehat{G}) = K + I$, as follows: the independent set I consists of n vertices $\widehat{v}_1, \widehat{v}_2, \ldots, \widehat{v}_n$ which correspond to the vertices $v_1, v_2, \ldots, v_n$ of the graph G and the clique K consists of m vertices $\widehat{u}_1, \widehat{u}_2, \ldots, \widehat{u}_m$ which correspond to the edges $e_1, e_2, \ldots, e_m$ of G. A vertex $\widehat{u}_t \in K$, $1 \leq t \leq m$, is connected to two vertices $\widehat{v}_i, \widehat{v}_j \in I$, $1 \leq i, j \leq n$, if and only if the corresponding vertices v_i and v_j are adjacent in G. Note that, every $\widehat{u}_i \in K$ sees all the vertices of the clique K and two vertices of the independent set I; thus, $|E(\widehat{G})| = \frac{m(m-1)}{2} + 2m$.

Moreover, we claim that the constructed split graph $\widehat{G}$ is also an undirected path graph. Indeed, we prove this by showing that the graph $\widehat{G}$ has a characteristic tree. Let $\mathcal{C}$ be the set of all maximal cliques of $\widehat{G}$. Note that K is a maximal clique for $\widehat{G}$, thus, we have $|\mathcal{C}| = |I| + 1$. Every vertex $\widehat{v}_i \in I$ belongs to exactly one maximal clique, i.e., $|C(\widehat{v}_i)| = 1$. Additionally, every vertex $\widehat{u}_i \in K$ belongs to exactly three maximal cliques, one of which is maximal clique K, i.e., $|C(\widehat{u}_i)| = |N[\widehat{u}_i]| - |K| + 1 = 3$.

Consider now a tree T with vertex set $\mathcal{C}$, such that the maximal clique K is connected by an edge to every maximal clique $C(\widehat{v}_i)$ for every $\widehat{v}_i \in I$, i.e., T is a star. We now show that T is a characteristic tree for $\widehat{G}$. Indeed, for every vertex $\widehat{v}_i \in I$, the subgraph $T[C(\widehat{v}_i)]$ induced by $C(\widehat{v}_i)$ is a path on one vertex, and also for every vertex $\widehat{u}_i \in K$, the subgraph $T[C(\widehat{u}_i)]$ is a path on three vertices. Therefore, the constructed graph $\widehat{G}$ has a characteristic tree and, thus, from Theorem 1 it follows that $\widehat{G}$ is a split undirected path graph.

We claim that the graph G has a chromatic number $\chi(G)$ if and only if the split undirected path graph $\widehat{G}$ has a harmonious chromatic number $h(\widehat{G}) = \chi(G) + m$. Note that the same arguments are used in [1] for proving the NP-completeness of the problem for split graphs.

Let $c_i \in \{1, \ldots, \chi(G)\}$ be the color assigned to the vertex $v_i \in G$, $1 \leq i \leq n$, in a minimum coloring of G. We assign the color c_i to the vertex $\widehat{v}_i$ of the set I and a distinct color from the set $\{\chi(G)+1, \ldots, \chi(G)+m\}$ to each vertex of the clique K. Since two adjacent vertices of G receive a different color, the neighbors of each $\widehat{u}_i \in K$ belonging to the independent set have distinct colors. Moreover, every vertex $\widehat{v}_i \in I$ sees $|N_G(v_i)|$ vertices of the clique K, where $N_G(v_i)$ is the neighborhood of the vertex v_i in G. Thus, every pair of colors appears in at most one edge. In addition, the number of colors assigned to the set I is equal to $\chi(G)$ and the number of colors assigned to the clique is equal to m. This results to a harmonious coloring of $\widehat{G}$ using $\chi(G) + m$ colors, which is minimum since the vertices of the set I cannot receive a color assigned to a vertex of the clique K.

Conversely, a harmonious coloring of $\widehat{G}$ using $h(\widehat{G}) = \chi(G) + m$ colors assigns m colors to the vertices of the clique K and $\chi(G)$ colors to the vertices of the set I. Note that, $\chi(G)$ is the minimum number of colors so that vertices $\widehat{v}_i, \widehat{v}_j$ having

a neighbor in common are assigned different colors. Since v_i, v_j are adjacent in G, it follows that we have a minimum coloring of G using $\chi(G)$ colors.

Thus, we have proved the following result.

Theorem 2. *The harmonious coloring problem is NP-complete for split undirected path graphs.*

Next, we show the following result.

Theorem 3. *Any split undirected path graph is a colinear graph.*

Proof. Let G be a split undirected path graph. Assume that G is not a colinear graph. Then, from Definition 3 there exists an induced subgraph G_A of G such that $\lambda(\overline{G}_A) \neq \chi(G_A)$; thus, due to Proposition 1, $\lambda(\overline{G}_A) > \chi(G_A)$.

From Theorem 1, we obtain that split undirected path graphs are hereditary, that is, every induced subgraph G_A of G is a split undirected path graph. Let $V(G_A) = K + I$ be a partition of the vertex set of G_A into a maximal clique K and an independent set I. Also, from Theorem 1 we have that G_A has a characteristic tree T with vertex set $\mathcal{C}$, where $\mathcal{C}$ is the set of all maximal cliques of G_A, such that for every vertex $v \in V(G_A)$, the subgraph $T[C(v)]$ of T induced by the vertex set $C(v)$ is a path in T.

In particular, since G_A is a split graph, for every vertex $v \in I$, the subgraph $T[C(v)]$ of T induced by the vertex set $C(v)$ is a vertex in T that corresponds to the unique maximal clique of G_A that v belongs to; we will denote this clique by C_v, i.e., $C_v = N_{G_A}[v]$ and $C(v) = \{C_v\}$ for every vertex $v \in I$. Also, for every vertex $v \in K$, the path $(C_u, \ldots, C_x, K, C_y, \ldots, C_z)$ of T induced by the vertex set $C(v)$, always passes from the vertex K; equivalently, for every vertex $v \in K$, the subgraph of T induced by the vertex set $C(v)$, corresponds to the vertex K and to at most two vertex disjoint paths $(C_y, \ldots, C_z)$ and $(C_x, \ldots, C_u)$ where C_y and C_x are adjacent to K in T. Moreover, observe that for any path $(K, C_{v_1}, C_{v_2}, \ldots, C_{v_k})$ of the characteristic tree T of G_A, we have $C_{v_1} \setminus \{v_1\} \supseteq C_{v_2} \setminus \{v_2\} \supseteq \ldots \supseteq C_{v_k} \setminus \{v_k\}$, since $C_{v_i} \setminus \{v_i\} = N_{G_A}(v_i) \subset K$, where $v_i \in I$ for every i, $1 \leq i \leq k$.

Let $\kappa : V(\overline{G}_A) \rightarrow \{1, 2, \ldots, \lambda(\overline{G}_A)\}$ be a colinear coloring of $\overline{G}_A$. In order to see how a colinear coloring can be assigned to the vertices of $\overline{G}_A$ we refer to the colinear coloring algorithm presented in [17]. In particular, the algorithm first constructs the directed acyclic graph (DAG) $D_{\overline{G}_A}$ associated to the graph $\overline{G}_A$ and, then, finds a minimum path cover of the transitive DAG $D_{\overline{G}_A}$. The size of the minimum path cover of $D_{\overline{G}_A}$ equals the colinear chromatic number $\lambda(\overline{G}_A)$. Also, the algorithm assigns a colinear coloring κ to the vertices of $\overline{G}_A$ such that a set of vertices are assigned the same color in κ if and only if they belong to the same path of the minimum path cover of $D_{\overline{G}_A}$. Moreover, the DAG $D_{\overline{G}_A}$ associated to the graph $\overline{G}_A$ is constructed as follows: $V(D_{\overline{G}_A}) = V(\overline{G}_A)$ and $E(D_{\overline{G}_A}) = \{\overrightarrow{xy} \ : \ x, y \in V(D_{\overline{G}_A})$ and $N_{\overline{G}_A}[x] \subseteq N_{\overline{G}_A}[y]\}$, where $\overrightarrow{xy}$ is a directed edge from x to y. Note that $D_{\overline{G}_A}$ is a transitive DAG [17]. For simplicity, throughout the proof we will denote the DAG $D_{\overline{G}_A}$ associated to the graph $\overline{G}_A$ by D.

The following observations will be useful in the rest of this proof. Two vertices $u, v \in V(D)$ are not adjacent in D if and only if neither $N_{\overline{G}_A}[v] \subseteq N_{\overline{G}_A}[u]$ nor $N_{\overline{G}_A}[v] \supseteq N_{\overline{G}_A}[u]$; we call two sets with this property incompatible. In $\overline{G}_A$ the vertices of I form a clique, therefore, for two vertices $u, v \in I$, u and v are not adjacent in D if and only if the sets $N_{\overline{G}_A}[u] \cap K$ and $N_{\overline{G}_A}[v] \cap K$ are incompatible. Note that, for any two vertices u, v of G_A, $N_{\overline{G}_A}[u] \subseteq N_{\overline{G}_A}[v]$ if and only if $N_{G_A}(u) \supseteq N_{G_A}(v)$. Additionally, for every vertex $u \in I$, we have $N_{G_A}(u) \subset K$.

Having assumed that $\lambda(\overline{G}_A) > \chi(G_A) = |K|$, there exists a minimum path cover of D with size $\lambda(\overline{G}_A) \geq |K| + 1$. The size of a minimum path cover of D equals the cardinality of a maximum independent set I_D of D [15]; thus, $|I_D| \geq |K| + 1$. Moreover, the independent set I_D corresponds to a collection C of mutually incompatible sets $N_{\overline{G}_A}[v]$, for all $v \in I_D$, that is, $C = \{N_{\overline{G}_A}[v] : v \in I_D\}$. Thus, $|C| \geq |K| + 1$ and the sets of C contain at most $|K|$ vertices of K. Also, recall that for any two vertices $u, v \in V(D)$ such that $u \in K$ and $v \in I$, if $uv \in E(\overline{G}_A)$ then $N_{\overline{G}_A}[u] \subset N_{\overline{G}_A}[v]$; thus, for any two vertices $u, v \in V(D)$ such that $u \in K$ and $v \in I$, u and v are adjacent in $\overline{G}_A$ if and only if u and v are adjacent in D.

Assume that $K \subset I_D$. Then, no vertex $v \in I$ can belong to I_D since every vertex of I is adjacent to at least one vertex of K in $\overline{G}_A$ and, thus, in D, due to our assumption that K is a maximal clique of G_A. Thus, not every vertex of K can belong to I_D, since $|I_D| \geq |K| + 1$. Assume that a vertex $u \in K$ belongs to I_D. Then, no vertex $v \in I$ that is adjacent to u in D and, thus, in $\overline{G}_A$, belongs to I_D; equivalently, $u \notin N_{\overline{G}_A}[v]$, for every vertex $v \in I_D$. Therefore, if we delete the vertex $u \in K$ from the set I_D, we obtain an independent set $I'_D = I_D \setminus \{u\}$ and a collection $C' = C \setminus \{N_{\overline{G}_A}[u]\}$ of at least $|K|$ mutually incompatible sets, which contain at most $|K| - 1$ vertices of K. Using the same arguments, if we delete every vertex of K from the independent set I_D, we obtain an independent set I''_D, such that $I''_D \subseteq I$ and $|I''_D| \geq k + 1$ (where $k \leq |K|$), which corresponds to a collection C'' of at least $k + 1$ mutually incompatible sets $N_{\overline{G}_A}[v]$, $v \in I$, which contain at most k vertices of K.

A collection C'' of at least $k + 1$ mutually incompatible sets $N_{\overline{G}_A}[v]$, $v \in I$, corresponds to a collection F of at least $k+1$ mutually incompatible sets $N_{G_A}(v)$, $v \in I$. Since, for every vertex $v \in I$ we have $N_{G_A}(v) = C_v \setminus \{v\}$, it follows that a collection F of at least $k + 1$ mutually incompatible sets $N_{G_A}(v)$, $v \in I$, corresponds to a collection of at least $k+1$ maximal cliques C_v of G_A, $v \in I$, each of which must belong to a different path $(K, C_{v_1}, C_{v_2}, \ldots, C_{v_k})$ of a characteristic tree T of G_A. However, every vertex $z \in K$ belongs to at most two such paths, therefore, every vertex $z \in K$ belongs to at most two sets of the collection F. Thus, every vertex $z \in K$ belongs to at least $|C''| - 2$ sets of the collection C''.

Summarizing, we have a collection C'' of at least $k+1$ mutually incompatible sets $N_{\overline{G}_A}[v]$, $v \in I$, which contain at most k vertices of K and, also, every vertex $z \in K$ belongs to at least $|C''| - 2$ sets of the collection C''. Recall that for two vertices $u, v \in I$, the sets $N_{\overline{G}_A}[u]$ and $N_{\overline{G}_A}[v]$ are incompatible if and only if the sets $N_{\overline{G}_A}[u] \cap K$ and $N_{\overline{G}_A}[v] \cap K$ are incompatible. Therefore, we have a

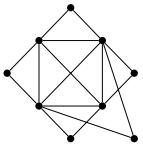


Fig. 3. A split graph G which is not a colinear graph, since $\chi(G) = 4$ and $\lambda(\overline{G}) = 5$. Also, G is not an undirected path graph

collection of at least $k+1$ mutually incompatible vertex sets on k vertices. It is easy to see that it is impossible to find a collection of at least $k+1$ mutually incompatible sets on k vertices, if every vertex belongs to at least k sets of the collection. This is a contradiction to our assumptions. Therefore, G is a colinear graph. □

Note that, not any split graph is a colinear graph (for example see Fig. 3). From Theorems 2 and 3, we obtain the following result.

Corollary 1. *The harmonious coloring problem is NP-complete on the class of colinear graphs.*

4 Harmonious Coloring on Split Strongly Chordal Graphs

In this section we show that the harmonious coloring problem admits a polynomial solution on the class of split strongly chordal graphs. Strongly chordal graphs form a known subclass of chordal graphs [4,11] and were first introduced by Farber [11]. A graph is strongly chordal iff it admits a strong elimination ordering; a vertex ordering $\sigma = (v_1, v_2, \ldots, v_n)$ is a strong elimination ordering of a graph G iff σ is a perfect elimination ordering and also has the property that for each i, j, k and ℓ, if $i < j$, $k < \ell$, $v_k, v_\ell \in N[v_i]$, and $v_k \in N[v_j]$, then $v_\ell \in N[v_j]$ [6,11].

Let us now give the definitions of a k-sun and an incomplete k-sun. An *incomplete k-sun* S_k ($k \geq 3$) is a chordal graph on $2k$ vertices whose vertex set can be partitioned into two sets, $U = \{u_1, u_2, \ldots, u_k\}$ and $W = \{w_1, w_2, \ldots, w_k\}$, so that W is an independent set, and w_i is adjacent to u_j if and only if $i = j$ or $i = j + 1 \pmod{k}$; the graph S_k ($k \geq 3$) is a *k-sun* if U is a complete graph.

The following characterization of strongly chordal graphs was proved by Farber [11] and turns up to be useful in obtaining a polynomial solution for the harmonious coloring problem on split strongly chordal graphs.

Proposition 3. *(Farber [11]) A chordal graph G is strongly chordal if and only if it contains no induced k-sun.*

Note also that a bipartite graph G is chordal bipartite if and only if the split graph obtained from G by making one of its two color classes complete is strongly chordal [19].

Next, we present a polynomial solution for the harmonious coloring problem on split strongly chordal graphs. Before describing our algorithm, we first construct a graph H_G from a split graph G, which we call *neighborhood intersection graph of* G, and we use it in the proposed algorithm.

The neighborhood intersection graph H_G of a split graph G. Let G be a split graph, and let $V(G) = K + I$ be a partition of its vertex set, where K induces a clique in G and I induces an independent set. We first compute the open neighborhood $N_G(v)$ of each vertex $v \in I$ and, then, we construct the following graph H_G, which depicts all intersection relations among the vertices' open neighborhoods: $V(H_G) = I$ and $E(H_G) = \{xy \ : \ x, y \in I \text{ and } N_G(x) \cap N_G(y) \neq \emptyset\}$. It is easy to see that the resulting graph H_G is unique up to isomorphism.

The following result is important for proving the correctness of our algorithm.

Lemma 1. *The neighborhood intersection graph H_G of a split strongly chordal graph G is a chordal graph.*

Proof. Let G be a split strongly chordal graph and let H_G be the neighborhood intersection graph of G. We will show that H_G is a chordal graph, i.e., that H_G is a C_k-free graph, for every $k \geq 4$. Since G is a split graph, there exists a partition of its vertex set $V(G) = K + I$, where K induces a clique and I induces an independent set in G. By the construction of H_G, there is a one to one correspondence between the vertices of $V(H_G)$ and the vertices of $V(G) \cap I$.

Assume that H_G is not a chordal graph and let $C_k = (v_1, v_2, \ldots, v_k)$ be a chordless cycle of H_G on k vertices, $k \geq 4$; thus, $v_i v_j \in E(H_G)$ if and only if $j = i + 1 \pmod k$. Therefore, we have that $N_G(v_i) \cap N_G(v_j) \neq \emptyset$ if and only if $j = i + 1 \pmod k$ or, equivalently, there exists at least one vertex $w_i \in K$ in G such that $w_i \in N_G(v_i) \cap N_G(v_j)$ if and only if $j = i + 1 \pmod k$; note that, the set $W = \{w_1, w_2, \ldots, w_k\}$ consists of distinct vertices, since C_k is a chordless cycle. Thus, $U = \{v_1, v_2, \ldots, v_k\}$ induces an independent set in G, $W = \{w_1, w_2, \ldots, w_k\}$ induces a clique in G, and w_i is adjacent to v_j if and only if $j = i$ or $j = i + 1 \pmod k$. Therefore, the subgraph of G induced by the vertices $U \cup W$ is a k-sun, $k \geq 4$. It follows that G is a split graph and, thus, it is a chordal graph, which contains a k-sun as an induced subgraph. This is a contradiction to our assumption that G is a strongly chordal graph due to Proposition 3. Therefore, we conclude that H_G is a chordal graph. □

The algorithm for a harmonious coloring of a split strongly chordal graph. The proposed algorithm computes a harmonious coloring and the harmonious chromatic number $h(G)$ of a split strongly chordal graph G, and works as follows:

Input: a split strongly chordal graph G, and a partition of its vertex set $V(G) = K + I$, where I induces an independent set in G and K induces a clique.

(i) `construct` the neighborhood intersection graph H_G of G.
(ii) `compute` a minimum proper vertex coloring $\kappa : V(H_G) \rightarrow \{1, 2, \ldots, \chi(H_G)\}$, and the chromatic number $\chi(H_G)$, of the chordal graph H_G (see e.g. [15]).
(iii) `compute` a coloring $\kappa' : V(G) \rightarrow \{1, 2, \ldots, h(G)\}$ of G, by assigning $\kappa'(v) = \kappa(v)$ to each vertex $v \in I$, and a distinct color $\kappa'(v)$ from the set $\{\chi(H_G) + 1, \chi(H_G) + 2, \ldots, \chi(H_G) + |K|\}$ to each vertex $v \in K$.
(iv) `return` the value $\kappa'(v)$ for each vertex $v \in V(G)$ and the size $\chi(H_G) + |K|$ of the number of different colors used in κ'; the coloring κ' is a harmonious coloring of G, and $\chi(H_G) + |K|$ equals the harmonious chromatic number $h(G)$ of G.

Correctness of the algorithm. Let G be a split strongly chordal graph, and let $V(G) = K+I$ be a partition of its vertex set, where I induces an independent set in G and K induces a clique. Let H_G be the neighborhood intersection graph of G.

We claim that the split strongly chordal graph G has a harmonious chromatic number $h(G) = |K|+r$, where r equals the chromatic number $\chi(H_G)$ of the graph H_G. Indeed, a harmonious coloring of G, using $h(G) = |K| + r$ colors, assigns a distinct color from the set $\{1, 2, \ldots, |K|\}$ to each vertex of the clique K, and also assigns r colors to the vertices of the set I. Note that, r is the minimum number of colors so that vertices $v_i, v_j \in I$ having a neighbor in common are assigned different colors. Since v_i, v_j are adjacent in H_G, it follows that r is the minimum number of colors for which a proper vertex coloring of H_G exists, i.e., $r = \chi(H_G)$.

Therefore, the split strongly chordal graph G has a harmonious chromatic number $h(G) = |K| + \chi(H_G)$, where $\chi(H_G)$ is the chromatic number of the neighborhood intersection graph H_G of G. Additionally, it is easy to see that the coloring κ' computed by the algorithm is a harmonious coloring of G using $h(G) = |K| + \chi(H_G)$ colors.

Complexity of the algorithm. Let G be a split strongly chordal graph on n vertices and m edges. Let $V(G) = K + I$ be a partition of its vertex set into a clique K and an independent set I, and let H_G be the neighborhood intersection graph of G. Step (i) of the algorithm, which includes the construction of the graph H_G, takes $O(n^3)$ time. Step (ii) computes a minimum proper vertex coloring of H_G; since from Lemma 1, H_G is a chordal graph, the problem is solvable in $O(n+m')$ time (see e.g. [15]), where $m' = |E(H_G)| = O(n^2)$. Finally, both Steps (iii) and (iv) can be executed in $O(n)$ time. Therefore, the complexity of the algorithm is $O(n^3)$ time.

Therefore, the following result holds.

Theorem 4. *The harmonious coloring problem has a polynomial solution on split strongly chordal graphs.*

5 Concluding Remarks

In this paper we show that the harmonious coloring problem is NP-complete on the classes of split undirected path graphs and colinear graphs. We also present a polynomial solution for the same problem on the class of split strongly chordal graphs. The interest of this result lies on the fact that the harmonious coloring problem is NP-complete on split graphs and strongly chordal graphs. In addition, polynomial solutions for the problem are only known for the classes of threshold graphs and connected quasi-threshold graphs; note that, the harmonious coloring problem is NP-complete on disconnected quasi-threshold graphs. Since linear graphs form a superclass of both split strongly chordal graphs and quasi-threshold graphs, the harmonious coloring problem is NP-complete on disconnected linear graphs, while it still remains open on connected linear graphs.

References

1. Asdre, K., Ioannidou, K., Nikolopoulos, S.D.: The harmonious coloring problem is NP-complete for interval and permutation graphs. Discrete Applied Math. 155, 2377–2382 (2007)
2. Asdre, K., Nikolopoulos, S.D.: NP-completeness results for some problems on subclasses of bipartite and chordal graphs. Theoret. Comput. Sci. 381, 248–259 (2007)
3. Bodlaender, H.L.: Achromatic number is NP-complete for cographs and interval graphs. Inform. Proc. Lett. 31, 135–138 (1989)
4. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. SIAM, Philadelphia (1999)
5. Cairnie, N., Edwards, K.: Some results on the achromatic number. J. Graph Theory 26, 129–136 (1997)
6. Chang, G.J.: Labeling algorithms for domination problems in sun-free chordal graphs. Discrete Applied Math. 22, 21–34 (1988)
7. Chvátal, V., Hammer, P.L.: Aggregation of inequalities for integer programming. Ann. Discrete Math. I, 145–162 (1977)
8. Civan, Y., Yalçin, E.: Linear colorings of simplicial complexes and collapsing. J. Comb. Theory A 114, 1315–1331 (2007)
9. Edwards, K.J.: The harmonious chromatic number and the achromatic number. In: Baily, R.A. (ed.) Surveys in Combinatorics, pp. 13–47. Cambridge University Press, Cambridge (1997)
10. Edwards, K.J., McDiarmid, C.: The complexity of harmonious coloring for trees. Discrete Applied Math. 57, 133–144 (1995)
11. Farber, M.: Characterizations of strongly chordal graphs. Discrete Math. 43, 173–189 (1983)
12. Farber, M.: Domination, independent domination, and duality in strongly chordal graphs. Discrete Applied Math. 7, 115–130 (1984)
13. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. W.H. Freeman, San Francisco (1979)
14. Gavril, F.: A recognition algorithm for the intersection graph of paths of a tree. Discrete Math. 23, 377–388 (1978)

15. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, New York (1980); 2nd edn. Annals of Discrete Mathematics, vol. 57. Elsevier, Amsterdam (2004)
16. Ioannidou, K., Nikolopoulos, S.D.: Colinear coloring and colinear graphs. Technical Report TR-2007-06, Department of Computer Science, University of Ioannina (2007)
17. Ioannidou, K., Nikolopoulos, S.D.: Colinear coloring on graphs. In: Das, S., Uehara, R. (eds.) WALCOM 2009. LNCS, vol. 5431, pp. 117–128. Springer, Heidelberg (2009)
18. Kratsch, D.: Finding dominating cliques efficiently, in strongly chordal graphs and undirected path graphs. Discrete Math. 86, 225–238 (1990)
19. McKee, T.A., McMorris, F.R.: Topics in Intersection Graph Theory. Society for Industrial and Applied Mathematics, Philadelphia (1999)
20. Monma, C.L., Wei, V.K.: Intersection graphs of paths of a tree. J. Comb. Theory B 41, 141–181 (1986)
21. Schäffer, A.A.: A faster algorithm to recognize undirected path graphs. Discrete Applied Math. 43, 261–295 (1993)
22. Yannakakis, M., Gavril, F.: Edge dominating sets in graphs. SIAM J. Applied Math. 38, 364–372 (1980)

Comparing RNA Structures with Biologically Relevant Operations Cannot Be Done without Strong Combinatorial Restrictions

Guillaume Blin[1], Sylvie Hamel[2], and Stéphane Vialette[1]

[1] Université Paris-Est, LIGM - UMR CNRS 8049, France
{gblin,vialette}@univ-mlv.fr
[2] DIRO - Université de Montréal - QC - Canada
hamelsyl@iro.umontreal.ca

Abstract. Arc-annotated sequences are useful for representing structural information of RNAs and have been extensively used for comparing RNA structures in both terms of sequence and structural similarities. Among the many paradigms referring to arc-annotated sequences and RNA structures comparison (see [2] for more details), the most important one is the general edit distance. The problem of computing an edit distance between two non-crossing arc-annotated sequences was introduced in [5]. The introduced model uses edit operations that involve either single letters or pairs of letters (never considered separately) and is solvable in polynomial-time [12].

To account for other possible RNA structural evolutionary events, new edit operations, allowing to consider either silmutaneously or separately letters of a pair were introduced in [9]; unfortunately at the cost of computational tractability. It has been proved that comparing two RNA secondary structures using a full set of biologically relevant edit operations is **NP**-complete. Nevertheless, in [8], the authors have used a strong combinatorial restriction in order to compare two RNA stem-loops with a full set of biologically relevant edit operations; which have allowed them to design a polynomial-time and space algorithm for comparing general secondary RNA structures.

In this paper we will prove theoretically that comparing two RNA structures using a full set of biologically relevant edit operations cannot be done without strong combinatorial restrictions.

1 Introduction

In computational biology, comparison of RNA molecules has recently attracted a lot of interest due to the rapidly increasing amount of known RNA molecules, especially non-coding RNAs. Very often, *arc-annotated sequences*, originally introduced in [5], are used to represent RNA structures. An arc-annotated sequence is a sequence over a given alphabet together with additional structural information specified by arcs connecting pairs of positions. The arcs determine the way the sequence folds into a three-dimensional space.

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 149–160, 2010.

The problem of computing an edit distance between two arc-annotated sequences was introduced in [5] with a model that used only three edit operations (deletion, insertion and substitution) either on single letters (letters in the sequence with no incident arc) or pairs of letters (letters connected by an arc). In this model, the two letters of an arc are never considered separately, and hence the problem of computing the edit distance between two arc-annotated sequences becomes equivalent (when no pair of arcs are crossing) to the tree edit distance problem, that can be solved in polynomial-time [12].

To account for other possible RNA structural evolutionary events, new edit operations, such as creation, deletion or modification of arcs between pairs of letters, were introduced in [9] at the cost of computational tractability. Indeed, it has been shown in [4] that in case of non-crossing arcs, the problem of computing the edit distance between two arc-annotated sequences under this model is **NP**-hard. Playing the game of applying constraints either on the legal edit operations or on the allowed alignments, several papers have shed new light on the borderline between tractability and intractability [8,2]. Of particular importance, in [8], the authors introduced the notion of *conservative edit distance and mapping* between two RNA stem-loops in order to design a polynomial-time algorithm for comparing general secondary RNA structures using the full set of biological edit operations introduced in [9]. This algorithm is based on a decomposition in stem-loop-like substructures that are pairwised compared and used to compare complete RNA secondary structures. As mentionned in [8], whereas in the very restrictive case of conservative distance and mapping, the computation of the general edit distance is polynomial-time solvable, it is not known if the general, *i.e.*, not conservative, edit distance between two stem-loops can be also computed in polynomial-time.

In this paper, we will show that this strong combinatorial restriction was necessary for the problem to become polynomial since it is **NP**-hard in the general case. Despite the fact that this result may be considered as purely theoretical, it proves that comparing two RNA structures using a full set of biologically relevant edit operations cannot be done without strong combinatorial restrictions.

2 Preliminaries

Given a finite alphabet Σ, an arc-annotated sequence is formally defined by a pair (S, P), where S is a string of Σ^* and P is a set of arcs connecting pairs of letters of S. In reference to RNA structures, letters are called *bases*. Bases with no incident arc are called *single bases*. In an arc-annotated sequence, two arcs (i_1, j_1) and (i_2, j_2) are crossing, if $i_1 < i_2 < j_1 < j_2$ or $i_2 < i_1 < j_2 < j_1$. An arc (i_1, j_1) is *embedded* into another arc (i_2, j_2) if $i_2 < i_1 < j_1 < j_2$. Evans [5] (see [8] for extensions) introduced five different levels of arc structure: Unlimited – no restriction at all; Crossing – there is no base incident to more than one arc; Nested – there is no base incident to more than one arc and no two arcs are crossing; Stem – there is no base incident to more than one arc and given any two arcs, one is embedded into the other; Plain – there is no arc. There is

an obvious inclusion relation between those levels: PLAIN ⊂ STEM ⊂ NESTED ⊂ CROSSING ⊂ UNLIMITED. An arc-annotated sequence (S_1, P_1) is said to *occur* in another arc-annotated sequence (S_2, P_2) if one can obtain the former from the latter by repeatedly deleting bases (deleting a base that is incident to an arc results in the deletion of the arc).

Among the many paradigms referring to arc-annotated sequences (see [2] for more details) we focus in this article on the LONGEST ARC-PRESERVING COMMON SUBSEQUENCE (LAPCS for short) [5,10,11] and the general edit distance (EDIT for short) [9,3]. Indeed, as shown in [2], those two paradigms are quite related since the LAPCS problem is a special case of EDIT when considering the complete set of edit operations defined in [9]. Therefore, the hardness results for LAPCS stands for EDIT.

Formally, the LONGEST ARC-PRESERVING COMMON SUBSEQUENCE problem is defined as follows: given two arc-annotated sequences (S_1, P_1) and (S_2, P_2), find the longest – in terms of sequence length – common arc-annotated subsequence that occurs in both (S_1, P_1) and (S_2, P_2). It has been shown in [9] that the LAPCS problem is **NP**-hard even for NESTED structures, *i.e.*, LAPCS(NESTED, NESTED). Still focussing on NESTED structures, Alber *et al.* [1] proved that the LAPCS(NESTED, NESTED) problem is solvable in $O(3^k\,|\Sigma|^k\,kn)$ time, where n is the maximum length of the two sequences and k is the length of the common subsequence searched for.

Here, we focus on the last open problem concerning LAPCS and EDIT over stem-loops by showing, with a unique proof, their hardness. More precisely, we prove that LAPCS(STEM, STEM) - which may be considered as a very restricted problem and thus not interesting - is **NP**-hard in order to infer the **NP**-hardness of EDIT(STEM, STEM) - which is for sure, according to [8], an interesting problem that can be used in a very simple way to compare complete RNA secondary structures. This results also prove that in any future work on comparing RNA structures with a full set of edit operations it will be necessary to introduce strong combinatorial restrictions in order to get an exact polynomial-time algorithm since even with the simpliest model, the general edit distance problem is still **NP**-complete. This is why this result is of particular interest since it closes the only remaining hope in RNA structures comparison.

3 Comparing RNA Stem-Loops Is NP-Complete

In this section, we prove that LAPCS over stem-loops (LAPCS(STEM, STEM)) is **NP**-complete (in Theorem 1); therefore answering an open question of [8]. This last result induces the **NP**-hardness of EDIT over stem-loops.

Theorem 1. LAPCS(STEM, STEM) *is* **NP***-complete.*

Corollary 1. *Comparing RNA structures with a full set of biologically relevant edit operations cannot be done without introducing strong combinatorial restrictions.*

In the following, we consider the decision version of the problem which corresponds to deciding if there exists an arc-preserving common subsequence of length greater or equal to a given parameter k'.

It is easy to see that the LAPCS problem is in **NP**. In order to prove its **NP**-hardness, we define a reduction from the **NP**-complete 3SAT problem [6] which is defined as follows: Given a collection $C_q = \{c_1, c_2, \ldots, c_q\}$ of q clauses, where each clause consists of a set of 3 literals (representing the disjunction of those literals) over a finite set of n boolean variables $V_n = \{x_1, x_2, \ldots, x_n\}$, is there an assignment of truth values to each variable of V_n s.t. at least one of the literals in each clause is true?

Let (C_q, V_n) be any instance of the 3SAT problem s.t. $C_q = \{c_1, c_2, \ldots, c_q\}$ and $V_n = \{x_1, x_2, \ldots, x_n\}$. For convenience, let L_i^j denote the j^{th} literal of the i^{th} clause (i.e. c_i) of C_q. In the following, given a sequence S over an alphabet Σ, let $\chi(i, c, S)$ denote the i^{th} occurrence of the letter c in S.

We build two arc-annotated sequences (S_1, P_1) and (S_2, P_2) as follows. An illustration of a full example is given in Figures 1 and 2, where $n = 4$ and $q = 3$. For readability reasons, the arc-annotated sequences resulting from the construction have been split into several parts and a schematic overview of the overall placement of each part is provided.

Let $S_1 = C_q^1 W_q C_{q-1}^1 ... C_2^1 W_2 C_1^1 W_1 S_M^1 V_1 P_1^1 V_2\ P_2^1 ... P_{q-1}^1 V_q P_q^1$ and $S_2 = C_q^2 W_q\ C_{q-1}^2 ... C_2^2 W_2 C_1^2 W_1\ S_M^2 V_1 P_1^2 V_2 P_2^2 ... P_{q-1}^2 V_q P_q^2$ s.t. for all $1 \leq i \leq q, 1 \leq k \leq n$,

- $C_i^1 = R_i^3 Q_i R_i^2 Q_i X_1^1 X_2^1 ...\ X_n^1 Q_i R_i^2 Q_i R_i^1$ with $X_k^1 = x_k s_j \overline{x_k}$ if $x_k = L_i^j$ or $\overline{x_k} = L_i^j$; $X_k^1 = x_k \overline{x_k}$ otherwise;
- $P_i^1 = Q_{q+i} Q_{q+i} R_{q+i}^3 X_n^1 ... X_{\frac{n}{2}+1}^1 R_{q+i}^2 X_{\frac{n}{2}}^1 ... X_1^1\ R_{q+i}^1 Q_{q+i} Q_{q+i}$ s.t. $X_k^1 = \overline{x_k} x_k$;
- $C_i^2 = X_1^2 ... X_n^2 R_i^3 Q_i X_1^2 ...\ X_{\frac{n}{2}}^2 R_i^2 X_{\frac{n}{2}+1}^2 ... X_1^2 Q_i\ R_i^1 X_1^2 ... X_n^2$ s.t. $\forall 1 \leq j \leq 3$, $\chi(j, X_k^2, C_i^2) = x_k \overline{x_k} s_j$ (resp. $s_j x_k \overline{x_k}$) if $x_k = L_i^j$ (resp. $\overline{x_k} = L_i^j$); $\chi(j, X_k^2, C_i^2) = x_k \overline{x_k}$ otherwise;
- $P_i^2 = X_n^2 ... X_1^2 R_{q+i}^1 Q_{q+i} X_n^2 ... X_{\frac{n}{2}+1}^2 R_{q+i}^2 X_{\frac{n}{2}}^2 ... X_1^2\ Q_{q+i} R_{q+i}^3 X_n^2 ... X_1^2$ with $X_k^2 = \overline{x_k} x_k$.

Moreover, let $S_M^1 = x_1 \overline{x_1} x_2 \overline{x_2} ... x_n \overline{x_n}$ and $S_M^2 = \overline{x_1} x_1 \overline{x_2} x_2 ... \overline{x_n} x_n$. Notice that, by construction, there is only one occurrence of each $\{s_1, s_2, s_3\}$ in C_i^2.

For all $1 \leq i \leq q$, let Q_i (resp. Q_{q+i}) be a segment of $n+1$ symbols y_i (resp. y_{q+i}). Moreover, for all $1 \leq i \leq q$, let W_i (resp. V_i) be a segment of $20(max\{q, n\}^2)$ symbols w_i (resp. v_i). Let us now define P_1 and P_2.

For all $1 \leq i \leq q-1$, (1) add an arc in P_1 between $\chi(1, x_k, C_i^1)$ (resp. $\chi(1, \overline{x_k}, C_i^1)$) and $\chi(1, x_k, P_{i+1}^1)$ (resp. $\chi(1, \overline{x_k}, P_{i+1}^1)$), $\forall 1 \leq k \leq n$ (see Figure 1.d and 2.b); (2) add an arc in P_2 between $\chi(j, x_k, C_i^2)$ (resp. $\chi(j, \overline{x_k}, C_i^2)$) and $\chi((4-j), x_k, P_i^2)$ (resp. $\chi((4-j), \overline{x_k}, P_i^2)$), $\forall 1 \leq k \leq n$ (see Figure 1.c, 2.a and 2.c); (3) add an arc in P_2 between $\chi(1, R_i^j, C_i^2)$ and $\chi(1, R_{q+i}^j, P_i^2)$, $\forall 1 \leq j \leq 3$ (see Figure 1.c, 2.a and 2.c).

Clearly, this construction can be achieved in polynomial-time, and yields to sequences (S_1, P_1) and (S_2, P_2) that are both of type STEM. We now give an intuitive description of the different elements of this construction.

Each clause $c_i \in C_q$ is represented by a pair (C_i^1, C_i^2) of sequences. The sequence C_i^2 is composed of three subsequences representing a selection mechanism of one of the three literals of c_i. The pair (S_M^1, S_M^2) of sequences is a control mechanism that will guarantee that a variable x_k cannot be true and false simultaneously. Finally, for each clause $c_i \in C_q$, the pair (P_i^1, P_i^2) of sequences is a propagation mechanism which aim is to propagate the selection of the assignment (i.e. true or false) of any literal x_k all over C_q. Notice that all the previous intuitive notions will be detailed and clarified afterwards.

In the rest of this article, we will refer to any such construction as a *snail-construction*. In order to complete the instance of the LAPCS(STEM, STEM) problem, we define the parameter $k' = 40q(max\{q,n\}^2) + 6qn + 8q + n$ which corresponds to the desired length of the solution. In the following, let (S_1, P_1) and (S_2, P_2) denote the arc-annotated sequences obtained by a snail-construction. We will denote S_d the set of symbols deleted in a solution of LAPCS problem on (S_1, P_1) and (S_2, P_2) (i.e. the symbols that do not belong to the common subsequence).

We start the proof that the reduction from 3SAT to LAPCS(STEM, STEM) is correct by giving some properties about any optimal solution.

Lemma 1. *In any optimal solution of* LAPCS *problem on* (S_1, P_1) *and* (S_2, P_2)*, at least one symbol incident to any arc would be deleted. Moreover, all the symbols of* V_i *and* W_i*, for* $1 \leq i \leq q$*, will not be deleted.*

Proof. By contradiction, let us suppose that there exist at least one arc s.t. the two symbols incident to this last are not deleted in a solution of LAPCS problem on (S_1, P_1) and (S_2, P_2). Then, by construction, it induces that at least one complete sequence V_j or W_j, for a given $1 \leq j \leq q$, has been deleted. Since they have the same length, we will consider w.l.o.g. afterwards that V_i has been deleted. Therefore, since S_1 is, by construction, smaller than S_2 the length of this optimal solution is at most $|S_1| - |V_j| = \sum_{i=1}^{q}(|C_i^1| + |P_i^1| + |V_i| + |W_i|) + |S_M^1| - |V_j| = \sum_{i=1}^{q}((6n+11)+(6n+7)+(20(max\{q,n\}^2))+(20(max\{q,n\}^2)))+ 2n - (20(max\{q,n\}^2)) = q[12n + 18 + 40(max\{q,n\}^2)] + 2n - (20(max\{q,n\}^2))$. Then, in order for this solution to be optimal, one should have $q[12n + 18 + 40(max\{q,n\}^2)] + 2n - (20(max\{q,n\}^2)) \geq 40q(max\{q,n\}^2) + 6qn + 8q + n$. This can be reduced to $6qn + 10q - 20(max\{q,n\}^2) + n \geq 0$. But, one can easily check that for any $n \geq 3$ (which is always the case in 3SAT instances), this is not true; a contradiction.

Lemma 2. *Any optimal solution of* LAPCS *problem on* (S_1, P_1) *and* (S_2, P_2) *is of length* $40q(max\{q,n\}^2) + 6qn + 8q + n$.

Proof. By construction, in S_1 there is (1) $\forall 1 \leq i \leq n$, $2q+1$ occurrences of x_i (resp. $\overline{x_i}$); (2) $\forall 1 \leq i \leq q$, 4 occurrences of Q_i (resp. Q_{q+i}); (3) $\forall 1 \leq i \leq q$, 1 occurrence of each $\{R_i^1, R_{q+i}^2, R_i^3, R_{q+i}^1, R_{q+i}^3, W_i, V_i, s_1, s_2, s_3\}$; (4) $\forall 1 \leq i \leq q$, 2 occurrences of R_i^2.

Whereas, in S_2, there is (1) $\forall 1 \leq i \leq n$, $6q+1$ occurrences of x_i (resp. $\overline{x_i}$); (2) $\forall 1 \leq i \leq q$, 2 occurrences of Q_i (resp. Q_{q+i}); (3) $\forall 1 \leq i \leq q$, 1 occurrence of each $\{R_i^1, R_i^2, R_i^3, R_{q+i}^1, R_{q+i}^2, R_{q+i}^3, W_i, V_i, s_1, s_2, s_3\}$.

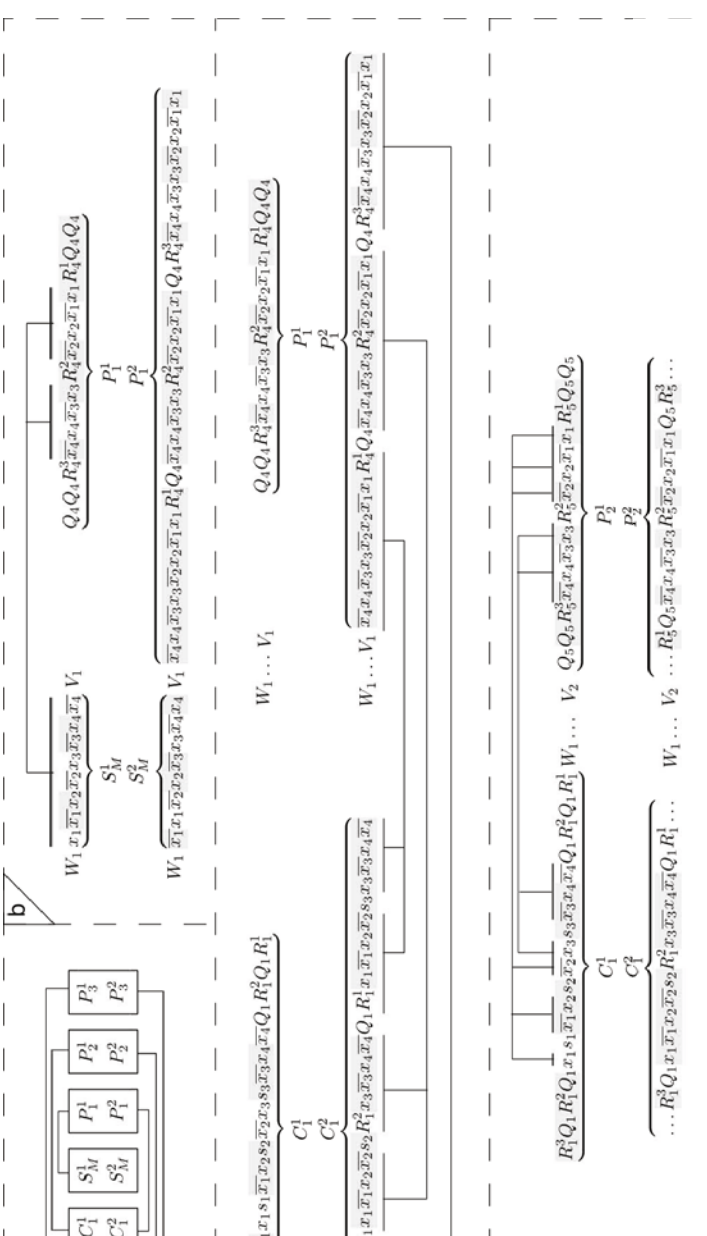


Fig. 1. Considering $C_q = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_4) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$. For readability, all the arcs have not been drawn, consecutive arcs are representing by a unique arc with lines for endpoints. Symbols over a grey background may be deleted to obtain an optimal LAPCS. a) A schematic view of the overall arrangement of the components of the two a.a. sequences. b) Description of S_M^1, S_M^2, P_1^1, P_1^2 and the corresponding arcs in P_1. c) Description of C_1^1, C_1^2, P_1^1, P_1^2 and the corresponding arcs in P_2. d) Description of C_1^1, C_1^2, P_2^1, P_2^2 and the corresponding arcs in P_1.

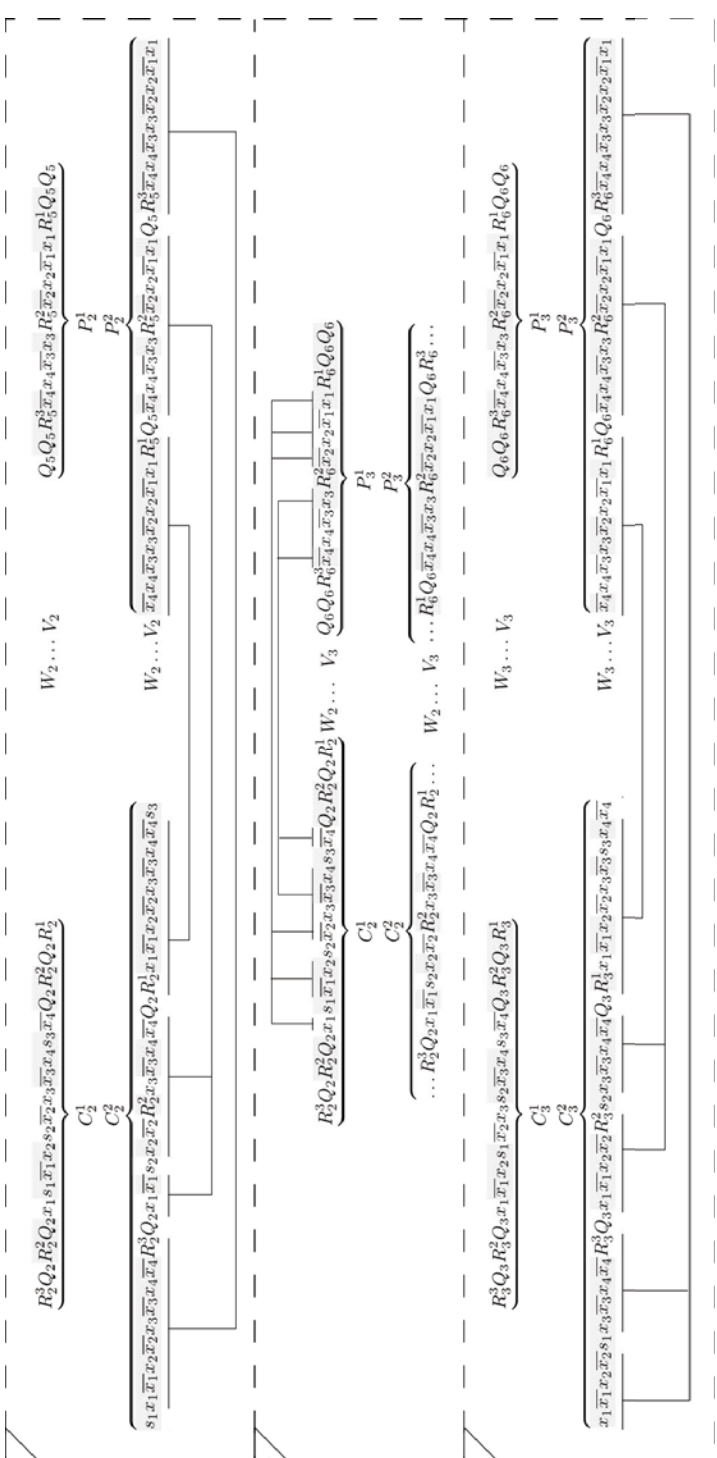


Fig. 2. Considering $C_q = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_4) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$. For readability all the arcs have not been drawn, consecutive arcs are representing by a unique arc with lines for endpoints. Symbols over a grey background may be deleted to obtain an optimal LAPCS. a) Description of C_2^1, C_2^2, P_2^1, P_2^2 and the corresponding arcs in P_2. c) Description of C_2^1, C_2^2, P_3^1, P_3^2 and the corresponding arcs in P_1. d) Description of C_3^1, C_3^2, P_3^1, P_3^2 and the corresponding arcs in P_2.

Therefore, in any optimal solution there may be only (1) $\forall 1 \leq i \leq n$, $2q+1$ occurrences of x_i (resp. $\overline{x_i}$); (2) $\forall 1 \leq i \leq q$, 2 occurrences of Q_i (resp. Q_{q+i}); (3) $\forall 1 \leq i \leq q$, 1 occurrence of each $\{R_i^1, R_i^2, R_i^3, R_{q+i}^1, R_{q+i}^2, R_{q+i}^3, W_i, V_i, s_1, s_2, s_3\}$.

More precisely, by Lemma 1, and since, by construction, there is an arc in P_2 between $\chi(1, R_i^j, C_i^2)$ and $\chi(1, R_{q+i}^j, P_i^2)$, $\forall j \in \{1,2,3\}$, in any optimal solution, $\forall 1 \leq i \leq q$, only half of the $\{R_i^1, R_i^2, R_i^3, R_{q+i}^1, R_{q+i}^2, R_{q+i}^3\}$ may be conserved.

Moreover, any x_i (resp. $\overline{x_i}$) of S_1 except in C_q^1, is linked by an arc to another x_i (resp. $\overline{x_i}$), therefore by Lemma 1, in any optimal solution, $\forall 1 \leq i \leq q-1$, only half of the occurrences of x_i (resp. $\overline{x_i}$) may be conserved.

Finally, in any optimal solution, only half of the occurrences of $\{x_i, \overline{x_i}\}$ and one over $\{s_1, s_2, s_3\}$ in C_q^1 and S_M^1 may be conserved. Indeed, by construction, if this is not the case in C_q^1 (resp. S_M^1), it implies that at least one complete sequence Q_q (resp. V_1 or W_1) is totally deleted – which is not optimal since it is of length $n+1$ (resp. $20(max\{q,n\}^2)$).

On the whole, the maximal total length of any solution is thus equal to $40q(max\{q,n\}^2) + 6qn + 8q + n$. Moreover, this solution is composed of (1) $\forall 1 \leq i \leq n$, $2q+1$ occurrences of either x_i or $\overline{x_i}$, (2) $\forall 1 \leq i \leq q$, 2 occurrences of Q_i and Q_{q+i}, (3) $\forall 1 \leq i \leq q$, 1 occurrence of each $\{W_i, V_i\}$ and either s_1, s_2 or s_3 and (4) $\forall 1 \leq i \leq q$, $R_i^{j_1}, R_i^{j_2}, R_{q+i}^{j_3}$ s.t. $\{j_1, j_2, j_3\} = \{1,2,3\}$.

Lemma 3. *In any optimal solution of* LAPCS *problem on* (S_1, P_1) *and* (S_2, P_2), *if* $\chi(1, x_k, S_M^1)$ *(resp.* $\chi(1, \overline{x_k}, S_M^1)$*) for a given* $1 \leq k \leq n$ *is deleted then,* $\forall 1 \leq j \leq q$, $\chi(1, x_k, C_j^1)$ *(resp.* $\chi(1, \overline{x_k}, C_j^1)$*) is deleted.*

Proof. By construction, $\forall 1 \leq k \leq n$ only one of $\{x_k, \overline{x_k}\}$ may be conserved between S_M^1 and S_M^2 since $\chi(1, x_k, S_M^1) < \chi(1, \overline{x_k}, S_M^1)$ whereas $\chi(1, \overline{x_k}, S_M^2) < \chi(1, x_k, S_M^2)$. By Lemma 1, at least one symbol incident to any arc is deleted. Therefore, $\forall 1 \leq k \leq n$ only one of $\{x_k, \overline{x_k}\}$ may be conserved between C_1^1 and C_1^2.

Let us suppose that for a given $1 \leq k \leq n$, $\chi(1, \overline{x_k}, S_M^1)$ is deleted. According to the proof of Lemma 2, in any optimal solution, $\forall 1 \leq k \leq n$ exactly one of $\{x_k, \overline{x_k}\}$ has to be deleted. Then $\chi(1, x_k, P_1^1)$ is deleted whereas $\chi(1, \overline{x_k}, P_1^1)$ is conserved.

By construction, in P_1^2, since according to the proof of Lemma 2, both occurrences of Q_{q+1} and $R_1^{j_1}, R_1^{j_2}, R_{q+1}^{j_3}$ s.t. $\{j_1, j_2, j_3\} = \{1,2,3\}$ have to be conserved, either (1) $\{R_1^1, R_1^2, R_{q+1}^3\}$, (2) $\{R_1^1, R_1^3, R_{q+1}^2\}$ or (3) $\{R_1^2, R_1^3, R_{q+1}^1\}$ are conserved.

Let us first consider that $\{R_1^1, R_1^2, R_{q+1}^3\}$ are conserved. Then one can check that the only solution is to conserve $\chi(2, R_1^2, C_1^1)$ since otherwise at least half of the x_k's would not be conserved. Consequently, the only solution is to conserve, $\forall 1 \leq k \leq n$, the first (resp. last) occurrence of any x_k or $\overline{x_k}$ in C_1^2 (resp. P_1^2) – i.e. the occurrences appearing before $\chi(1, Q_1, C_1^2)$ (resp. after $\chi(2, Q_{q+1}, P_1^2)$). Since by construction, there is an arc between $\chi(1, x_k, C_1^2)$ (resp. $\chi(1, \overline{x_k}, C_1^2)$) and $\chi(3, x_k, P_1^2)$ (resp. $\chi(3, \overline{x_k}, P_1^2)$), in order for $\chi(1, \overline{x_k}, P_1^1)$ to be conserved, one has to conserved $\chi(3, \overline{x_k}, P_1^2)$. Thus, by Lemma 1, $\chi(1, \overline{x_k}, C_1^2)$ has to be deleted and, according to the proof of Lemma 2, $\chi(1, x_k, C_1^2)$ has to be conserved.

Let us now consider that $\{R_1^1, R_1^3, R_{q+1}^2\}$ are conserved. By a similar reasoning, one can check that the only solution is to conserve, $\forall 1 \leq k \leq n$, the second occurrence of any x_k or $\overline{x_k}$ in C_1^2 (resp. P_1^2) – i.e. the occurrences appearing between $\chi(1, Q_1, C_1^2)$ and $\chi(2, Q_1, C_1^2)$ (resp. $\chi(1, Q_{q+1}, P_1^2)$ and $\chi(2, Q_{q+1}, P_1^2)$). Since by construction, there is an arc between $\chi(2, x_k, C_1^2)$ (resp. $\chi(2, \overline{x_k}, C_1^2)$) and $\chi(2, x_k, P_1^2)$ (resp. $\chi(2, \overline{x_k}, P_1^2)$), in order to $\chi(1, \overline{x_k}, P_1^1)$ to be conserved, one has to conserved $\chi(2, \overline{x_k}, P_1^2)$. Thus, by Lemma 1, $\chi(2, \overline{x_k}, C_1^2)$ has to be deleted and, according to the proof of Lemma 2, $\chi(2, x_k, C_1^2)$ has to be conserved.

Finally, let us consider that $\{R_1^2, R_1^3, R_{q+1}^1\}$ are conserved. Once again, by a similar reasoning, one can check that the only solution is to conserve $\chi(1, R_1^2, C_1^1)$ since otherwise at least half of the x_k's would not be conserved. Consequently, the only solution is to conserve, $\forall 1 \leq k \leq n$, the last (resp. first) occurrence of any x_k or $\overline{x_k}$ in C_1^2 (resp. P_1^2) – i.e. the occurrences appearing after $\chi(2, Q_1, C_1^2)$ (resp. before $\chi(1, Q_{q+1}, P_1^2)$). Since by construction, there is an arc between $\chi(3, x_k, C_1^2)$ (resp. $\chi(3, \overline{x_k}, C_1^2)$) and $\chi(1, x_k, P_1^2)$ (resp. $\chi(1, \overline{x_k}, P_1^2)$), in order to $\chi(1, \overline{x_k}, P_1^1)$ to be conserved, one has to conserved $\chi(1, \overline{x_k}, P_1^2)$. Thus, by Lemma 1, $\chi(3, \overline{x_k}, C_1^2)$ has to be deleted and, according to the proof of Lemma 2, $\chi(3, x_k, C_1^2)$ has to be conserved.

Therefore, in the three cases, if for a given $1 \leq k \leq n$, $\chi(1, x_k, S_M^1)$ is conserved then so does $\chi(1, x_k, C_1^1)$. It is easy to see that, by a similar reasoning, if for a given $1 \leq k \leq n$, $\chi(1, \overline{x_k}, S_M^1)$ is conserved then so does $\chi(1, \overline{x_k}, C_1^1)$.

With a similar reasoning, by reccurence, since, $\forall 1 \leq i \leq q, 1 \leq k \leq n$, there is an arc in P_1 between $\chi(1, x_k, C_i^1)$ (resp. $\chi(1, \overline{x_k}, C_i^1)$) and $\chi(1, x_k, P_{i+1}^1)$ (resp. $\chi(1, \overline{x_k}, P_{i+1}^1)$), if $\chi(1, x_k, C_i^1)$ is conserved then $\chi(1, x_k, P_{i+1}^1)$ is deleted. And therefore, with similar arguments, $\chi(1, x_k, C_{i+1}^1)$ is conserved. Once more, it is easy to see that this result still holds if $\chi(1, \overline{x_k}, C_i^1)$ is conserved.

Theorem 2. *Given an instance of the problem 3SAT with n variables and q clauses, there exists a satisfying truth assignment iff the* Lapcs *of (S_1, P_1) and (S_2, P_2) is of length $k' = 40q(max\{q, n\}^2) + 6qn + 8q + n$.*

Proof. ($\Rightarrow$) An optimal solution for $C_q = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_4) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$ – i.e. $x_1 = x_3 = true$ and $x_2 = x_4 = false$ – is illustrated in Figures 1 and 2 where any symbol over a grey background have to be deleted. Suppose we have a solution of 3SAT, that is an assignment of each variable of V_n satisfying C_q. Let us first list all the symbols to delete in S_1.

For all $1 \leq k \leq n$, if $x_k = false$ then delete, $\forall 1 \leq j \leq q$, $\{\chi(1, x_k, C_j^1), \chi(1, \overline{x_k}, P_j^1)\}$ and $\chi(1, x_k, S_M^1)$; otherwise delete, $\forall 1 \leq j \leq q$, $\{\chi(1, \overline{x_k}, C_j^1), \chi(1, x_k, P_j^1)\}$ and $\chi(1, \overline{x_k}, S_M^1)$.

For each L_i^j satisfying c_i with the biggest index j with $1 \leq i \leq q$,

if (1) $j = 1$ then delete $\{\chi(1, R_i^3, C_i^1), \chi(1, Q_i, C_i^1), \chi(1, R_i^2, C_i^1), \chi(2, Q_i, C_i^1), \chi(1, s_2, C_i^1), \chi(1, s_3, C_i^1), \chi(1, R_{q+i}^2, P_i^1), \chi(1, R_{q+i}^1, P_i^1), \chi(3, Q_{q+i}, P_i^1), \chi(4, Q_{q+i}, P_i^1)\}$ (cf Figure 1.a);

if (2) $j = 2$ then delete $\{\chi(1, R_i^2, C_i^1), \chi(2, Q_i, C_i^1), \chi(1, s_1, C_i^1), \chi(1, s_3, C_i^1), \chi(3, Q_i, C_i^1), \chi(2, R_i^2, C_i^1), \chi(2, Q_{q+i}, P_i^1), \chi(1, R_{q+i}^3, P_i^1), \chi(1, R_{q+i}^1, P_i^1), \chi(3, Q_{q+i}, P_i^1)\}$ (cf Figure 2.a);

if (3) $j = 3$ then delete $\{\chi(1, s_1, C_i^1), \chi(1, s_2, C_i^1), \chi(3, Q_i, C_i^1), \chi(2, R_i^2, C_i^1), \chi(4, Q_i, C_i^1), \chi(1, R_i^1, C_i^1), \chi(1, Q_{q+i}, P_i^1), \chi(2, Q_{q+i}, P_i^1), \chi(1, R_{q+i}^3, P_i^1), \chi(1, R_{q+i}^2, P_i^1)\}$ (cf Figure 2.c);

Let us now list all the symbols in S_2 to be deleted.

For all $1 \leq k \leq n$, if $x_k = false$ then delete $\chi(1, x_k, S_M^2)$; otherwise delete $\chi(1, \overline{x_k}, S_M^2)$.

For each L_i^j satisfying c_i with the biggest index j with $1 \leq i \leq q$,

if (1) $j = 1$ then delete $\forall 1 \leq k \leq n$ $\{\chi(1, R_i^3, C_i^2), \chi(1, s_2, C_i^2), \chi(2, x_k, C_i^2), \chi(2, \overline{x_k}, C_i^2), \chi(1, s_3, C_i^2), \chi(3, x_k, C_i^2), \chi(3, \overline{x_k}, C_i^2), \chi(1, x_k, P_i^2), \chi(1, \overline{x_k}, P_i^2), \chi(1, R_{q+i}^1, P_i^2), \chi(1, R_{q+i}^2, P_i^2), \chi(2, x_k, P_i^2), \chi(2, \overline{x_k}, P_i^2)\}$. Moreover, if $x_k{=}false$ with $1 \leq k \leq n$ then delete, $\{\chi(1, x_k, C_i^2), \chi(3, \overline{x_k}, P_i^2)\}$; otherwise delete $\{\chi(1, \overline{x_k}, C_i^2), \chi(3, x_k, P_i^2)\}$ (cf Figure 1.a);

if (2) $j = 2$ then delete $\forall 1 \leq k \leq n$ $\{\chi(1, R_i^2, C_i^2), \chi(1, s_1, C_i^2), \chi(1, x_k, C_i^2), \chi(1, \overline{x_k}, C_i^2), \chi(1, s_3, C_i^2), \chi(3, x_k, C_i^2), \chi(3, \overline{x_k}, C_i^2), \chi(1, x_k, P_i^2), \chi(1, \overline{x_k}, P_i^2), \chi(1, R_{q+i}^1, P_i^2), \chi(1, R_{q+i}^3, P_i^2), \chi(3, x_k, P_i^2), \chi(3, \overline{x_k}, P_i^2)\}$. Moreover, if $x_k{=}false$ with $1 \leq k \leq n$ then delete, $\{\chi(2, x_k, C_i^2), \chi(2, \overline{x_k}, P_i^2)\}$; otherwise delete $\{\chi(2, \overline{x_k}, C_i^2), \chi(2, x_k, P_i^2)\}$ (cf Figure 2.a);

if (3) $j = 3$ then delete $\forall 1 \leq k \leq n$ $\{\chi(1, R_i^1, C_i^2), \chi(1, s_1, C_i^2), \chi(1, x_k, C_i^2), \chi(1, \overline{x_k}, C_i^2), \chi(1, s_2, C_i^2), \chi(2, x_k, C_i^2), \chi(2, \overline{x_k}, C_i^2), \chi(2, x_k, P_i^2), \chi(2, \overline{x_k}, P_i^2), \chi(1, R_{q+i}^2, P_i^2), \chi(1, R_{q+i}^3, P_i^2), \chi(3, x_k, P_i^2), \chi(3, \overline{x_k}, P_i^2)\}$. Moreover, if $x_k{=}false$ with $1 \leq k \leq n$ then delete, $\{\chi(3, x_k, C_i^2), \chi(1, \overline{x_k}, P_i^2)\}$; otherwise delete $\{\chi(3, \overline{x_k}, C_i^2), \chi(1, x_k, P_i^2)\}$ (cf Figure 2.c);

By construction, the natural order of the symbols of S_1 and S_2 allows the corresponding set of undeleted symbols to be conserved in a common arc-preserving common subsequence between (S_1, P_1) and (S_2, P_2). Let us now prove that the length of this last is k'. One can easily check that this solution is composed of $\forall 1 \leq k \leq n$, (1) $2q + 1$ occurrences of either x_k or $\overline{x_k}$, (2) $\forall 1 \leq i \leq q$, 2 occurrences of Q_i and Q_{q+i}, (3) $\forall 1 \leq i \leq q$, 1 occurrence of each $\{W_i, V_i\}$ and either s_1, s_2 or s_3 and (4) $\forall 1 \leq i \leq q$, $R_i^{j_1}, R_i^{j_2}, R_{q+i}^{j_3}$ s.t. $\{j_1, j_2, j_3\} = \{1, 2, 3\}$. Thus, the length of the solution is $40q(max\{q, n\}^2) + 6qn + 8q + n$.

($\Leftarrow$) Suppose we have an optimal solution – i.e. a set of symbols S_d to delete – for Lapcs of (S_1, P_1) and (S_2, P_2). Let us define the truth assignment of V_n s.t., $\forall 1 \leq i \leq q$, if $\chi(1, s_j, C_i^1) \notin S_d$ then L_i^j is true. Let us prove that it is a solution of 3SAT.

By construction, if $L_i^j = x_k$ (resp. $\overline{x_k}$) then in C_i^1, s_j appears between x_k and $\overline{x_k}$ whereas in C_j^2 it appears after $\overline{x_k}$ (resp. before x_k). Thus, if $\chi(1, s_j, C_i^1)$ is not deleted then $\overline{x_k}$ (resp. x_k) in C_i^1 is deleted if $L_i^j = x_k$ (resp. $\overline{x_k}$). Consequently, according to the proof of Lemma 3, if $\chi(1, s_j, C_i^1)$ is not deleted then $\overline{x_k}$ (resp. x_k) in all $C_{i'}^1$, with $1 \leq i' \leq q$ is deleted if $L_i^j = x_k$ (resp. $\overline{x_k}$). Therefore, we can ensure that one cannot obtain L_i^j and $L_{i'}^{j'}$ being true whereas $L_i^j = \overline{L_{i'}^{j'}}$ (that is a

variable cannot be simultaneously true and false). By Lemma 2, we can ensure that for any $1 \leq i \leq q$ exactly one of $\{s_1, s_2, s_3\}$ is conserved in C_i^1. Therefore, for any clause c_i at least one of its literal is set to true. This ensures that our solution is a solution of 3SAT.

Lemma 4. LAPCS(STEM, STEM) *is solvable in* $O(2^{k-1}\,|\Sigma|^k\,kn)$

Proof. We use a straightforward brute-force algorithm for arc-annotated sequences [1]: (i) generate all possible sequences of length k with all possible STEM arc annotations, and (ii) for each of these arc-annotated candidate sequences, check whether or not it occurs as a pattern in both S_1 and S_2.

At the heart of this approach is the fact that it can be decided in $O(n\,k)$ time whether or not this sequence occurs as an arc-preserving common subsequence [7]. It is easily seen that the above algorithm reduces to $O(2^{k-1}\,|\Sigma|^k\,km)$ time for LAPCS(STEM, STEM). Indeed, there exist $|\Sigma|^k$ sequences of length k and hence, for a given sequence of length k, there exist $\binom{k}{2i}$ different arc-annotations with i arcs. Therefore, there exist $\sum_{i=0}^{\lfloor k/2 \rfloor} \binom{k}{2i} = 2^{k-1}$ arc-annotations of a given sequence of length k.

4 Future Work

From a computational biology point of view, especially for comparing stems, one may, however, be mostly interested in the case k (length of the common subsequence searched) might not be assumed too small compared to n. A first approach is provided in [1] where it is proved that, given two sequences of length at most n and nested arc structure, an arc-preserving common subsequence can be determined (if it exists) in $O(3.31^{k_1+k_2}\,n)$ time; obtained by deleting (together with corresponding arcs) k_1 letters from the first and k_2 letters from the second sequence. Improving the running time of the parameterization in case of stem arc structures appears to be a promising line of research.

References

1. Alber, J., Gramm, J., Guo, J., Niedermeier, R.: Computing the similarity of two sequences with nested arc annotations. Theoretical Computer Science 312(2-3), 337–358 (2004)
2. Blin, G., Denise, A., Dulucq, S., Herrbach, C., Touzet, H.: Alignment of RNA structures. IEEE/ACM Transactions on Computational Biology and Bioinformatics (2008) (to appear)
3. Blin, G., Fertin, G., Herry, G., Vialette, S.: Comparing RNA structures: Towards an intermediate model between the EDIT and the LAPCS problems. In: Sagot, M.-F., Walter, M.E.M.T. (eds.) BSB 2007. LNCS (LNBI), vol. 4643, pp. 101–112. Springer, Heidelberg (2007)
4. Blin, G., Fertin, G., Rusu, I., Sinoquet, C.: Extending the hardness of RNA secondary structure comparison. In: Chen, B., Paterson, M., Zhang, G. (eds.) ESCAPE 2007. LNCS, vol. 4614, pp. 140–151. Springer, Heidelberg (2007)

5. Evans, P.: Algorithms and Complexity for Annotated Sequences Analysis. PhD thesis, University of Victoria (1999)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability: a guide to the theory of NP-completeness. W.H. Freeman, New York (1979)
7. Gramm, J., Guo, J., Niedermeier, R.: Pattern matching for arc-annotated sequences. ACM Transactions on Algorithms 2(1), 44–65 (2006) (to appear)
8. Guignon, V., Chauve, C., Hamel, S.: An edit distance between RNA stem-loops. In: Consens, M.P., Navarro, G. (eds.) SPIRE 2005. LNCS, vol. 3772, pp. 335–347. Springer, Heidelberg (2005)
9. Jiang, T., Lin, G., Ma, B., Zhang, K.: A general edit distance between RNA structures. Journal of Computational Biology 9(2), 371–388 (2002)
10. Jiang, T., Lin, G., Ma, B., Zhang, K.: The longest common subsequence problem for arc-annotated sequences. Journal of Dicrete Algorithms, 257–270 (2004)
11. Lin, G., Chen, Z.-Z., jiang, T., Wen, J.: The longest common subsequence problem for sequences with nested arc annotations. Journal of Computer and System Sciences 65, 465–480 (2002)
12. Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM Journal of Computing 18(6), 1245–1262 (1989)

The 1.375 Approximation Algorithm for Sorting by Transpositions Can Run in $O(n \log n)$ Time

Jesun S. Firoz[1], Masud Hasan[1], Ashik Z. Khan[1], and M. Sohel Rahman[1,2]

[1] Department of Computer Science and Engineering, BUET, Dhaka-1000, Bangladesh
[2] Department of Computer Science, King's College London, UK
{jesunsahariar,ashrik}@gmail.com,
{masudhasan,msrahman}@cse.buet.ac.bd

Abstract. We improve the running time from $O(n^2)$ to $O(n \log n)$ of the existing best known $1.375-$approximation algorithm for sorting by transpositions with the help of the permutation tree data structure.

1 Introduction

In computational biology, comparison of two genomes is significant because it provides us some insight on how far away genetically these species are. Various global rearrangements, such as reversals, transpositions, translocations, fissions, fusions and block-interchanges, applied on genes have been proposed to determine the evolutionary distance between two related genomes by comparing the gene orders.

In this paper, we are interested in the transposition operation. A transposition is a rearrangement operation for a permutation in which a segment is cut out of the permutation and pasted in a different location. Sorting permutation by transpositions was first studied by Bafna and Pevzner [1], who discussed the first $1.5-$approximation algorithm which had quadratic running time. Eriksson et al. [4] gave an algorithm that sorts any given permutation of n elements by at most $\frac{2}{3}n$ transpositions.

Later, Hartman and Shamir used the concept of simplified breakpoint graph to design another $1.5-$approximation algorithm with $O(n^2)$ running time [8]. They further used the splay tree to implement this simplified algorithm and thereby reducing the time complexity to $O(n^{\frac{3}{2}}\sqrt{\log n})$ [8,9]. Finally, Elias and Hartman presented an $1.375-$approximation algorithm in [3], which is the best known approximation algorithm for sorting by transpositions in the literature so far. The running time of that algorithm [3] however is $O(n^2)$.

Very recently, in [5], Feng and Zhu presented a new data structure named the permutation tree. Using the permutation tree, Feng and Zhu improved the running time of the $1.5-$approximation algorithm of [8] to $O(n \log n)$. In this paper, we use the permutation tree data structure and follow the path of Feng and Zhu and make an effort to improve the running time of the $1.375-$approximation algorithm of [3]. In particular, with the help of the permutation tree data structure we improve the running time of the sorting by transpositions algorithm of [3] to $O(n \log n)$. This improvement in running time is bound to have serious impact in the relevant research in computational biology especially due to huge amount of various genomic data (DNA, RNA, and protein sequences) becoming available for evolutionary distance measurement.

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 161–166, 2010.

2 Preliminaries

In this section we discuss some preliminary definitions and notations, which mostly follow from [5,3]. Let $L = \{1, 2, 3, \ldots, n\}$. A permutation $\pi = (\pi_1 \pi_2 ... \pi_n)$ of L is an ordered arrangement of the elements in L. The permutation $\sigma = (1, 2, \ldots, n)$ is called the *identity* permutation. Let us define a segment X of π as a sequence of consecutive elements $\pi_i, \ldots, \pi_k, k \geq i$. Two segments $X = \pi_i, \ldots, \pi_k$ and $Y = \pi_j, \ldots, \pi_l$ are *contiguous* if $j = k+1$ or $i = l+1$. A *transposition* τ on π is an exchange of two disjoint contiguous segments. If the segments are $X = \pi_i, ..., \pi_{j-1}$ and $Y = \pi_j, ..., \pi_{k-1}$, then the result of applying τ on π, denoted $\tau.\pi$, is $(\pi_1 ... \pi_{i-1} \pi_j ... \pi_{k-1} \pi_i ... \pi_{j-1} \pi_k ... \pi_n)$. We sometimes use $trans(i, j, k)$ to denote the above transposition operation.

Given a permutation π, the *sorting by transpositions* problem asks to find a sequence of transpositions $\tau_1, \tau_2, ..., \tau_t$ to transform π into σ such that the number of transpositions t is minimized. The *transposition distance* of a permutation π, denoted by $d(\pi)$, is the smallest possible value of t.

The breakpoint graph [1] is a graph representation of a permutation (i.e. genome). Let $\pi = (\pi_0 ... \pi_{n-1})$ be a permutation. The breakpoint graph $G(\pi)$ is an edge-colored graph on $2n$ vertices $\{l_0, r_0, l_1, r_1, \ldots, l_{n-1}, r_{n-1}\}$. For every $0 \leq i \leq n-1$, r_i and l_{i+1} are connected by a grey edge, and for every π_i, l_{π_i} and $r_{\pi_{i-1}}$ are connected by a black edge, denoted by b_i.

The breakpoint graph uniquely decomposes into cycles. We denote the number of cycles in G(π) by c(π). The number of black edges in a cycle of breakpoint graph is called the *length* of the cycle. A cycle of length k is called a $k-$*cycle* and a $k-$cycle is *odd/even* if k is odd/even. The number of odd cycles is denoted by $c_{odd}(\pi)$, and we define $\Delta c_{odd}(\pi, \tau) = c_{odd}(\tau.\pi) - c_{odd}(\pi)$. A transposition τ is a $k-$*move* if $\Delta c_{odd}(\pi, \tau) = k$. A cycle is called *oriented* if there is a 2-move that is applied on three of its black edges; otherwise, it is *unoriented*. A k-cycle in the breakpoint graph is called *short* if $k \leq 3$; otherwise, it is called *long*. π and its corresponding breakpoint graph $G(\pi)$ are called *simple* if $G(\pi)$ contains only short cycles. A permutation π is *2-permutation* (*3-permutation*) if $G(\pi)$ is contains only 2-cycles (3-cycles). We generally transform permutations with long cycles into simple permutations by inserting new elements into the permutations and thereby splitting the long cycles [6].

Two pairs of black edges (a, b) and (c, d) are said to *intersect* if their edges occur in alternated order in the breakpoint graph, i.e., in order a, c, b, d. Cycles C and D intersect if there is a pair of black edges in C that intersects with a pair of black edges in D. A *configuration* of cycles is a subgraph of the breakpoint graph that is induced by one or more cycles. Configuration A is *connected* if for any two cycles c_1 and c_k of A there are cycles $c_2, \ldots, c_{k-1} \in A$ such that, for each $i \in [1, k-1]$, c_i intersects with c_{i+1}. A *component* is a maximal connected configuration in a breakpoint graph. The *size* of a configuration or a component is the number of cycles it contains. A configuration (similarly, a component) is called *unoriented* if all of its cycles are unoriented. A configuration (component) is *small* if its size is at most 8; otherwise it is *big*. Small components that do not have an $\frac{11}{8}$-sequence are called *bad small components* [3]. In a configuration, an *open gate* is a pair of black edges of a 2-cycle or an unoriented 3-cycle that does not intersect with another cycle of that configuration. A configuration not containing open gates is referred to as a full configuration.

An $(x, y)-$sequence of transpositions on a simple permutation (for $x \geq y$) is a sequence of x number of transpositions, such that at least y of them are 2-moves and that leaves a simple permutation at the end.

A *permutation tree* [5] is firstly a balanced binary tree T with root r, where each internal node of T has two children. Let t be a node of T. The left and right children of t are denoted as $L(t)$ and $R(t)$, respectively. The height of a leaf node is defined to be zero. The height of an internal node is defined to be $H(t) = max\{H(L(t)), H(R(t))\} + 1$. Since the tree is balanced, for any node t of T, we have $|H(L(t)) - H(R(t))| \leq 1$. The height of T is defined to be the height of the root $H(T) = H(r)$. Secondly, a permutation tree must correspond to a permutation. The permutation tree corresponding to $\pi = (\pi_1\pi_2 \dots \pi_n)$ has n leaf nodes, labeled by $\pi_1, \pi_2, \dots, \pi_n$ respectively. Each node of T corresponds to an interval of π and is labeled by the maximum number in the interval. For any internal node t of T, the interval corresponding to t must be the concatenation of the two intervals corresponding to $L(t)$ and $R(t)$. The number labeled to t is called the *value* of t.

Theorem 1. [5] The height of the permutation tree corresponding to $\pi = (\pi_1\pi_2 \dots \pi_n)$ is bounded by $O(\log n)$.

There are three operations for a permutation tree [5]. They are *Build*, which builds a permutation tree corresponding to a given permutation, *Join*, which joins two trees into one, and *Split*, which splits one tree into two. The following theorems report the time complexity of these three operations.

Theorem 2. [5] Time complexity of the Build operation is $O(n)$.

Theorem 3. [5] If t_1 corresponds to $(\pi_1\pi_2 \dots \pi_m)$, and t_2 corresponds to $(\pi_{m+1}\pi_{m+2} \dots \pi_n)$, then $Join(t_1, t_2)$ returns a permutation tree corresponding to $(\pi_1\pi_2 \dots \pi_m\pi_{m+1}\pi_{m+2} \dots \pi_n)$. The time complexity of $Join(t_1, t_2)$ is $O(H(t_1) - H(t_2))$.

Theorem 4. [5] Let T be a permutation tree corresponding to $\rho = (\pi_1\pi_2 \dots \pi_{m-1}\pi_m\pi_{m+1} \dots \pi_n)$. $Split(T, m)$ always returns t_l corresponding to $\rho_l = ((\pi_1\pi_2 \dots \pi_{m-1})$ and $\rho_r = ((\pi_m \dots \pi_n)$. Moreover, $Split(T, m)$ takes $O(\log n)$ time.

3 Faster Running Time for Elias and Hartman's Algorithm

As has been mentioned above, the $1.375-$approximation algorithm of Elias and Hartman (Algorithm 1) is the best approximation algorithm for sorting by transpositions in the literature. However, the running time of this algorithm is $O(n^2)$. Now, our goal is to improve the running time of this algorithm using the permutation tree data structure. In what follows we will refer to the algorithm of Elias and Hartman as the EH algorithm.

To achieve our goal we need to be able to use the permutation tree for applying $(x, y)-$sequence and $k-$move. Additionally, given a pair of black edges we can find, with the help of a permutation tree, another pair of black edges such that these two pairs intersect. Feng and Zhu [5] used the following lemma to find such a pair of black edges.

Lemma 1. [1] Let b_i and b_j are two black edges in an unoriented cycle C such that $i < j$. Let $\pi_k = \max_{i<m\leq j} \pi_m$ and $\pi_l = \pi_k + 1$. Then the black edges b_k and b_{l-1} belong to the same cycle and the pair $\langle b_k, b_{l-1}\rangle$ intersects the pair $\langle b_i, b_j\rangle$.

Feng and Zhu suggested that a permutation tree can be used for query and for transposition as follows. Let $\pi = (\pi_1 \dots \pi_n)$ be a simple permutation. Assume that the permutation tree T corresponding to π has been constructed by procedure $Build$. Now, Procedure $Query(\pi, i, j)$ finds a pair of black edges intersecting the pair $\langle b_i, b_j\rangle$ given a permutation π and Procedure $Transposition(\pi, i, j, k)$, applies a transposition $trans(i, j, k)$ on π. These two procedures can be implemented as follows.

Algorithm 1. EH Algorithm

```
1: Transform permutation π into a simple permutation π̂ .
2: Check if there is a (2, 2)-sequence. If so, apply it.
3: While G(π̂) contains a 2-cycle, apply a 2-move.
4: π̂ is a 3-permutation. Mark all 3-cycles in G(π̂).
5: while G(π̂) contains a marked 3-cycle C do
6:    if C is oriented then
7:       apply a 2-move on it.
8:    else
9:       Try to sufficiently extend C eight times
10:      if sufficient configuration has been achieved then
11:         apply an 11/8-sequence.
12:      else
13:         it must be a small component. If an 11/8-sequence is still possible apply it.
14:         if Applying a 11/8-sequence is not possible then
15:            This must be a bad small component. Unmark all cycles of the component.
16:         end if
17:      end if
18:   end if
19: end while
20: Now, G(π̂) contains only bad small components.While G(π̂) contains
    at least 8 cycles,apply an 11/8-sequence.
21: While G(π̂) contains a 3-cycle,apply a (3,2)-sequence.
22: Mimic the sorting of π using the sorting of π̂.
```

$Query(\pi, i, j)$**.** Split T into three permutation trees: t_1, which corresponds to $[\pi_1, \ldots, \pi_i]$; t_2, which corresponds to $[\pi_i + 1, \ldots, \pi_j]$; and t_3, which corresponds to $[\pi_j + 1, \ldots, \pi_n]$. Clearly this can be done in $O(\log n)$ time by two splitting operations of T. The value of the root of t_2 is the largest element (let it be π_k) in the interval $[\pi_i + 1 \ldots \pi_j]$. Assume that $\pi_l = \pi_k + 1$. By Lemma 1, pair $\langle b_k, b_{l-1}\rangle$ intersects pair $\langle b_i, b_j\rangle$. By Theorems 3 and 4, $Query(\pi, i, j)$ takes $O(\log n)$ time.

$Transposition(\pi, i, j, k)$**.** Split T into four permutation trees: t_1, which corresponds to $[\pi_1, \ldots, \pi_{i-1}]$; t_2, which corresponds to $[\pi_i, \ldots, \pi_{j-1}]$; t_3, which corresponds to $[\pi_j, \ldots, \pi_k - 1]$; and t_4, which corresponds to $[\pi_k, \ldots, \pi_n]$. Then, join the four trees by executing $Join(Join(Join(t_1, t_3), t_2), t_4)$. Clearly, adjusting the permutation tree T can be done by three splitting and three joining operations. By Theorems 3 and 4, $Transposition(\pi, i, j, k)$ takes $O(\log n)$ time as well.

Lemma 2. [5] Both $Query$ and $Transposition$ run in $O(\log n)$ time.

In what follows, we will show how the results on permutation tree can be applied to improve the running time of the EH algorithm. In particular we will state and prove a number of lemmas concerning the running time of different steps of the the EH algorithm, achieving an $O(n \log n)$ running time for the algorithm in the sequel.

Lemma 3. *Step 7 of the EH algorithm can be implemented in $O(n)$ time.*

Proof. A permutation π is made simple by (g, b)-splits acting on the breakpoint graph $G(\pi)$. A (g, b)-split for $G(\pi)$ splits one cycle into two shorter ones. Equivalently, this

operation inserts a new element into π [7]. A breakpoint graph $G(\pi)$ can be transformed into $G(\hat{\pi})$ containing only 1-cycles, 2-cycles, and 3-cycles by a series of (g, b)-splits [8], that is, the permutation corresponding to $G(\hat{\pi})$ beomces simple. This can be done by scanning the permutation linearly and inserting a new element when necessary. Thus Step 1 can be implemented in $O(n)$ time. □

Lemma 4. *Step 2 of the EH algorithm can be implemented in $O(n \log n)$ time.*

Proof. To check whether a $(2, 2)$-sequence exists, the following steps are executed:

(a) We check whether there are (at least) four 2-cycles. If yes, then we are done; otherwise we go to the next step.
(b) If there are two intersecting 2-cycles then a $(2, 2)$-sequence exists and we are done [3]. Otherwise we go to the following step.
(c) If there are two nonintersecting 2-cycles, we apply a transposition on three of the four black edges of the two 2-cycles (check all four possibilities). Clearly, this is a 2-move [2]. Now, there is a (2,2)-sequence iff in the resulting graph there is an oriented cycle. Otherwise we go to the following step.
(d) In this case the permutation is a 3-permutation. Here, if all cycles are unoriented, there is no (2,2)-sequence. Otherwise, for each oriented 3-cycle, we need to check if, after applying a 2-move on it, there is an oriented cycle in the resulting graph. There is a (2,2)-sequence iff the answer is yes for some cycle.

Clearly, the complexity depends on steps c and d as these two cases involve applying the 2-move and the transpositions. Hence, by Lemma 2, the result follows. □

Lemma 5. [5] The number of even cycles in a breakpoint graph must be even.

Lemma 6. *Step 3 of the EH algorithm can be implemented in $O(n \log n)$ time.*

Proof. By Lemma 5, there is an even number of 2-cycles in the breakpoint graph for a simple permutation. A 2-move in Step 3 transforms two 2-cycles into a 1-cycle and a 3-cycle. The 2-cycles of G(π) can be found in linear time and be eliminated by at most $\frac{n}{2}$ 2-moves. Since a transposition takes $O(\log n)$ time (Lemma 2), the result follows. □

Lemma 7. *Step 4 of the EH algorithm can be implemented in $O(n)$ time.*

Proof. All the 3-cycles can be marked by a linear scan of the breakpoint graph. Clearly, this takes at most $O(n)$ time. □

Lemma 8. *Step 7 of the EH algorithm can be implemented in* O(log*n) time.*

Proof. To apply a 2-move, we use the transposition operation on the permutation tree which can be done in $O(\log n)$ time. □

Lemma 9. *Steps 9 to 15 of the EH algorithm can be implemented in $O(\log n)$ time.*

Proof. There are two types of extensions that are sufficient for extending any cycle C: **Type 1:** Extensions closing open gates, and **Type 2:** Extensions of full configurations such that the extended configuration has at most one open gate.

To do a sufficient extension of Type 1 (add a cycle that closes an open gate), we need to pick an arbitrary open gate and find another cycle that intersects with the open gate. For this, we query the permutation tree with the black edge $\langle b_i, b_j \rangle$ of the open gate

under consideration. The query procedure in turn returns the intersecting pair $\langle b_k, b_{l-1} \rangle$ as stated above. This step takes $O(\log n)$ time.

If the configuration is full, i.e., there are no open gates, we do sufficient extension of Type 2. To do this, we query the permutation tree with each pair of black edges of each cycle in the configuration, until we find a cycle that intersects with a pair. If such a cycle is found, we extend the configuration by this cycle to find a component of size greater than or equal to 9. As there can be atmost 24 such pairs of black edges, this step takes $O(\log n)$ time as well.

Finally, we apply an $\frac{11}{8}$-sequence by using the transposition procedure of permutation tree which takes $O(\log n)$ time. Hence the result follows. □

Lemma 10. *The while loop at Step 5 of the EH algorithm can be implemented in $O(n \log n)$ time.*

Proof. The loop iterates at most n times and each iteration takes $O(\log n)$ time by Lemmas 8 and 9. □

Lemma 11. *Step 20 of the EH algorithm can be implemented in $O(n \log n)$ time.*

Proof. In this step we apply an $\frac{11}{8}$-sequence while there are at least 8 cycles. Application of an $\frac{11}{8}$-sequence takes $O(\log n)$ time and this step iterates at most $O(n)$ times. □

Lemma 12. *Step 21 of the EH algorithm can be implemented in $O(n \log n)$ time.*

Proof. Applying a (3,2)-sequence is essentially equivalent to applying 3 transpositions such that at least 2 of them are 2-moves. By Lemma 2, each transposition can take at most $O(\log n)$ time. Moreover, there can be no more than n 3-cycles. So, the lemma follows. □

Lemma 13. [5] *Step 22 of the EH algorithm can be implemented in $O(n \log n)$ time.*

From the above results the following theorem follows easily.

Theorem 5. *The EH algorithm implemented with permutation tree runs in $O(n \log n)$ time.*

References

1. Bafna, V., Pevzner, P.A.: Sorting by transpositions. SIAM J. Discrete Math. 11(2), 224–240 (1998)
2. Christie, D.: Genome rearrangement problem. Ph.D. Thesis, University of Glasgow (1999)
3. Elias, I., Hartman, T.: A 1.375-approximation algorithm for sorting by transpositions. IEEE/ACM Trans. Comput. Biology Bioinform. 3(4), 369–379 (2006)
4. Eriksson, H., Eriksson, K., Karlander, J., Svensson, L.J., Wästlund, J.: Sorting a bridge hand. Discrete Mathematics 241(1-3), 289–300 (2001)
5. Feng, J., Zhu, D.: Faster algorithms for sorting by transpositions and sorting by block interchanges. ACM Transactions on Algorithms 3(3) (2007)
6. Gu, Q.-P., Peng, S., Sudborough, I.H.: A 2-approximation algorithm for genome rearrangements by reversals and transpositions. Theor. Comput. Sci. 210(2), 327–339 (1999)
7. Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. J. ACM 46(1), 1–27 (1999)
8. Hartman, T., Shamir, R.: A simpler and faster 1.5-approximation algorithm for sorting by transpositions. Inf. Comput. 204(2), 275–290 (2006)
9. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. J. ACM 32(3), 652–686 (1985)

Parallel Algorithms for Encoding and Decoding Blob Code

Saverio Caminiti and Rossella Petreschi

Computer Science Department, Sapienza University of Rome
Via Salaria, 113 - I00198 Rome, Italy
{caminiti,petreschi}@di.uniroma1.it

Abstract. A bijective code is a method for associating labeled n-trees to $(n-2)$-strings of node labels in such a way that different trees yield different strings and vice versa. For all known bijective codes, optimal sequential encoding and decoding algorithms are presented in literature, while parallel algorithms are investigated only for some of these codes. In this paper we focus our attention on the Blob code: a code particularly considered in the field of Genetic Algorithms. To the best of our knowledge, here we present the first parallel encoding and decoding algorithms for this code. The encoding algorithm implementation is optimal on an EREW PRAM, while the decoding algorithm requires $O(\log n)$ time and $O(n)$ processors on CREW PRAM.

1 Introduction

A labeled n-tree is an unrooted tree on n nodes, each of which has a distinct label selected in the set $[0, n-1]$. An interesting data structure for representing these trees can be obtained by encoding them by means of strings of node labels. This data structure is useful in many practical applications: Genetic Algorithms [20,28], random uniformly distributed trees generation [10], fault dictionary storage [1], and distributed spanning tree maintenance [14].

The *naïve* method to obtain a string C representing a labeled n-tree T, regarded as rooted in a fixed node (e.g., node 0), consists in associating each node x with its parent $p(x)$: C is a string over the alphabet $[0, n-1]$ whose i-th element is $p(i)$. C has cardinality $n-1$ since the root has no parent and can be omitted. It should be noted that an arbitrary string of length $n-1$ over the alphabet $[0, n-1]$ does not necessarily correspond to a tree (it may represents a graph which has cycles or is not connected). Here we are interested in those codes that define a bijection between the set of labeled n-trees and the set of strings over $[0, n-1]$. Since Cayley proved that the number of labeled trees on $n \geq 2$ nodes is n^{n-2} [8], we know that this kind of one-to-one mapping requires the length of the string to be equal to $n-2$. In his proof of Cayley's theorem, Prüfer presented the first bijective string based code for trees [27]. The Prüfer code proceeds recursively, deleting, at each step, the leaf with smallest label from the tree; whenever a leaf is deleted, the label of its parent is added to the codeword. Over the years since then, many codes behaving like the Prüfer one,

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 167–178, 2010.

i.e., recursively eliminating leaves according to some rules, have been introduced by Neville [22], Moon [21], and Deo and Micikevičius [11].

Other codes, based on ideas completely different from the recursive leaves elimination, have been presented along the years. Namely, the ϑ_n bijection by Eğecioğlu and Remmel [12]; the code due to Kreweras–Moszkowski [19]; the Chen code [9]; the Blob code, the Happy code, and the Dandelion code due to Picciotto [26]; and the MHappy code due to Caminiti and Petreschi [6]. In the last years, almost all these codes have been reinterpreted in a unified framework where their behavior is described as a transformation of the tree into a functional digraph [6]. This reinterpretation allowed researchers to highlight strong similarities among a set of codes, now called Dandelion-like codes [24] (the romantic name comes from the Dandelion code that transforms a tree into a graph similar to a dandelion flower).

Encoding and decoding in sequential linear time is possible for all bijective codes presented in this introduction (see [5] for Prüfer-like codes, see [6] for Dandelion-like codes and [4] for a survey). Concerning parallel algorithms, studies have been performed and algorithms are known both for Prüfer-like codes [5] and for Dandelion-like codes [7]. The interested reader may found a complete survey on all these codes in [3].

In this paper we focus our attention on the *Blob code* introduced by Picciotto in her PhD thesis [26]. This code is based on the Orlin's proof of Cayley's theorem and makes explicit a bijection implicitly presented in that proof [23]. In its original description, the Blob code considers all nodes in decreasing label order, detaches them from their parents and adds them to a macro node called *blob*. During this process a string is generated (further details are given in Section 3). Both the original encoding and decoding algorithms require $O(n^2)$ time. By reinterpreting this code as transformation of the tree into a functional digraph, Caminiti and Petreschi [6] designed linear time encoding and decoding sequential algorithms. This reinterpretation also allowed Paulden and Smith to recognize that the Blob code is indeed equivalent to the Kreweras–Moszkowski code [24]. Moreover, several experimental analyses have been performed in the field of Genetic Algorithms to test Blob code performances focusing on certain desirable properties (namely *locality* and *heritability*) [17,18]. Also theoretical investigations of the Blob code properties have been made [25].

For the best of our knowledge, here we present the first encoding and decoding parallel algorithms for the Blob code. The encoding algorithm can be parallelized on an EREW PRAM to run in $O(\log n)$ time with $O(n/\log n)$ processors: the overall cost is linear and the algorithm is optimal. The decoding algorithm costs $O(n \log n)$ on a CREW PRAM since it runs in $O(\log n)$ time with $O(n)$ processors. Our algorithms exhibit asymptotic behavior similar to that of parallel algorithms known in the literature for other bijective codes (see Table 1). It remains an open problem to decrease the cost of a decoding algorithm in order to reach the optimality.

The paper is organized as follows: after a few preliminary definition, in Section 3, we recall the Blob code both in its original formulation and as

Table 1. Costs of known parallel algorithms for bijective codes. Costs are expressed as the number of processors multiplied by the maximum time required by a single processor working on a PRAM (EREW or CREW).

		Encoding		Decoding	
Prüfer-like	Prüfer	$O(n)$	EREW	$O(n \log n)$	EREW
	2nd Neville	$O(n\sqrt{\log n})$	EREW	$O(n\sqrt{\log n})$	EREW
	3rd Neville	$O(n)$	EREW	$O(n\sqrt{\log n})$	EREW
	Stack-Queue	$O(n\sqrt{\log n})$	EREW	$O(n\sqrt{\log n})$	EREW
Dandelion-like	Dandelion	$O(n)$	EREW	$O(n \log n)$	CREW
	ϑ_n bijection	$O(n)$	EREW	$O(n \log n)$	CREW
	Happy	$O(n)$	EREW	$O(n \log n)$	CREW
	MHappy	$O(n)$	EREW	$O(n \log n)$	CREW

reinterpreted in [6]. Our main result is given in Section 4 where we present the first parallel encoding and decoding algorithms for Blob code.

2 Preliminaries

In this section, we introduce some definitions that will be useful in the rest of the paper.

Definition 1. *Given a function* $g : [0, n] \to [0, n]$*, the* functional digraph $G = (V, E)$ *associated with* g *is a directed graph with* $V = \{0, \ldots, n\}$ *and* $E = \{(v, g(v))$ *for each* $v \in V\}$*.*

Lemma 1. *A digraph* $G = (V, E)$ *is a functional digraph if and only if the outer degree of each node is equal to 1.*

It is well known that each connected component of a functional digraph is composed of several trees, each of which is rooted in a node belonging to the *core* of the component, which is either a cycle or a loop (see Figure 1a).

Functional digraphs are easily generalizable to represent functions undefined in some values: if $g(x)$ is not defined, the node x in G does not have any outgoing edge. In this case, the connected component of G containing x is a tree rooted at x without cycles or loops (see Figure 1b).

In this paper we only deal with labeled n-trees (simply trees), and each tree T will always be considered as rooted in the fixed node 0 with all edges oriented upwards from a node v to its parent $p(v)$, i.e., T is the functional digraph associated with the function p.

The notation $P_G(u, v)$ identify the set of nodes in the unique directed path in G between u and v (both excluded); when no ambiguity arises subscript G is omitted. As usual in mathematics, we use square brackets to include one or both

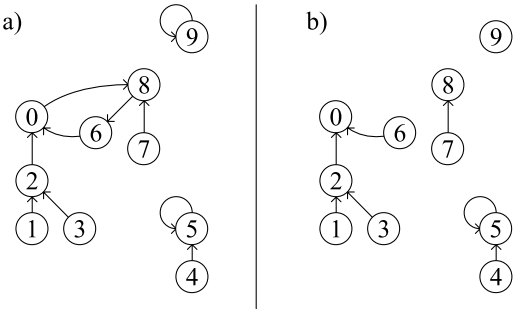


Fig. 1. a) A functional digraph associated with a fully defined function; b) A functional digraph associated with a function undefined in 0, 8, and 9

endpoints. As an example, $P(3,6]$ represents the set $\{0,2,6,8\}$ in the digraph of Figure 1a.

Let us call *n-string* a string of n elements over the alphabet $[0, n+1]$.

Definition 2. *A code is a method for associating trees to strings in such a way that different trees yield different strings. A* bijective code *is a code associating n-trees to* $(n-2)$*-strings.*

3 Blob Code

In this section, we recall the Blob code as described by Piccitto in [26] and then we reinterpret this code in terms of transformation of a tree into a functional digraph as done in [6].

Given an n-tree rooted in 0, the encoding algorithm for the Blob code considers all nodes but 0 in decreasing label order. Each node is detached from its parent and added to a macro node called *blob*. This macro node has a parent in the tree (a normal node) but it contains many other nodes; each node included in the *blob* preserves its own subtree, if any, even though this subtree is not necessarily included in the *blob*. Some nodes force the *blob* to change its parent, others do not. The formers add the parent of *blob* to the codeword, while the others simply add their own parents.

Formally the encoding algorithm can be described as follows and an example of its execution is given in Figure 2.

Algorithm: Blob Encoding
Input: a n-tree T rooted in 0 with edges oriented upward
Output: an $(n-2)$-string C

1. `Initialize` C `as an empty vector indexed form 1 to` $n-1$
2. $blob = \{n-1\}$
3. `add edge` $(blob, p(n-1))$ `and delete edge` $(n-1, p(n-1))$

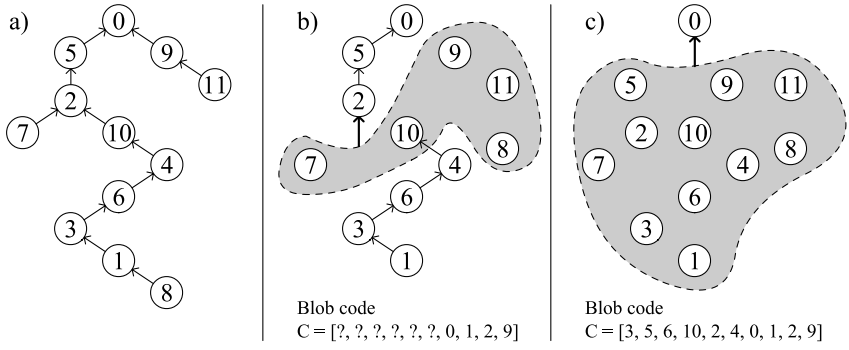


Fig. 2. a) A sample tree T rooted in 0; b) An intermediate step of the execution of the BLOB ENCODING algorithm. The grey area identifies the *blob*, question marks in the code correspond to unassigned values; c) The resulting blob (and the codeword) at the end of the execution.

```
4.  for v = n − 2 to 1 do
5.    if (P(v, 0) ∩ blob) ≠ ∅) then
6.      C[v] = p(v)
7.      delete edge (v, p(v))
8.      insert v in blob
9.    else
10.     C[v] = p(blob)
11.     delete edge (blob, p(blob)) and add edge (blob, p(v))
12.     delete (v, p(v))
13.     insert v in blob
```

Let us now reinterpret the Blob code as done in [6]. For the sake of clearness, we explicitly report some lemmas (and proofs) in order to better clarify concepts that will be useful in Section 4. We will call *stable* (*unstable*) all nodes that affirmatively (negatively) satisfy the test of Line 5. Each stable node v let the *blob* parent unchanged and its value in the codeword C is simply $p(v)$. Moreover, it is possible to see that the condition in Line 5 is not strictly connected with the incremental construction of the *blob*, but it can be computed a priori as the following lemma asserts:

Lemma 2. *Stable nodes are all nodes v such that $v < \max(P(v, 0))$.*

Proof. When node v is considered by the BLOB ENCODING algorithm set *blob* contains all the nodes from $v+1$ to n. Then the condition of Line 5 holds if and only if at least a node greater than v occurs in $P(v, 0)$. □

As a consequence of Lemma 2, we are able to characterize the value written in the codeword by each unstable node:

Lemma 3. *In the codeword, the value corresponding to each unstable node v is $p(z)$, where z is the smallest unstable node greater than v.*

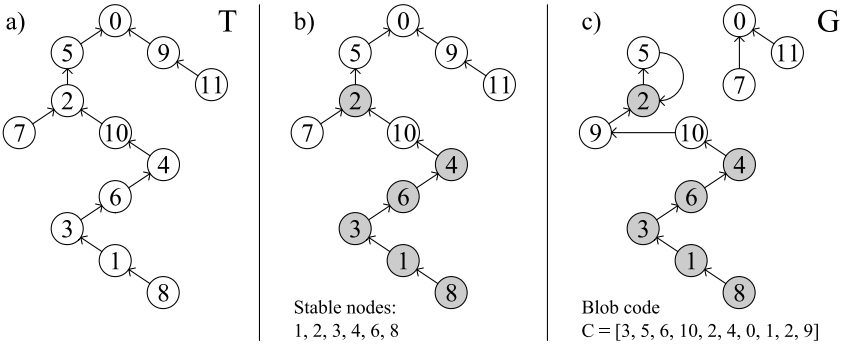


Fig. 3. a) A sample tree T rooted in 0; b) Stable nodes of T marked in grey; c) $G = \varphi_b(T)$ and the Blob code representing T.

Proof. In Line 10 of Blob Encoding algorithm the current parent of *blob* defines the code value corresponding to an unstable node v. In subsequent lines the *blob* becomes child of $p(v)$. It implies $p(blob)$ equal to the parent of the smallest unstable node greater than v, i.e., $p(z)$. □

Lemma 3 allows us to define a function φ_b that constructs a functional digraph G from a tree T in the following way: for each unstable node $v > 0$, remove the edge $(v, p(v))$ and add the edge $(v, p(z))$, where $z = min\{u \mid u > v \text{ and } u \text{ is unstable}\}$. If z does not exist (it must be $v = n - 1$), edge $(n - 1, 0)$ is added. In Figure 3a and 3c a tree T and its corresponding graph $G = \varphi_b(T)$ are depicted. Figure 3b shows all stable nodes of T.

Lemmas 2 and 3 guarantee that the codeword computed by Blob Encoding Algorithm is equal to the string $C = [g(1), g(2), \ldots, g(n-2)]$, where g is the function associated with the functional digraph $G = \varphi_b(T)$.

Let us now describe how it is possible to reconstruct the original tree T starting from its codeword C, i.e., the decoding algorithm. Obtaining G from C is straightforward, indeed $g(0)$ is always undefined and $g(n-1)$ is always 0. Thus, we will focus on proving that φ_b is invertible so that the tree can be obtained as $T = \varphi_b^{-1}(G)$. We need the following lemma:

Lemma 4. *Each path in T from a stable node v to $m = \max(P(v, 0))$ is preserved in $G = \varphi_b(T)$.*

Proof. Let v be a stable node and let assume by contradiction that the path from v to $m = \max(P(v, 0))$ is in T but not in $G = \varphi_b(T)$. This means that in the transformation from T to G at least one node w in $P(v, m)$ has changed its parent. Since φ_b changes only edges outgoing from unstable nodes, w should be unstable and then $w > \max(P(w, 0))$. $w \in P(v, m)$ implies $m \in P(w, 0)$, then w should be greater than m contradicting $m = \max(P(v, 0))$. □

In order to invert the transformation operated by φ_b, all cycles in G have to be broken, and stable and unstable nodes have to be recomputed. Each cycle Γ

(loops as regarded as cycles of length 1) is broken deleting the edge outgoing from γ, the maximum node in Γ. Lemma 4 implies that γ is unstable in T, otherwise a node greater than γ would appear in Γ. Notice that γ becomes the root of its own connected component, while 0 is the root of the only connected component not containing cycles. We call stable in G each node v such that $v < \max(P(v, \gamma_v])$, where γ_v is the root of the connected component containing v. Lemma 4 guarantees that if a node is stable in T it is also stable in G. As proved in [6] vice versa is also true. Once stable nodes have been identified, the original tree T can be easily recomputed inverting the changes operated during the encoding.

The computational complexity of the original Blob Encoding algorithm is quadratic in the number of nodes of the tree, due to the test in Line 5; the original decoding algorithm (not reported in this paper) is also quadratic. Our characterization of stable nodes (Lemma 2) decreases the complexity of the Blob Encoding algorithm to $O(n)$. Linear complexity for both encoding and decoding algorithms can be achieved exploiting the transformation of the tree into a functional digraph. Indeed both φ_b and φ_b^{-1} can be implemented in $O(n)$ sequential time: the computation of maximum nodes in the upper path (coding) and the cycles identification (decoding) can both be implemented by simple search techniques [6].

4 Parallel Algorithms

Following the ideas introduced in the previous section, here we present parallel encoding and decoding algorithms for the Blob code.

4.1 Model of Computation

A wide range of parallel machines have been constructed in the past few years for executing instructions in parallel. Parallel processors may communicate either via shared memory or via fixed connections, in synchronous or asynchronous ways. The machine may be SIMD (all the processors execute the same instructions at the same time) or MIMD (the processors may execute different instructions simultaneously). It is possible to have concurrent or exclusive reading and writing operations. Consequently, each time a parallel algorithm is designed, it is necessary to specify the underling parallel machine. In this presentation we choose not to address a specific parallel architecture, but we describe our algorithms referring to the classical synchronous SIMD, shared memory, Exclusive Write PRAM model. Even though many researchers consider this model too abstract, we are convinced that PRAM is a robust theoretical framework suitable to describe high level parallel algorithms.

Our choice is supported by the fact that this model is well studied both from a theoretical and a from practical point of view. Indeed, a great effort in designing a general purpose computer architecture to implement PRAM algorithms has been made in the last years, within the project PRAM-On-Chip at the University

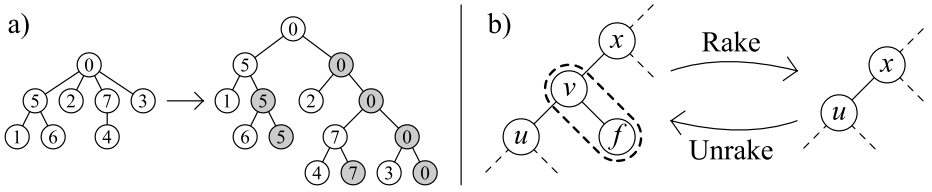


Fig. 4. a) Transformation of a tree into a regular binary tree (dummy nodes are marked in grey); b) Nodes involved in Rake and Unrake operations

of Maryland (see [31]). Moreover, a SIMD multi-thread extension of C language for providing an easy programming tool to implement PRAM algorithms has been developed. The results obtained so far seem to confirm the simplicity and the efficiency of the PRAM model. For more information we refer to [30].

Due to the lack of space, we cannot detail the basic parallel techniques used in this section (Prefix sum, Pointer Jumping and Parallel Tree Contraction), so we refer the reader to the classical book by Jájá [16]. Here we only recall Brent's theorem:

Brent's scheduling theorem [2]: let $n \in \mathbb{N}$ represent the input size and $p(n)$ be a processor bound function. Let A be an EREW PRAM algorithm that requires $w(n)$ computational operations and $t(n)$ time. If each of the $p(n)$ processors can determine in time $O(t(n))$ which steps of A it needs to simulate, then parallel algorithm A can be simulated using $O(w(n)/p(n) + t(n))$ time and $p(n)$ processors on an EREW PRAM.

In the following costs are expressed as the number of processors multiplied by the maximum time required by a single processor.

4.2 Encoding Algorithm

Given a tree T, the first step in implementing the encoding phase, according with the discussion in Section 3, consists in identifying all stable nodes. To this purpose we compute, for each node v, the maximum value in the ascending path from v to 0: we call this information $\mu(v)$. Stable nodes are those nodes v such that $v < \mu(v)$ (see Lemma 2). In order to efficiently perform μ computation we use the Rake operation to obtain Parallel Tree Contraction and Decontraction.

Initially T is transformed into a regular binary tree T_R, i.e., a tree such that each internal node has exactly 2 children: for each node $v \in T$ with $d \geq 1$ children $u_1, u_2, \ldots, u_d$, T_R has $d+1$ nodes $v_1, v_2, \ldots, v_{d+1}$ where v_1 corresponds to v while the other nodes are new dummy nodes. Each v_i has v_{i+1} as right child and u_i as left child. Figure 4a show an example of this transformation.

During this process at most $O(n)$ dummy nodes are introduced, thus the size of the tree does not asymptotically change. Dummy nodes are labeled with values that do not affect the computation of μ (e.g., negative values). Since T_R is a regular binary tree the Parallel Tree Contraction can be performed using

exclusively Rake operations. For the sake of clearness, from now on we will call T_C the tree that undergoes contraction even though the algorithm does not actually need to maintain two copies of the regular binary tree. Initially $T_C = T_R$ and each node v set $\mu(v)$ equal to $p(v)$ in T_C (for the root node we set $\mu(0) = 0$). When a Rake operation is performed to remove a node v and its leaf child f (see Figure 4b), the value of the other child u is updated as $\mu(u) = \max(\mu(u), \mu(v))$. As a consequence of this update, the following invariant is preserved at each step of the contraction process:

(1) For each node v in T_C it holds $\mu(u) = \max(P_{T_R}(u, x])$.

Notice that the path from u to x is considered with respect to the uncontracted tree T_R. Once the contraction is done and the tree is reduced to exactly 3 nodes (the root 0 and two leaves) the decontraction phase begins. All nodes removed during rake operations are reinserted backward into the tree by means of Unrake operations. When an Unrake operation is performed to reinsert node v and its leaf child f in between nodes x and u (see Figure 4b), the following updates are performed: $\mu(v) = \max(\mu(v), \mu(x))$ and $\mu(f) = \max(\mu(f), \mu(v))$. So, along the uncontraction process the following invariant is preserved:

(2) For each node v in T_C it holds $\mu(v) = \max(P_{T_R}(v, 0])$.

We underline that, at the beginning of the uncontraction process, Invariant (2) holds for all the 3 nodes as a consequence of Invariant (1). To see that Invariant (2) holds for each other node v, let's consider that, when v is reinserted by an Unrake operation, the following facts hold: by Invariant (1), value $\mu(v)$, computed during the contraction phase, is equal to $\max(P_{T_R}(u, x])$; by Invariant (2), $\mu(x) = \max(P_{T_R}(x, 0])$; then the new value $\mu(v) = \max(\mu(v), \mu(x)) = \max(P_{T_R}(v, 0])$. A similar argument holds for node f reinserted together with v by the Unrake.

Once $\mu(v)$ is known for each node $v \in T$, a vector U containing all unstable nodes in increasing order can be computed. In order to avoid expensive sorting algorithms we enumerate unstable nodes by means of Prefix Sum [13] computation. Finally, in order to obtain the functional digraph $G = \varphi_b(T)$, we simply replace the parent of the i-th unstable node in U with the parent of the $(i+1)$-th unstable node in parallel. Notice that the last element in U is always node $n-1$, and node 0 appears in U (even though we don't use it).

Algorithm: Blob Parallel Encoding
Input: a n-tree T rooted in 0 represented by its parent vector p
Output: an $(n-2)$-string C indexed from 1 to $n-1$

1. `Compute` $\mu(v)$ `for each` $v \in T$
   Create U containing all unstable nodes in increasing order
2. **for** $v = 0$ **to** $n-1$ **in parallel do**
3. **if** $v \geq \mu(v)$ **then** $A[v] = 1$ **else** $A[v] = 0$
4. `Execute Prefix Sum on` A
5. **for** $v = 0$ **to** $n-1$ **in parallel do**
6. **if** $v \geq \mu(v)$ **then** $U[A[v]] = v$

Exchange parents
7. **for** $v = 1$ **to** $|U| - 2$ **in parallel do**
8. $\quad p[U[v]] = p[U[v+1]]$
Generate the codeword
9. **for** $v = 1$ **to** $n - 2$ **in parallel do**
10. $\quad C[v] = p[v]$

The computational complexity of this algorithm is optimal, indeed, Parallel Tree contraction can be implemented in $O(\log n)$ time with $O(n/\log n)$ processors on an EREW PRAM (for a detailed description see [29]). Details on efficient transformation of a tree into a regular binary tree can be found in [15]. The same bounds hold for Prefix Sum computation. The four parallel cycles appearing in the algorithm require O(1) time with n processors and do not imply concurrent reading or writing. Applying Brent's theorem all these operations can be scheduled on $O(n/\log n)$ processors in $O(\log n)$ time. The overall cost is linear and thus the algorithm is optimal.

4.3 Decoding Algorithm

Let us now focus on the decoding phase. Graph G can be easily obtained from the codeword, then the most demanding step is the identification of stable and unstable nodes. We recall that a node v is stable in G if and only if $v < \max(P(v, \gamma_v])$, where γ_v is the maximum node in the core of the connected component containing v. In other words we have to compute the maximum value in the ascending path of each node v, let us call this value $\eta(v)$.

Differently from the encoding phase, here we cannot use Parallel Tree Contraction since the graph is not a tree: the ascending path of a node leads to the core of its connected component that may be a cycle or a loop. This computation can be obtained in a Pointer Jumping like fashion: for each node v we follow the outgoing edge $(v, g[v])$ searching for the maximum value in the ascending path. After each step we set $g[v] = g[g[v]]$ and we stop after $\lceil \log n \rceil$ steps to avoid infinite recursion inside cycles. The procedure details are as follows:

Algorithm: COMPUTE η
Input: a functional digraph G represented by g
Output: $\eta(v)$ for each node v
1. $g[0] = 0$
2. **for** $v = 0$ **to** $n - 1$ **in parallel do**
3. $\quad \eta(v) = 0$
4. **for** $step = 1$ **to** $\lceil \log n \rceil$ **do**
5. $\quad$ **for** $v = 0$ **to** $n - 1$ **in parallel do**
6. $\quad\quad \eta(v) = \max(\eta(v), g[v], \eta(g[v]))$
7. $\quad\quad g[v] = g[g[v]]$

Once η has been computed, a vector U containing all unstable nodes in increasing order can be generated as done in the BLOB PARALLEL ENCODING algorithm. The last step consists in inverting the parent exchanges performed in the encoding.

Algorithm: Blob Parallel Decoding
Input: an $(n-2)$-string C indexed from 1 to $n-1$
Output: a n-tree T represented by its parent vector p

1. $p[0] = g[0] =$ `undefined`
2. **for** $v = 1$ **to** $n-2$ **in parallel do**
3. $\quad p[v] = g[v] = C[v]$
4. $p[n-1] = g[n-1] = 0$
5. `Compute` $\eta(v)$ `for each node` v `on vector` g
   Create U containing all unstable nodes in increasing order
6. **for** $v = 0$ **to** $n-1$ **in parallel do**
7. $\quad$ **if** $v \geq \mu(v)$ **then** $A[v] = 1$ **else** $A[v] = 0$
8. `Execute Prefix Sum on` A
9. **for** $v = 0$ **to** $n-1$ **in parallel do**
10. $\quad$ **if** $v \geq \mu(v)$ **then** $U[A[v]] = v$
    Exchange parents backward
11. **for** $v = 2$ **to** $|U|-1$ **in parallel do**
12. $\quad p[U[v]] = p[U[v-1]]$
13. $p[U[1]] = 0$

The more demanding step of Blob Parallel Decoding is the computation of η realized by algorithm Compute η. It requires $O(\log n)$ sequential iterations, each of which uses $O(n)$ processors. Moreover, for this algorithm the concurrent read model is required, indeed, in Line 6, several nodes may access the same $g[v]$ and $\eta(g[v])$ concurrently. Once η is known, the computation of vector U, as well as all the parallel cycles, can be implemented in $O(\log n)$ time with $O(n/\log n)$ processors on a EREW PRAM. The overall cost of Blob Parallel Decoding is $O(n \log n)$ on a CREW PRAM.

References

1. Boppana, V., Hartanto, I., Fuchs, W.K.: Full Fault Dictionary Storage Based on Labeled Tree Encoding. In: Proc. of IEEE VTS, pp. 174–179 (1996)
2. Brent, R.P.: The Parallel Evaluation of General Arithmetic Expressions. J. of ACM 21(2), 201–206 (1974)
3. Caminiti, S.: On Coding Labeled Trees. PhD thesis, Sapienza University of Rome (December 2007)
4. Caminiti, S., Deo, N., Micikevičius, P.: Linear-time Algorithms for Encoding Trees as Sequences of Node Labels. Congr. Num. 183, 65–75 (2006)
5. Caminiti, S., Finocchi, I., Petreschi, R.: On Coding Labeled Trees. TCS 382(2), 97–108 (2007)
6. Caminiti, S., Petreschi, R.: String Coding of Trees with Locality and Heritability. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 251–262. Springer, Heidelberg (2005)
7. Caminiti, S., Petreschi, R.: Parallel Algorithms for Dandelion-Like Codes. In: Allen, G., et al. (eds.) ICCS 2009, Part I. LNCS, vol. 5544, pp. 611–620. Springer, Heidelberg (2009)
8. Cayley, A.: A Theorem on Trees. Quart. J. of Math. 23, 376–378 (1889)

9. Chen, W.Y.C.: A General Bijective Algorithm for Trees. Proc. of NAS 87, 9635–9639 (1990)
10. Deo, N., Kumar, N., Kumar, V.: Parallel Generation of Random Trees and Connected Graphs. Congr. Num. 130, 7–18 (1998)
11. Deo, N., Micikevičius, P.: A New Encoding for Labeled Trees Employing a Stack and a Queue. Bull. of ICA 34, 77–85 (2002)
12. Eğecioğlu, Ö., Remmel, J.B.: Bijections for Cayley Trees, Spanning Trees, and Their q-Analogues. J. of Comb. Th. 42A(1), 15–30 (1986)
13. Fischer, M.J., Ladner, R.E.: Parallel Prefix Computation. J. of ACM 27(4), 831–838 (1980)
14. Garg, V.K., Agarwal, A.: Distributed Maintenance of a Spanning Tree Using Labeled Tree Encoding. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 606–616. Springer, Heidelberg (2005)
15. Greenlaw, R., Petreschi, R.: Computing Prüfer Codes Efficiently in Parallel. DAM 102(3), 205–222 (2000)
16. Jájá, J.: An Introduction to Parallel Algorithms. Addison-Wesley, Reading (1992)
17. Julstrom, B.A.: The Blob Code: A Better String Coding of Spanning Trees for Evolutionary Search. In: Proc. of ROPNET, pp. 256–261 (2001)
18. Julstrom, B.A.: The Blob Code is Competitive with Edge-Sets in Genetic Algorithms for the Minimum Routing Cost Spanning Tree Problem. In: Proc. of GECCO, pp. 585–590 (2005)
19. Kreweras, G., Moszkowski, P.: Tree Codes that Preserve Increases and Degree Sequences. J. of Disc. Math. 87(3), 291–296 (1991)
20. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press, Cambridge (1996)
21. Moon, J.W.: Counting Labeled Trees. William Clowes and Sons, London (1970)
22. Neville, E.H.: The Codifying of Tree-Structure. Proc. of Cambridge Phil. Soc. 49, 381–385 (1953)
23. Orlin, J.B.: Line-Digraphs, Arborescences, and Theorems of Tutte and Knuth. J. of Comb. Th. 25, 187–198 (1978)
24. Paulden, T., Smith, D.K.: Recent Advances in the Study of the Dandelion Code, Happy Code, and Blob Code Spanning Tree Representations. In: Proc. of CEC, pp. 2111–2118 (2006)
25. Paulden, T., Smith, D.K.: Some Novel Locality Results for the Blob Code Spanning Tree Representation. In: Proc. of GECCO, pp. 1320–1327 (2007)
26. Picciotto, S.: How to Encode a Tree. PhD thesis, University of California, San Diego (1999)
27. Prüfer, H.: Neuer Beweis eines Satzes über Permutationen. Archiv der Mathematik und Physik 27, 142–144 (1918)
28. Reeves, C.R., Rowe, J.E.: Genetic Algorithms: A Guide to GA Theory. Springer, Heidelberg (2003)
29. Reif, J.H.: Synthesis of Parallel Algorithms. Morgan Kaufmann, San Francisco (1993)
30. Vishkin, U., Caragea, G.C., Lee, B.: Models for Advancing PRAM and other Algorithms into Parallel Programs for a PRAM-On-Chip Platform. In: Handbook of Parallel Computing: Models, Algorithms and Applications, ch. 5. CRC Press, Boca Raton (2008)
31. Wen, X., Vishkin, U.: PRAM-on-Chip: First Commitment to Silicon. In: Proc. of SPAA, pp. 301–302 (2007)

A Rooted-Forest Partition with Uniform Vertex Demand*

Naoki Katoh and Shin-ichi Tanigawa

Department of Architecture and Architectural Engineering, Kyoto University,
Kyoto Daigaku Katsura, Nishikyo-ku, Kyoto 615-8540 Japan
{naoki,is.tanigawa}@archi.kyoto-u.ac.jp

Abstract. A rooted-forest is a graph having self-loops such that each connected component contains exactly one loop, which is regarded as a root, and there exists no cycle consisting of non-loop edges. In this paper, we shall study on a partition of a graph into edge-disjoint rooted-forests such that each vertex is spanned by exactly d components of the partition, where d is a positive integer.

1 Introduction

The Tutte-Nash-Williams tree-packing theorem [11, 19] is one of fundamental results on combinatorial optimization, which asserts that an undirected graph $G = (V, E)$ contains k edge-disjoint spanning trees if and only if $|E_G(\mathcal{P})| \geq k|\mathcal{P}| - k$ holds for any partition $\mathcal{P}$ of V, where $E_G(\mathcal{P})$ denotes the set of edges of G connecting between two distinct components of $\mathcal{P}$ and $|\mathcal{P}|$ denotes the number of components of $\mathcal{P}$. Equivalently, it is well known that G can be partitioned into k edge disjoint spanning trees if and only if $|E| = k|V| - k$ and $|F| \leq k|V(F)| - k$ for any nonempty $F \subseteq E$, where $V(F)$ denotes the number of vertices spanned by F (see e.g. [15, Chapter 51] for more details).

As a variant of the commonly studied trees or forests, a graph is called a *pseudoforest* if each connected component contains at most one cycle [5]. An equivalent formula of the existence of a pseudoforest is described in terms of a counting condition [4]: $|F| \leq |V(F)|$ for any $F \subseteq E$. Whiteley [21] has proved a generalization of the tree-packing theorem by mixing spanning trees and spanning pseudoforests: for two integers k and l with $k \geq l$, a graph $G = (V, E)$ can be partitioned into edge-disjoint l spanning trees and $k - l$ spanning pseudoforests if and only if $|E| = k|V| - l$ and $|F| \leq k|V(F)| - l$ for any nonempty $F \subseteq E$. Haas [6] has broadened the range of l: for two integers k and l with $k \leq l \leq 2k - 1$, a graph $G = (V, E)$ satisfies $|E| = k|V| - l$ and $|F| \leq k|V(F)| - l$ for any nonempty $F \subseteq E$ if and only if it can be partitioned into l edge-disjoint trees such that each vertex is spanned by exactly k of them and any distinct l subtrees (with at least one edge) among them do not span a same vertex subset.

* The first author is supported by Grant-in-Aid for Scientific Research (B) and Grant-in-Aid for Scientific Research (C), JSPS. The second author is supported by Grant-in-Aid for JSPS Research Fellowships for Young Scientists.

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 179–190, 2010.

Also, Frank and Szegö [3] and Fekete and Szegö [2] have provided constructive characterizations of these sparse graphs.

This paper focuses on a *rooted-forest*, that is, a forest such that each connected component has a unique root. By regarding each self-loop in a graph as a root, a graph is said to be a *rooted-forest* if and only if each connected component contains exactly one loop but there exists no cycle consisting of non-loop edges. In this setting, a collection of subgraphs of rooted-forests forms a graph class that lies between forests and pseudoforests.

In this paper, we newly prove a necessary and sufficient condition of a graph (having self-loops) to be decomposed into edge-disjoint rooted-forests with two additional (nontrivial) conditions, a *precolored condition of roots* and a *vertex demand condition*. To impose these conditions, throughout the paper, we shall consider a *loop-colored graph* $G = (V, E)$, i.e., each loop has some prespecified color (as shown in Figure 1(a)), and let $\{c_1, c_2, \ldots, c_k\}$ be the set of colors appearing in the loop set of G. Roughly speaking, the color set of the roots represents the set of distinct types of supplies, and among k types of roots we require that every vertex receives d distinct types of supplies through d rooted-forests.

To define our partition problem more formally, let us introduce the following terminologies. For $F \subseteq E$, let $L(F)$ denote the set of loops contained in F. A subgraph $G' = (V', E')$ of G is said to be a *rooted-forest colored in* c_i if it satisfies the following three conditions:

(F1) $(V', E' \setminus L(E'))$ is a forest,
(F2) E' does not contain any loop colored in c_j with $j \neq i$, and
(F3) each connected component of G' contains exactly one loop colored in c_i.

G' is further called a *spanning rooted-forest* (colored in c_i) if E' spans V in addition to (F1)∼(F3) (see Figure 1(b)). Also, if $G[F] = (V(F), F)$ forms a rooted-forest for $F \subseteq E$, then F is simply called a rooted-forest.

Given a loop-colored graph $G = (V, E)$ and a positive integer d, we generalize a concept of the forest partition to a partition $\mathcal{E} = \{E_1, E_2, \ldots, E_k\}$ of E into k components such that

(P1) each E_i is a rooted-forest colored in c_i,
(P2) each vertex is spanned by exactly d components, i.e., $|\{i : \delta_E(v) \cap E_i \neq \emptyset\}| = d$ holds for each $v \in V$, where $\delta_E(v)$ denotes the set of edges of E incident to v.

If a partition of E satisfies the two conditions (P1) and (P2), we say that E (or G) admits a (k, d)-*rooted-forest partition*. Figure 1(c) shows an example of a $(4, 3)$-rooted-forest partition. Finding such a partition may be considered as a coloring problem of non-loop edges into k colors such that each color induces a rooted-forest and the number of distinct colors appearing around a vertex is equal to d.

For an edge set $F \subseteq E$, let $\chi(F)$ be the total number of distinct colors appearing in $L(F)$. The following forest partition theorem is our main result.

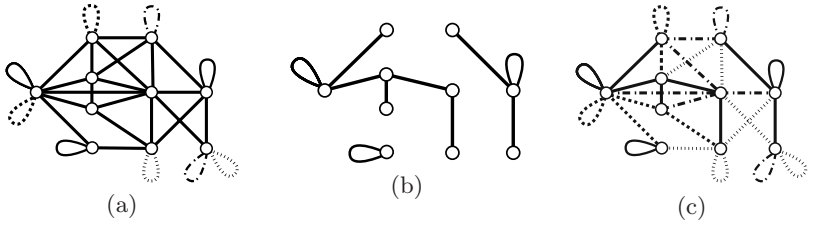


Fig. 1. (a) A loop-colored graph with $k = 4$. (b) A spanning rooted-forest. (c) A $(4, 3)$-rooted-forest partition.

Theorem 1. *Let G be a loop-colored graph, k be the number of colors used in G, and d be a positive integer with $d \leq k$. Then, G admits a proper (k, d)-rooted-forest partition if and only if it satisfies the following counting condition:*

(C1) $|E| = d|V|$, *and*
(C2) $|F| \leq d|V(F)| - d + \min\{d, \chi(F)\}$ *for any nonempty* $F \subseteq E$.

We simply say that G satisfies the *counting condition* if it satisfies (C1) and (C2) throughout the paper. Notice that, a set of edges satisfying the counting condition is a base of the matroid induced by the integer-valued nondecreasing submodular function $\mu : 2^E \to \mathbb{Z}$ defined as

$$\mu(F) = d|V(F)| - d + \min\{d, \chi(F)\} \quad (F \subseteq E). \tag{1}$$

Therefore, Theorem 1 implies that a set of edges admitting a (k, d)-rooted-forest partition is characterized in terms of a matroid as the well-known characterization of forest-partitions in terms of the union of graphic matroids, see e.g. [15].

Motivations. Let us explain a less obvious application of the tree-packing theorem to the rigidity theory. The rigidity theory, which concerns with rigidity and flexibility of structural frameworks, has a wide range of applications, e.g. in structural engineering, in molecular biology, in computer aided design, in localization problems of sensor networks and so on. One major direction towards the mathematical developments of the rigidity theory tries to extend combinatorial characterizations of the static/first-order rigidity of structural frameworks. The celebrated Maxwell-Laman theorem [10] states that a bar-joint framework in two dimensional space (see Figure 2(a)) is *(bar-inclusionwise) minimally rigid* if and only if the underlying graph $G = (V, E)$ satisfies the following condition: $|E| = 2|V| - 3$ and $|F| \leq 2|V(F)| - 3$ for any nonempty $F \subseteq E$. The tree-packing theorem further tells us the following combinatorial characterizations of rigid frameworks:

(Recki's 2spanning-tree partition [14]). G is a graph of a minimally rigid framework if and only if duplicating any edge results in a graph which can be partitioned into two edge-disjoint spanning trees.

(Crapo's proper 3tree2 partition [1]). G is a graph of a minimally rigid framework if and only if it can be partitioned into three trees such that each

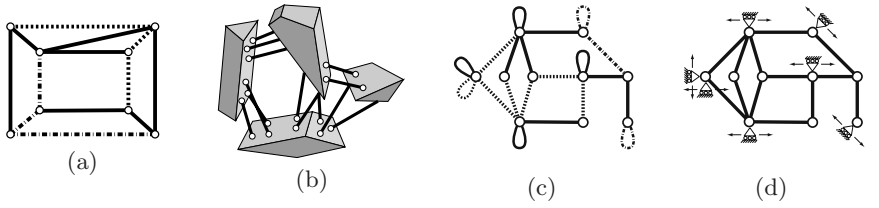


Fig. 2. (a) A graph of minimally rigid bar-joint framework in two dimensional space, where solid/dotted/broken lines indicate a proper 3tree2-partition. (b) A body-bar framework in three dimensional space. (c) A proper $(3, 2)$-rooted-forest partition of a loop-colored graph, and (d) its rigid realization as a bar-joint-slider framework.

vertex is spanned by exactly two of them and any two subtrees of them do not span a same vertex subset. (A partition satisfying the first condition is called "3tree2" and the second condition is called "proper" (see Figure 2(a)).)

Tay [18] has provided a proof of Crapo's theorem without using Laman's theorem, which indicates a usefulness of tree-packing theorems in the rigidity theory. A more direct connection between a tree-packing and the rigidity has appeared in body-bar frameworks, that is, structures consisting of rigid bodies connected by rigid bars (see Figure 2(b)). Tay [16] has proved, by using the tree-packing theorem, that a body-bar framework is generically rigid in d-dimensional space if and only if the underlying graph (obtained by regarding each body as a vertex and each bar as an edge) contains $\binom{d+1}{2}$ edge-disjoint spanning trees. Tay [17] and Whiteley [20] independently extended this result; they relate the infinitesimal rigidity of body-hinge frameworks to edge-disjoint spanning trees of the underlying graphs, and Katoh and Tanigawa [8] recently proved that this combinatorial characterization further apply to panel-hinge frameworks.

Recently investigating the rigidity of frameworks with external constraints is becoming popular, and in such a context an external constraint imposed on a vertex can be regarded as a self-loop (or self-loops) when extracting the underlying graphs. The following our recent result [9] on bar-joint frameworks under vertex-slider constraints (called bar-joint-slider frameworks) shows a direct application of (k, d)-rooted-forest partitions.

Theorem 2 ([9]). *Let G be a loop-colored graph and let $\mathbf{d}$ be a mapping from a color to a direction (i.e. a vector in $\mathbb{R}^2$). Then, G can be realized as a minimally rigid bar-joint-slider framework in two dimensional space such that each loop colored in c is realized as a slider directed to $\mathbf{d}(c)$ if and only if G admits a proper $(k, 2)$-rooted-forest partition.*

Figures 2 (c) and (d) show an example of a proper $(k, 2)$-rooted-forest partition and its bar-joint-slider realization. This result generalizes Crapo's proper 3tree2-partition to the case of bar-joint-slider frameworks. As in the two dimensional case, we strongly believe that our main theorem on (k, d)-rooted-forest partitions

will be able to bridge between the rigidity of higher dimensional frameworks with external constraints and the corresponding counting conditions.

Contribution. This paper is a succession of our recent paper [9] on the infinitesimal rigidity of bar-joint-slider frameworks explained above. In that paper, we have proved Theorem 1 for a special case of $d = 2$. The proof for $k = d$ given in Section 3.1 is directly followed from an argument given in [9]. A nontrivial part of this paper is Section 3.2 for $k > d$; the argument used in the proof in [9] cannot be extended to the case of $d > 2$.

2 Preliminaries

We skip the definition, basic terminologies and fundamental properties of matroids (see e.g. [12]). We will use the following preliminary result concerning the matroid induced by a nondecreasing submodular function, which can be found in [12, Chapter 12]. The function $f : 2^E \to \mathbb{R}$ is called *submodular* if $f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$ for any $X, Y \in 2^E$ and *nondecreasing* if $f(X) \leq f(Y)$ for any $X \subseteq Y \subseteq E$. Let $f : 2^E \to \mathbb{Z}$ be an integer-valued nondecreasing submodular function. It is known that f induces a matroid on E, denoted by $\mathcal{M}_f$, whose collection of independent sets is written by $\mathcal{I}(\mathcal{M}_f) = \{I \subseteq E : |I'| \leq f(I') \text{ for all nonempty } I' \subseteq I\} \cup \{\emptyset\}$. For an edge set F, $\mathcal{P}(F)$ denotes the collection of all possible partitions $\{F_0, F_1, \ldots, F_m\}$ of F for some integer m with $0 \leq m \leq |F|$ such that $F_i \neq \emptyset$ for each $i = 1, \ldots, m$ (and F_0 may be empty). The following proposition provides an explicit formula expressing the rank function r_f of $\mathcal{M}_f$, which is in the slightly different form from that given in e.g. [12, Chapter 12] or [15].

Proposition 1. *Let f be an integer-valued nondecreasing submodular function on E satisfying $f(F) \geq 0$ for every nonempty $F \subseteq E$. Then, for any nonempty $F \subseteq E$, the rank $r_f(F)$ of F in $\mathcal{M}_f$ is given by*

$$r_f(F) = \min\{|F_0| + \textstyle\sum_{i=1}^{m} f(F_i) : \{F_0, \ldots, F_m\} \in \mathcal{P}(F)\}. \tag{2}$$

Consider the matroid union of $\mathcal{M}_f$ and $\mathcal{M}_g$ induced by integer-valued nondecreasing submodular functions f and g on E (see e.g. [15] for the union of matroids). Pym and Perfect [13] proved that $\mathcal{M}_f \vee \mathcal{M}_g$ is the matroid induced by the submodular function $f + g$, i.e. $\mathcal{M}_f \vee \mathcal{M}_g = \mathcal{M}_{f+g}$, if $f(F) \geq 0$ and $g(F) \geq 0$ hold for every $F \subseteq E$ including $\emptyset$. Although $\mathcal{M}_f \vee \mathcal{M}_g = \mathcal{M}_{f+g}$ may not hold in general in the case of $f(\emptyset) < 0$ or $g(\emptyset) < 0$, a sufficient condition for that equation is known.

Lemma 1 ([9]). *Let f and g be integer-valued nondecreasing submodular functions on E satisfying $f(F) \geq 0$ and $g(F) \geq 0$ for every nonempty $F \subseteq E$. Then, $\mathcal{M}_f \vee \mathcal{M}_g = \mathcal{M}_{f+g}$ holds if, for any $F \subseteq E$, there exists a partition $\{F_0, F_1, \ldots, F_m\} \in \mathcal{P}(F)$ that takes the minimum values of (2) for $r_f(F)$ and $r_g(F)$ simultaneously.*

3 Proof of Theorem 1

3.1 Case of $k = d$

We shall first consider a special case of Theorem 1: k, the number of colors appearing in G, is equal to the vertex demand d. Let us denote by $c_1, c_2, \ldots, c_d$ the colors that we shall consider in this subsection. We prove that, if $k = d$, then (d, d)-rooted-forest partitions can be characterized in terms of the union of d matroids.

For a vertex set V, let $K(V)$ be the complete graph on V, and $K^+(V)$ be the loop-colored graph obtained from $K(V)$ by attaching a loop colored in c_i to each vertex for every $1 \leq i \leq d$. (Namely $d|V|$ loops are inserted in total.) We simply denote by $K^+(V)$ the edge set of $K^+(V)$ if it is clear from the context. For an edge set F, $G[F]$ denotes the graph edge-induced by F, i.e. $G[F] = (V(F), F)$.

For each color c_i, let us first consider the following function $\tau_i : 2^{K^+(V)} \to \mathbb{Z}$; for $F \subseteq K^+(V)$,

$$\tau_i(F) = |V(F)| - 1 + \chi_i(F), \tag{3}$$

where $\chi_i(F)$ is defined by $\chi_i(F) = 1$ if $L(F)$ contains a loop colored in c_i and otherwise $\chi_i(F) = 0$. Then, it is not difficult to see that τ_i is submodular. Also τ_i is nondecreasing, and hence it induces a matroid, denoted by $\mathcal{M}_{\tau_i}$, on $K^+(V)$.

Lemma 2. *An edge set $F \subseteq K^+(V)$ is a base of $\mathcal{M}_{\tau_i}$ if and only if it is a spanning rooted-forest colored in c_i.*

Let us consider how independent sets of $\bigvee_{i=1}^{d} \mathcal{M}_{\tau_i}$ can be characterized in terms of the counting condition. To apply Lemma 1, we just need to show the following property.

Lemma 3. *Let c_i be a color and let $F \subset K^+(V)$. Let m be the total number of connected components of $G[F]$ and $\{F_1, \ldots, F_m\}$ be a partition of F such that, for each $j = 1, \ldots, m$, F_j is the edge set of a connected component of $G[F]$. Also, let $F_0 = \emptyset$. Then, $\{F_0, F_1, \ldots, F_m\} \in \mathcal{P}(F)$ takes the minimum value of (2). Namely, the rank $r_{\tau_i}(F)$ of F in $\mathcal{M}_{\tau_i}$ can be written as $r_{\tau_i}(F) = \sum_{l=1}^{m} \tau_i(F_l)$.*

Combining Lemma 1 and Lemma 3, we obtain $\bigvee_{i=1}^{d} \mathcal{M}_{\tau_i} = \mathcal{M}_{\sum \tau_i}$, and the following lemma follows from $\sum \tau_i = \sum(|V(F)|-1+\chi_i(F)) = d|V(F)|-d+\chi(F)$.

Lemma 4. *Let $F \subseteq K^+(V)$. Then, F is a base of $\bigvee_{i=1}^{d} \mathcal{M}_{\tau_i}$ if and only if it satisfies the counting condition.*

We are now ready to show Theorem 1 for $k = d$.

Proof (Proof of Theorem 1 for $k = d$). Lemma 2 implies that a base of $\mathcal{M}_{\tau_i}$ is exactly a spanning rooted-forest colored in c_i. Hence, an edge set is a base of $\bigvee_{i=1}^{d} \mathcal{M}_{\tau_i}$ if and only if it can be partitioned into d edge subsets E_i each of which is a spanning rooted-forest colored in c_i, and equivalently it admits a (d, d)-rooted-forest partition. Combining Lemma 4 with this fact, we conclude that an edge set admits a (d, d)-rooted-forest partition if and only if it satisfies the counting condition. □

3.2 Case of $k > d$

Let us first show the necessity of Theorem 1.

Proof (the necessity of Theorem 1). Let $\{E_1, \ldots, E_k\}$ be a (k, d)-rooted-forest partition of E. Note that (P1) implies $|E_i| = |V(E_i)|$ for each i. Also, notice that (P2) implies $\sum_{i=1}^{k} |V(E_i)| = d|V|$ since each vertex appears in exactly d sets among $V(E_i), i = 1, \ldots, k$. Therefore, we have $|E| = \sum_{i=1}^{k} |E_i| = \sum_{i=1}^{k} |V(E_i)| = d|V|$, which implies (C1).

(C2) can be shown in a similar manner. For any $F \subseteq E$, let $F_i = F \cap E_i, i = 1, \ldots, k$. Let $s = \chi(F)$ and $t = |\{i : F_i \neq \emptyset\}|$. Note that $s \leq t$. Without loss of generality, we assume that $F_i \neq \emptyset$ for $1 \leq i \leq t$ (and $F_i = \emptyset$ for $t + 1 \leq i \leq k$), and assume $L(F_i) \neq \emptyset$ for $1 \leq i \leq s$ (and $L(F_i) = \emptyset$ for $s + 1 \leq i \leq k$). Then, by (P1), we have the following fact: $|F_i| \leq |V(F_i)|$ for each $1 \leq i \leq s$, while $|F_i| \leq |V(F_i)| - 1$ for each $s + 1 \leq i \leq t$. Also, since each vertex of $V(F)$ is spanned by at most d sets among $F_i, i = 1, \ldots, t$ by (P2), we have

$$\sum_{i=1}^{t} |V(F_i)| \leq d|V(F)|. \tag{4}$$

To see (C2), suppose for a contradiction that $F \subsetneq E$ violates the counting condition, i.e., $|F| \geq \mu(F) + 1 = d|V(F)| - d + \min\{d, s\} + 1$ (recall $s = \chi(F)$). Then, we have $|F| = \sum_{i=1}^{t} |F_i| \leq \sum_{i=1}^{s} |V(F_i)| + \sum_{i=s+1}^{t} |V(F_i)| - 1 \leq t|V(F)| - (t - s)$, and hence

$$d|V(F)| - d + \min\{d, s\} + 1 \leq |F| \leq t|V(F)| - (t - s). \tag{5}$$

On the other hand, by using (4), we also have $|F| = \sum_{i=1}^{t} |F_i| \leq (\sum_{i=1}^{t} |V(F_i)|) - (t - s) \leq d|V(F)| - (t - s)$, and hence

$$d|V(F)| - d + \min\{d, s\} + 1 \leq |F| \leq d|V(F)| - (t - s). \tag{6}$$

If $t < d$, then (5) implies $0 \geq (d - t)|V(F)| - d + \min\{d, s\} + 1 + (t - s) \geq \min\{d, s\} - s + 1 = 1$ since $|V(F)| \geq 1$ and $s \leq t < d$. This is a contradiction.

If $t \geq d$, then $t + \min\{d, s\} \geq d + s$ holds for any t and s (with $t \geq d$ and $t \geq s$). Hence, (6) implies $1 \leq 0$, which is a contradiction. □

To show the sufficiency, we need some terminologies. Suppose that a loop-colored graph $G = (V, E)$ satisfies the counting condition. An edge set $F \subseteq E$ is called *tight* if $|F| = \mu(F)$. A tight set F is said to be *small* if $|V(F)| = 1$ (and hence F consists of loops attached to a vertex), and otherwise *large*. Also, a tight set is *connected* if it induces a connected subgraph, and otherwise *disconnected*. We need the following easy observation.

Lemma 5. *Let G be a loop-colored graph satisfying the counting condition. Suppose that G is disconnected. Then, each connected component of G satisfies the counting condition.*

Let us start the proof of the sufficiency of our main theorem.

Proof (the sufficiency of Theorem 1). By Lemma 5, we can assume that G is connected. Also, we can assume $|V| \geq 2$ since Theorem 1 trivially holds if $|V| = 1$.

The proof is done by induction on $|E \setminus L(E)|$. The following two lemmas cope with the case when G have a large tight set.

Lemma 6. *If G contains a large tight set F with $\chi(F) < d$, then G admits a (k, d)-rooted-forest partition.*

Lemma 7. *If G contains a large and connected tight set F with $\chi(F) \geq d$ and $V(F) \subsetneq V$, then G admits a (k, d)-rooted-forest partition.*

By the previous two lemmas, we now concentrate on G which contains no large and connected tight set F with $V(F) \subsetneq V$. Let us assume that G is a counterexample of the statement of Theorem 1, and prove a property of this counterexample in the next lemma. We then prove that the existence of a counterexample G implies a contradiction.

Lemma 8. *Suppose that G admits no (k, d)-rooted-forest partition (i.e., G is a counterexample). Let l be a loop colored in c_i attached to a vertex v in G. Then, any vertex u adjacent to v possesses (at least) one of the following properties in G: (i) u is incident to a loop colored in c_i, or (ii) u is incident to d loops.*

Proof. Let e be an edge connecting u and v in G. Consider the following operation that generates a new graph: remove e and then insert a new loop l' colored in c_i to u. If the resulting graph, denoted by G', satisfies the counting condition, then G' admits a (k, d)-rooted-forest partition $\mathcal{E}' = \{E'_1, \ldots, E'_k\}$ by induction. Define $\mathcal{E}$ as $\{E'_1, \ldots, E'_{i-1}, E'_i \setminus \{l'\} \cup \{e\}, E'_{i+1}, \ldots, E_k\}$. Then, it is not difficult to see that $\mathcal{E}$ is a (k, d)-rooted-forest partition of E, contradicting that G is a counterexample.

Therefore, G' cannot satisfy the counting condition, and there exists an edge subset C in G' satisfying $|C| \geq \mu(C) + 1$. Let us take C as an inclusionwise minimal edge subset satisfying this inequality. Clearly $l' \in C$ (since otherwise an edge subset of G violates the counting condition). We claim

$$v \notin V(C). \tag{7}$$

To see this, suppose $v \in V(C)$. Let $C' = C \setminus \{l'\} \cup \{e, l\}$. We then have $V(C') = V(C)$ by $v \in V(C)$, and we also have $|C'| \geq |C|$ by $l' \in C$ and $e \notin C$. Also, $\chi(C') = \chi(C)$ holds since l has the same color as l'. Thus, $|C'| \geq |C| \geq d|V(C)| - d + \min\{d, \chi(C)\} + 1 = d|V(C')| - d + \min\{d, \chi(C')\} + 1 = \mu(C') + 1$, contradicting that G satisfies the counting condition since $C' \subseteq E$. Hence (7) holds.

(7) implies $V(C) \subsetneq V$. Consider $C \setminus \{l'\}$. Then, by $C \setminus \{l'\} \subset E$, $|C \setminus \{l'\}| \leq \mu(C \setminus \{l'\})$ holds. Combining this inequality, $|C| \geq \mu(C) + 1$, and $\mu(C \setminus \{l'\}) \leq \mu(C)$, we obtain $|C \setminus \{l'\}| = \mu(C \setminus \{l'\})$. Therefore, $C \setminus \{l'\}$ is a tight set with $V(C \setminus \{l'\}) \subsetneq V$. Recall that G cannot have any large connected tight set F with $V(F) \subsetneq V$ by Lemmas 6 and 7. Moreover, Lemma 5 implies that every

connected component of a disconnected tight set is again a tight set. Hence the minimality of C implies that $C \setminus \{l'\}$ is connected. As a result, $C \setminus \{l'\}$ must be a small tight set that spans u, i.e., $C \setminus \{l'\}$ consists of a set of loops attached to u. Recall that C violates the counting condition. Namely, adding a loop l' colored in c_i into $C \setminus \{l'\}$ violates the counting condition, implying that either $C \setminus \{l'\}$ contains a loop colored in c_i or $C \setminus \{l'\}$ consists of d loops according to the definition of μ (see (1)). □

By using the last lemma, we now show that the existence of a counterexample derives a contradiction. Suppose, for a contradiction, that G admits no (k, d)-rooted-forest partition. If G has a vertex incident to d loops, then the properties (i) and (ii) of Lemma 8 imply that any vertex adjacent to v is also incident to d loops. Since G is connected (due to Lemma 5), we see that every vertex of G is incident to d loops. In total we have $d|V|$ loops in G. However, since G is connected with $|V| \geq 2$, $E \setminus L(E) \neq \emptyset$ holds. Therefore, we obtain $|E| > d|V|$, contradicting that G satisfies the counting condition (C1).

Hence G has no vertex v incident to d loops. The property (i) of Lemma 8 then implies that, if there is a vertex incident to a loop colored in c_i, then any adjacent vertex also has a loop colored in c_i. Since G is connected, continuing this process, we eventually see that every vertex is incident to a loop colored in c_i. As a result, every vertex is incident to k loops (k denotes the number of colors appearing in the set of loops of G). By $k \geq d$, this contradicts that G has no vertex incident to d loops.

This completes the proof of Theorem 1. □

Note that the above proof is constructive, i.e., it provides an explicit way to construct a (k, d)-rooted-forest partition.

4 Algorithms

In this section we shall briefly discuss how to check whether a loop-colored graph satisfies the counting condition (i.e. (C1) and (C2)) or not. Imai [7] has shown that, for a given graph $H = (V, E)$ and two integers k and l, it can be decided whether H satisfies $|F| \leq k|V(F)| - l$ for every nonempty $F \subseteq E$ in polynomial time. Our algorithm is an extension of Imai's technique. Based on this algorithm, we can show how to compute a large and connected tight set, and provide an algorithm for constructing a (k, d)-rooted-forest partition. Due to the space limitation, we omit the detailed description.

We use the following conventional notations. In a digraph $D = (U, A)$ with a node set U and an arc set A, $a = (u, v) \in A$ indicates an arc from u to v and called a leaving arc from u (and an entering arc to v). Similarly, for any $S \subseteq U$, $a = (u, v)$ is a leaving arc from S if $u \in S$ and $v \notin S$. A network $N = (D, c)$ is a pair of a digraph D and a capacity function c on A. In the subsequent discussions we will focus on a network having two designate nodes s and t, called a source and a sink, respectively. An $s-t$ flow f is a function on A with $0 \leq f(a) \leq c(a)$ for every $a \in A$ satisfying the flow conservation law at each node (except for s

and t). The value of f is the sum of $f(a)$ over all leaving arcs a from s. For any $S \subseteq U$ with $s \in S$ and $t \notin S$, the set $\delta_N(S)$ of arcs leaving from S is called an $s-t$ cut, and its weight is defined as the sum of $c(a)$ over all $a \in \delta_N(S)$.

Let us first claim our result.

Theorem 3. *Let $G = (V, E)$ be a loop-colored graph, k be the number of colors appearing in G, d be an integer with $d \leq k$. Then, one can check whether G satisfies the counting condition or not in $O(d^3|V|^2)$ time.*

Proof. Since (C1) can be trivially checked from the input, we assume that $|E| = d|V|$ holds. Now let us consider how to check (C2).

To explain the basic idea, let us first consider how to decide whether G satisfies $|F| \leq d|V(F)|$ for every $F \subseteq E$. We shall define an auxiliary network $N_1 = (D_1, c_1)$ as follows. The node set of D_1 is defined as $E \cup V \cup \{s, t\}$, where s and t are a sink and a source, respectively, and the arc set A_1 of D_1 is defined as $A_1 = \{(e, v) \in E \times V : \ e \text{ is incident to } v \text{ in } G\} \cup \{(s, e) \mid e \in E\} \cup \{(v, t) \mid v \in V\}$. Namely, each non-loop edge has two leaving arcs while a loop has one leaving arc in N_1 (see Figures 3(a) and (b) for an example). The capacity function c_1 on A_1 is defined as $c_1(a) = \infty$ if $a \in E \times V$, $c_1(a) = 1$ if $a \in \{s\} \times E$, and $c_1(a) = d$ if $a \in V \times \{t\}$. Then, the network N_1 has the following property on $s-t$ cuts. For any $F \subseteq E$ and for any node subset $S \subseteq E \cup V \cup \{s\}$ of N_1 with $s \in S$ and $S \cap E = F$, the weight of the $s-t$ cut $\delta_{N_1}(S)$ is at least $|E \setminus F| + d|V(F)|$, and moreover this value can be achieved when $S \cap V$ is the set of all vertices spanned by F in G. Therefore, the max-flow min-cut theorem implies that $|F| \leq d|V(F)|$ holds for any $F \subseteq E$ if and only if the value of a maximum $s-t$ flow in N_1 is equal to $|E|$ (see [7, Lemma 2.2] for more details). Thus, one can check whether G satisfies $|F| \leq d|V(F)|$ for every $F \subseteq E$ by a maximum flow algorithm.

Checking $|F| \leq d|V(F)| - d$ can be performed by extending the above technique. For an edge $e \in E$, define a capacity function $c_{1,e}$ on A_1 as $c_{1,e}(a) = d+1$ if $a = (s, e)$ and otherwise $c_{1,e}(a) = c_1(a)$. Let $N_{1,e} = (D_1, c_{1,e})$. Namely, we increase the capacity of the arc (s, e) from 1 to $d+1$. Suppose that there exists a maximum $s-t$ flow f in N_1 whose value is equal to $|E|$. Then, as in the case of N_1, it can be observed from the max-flow min-cut theorem that $|F| \leq d|V(F)| - d$ holds for any $e \in F \subseteq E$ if and only if the value of a maximum flow in $N_{1,e}$ is equal to $|E| + d$. Therefore, one can check whether $G = (V, E)$ satisfies $|F| \leq d|V(F)| - d$ for any $F \subseteq E$ by computing the values of maximum $s-t$ flows in $N_{1,e}$ for all $e \in E$ (see [7, Theorem 2.1]). Observe that one can compute a maximum $s-t$ flow of $N_{1,e}$ from that of N_1 by at most d applications of a flow augmentation. Thus, we have a way to check the condition $|F| \leq d|V(F)| - d$ in $T_{max\_flow}(N_1) + O(d|E||A_1|)$ time, where $T_{max\_flow}(N_1)$ is the time for computing a maximum $s-t$ flow in N_1.

In order to take into account the term, $\min\{F, \chi(F)\}$, appearing in our counting condition (C2), we shall insert a gadget into N_1. Let us define a network $N_2 = (D_2, c_2)$ as follows. The node set of D_2 is $L(E) \cup \mathcal{C} \cup \{s, t, n\}$, where $\mathcal{C}$ denotes the set $\{c_1, \ldots, c_k\}$ of colors appearing in G, s and t are a source and a sink, respectively, and n is a special node. The arc set A_2 of D_2 is defined as $A_2 = \{(e, c) \in L(E) \times \mathcal{C} : \ e \text{ is colored in } c\} \cup \{(s, e) \mid e \in L(E)\} \cup \{(c, n) \mid c \in$

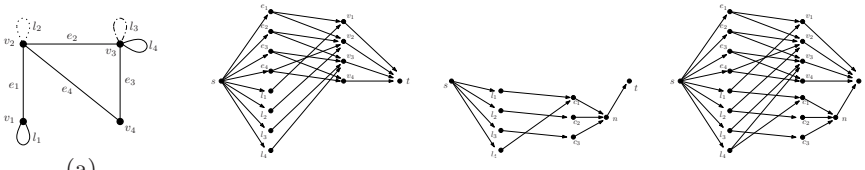


Fig. 3. (a)A loop-colored graph $G = (V, E)$ with $k = 3$, (b)$D_1 = (E \cup V \cup \{s, t\}, A_1)$, (c)$D_2 = (L(E) \cup \mathcal{C} \cup \{s, t, n\}, A_2)$, and (d)$D = (E \cup V \cup \mathcal{C} \cup \{s, t, n\}, A_1 \cup A_2)$

$\mathcal{C}\} \cup \{(n, t)\}$ (see Figure 3(c)). Also, the capacity function c_2 on A_2 is defined as $c_2(a) = \infty$ if $a \in L(E) \times \mathcal{C}$, $c_2(a) = 1$ if $a \in \{s\} \times L(E)$, $c_2(a) = 1$ if $a \in \mathcal{C} \times \{n\}$, and $c_2(a) = d$ if $a = (n, t)$. Then, the network N_2 has the following property: for any nonempty $F \subseteq L(E)$ and for any node subset $S \subseteq L(E) \cup \mathcal{C} \cup \{s, n\}$ of N_2 with $s \in S$ and $S \cap L(E) = F$, the weight of the $s - t$ cut $\delta_{N_2}(S)$ is at least $|L(E) \setminus F| + \min\{d, \chi(F)\}$ (and this value can be achieved).

We now put N_1 and N_2 together by overlapping the node set $L(E) \cup \{s, t\}$, and denote the resulting network by $N = (D, c)$. Namely, D is a digraph on $E \cup V \cup \mathcal{C} \cup \{s, t, n\}$ whose arc set is $A = A_1 \cup A_2$ (see Figure 3(d)), and c is a capacity function defined as $c(a) = \infty$ if $a \in E \times (V \cup \mathcal{C})$, $c(a) = 1$ if $a \in \{s\} \times E$, $c(a) = d$ if $a \in V \times \{t\}$, $c(a) = 1$ if $a \in \mathcal{C} \times \{n\}$, and $c(a) = d$ if $a = (n, t)$. Then, for any $F \subseteq E$ and any node subset S of N with $s \in S$ and $S \cap E = F$, the weight of the cut $\delta_N(S)$ is at least $|E \setminus F| + d|V(F)| + \min\{d, \chi(F)\}$ (and this value can be achieved). Therefore, by the max-flow min-cut theorem, $|F| \leq d|V(F)| + \min\{d, \chi(F)\}$ holds for any $F \subseteq E$ if and only if the value of a maximum $s - t$ flow in N is equal to $|E|$.

The extension of this algorithm for coping with the term $-d$ (i.e. checking $|F| \leq d|V(F)| - d + \min\{d, \chi(F)\}$) can be done in the same technique as above; the algorithm checks whether the value of a maximum flow of N can be augmented from $|E|$ to $|E| + d$ when increasing the capacity of an arc (s, e) from 1 to $d + 1$ for each $e \in E$. Therefore, one can check whether $|F| \leq d|V(F)| - d + \min\{d, \chi(F)\}$ is satisfied for any nonempty $F \subseteq E$ in $T_{max\_flow}(N) + O(d|E||A|)$ time.

It can be shown that a maximum $s-t$ flow of N can be computed in $O((d(|E| + |V|))^{3/2})$ time by replacing each arc a with the capacity $c(a)$ by $c(a)$ parallel arcs with unit capacity (see e.g. [15, Corollary 9.6.a]). Therefore, putting $|E| = d|V|$ and $|A| = O(d|V|)$, we obtain an $O(d^3|V|^2)$ time algorithm. This completes the proof.

Although the detailed description is omitted in this extended abstract, we can provide an algorithm for constructing (k, d)-rooted-forest partition based on the proof of Theorem 1 and this auxiliary graph.

Theorem 4. *Let $G = (V, E)$ be a loop-colored graph satisfying the counting condition, k be the number of colors used in G, and d be an integer with $d \geq k$. Then, one can find a (k, d)-rooted-forest partition in polynomial time.*

References

1. Crapo, H.: On the generic rigidity of plane frameworks. Technical report, Institut National de Recherche en Informatique et en Automatique (1990)
2. Fekete, Z., Szegö, L.: A note on $[k, l]$-sparse graphs. In: Graph Theory in Paris; A Conference in Memory of Claude Berge, pp. 169–177 (2004)
3. Frank, A., Szego, L.: Constructive characterizations for packing and covering with trees. Discrete Applied Mathematics 131(2), 347–371 (2003)
4. Gabow, H., Tarjan, R.: A linear-time algorithm for finding a minimum spanning pseudoforest. Information Processing Letters 27(5), 259–263 (1988)
5. Gabow, H., Westermann, H.: Forests, frames, and games: algorithms for matroid sums and applications. Algorithmica 7(1), 465–497 (1992)
6. Haas, R.: Characterizations of arboricity of graphs. Ars Combinatoria 63, 129–138 (2002)
7. Imai, H.: Network flow algorithms for lower truncated transversal polymatroids. Journal of the Operations Research Society of Japan 26(3), 186–210 (1983)
8. Katoh, N., Tanigawa, S.: A proof of the molecular conjecture. In: Proceedings of the 25th annual symposium on Computational geometry, pp. 296–305. ACM, New York (2009)
9. Katoh, N., Tanigawa, S.: On the infinitesimal rigidity of bar-and-slider frameworks. In: Dong, Y., Du, D.-Z. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 524–533. Springer, Heidelberg (2009)
10. Laman, G.: On graphs and rigidity of plane skeletal structures. Journal of Engineering mathematics 4(4), 331–340 (1970)
11. Nash-Williams, C.: Edge-disjoint spanning trees of finite graphs. Journal of the London Mathematical Society 1(1), 445 (1961)
12. Oxley, J.: Matroid theory. Oxford University Press, USA (1992)
13. Pym, J., Perfect, H.: Submodular functions and independence structures. J. Math. Anal. Appl. 30(1-31), 33 (1970)
14. Recski, A.: Network theory approach to the rigidity of skeletal structures. Part II. Laman's theorem and topological formulae. Discrete Appl. Math. 8(1), 63–68 (1984)
15. Schrijver, A.: Combinatorial optimization: polyhedra and efficiency. Springer, Heidelberg (2003)
16. Tay, T.: Rigidity of multi-graphs. I. Linking rigid bodies in n-space. Journal of combinatorial theory. Series B 36(1), 95–112 (1984)
17. Tay, T.: Linking $(n-2)$-dimensional panels in n-space II:$(n-2, 2)$-frameworks and body and hinge structures. Graphs and Combinatorics 5(1), 245–273 (1989)
18. Tay, T.: A new proof of Lamans theorem. Graphs and combinatorics 9(2), 365–370 (1993)
19. Tutte, W.: On the problem of decomposing a graph into n connected factors. J. London Math. Soc. 36, 221–230 (1961)
20. Whiteley, W.: The union of matroids and the rigidity of frameworks. SIAM Journal on Discrete Mathematics 1, 237 (1988)
21. Whiteley, W.: Some matroids from discrete applied geometry. Contemporary Mathematics 197, 171–312 (1996)

A Simple and Faster Branch-and-Bound Algorithm for Finding a Maximum Clique*

Etsuji Tomita**, Yoichi Sutani, Takanori Higashi,
Shinya Takahashi, and Mitsuo Wakatsuki

Advanced Algorithms Research Laboratory,
Department of Information and Communication Engineering
The University of Electro-Communications
Chofugaoka 1-5-1, Chofu, Tokyo 182-8585, Japan
tomita@ice.uec.ac.jp

Abstract. This paper proposes new approximate coloring and other related techniques which markedly improve the run time of the branch-and-bound algorithm MCR (J. Global Optim., 37, 95–111, 2007), previously shown to be the fastest maximum-clique-finding algorithm for a large number of graphs. The algorithm obtained by introducing these new techniques in MCR is named MCS. It is shown that MCS is successful in reducing the search space quite efficiently with low overhead. Consequently, it is shown by extensive computational experiments that MCS is remarkably faster than MCR and other existing algorithms. It is faster than the other algorithms by an order of magnitude for several graphs. In particular, it is faster than MCR for difficult graphs of very high density and for very large and sparse graphs, even though MCS is not designed for any particular type of graphs. MCS can be faster than MCR by a factor of more than 100,000 for some extremely dense random graphs.

1 Introduction

A *clique* is a subgraph in which all pairs of vertices are adjacent to each other. Finding a maximum clique in a graph is an NP-hard problem, and it is difficult to obtain the exact solution efficiently [3]. It is also difficult to obtain even a satisfactory approximate solution [12]. Nevertheless, many practical problems can be formulated as maximum clique problems (e.g., see [3], [6], [1], [5], [15], and others). Therefore, it is required to develop exact maximum-clique-finding algorithms that run very fast in practice.

* This research was supported in part by Grants-in-Aid for Scientific Research Nos. 16300001, 19500010, and 21300047 from the Ministry of Education, Culture, Sports, Science and Technology, Japan. It was also partially supported by a Special Grant for the Strategic Information and Communications R&D Promotion Programme (SCOPE) Project from the Ministry of Internal Affairs and Communications, Japan.

** Corresponding author. The author is also with the Research and Development Initiative, Chuo University, Kasuga 1-13-27, Bunkyo-ku, Tokyo 112-8551, Japan.

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 191–203, 2010.

One standard approach to develop a fast algorithm is based on the branch-and-bound method, where the focus is on reducing the search space efficiently with *low overhead*. We developed a *simple* branch-and-bound algorithm that is referred to as MCR [23]; that was successful in reducing the search space with low overhead.

Here, *simplicity* is very important to *make the overhead as low as possible*. It was shown in computational experiments that MCR clearly outperformed other existing algorithms in finding a maximum clique. However, it is not sufficiently fast to solve large practical problems. Hence, much faster algorithms are still in great demand.

In this paper, we propose a new approximate coloring that can play a crucial role in the branch-and-bound algorithm. Subsequently, we introduce a new adjunct ordered set of vertices for approximate coloring. Following this ordered set of vertices, we present a new technique for reconstructing the adjacency matrix of a graph. The algorithm that is obtained by introducing these new techniques in MCR is named MCS. While MCS inherits the simplicity of MCR to a large extent, MCS is much more successful in reducing the search space quite efficiently. The main difference between the search spaces of MCR and MCS lies in the new approximate coloring together with the adjunct ordered set of vertices introduced in MCS. The resulting overhead in MCS is still low due to the *simplicity* of the newly introduced techniques. Consequently, extensive computational experiments have shown that MCS is remarkably faster than MCR and other algorithms. MCS is faster than other algorithms by an order of magnitude for several graphs. In particular, it is faster than MCR for difficult graphs with very high density and for very large and sparse graphs, even though MCS is not designed for any particular type of graphs.

MCR is only briefly described in Sect. 3 due to the page limitation, and the reader is advised to refer to [23] for further details.

2 Definitions and Notation

(1) We consider a simple undirected graph $G = (V, E)$ with a finite set V of vertices and a finite set E of edges that comprises *unordered* pairs $(v, w)(= (w, v))$ of distinct vertices. The set V of vertices is considered to be *ordered*, and the i-th element in it is denoted by $V[i]$. A pair of vertices v and w are said to be adjacent if $(v, w) \in E$.

(2) For a vertex $v \in V$, let $\Gamma(v)$ be the set of all vertices that are adjacent to v in $G = (V, E)$, i.e., $\Gamma(v) = \{w \in V | (v, w) \in E\}$. We call $|\Gamma(v)|$ the degree of v. Here, the number of elements in a set S is denoted by $|S|$.

(3) For a subset $R \subseteq V$ of vertices, $G(R) = (R, E \cap (R \times R))$ is an *induced* subgraph. An induced subgraph $G(Q)$ is said to be a *clique* if $(v, w) \in E$ for all $v, w \in Q \subseteq V$, with $v \neq w$. In this case, we may simply say that Q is a clique. The largest clique in a graph is called a *maximum clique*, and the number of vertices in a maximum clique in an induced subgraph $G(R)$ is denoted by $\omega(R)$.

3 Maximum Clique Algorithm MCR

3.1 Branch-and-Bound Algorithm

The basic branch-and-bound algorithm MCR [23] begins with a small clique and continues finding larger and larger cliques until one is found that can be verified to have the maximum size. To be more precise, we maintain global variables Q and Q_{max}, where Q consists of the vertices of the current clique and Q_{max} consists of the vertices of the largest clique found so far. Let $R \subseteq V$ consist of vertices (candidates) that may be added to Q. We begin the algorithm by letting $Q := \emptyset$, $Q_{max} := \emptyset$, and $R := V$ (the set of all vertices). We select a certain vertex p from R, add it to Q ($Q := Q \cup \{p\}$), and then compute $R_p := R \cap \Gamma(p)$ as the new set of candidate vertices. This procedure is applied recursively while $R_p \neq \emptyset$.

When $R_p = \emptyset$ is reached, Q constitutes a maximal clique. If Q is maximal and $|Q| > |Q_{max}|$ holds, we replace Q_{max} by Q. We then backtrack by removing p from Q and R. We select a new vertex p from the resulting R and continue the same procedure until $R = \emptyset$.

3.2 Greedy Approximate Coloring

In order to prune unnecessary searching, we used *greedy approximate coloring* of the vertices in MCR. That is, each $p \in R$ is *sequentially* assigned a minimum possible positive integer value $No[p]$, called the *Number* or *Color* of p, such that $No[p] \neq No[r]$ if $(p, r) \in E$. Consequently, we have that $\omega(R) \leq \mathrm{M}ax\{No[p]|p \in R\}$.

Hence, if $|Q| + \mathrm{M}ax\{No[p]|p \in R\} \leq |Q_{max}|$ holds, we need not continue the search for R.

After $Numbers$ ($Colors$) are assigned to all vertices in R, we sort the vertices in ascending order with respect to their $Numbers$. We refer to the numbering and sorting procedure as NUMBER-SORT [23]. In each step, select a vertex p in R, beginning from the last (right) vertex and ending at the first (left) vertex. As the result, a vertex with the *maximum* $Number$ is selected in constant time in each step. This is the reason why we sort the vertices in R with respect to their $Numbers$.

Let $\mathrm{M}ax\{No[r]|r \in R\} = maxno$ and $C_i = \{r \in R|No[r] = i\}$, where $i = 1, 2, \ldots, maxno$. In other words, C_i is a set of vertices whose $Number$ ($Color$) is i. Thus, when the NUMBER-SORT has been applied to R, we have that $R = C_1 \cup C_2 \cup \ldots \cup C_{maxno}$, where the vertices in R are *ordered* in a manner such that first appear the vertices in C_1, and then the vertices in C_2 follow, and so on.

3.3 Initial Sorting and Initial Numbering

In the first stage of algorithm MCQ [24], which is a predecessor of MCR, vertices are sorted in descending order with respect to their degrees and are assigned simple initial $Numbers$, At the beginning of MCR, vertices are sorted and assigned

initial *Numbers* in a similar but more sophisticated manner. To be more precise, the steps from {SORT} to just above EXPAND(V, No) in Fig.4 (Algorithm MCR) in [23] is named *EXTENDED INITIAL SORT-NUMBER* to V.

4 New Algorithm

4.1 New Approximate Coloring

Approximate coloring is generally quite effectively used in branch-and-bound algorithms for finding a maximum clique. Here, we should note that the *minimization* of the number of colors is *not* necessarily most important. It is more important to *reduce the number of vertices from which searching is necessary*. In this paper, we propose a new approximate coloring following greedy approximate coloring in Sect. 3.2 along this line [10].

Because of the *bounding condition* mentioned in Sect. 3.2, if $No[r] \leq |Q_{max}| - |Q|$, then it is not necessary to search from vertex r. The number of vertices to be searched can be reduced if the *Number* $No[p]$ of vertex p for which $No[p] > |Q_{max}| - |Q|$ can be changed to a value less than or equal to $|Q_{max}| - |Q|$. When we encounter such vertex p with $No[p] > |Q_{max}| - |Q|$, we attempt to change it's *Number* in the following manner. Let No_p denote the original value of $No[p]$.

[Re-NUMBER p]

0) Let $No_{th} := |Q_{max}| - |Q|$. ($No_{th}$ stands for $No_{threshold}$.)
1) Attempt to find a vertex q in $\Gamma(p)$ such that $No[q] = k_1 \leq No_{th}$, with $|C_{k_1}| = 1$.
2) If such q is found, then attempt to find *Number* k_2 such that no vertex in $\Gamma(q)$ has *Number* k_2.
3) If such number k_2 is found, then change the *Number* of q and p so that $No[q] = k_2$ and $No[p] = k_1$.

(If no vertex q with *Number* k_2 is found, nothing is done.)

When the vertex q with *Number* k_2 is found, $No[p]$ is changed from No_p to k_1 ($\leq No_{th}$); thus, *it is no longer necessary to search from* p.

The exact procedure Re-NUMBER is shown in Fig. 1. To save time, we use it only when $No[p] = maxno$. The new approximate coloring is described in the first part of Fig. 2 under the heading {NUMBER}; it can be seen that Re-NUMBER follows the conventional greedy approximate coloring. The second part of Fig. 2, under the heading {SORT}, describes the sorting of the vertices in R in ascending order with respect to their *Numbers* (Refer to the end of Sect. 3.2). Note that as shown in Fig. 2, vertex r with $No[r] \leq No_{th}$ need not be sorted since the searching operation need not begin from r according to the *bounding condition*.

In Fig. 2, assume that V_a is identical to R for a while (until V_a is introduced in Sect. 4.2).

We employ the new procedure Re-NUMBER-SORT (in Fig. 2) instead of the procedure NUMBER-SORT used in MCR [23] in order to make more effective use of the bounding condition.

```
procedure Re-NUMBER(p, No_p, No_th, C_1, C_2, ..., C_maxno)
begin
  for k_1 := 1 to No_th − 1 do
    if |C_k1 ∩ Γ(p)| =1 then
      q := the element in (C_k1 ∩ Γ(p)) ;
      for k_2 := k_1 + 1 to No_th do
        if C_k2 ∩ Γ(q) = ∅ then
          {Exchange the Numbers of p and q.}
          C_No_p := C_No_p − {p};
          C_k1 := (C_k1 − {q}) ∪ {p};
          C_k2 := C_k2 ∪ {q};
          return
        fi
      od
    fi
  od
end { of Re-NUMBER}
```

Fig. 1. Procedure Re-NUMBER

The time complexity of Re-NUMBER-SORT is $O(|R|^3)$, while that of NUMBER-SORT [23] is $O(|R|^2)$. Here, $|R|$ is the number of vertices of the concerned subgraph $G(R)$.

4.2 Adjunct Ordered Set of Vertices for Approximate Coloring

As noted in [7], [24], and [23], the ordering of vertices is crucial in algorithms for finding a maximum clique. The result of approximate coloring greatly depends on the order of vertices because *sequential* coloring is the main component in the procedure. In MCR, the vertices are sorted in descending order mainly with respect to their degrees. When *Numbering* procedures are applied, the vertices are sorted in ascending order with respect to their *Numbers*, and the initial order of the vertices with the same *Number* is *inherited* in the subsequent subproblems [23]. However, the application of *Re-NUMBER*, which is described in Sect. 4.1, changes the *Numbers* of the vertices, thereby making the vertices disordered with respect to their degrees. We can reduce the search space by *sorting* vertices in *R in descending order with respect to their degrees* before every application of approximate coloring. That is, the reduction of the search space is most effective if *the minimum possible Number* is assigned to *a vertex with the maximum degree* in each step of greedy approximate coloring [9], [20]. However, the sorting of vertices is a computational burden and reduces the overall running time only for *dense* graphs [20]. The aim of the present study is to develop a faster algorithm whose use is not confined to any particular type of graphs. So, in addition to the ordered set R of vertices, we simply introduce a new particular *adjunct ordered set* V_a of vertices that preserves the order of the vertices *sorted in descending*

```
procedure Re-NUMBER-SORT(V_a, R, No)
begin
  {NUMBER}
    maxno := 0;
    C_1 := ∅;
    for i := 1 to |V_a| do
      { Conventional greedy approximate coloring }
      p := V_a[i] ;
      k := 1;
      while C_k ∩ Γ(p) ≠ ∅
        do k := k + 1 od
      if k > maxno then
        maxno := k;
        C_maxno := ∅
      fi
      C_k := C_k ∪ {p};

      { - Re-NUMBER starts - }
      No_th := |Q_max| − |Q|;
      if (k > No_th) and (k = maxno) then
        Re-NUMBER(p, k, No_th, C_1, C_2, ..., C_maxno) ;
        if C_maxno = ∅ then
          maxno := maxno − 1
        fi
      fi
      { - Re-NUMBER ends - }

    od
  {SORT (vertices in R in ascending order w.r.t. their Numbers)}
    i := |V_a|;
    if No_th < 0 then No_th := 0 fi
    for k := maxno downto No_th + 1 do
      for j := |C_k| downto 1 do
        R[i] := C_k[j];
        No[R[i]] := k;
        i := i − 1
      od
    od
    if i ≠ 0 then
      R[i] := C_{k−1}[|C_{k−1}|];
      No[R[i]] := No_th
    fi
end { of Re-NUMBER-SORT }
```

Fig. 2. Procedure Re-NUMBER-SORT

order with respect to their degrees in the first stage [22]. We apply the procedure Re-NUMBER-SORT shown in Fig. 2 to the vertices in V_a, begining from the first (left) vertex and ending at the last (right) vertex. Thus, we can avoid the undesirable effect of Re-NUMBER.

```
procedure MCS(G = (V, E))
begin
  global Q := ∅; global Q_max := ∅;
  {EXTENDED INITIAL SORT-NUMBER}
  Apply EXTENDED INITIAL SORT-NUMBER to V (see Sect. 3.3);
  Reconstruct the adjacency matrix as described in Sect. 4.3;
  EXPAND (V, V, No);
  output Q_max  {Maximum clique}
end { of MCS }

procedure EXPAND(V_a, R, No)
begin
  while R ≠ ∅ do
    p := the last vertex in R (i.e., a vertex with the maximum Number in R);
    if |Q| + No[p] > |Q_max| then
      Q := Q ∪ {p};
      V_p := V_a ∩ Γ(p);  {preserving the order}
      if V_p ≠ ∅ then
       Re-NUMBER-SORT(V_p, newR, newNo);
       {The initial values of newR and newNo have no significance}
       EXPAND(V_p, newR, newNo)
      else if |Q| > |Q_max| then Q_max := Q fi
      fi
    else return
    fi
    Q  := Q − {p};
    R  := R − {p};
    V_a := V_a − {p}  {preserving the order}
  od
end { of EXPAND }
```

Fig. 3. Algorithm MCS

As mentioned in Sect. 3.1, we select a vertex in the ordered set R for *searching*, beginning from the last (right) vertex and continuing up to the first (left) vertex, as shown in Fig. 3.

4.3 Reconstruction of the Adjacency Matrix

Each graph is stored as an adjacency matrix in the computer memory. Sequential numbering in Re-NUMBER-SORT is carried out according to the initial order of vertices in the adjunct ordered set V_a, as described in Sect. 4.2. Taking this into account, we *rename* the vertices of the graph and *reconstruct* the adjacency matrix so that the vertices are *consecutively ordered* in a manner identical to *the initial order of vertices* obtained at the beginning of MCR. The above-mentioned reconstruction of the adjacency matrix results in a more effective use of the cache memory since it facilitates the use of localized memory.

4.4 Algorithm MCS

The new algorithm obtained by introducing the techniques described in Sects. 4.1–4.3 in MCR is named MCS and is shown in Fig. 3.

4.5 Effectiveness of the Reduction of the Search Space

We confirm the effectiveness of the algorithm MCS in reducing the search space. Some characteristic results of computational experiments conducted under the conditions described in Sect. 5 (Computational experiments) for MCR and MCS are listed in Table 1.

Table 1. Comparison of branches

Graph		$Branches \times 10^{-3}$			CPU time
Name	ω	MCR	MCS	$(MCR/MCS)_b$	$(MCR/MCS)_t$
r200.9	40–44	97,627	6,608	15	9
r200.95	58–66	104,801	2,735	38	22
r200.98	90–103	2,357	4	589	155
r300.98	120	4.03×10^6	31,619	127	108
r500.994	263	$> 4.29 \times 10^6$	70	$> 61,286$	$> 256,410$
MANN_a45	345	2,952	225	13	11
p_hat500-3	50	138,300	7,923	18	12
p_hat700-3	62	3,733,665	88,168	42	29
san400_0.9_1	100	74	2	37	28
gen200_p0.9_44	44	583	35	17	12
gen200_p0.9_55	55	2,335	112	21	13
gen400_p0.9_55	55	$> 4.29 \times 10^6$	2,894,935	> 1.5	100
gen400_p0.9_65	55	$> 4.29 \times 10^6$	3,332,982	> 1.3	> 66

Table 1 lists the number of branches, that is, the total number of EXPAND() calls excluding the first call, of MCR and MCS for random graphs r200.9 – r500.994 and several DIMACS benchmark graphs in the leftmost column. The random graphs r200.9, r200.95, and r200.98 are graphs with 200 vertices and with edge probabilities 0.9, 0.95, and 0.98, respectively. The number of branches specified for r200.9 is the average over 10 graphs, and the number of branches given for r200.95 and r200.98 is the average over 100 graphs. The second column (ω) lists the ranges of the maximum clique sizes obtained.

In Table 1, the values for graphs with names of the form r$n.p$ ($n = 300, 500$ and $p = 0.98, 0.994$) are obtained from one random graph with n vertices and with edge probability p ($4.29 \times 10^9 = 2^{32}$). The number of branches is related to the size of the search space. The fifth column $(MCR/MCS)_b$ lists the ratio of the number of branches of MCR to that of MCS.

The ratio of the CPU time required by MCR to that of MCS for each graph is given in the last column $(MCR/MCS)_t$ for reference and has been obtained from Tables 2 and 3 in Sect. 5.

Table 1 confirms that MCS is quite successful in reducing the search space. In addition, we can see that the reduction of the search space by MCS effectively contributes to the reduction of the running time. We have confirmed that the search space of MCS is considerably smaller than that of MCR for all graphs in Sect. 5.

5 Computational Experiments

We carried out computational experiments in order to demonstrate the overall superiority of MCS over MCR. Both MCR and MCS were implemented in exactly the same manner in the programming language C. The computer used, which had a Linux operating system, is described in Appendix. We also executed the DIMACS benchmark program dfmax [13], [14] as a standard. The computation times for other algorithms are calibrated using the ratios shown in Appendix.

5.1 Results for Random Graphs

Random graphs are generated for each pair of n (number of vertices) and p (edge probability) listed in Table 2. These graphs are generated such that there exists an edge with probability p for each pair of vertices. The average CPU times [sec] required to solve these graphs when using dfmax, MCR, and MCS are listed in Table 2. The CPU times are averaged over 10 random graphs for each pair of n and p. However, when the CPU time [sec] is greater than 10^5, the individual value of the graph, instead of the average, is listed. The CPU times required to solve the graphs with $n \leq 200$ and $p \geq 0.95$ are averaged over 100 graphs because of the large variations in these graphs and the short running time of MCR and MCS. For graphs with $n \geq 300$ and $p \geq 0.9$, the CPU time for only one graph is considered for each pair of n and p (10^5 seconds $\simeq$ 1.16 days, and 10^7 seconds $\simeq$ 116 days). The third column (ω) lists the ranges of the sizes of the maximum cliques obtained.

The calibrated CPU times for New [16] and COCR [18] are also listed for reference. The boldface entries indicate the fastest time in the row. In Table 2, it is observed that MCS is faster than MCR for all graphs. MCS is particularly faster than MCR for dense graphs. MCS is the fastest for all the random graphs listed in Table 2, except for that with $[n = 200, p = 0.9]$. For this exceptional graph, COCR is approximately twice as fast as MCS. COCR is specially designed for solving the maximum clique problem for *dense* graphs. For the graphs with $p \geq 0.99$ in Table 2, MCS is faster than MCR by a factor of greater than 100,000.

Regarding dfmax, it was stated in [14] that "It ... may be hard to beat on sparser graphs, especially random ones." Prior to the development of MCQ [24], dfmax was widely recognized as the fastest maximum clique algorithm for sparse graphs, as stated in [8] and [16]. MCQ and its successors are faster than dfmax, even for sparse graphs. MCS is the *only* algorithm that is *more than twice as fast as dfmax* for sparse graphs with 10,000 or more vertices (Table 2).

5.2 Results for DIMACS Benchmark Graphs

Table 3 lists the CPU times required by MCS and other algorithms to solve the DIMACS benchmark graphs [13], where the calibrated CPU times for New [16] and ILOG [17] are included for reference. In this table, *density* represents the edge density of the graph. The boldface entries indicate the fastest time among the times obtained within the time limits in the row. From this table, it is

Table 2. CPU time[sec] for random graphs

Graph			dfmax	MCR		MCS	New	COCR
n	p	ω	[13]	[23]			[16]	[18]
	0.8	19-21	0.140	0.014	·	**0.008**	0.065	0.150
100	0.9	29-32	3.67	0.038	○	**0.013**	0.663	0.196
	0.95	39-48	23.736	0.011	○	**0.003**	0.196	
	0.98	56-68	26.5401	0.0012		**0.0009**		
	0.8	23	6.88	0.55	○	**0.23**		0.75
150	0.9	36-39	1058.96	5.26		**1.00**		1.16
	0.95	50-59	37,436.79	3.94	★	**0.35**		
	0.98	73-85	$> 10^5$	0.243	★○	**0.006**		
	0.8	24-27	192.7	12.3	·	**4.5**	147.3	8.7
200	0.9	40-44	$> 10^5$	647		74		○ **37**
	0.95	58-66	$> 10^5$	1,272	★○	**59**		
	0.98	90-103	$> 10^5$	30.9	★★	**0.2**		
	0.6	15-16	144.1	1.4		**1.0**	3.5	5.0
	0.7	19-21	26,236	23	·	**12**	121	
300	0.8	28-29	$> 10^5$	1,264	○	**394**		
	0.9	49		1,475,387	★○	**62,607**		
	0.98	120		284,534	★★	**2,623**		
	0.5	13-14	9.0	3.6		**2.8**	7.3	17.4
500	0.6	17	242	63	·	**40**	183	
	0.7	22-23	24,998	3,268	○	**1,539**		
	0.994	263	$> 1.5 \times 10^7$	$> 10^7$	★★★★★	**39**		
	0.4	12	33.3	16.1		**13.2**	23.2	
	0.5	15	1,107	395		**290**		
1,000	0.6	19-20	106,776	24,986	·	**15,317**		
	0.66	23		555,089	○	**275,964**		
	0.998	618		$> 10^7$	★★★★★	**46**		
2,000	0.9995	1,453		$> 10^7$	★★★★★	**61**		
	0.1	7	6.3	5.3	·	**3.3**		
5,000	0.2	9	259	197		**138**		
	0.3	12	14,008	8,668		**5,818**		
10,000	0.1	7-8	137	100	·	**60**		
	0.2	10	9,417	8,055	·	**4,389**		
15,000	0.1	8	793	511	·	**327**		
20,000	0.1	8	2,665	1,737		**1,179**		

Entries marked ★★★★★, ★★,★○, ★, ○, and · are respectively at least 100,000, 100, 20, 10, 2, and 1.5 times faster than any of the others in the same row.

confirmed that MCS is almost always faster than MCR and the other algorithms in Table 3.

MCS is almost always considerably faster than χ+DF [8], COCR [18], MIPO [2], SQUEEZE [4], and Target [21] (see Table 4 in [23]). Although COCR is specially designed to efficiently find a maximum clique in dense graphs, MCS

Table 3. CPU time[sec] for DIMACS benchmark graphs (2)

Graphs				dfmax	MCR	MCS		New	ILOG
Name	n	*density*	ω	[13]	[23]			[16]	[17]
brock400_1	400	0.75	27	22,051	1,771	○	**693**		8,401
brock400_2	400	0.75	29	13,519	726	○	**297**		5,860
brock400_3	400	0.75	31	14,795	1,200	○	**468**		3,316
brock400_4	400	0.75	33	10,633	639	○	**248**		4,483
brock800_1	800	0.65	23	$> 10^5$	17,789	·	**9,347**		$> 10,667$
brock800_2	800	0.65	24	$> 10^5$	16,048	·	**8,368**		$> 10,667$
brock800_3	800	0.65	25	91,031	10,853	·	**5,755**		$> 10,667$
brock800_4	800	0.65	26	78,737	7,539	·	**3,997**		$> 10,667$
MANN_a27	378	0.990	126	$> 10^5$	2.5	○	**0.8**	$> 2,232$	13.7
MANN_a45	1,035	0.996	345	$> 10^5$	3,090	★	**281**		$> 10,667$
p_hat300-3	300	0.744	36	779.7	10.8	○	**2.5**		30.2
p_hat500-2	500	0.505	36	132.9	3.1	○	**0.7**	95.7	24.2
p_hat500-3	500	0.752	50	$> 10^5$	1,788	★	**150**		9,441
p_hat700-2	700	0.498	44	5,299.9	44.4	●	**5.6**		189.5
p_hat700-3	700	0.748	62	$> 10^5$	68,187	★○	**2,392**		$> 10,667$
p_hat1000-2	1,000	0.489	46	$> 10^5$	2,434	★	**221**		12,478
p_hat1500-2	1,500	0.506	65	$> 10^5$	722,733	★○	**16,512**		$> 10,667$
san200_0.9_1	200	0.900	70	$> 10^5$	1.20		0.22	○ **0.06**	0.77
san200_0.9_2	200	0.900	60	$> 10^5$	4.2	○	**0.4**	1.0	1.9
san400_0.7_1	400	0.700	40	$> 10^5$	1.76	○	**0.54**	$> 2,232$	17.2
san400_0.7_2	400	0.700	30	$> 10^5$	0.33	○	**0.13**	112.97	50.0
san400_0.7_3	400	0.700	22	$> 10^5$	3.6	○	**1.4**		202.4
san400_0.9_1	400	0.900	100	$> 10^5$	3.4	★○	**0.1**		1,259.3
san1000	1,000	0.502	15	$> 10^5$	4.8		2.1	★○ **0.1**	76.1
sanr200_0.7	200	0.702	18	3.06	0.57	·	**0.34**	3.15	3.2
sanr200_0.9	200	0.898	42	86,954	289	●	**41**		111
sanr400_0.7	400	0.700	21	2,426	379	○	**181**		2,325
gen200_p0.9_44	200	0.900	44	48,262	5.39	★	**0.47**		
gen200_p0.9_55	200	0.900	55	9,281.0	15.0	★	**1.2**		
gen400_p0.9_55	400	0.900	55		5,846,951	★★	**58,431**		
gen400_p0.9_65	400	0.900	65		$> 10^7$	★●	**151,597**		
gen400_p0.9_75	400	0.900	75		$> 10^7$	★○	**294,175**		
C250.9	250	0.899	44	$> 10^5$	44,214	★	**3,257**		

Entries marked ★★,★●, ★○, ★, ●, ○, and · are respectively at least 100, 50, 20, 10, 5, 2, 1.5 times faster than any of the others within the time limits in the same row.

is faster than COCR by 3.5 times for MANN_a27, whose density is very high (*density* = 0.990). Further, MCS is confirmed to be much faster than MC of Wood [26] and CP+SDP of Hoeve [11], as is evident in [17].

6 Concluding Remarks

Our new algorithm, MCS, retains the *simplicity* of our earlier algorithms while *further reducing the search space* quite efficiently with *low overhead*; hence, it runs remarkably faster than MCR and the other algorithms.

Owing to the page limitation of this paper, we cannot describe the details of the individual contribution of techniques in Sects. 4.1-4.3, but it is noted that effectiveness of MCS is established by the combination of all of these techniques. For example, a single introduction of the new approximate coloring (in Sect. 4.1) in MCR results in requiring more than 10^5 seconds to solve MANN_a45 [10]. A single introduction of the adjunct ordered set V_a (in Sect. 4.2) in MCR is almost always effective, but is not effective for MANN_a45 [22].

Our present techniques can be useful for generating large maximal cliques [25]. Some theoretical analysis of maximum-clique-finding algorithms is on the way based upon [25] and [19].

Acknowledgements. We thank T. Nakagawa for his contribution to the computational experiments. We express our sincere gratitude to E. Harley and the referees for their helpful detailed comments. Useful discussions and kind help by T. Akutsu and others are also acknowledged.

References

1. Bahadur, D.K.C., Tomita, E., Suzuki, J., Horimoto, K., Akutsu, T.: Protein threading with profiles and distance constraints using clique based algorithms. J. Bioinformatics and Computational Biology 4, 19–42 (2006)
2. Balas, E., Ceria, S., Cornuéjols, G., Pataki, G.: Polyhedral methods for the maximum clique problem. In: Johnson, Trick (eds.) [13], pp. 11–28 (1996)
3. Bomze, I.M., Budinich, M., Pardalos, P.M., Pelillo, M.: The maximum clique problem. In: Du, D.-Z., Pardalos, P.M. (eds.) Handbook of Combinatorial Optimization, Supplement vol. A, pp. 1–74 (1999)
4. Bourjolly, J.-M., Gill, P., Laporte, G., Mercure, H.: An exact quadratic 0-1 algorithm for the stable set problem. In: Johnson, Trick (eds.) [13], pp. 53–73 (1996)
5. Brown, J.B., Bahadur, D.K.C., Tomita, E., Akutsu, T.: Multiple methods for protein side chain packing using maximum weight cliques. Genome Inform. 17, 3–12 (2006)
6. Butenko, S., Wilhelm, W.E.: Clique-detection models in computational biochemistry and genomics - invited review -. European J. Operational Research 173, 1–17 (2006)
7. Carraghan, R., Pardalos, P.M.: An exact algorithm for the maximum clique problem. Operations Research Letters 9, 375–382 (1990)
8. Fahle, T.: Simple and Fast: Improving a branch-and-bound algorithm for maximum clique. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 485–498. Springer, Heidelberg (2002)
9. Fujii, T., Tomita, E.: On efficient algorithms for finding a maximum clique. Technical Report of IECE, AL81-113, pp. 25–34 (1982)
10. Higashi, T., Tomita, E.: A more efficient algorithm for finding a maximum clique based on an improved approximate coloring. Technical Report of Univ. Electro-Commun, UEC-TR-CAS5-2006 (2006)
11. van Hoeve, W.J.: Exploiting semidefinite relaxations in costraint programming. Computers & Operations Research 33, 2787–2804 (2006)
12. Håstad, J.: Clique is hard to approximate within $n^{1-\varepsilon}$. Acta Mathematica 182, 105–142 (1999)

13. Johnson, D.S., Trick, M.A. (eds.): Cliques, Coloring, and Satisfiability, DIMACS Series in Discr. Math. and Theoret. Comput. Sci., vol. 26. American Math. Soc., Providence (1996)
14. `http://www.cs.sunysb.edu/~algorith/implement/dimacs/distrib/color/graph/form`
15. Matsunaga, T., Yonemori, C., Tomita, E., Muramatsu, M.: Clique-based data mining for related genes in a biomedical database. BMC Bioinformatics 10 (2009)
16. Östergård, P.R.J.: A fast algorithm for the maximum clique problem. Discrete Applied Math. 120, 197–207 (2002)
17. Régin, J.C.: Using constraint programming to solve the maximum clique problem. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 634–648. Springer, Heidelberg (2003)
18. Sewell, E.C.: A branch and bound algorithm for the stability number of a sparse graph. INFORMS J. Computing 10, 438–447 (1998)
19. Shindo, M., Tomita, E.: A simple algorithm for finding a maximum clique and its worst-case time complexity. Systems and Computers in Japan 21, 1–13 (1990)
20. Shindo, M., Tomita, E., Maruyama, Y.: An efficient algorithm for finding a maximum clique. Technical Report of IEC, CAS86-5, pp. 33–40 (1986)
21. Stix, V.: Target-oriented branch and bound method for global optimization. J. Global Optim. 26, 261–277 (2003)
22. Sutani, Y., Tomita, E.: Computational experiments and analyses of a more efficient algorithm for finding a maximum clique. Technical Report of IPSJ, 2005-MPS-57, pp. 45–48 (2005)
23. Tomita, E., Kameda, T.: An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. J. Global Optim. 37, 95–111 (2007); J. Global Optim. 44, 311 (2009)
24. Tomita, E., Seki, T.: An efficient branch-and-bound algorithm for finding a maximum clique. In: Calude, C.S., Dinneen, M.J., Vajnovszki, V. (eds.) DMTCS 2003. LNCS, vol. 2731, pp. 278–289. Springer, Heidelberg (2003)
25. Tomita, E., Tanaka, A., Takahashi, H.: The worst-case time complexity for generating all maximal cliques and computational experiments (An invited paper in the Special Issue on COCOON 2004). Theoret. Comput. Sci. 363, 28–42 (2006); The preliminary version appeared in Tomita, E., Tanaka, A. Takahashi, H.: The worst-case time complexity for generating all maximal cliques. In: Chwa, K.-Y., Munro, J.I.J.(eds.) COCOON 2004. LNCS, vol. 3106, pp.161–170. Springer, Heidelberg (2004)
26. Wood, D.R.: An algorithm for finding a maximum clique in a graph. Operations Research Letters 21, 211–217 (1997)

Appendix

Clique Benchmark Results

Type of Machine: Pentium 4 3.6 GHz, *Compiler and flags used:* gcc -O2.
Our user time (T_1) *for DIMACS benchmark instances:* `r100.5`, `r200.5`, `r300.5`, `r400.5`, and `r500.5` are 2.13×10^{-3}, 6.35×10^{-2}, 0.562, 3.48, and 13.3 seconds, respectively. From Östergård's [16] user time (T_2) and Sewell's [18] user time (T_3) for the same instances, we obtained the average values of T_2/T_1 and T_3/T_1 as 4.48 and 86.76, respectively, in the same way as in [23]. For Régin's [17] user time (T_5), we obtained the average value of T_5/T_1 to be 1.35 by referring to the $\chi + DF$ (Fahle's) [8] running time in [17].

On Some Simple Widths

Ling-Ju Hung and Ton Kloks*

Department of Computer Science and Information Engineering
National Chung Cheng University, Min-Hsiung, Chia-Yi 621, Taiwan
hunglc@cs.ccu.edu.tw

Abstract. The $\mathcal{G}$-width of a class of graphs $\mathcal{G}$ is defined as follows. A graph G has $\mathcal{G}$-width k if there are k independent sets $\mathbb{N}_1, \ldots, \mathbb{N}_k$ in G such that G can be embedded into a graph $H \in \mathcal{G}$ such that for every edge e in H which is not an edge in G, there exists an i such that both endpoints of e are in $\mathbb{N}_i$. For the class $\mathfrak{C}$ of cographs we show that $\mathfrak{C}$-width is NP-complete. We show that the recognition is fixed-parameter tractable, and we show that there exists a finite obstruction set. We introduce simple-width as an alternative for rankwidth and we characterize the graphs with simple-width at most two.

1 Cograph-Width

Definition 1 ([1]). *Let $\mathcal{G}$ be a class of graphs which contains all cliques. The* $\mathcal{G}$-width *of a graph G is the minimum number k of independent sets $\mathbb{N}_1, \ldots, \mathbb{N}_k$ in G such that there exists an embedding $H \in \mathcal{G}$ of G with the property that for every edge $e = (x, y)$ in H which is not an edge of G, there exists an i with $x, y \in \mathbb{N}_i$.*

Definition 2 ([2]). *A graph is a* cograph *if it has no induced P_4,* i.e. *an induced path with 4 vertices.*

Theorem 1 ([3]). *A graph $G = (V, E)$ is a cograph if and only if for every $W \subseteq V$ with $|W| \geq 2$, there exists a partition $\{W_1, W_2\}$ of W with $W_i \neq \varnothing$, $i = 1, 2$, such that either*

(a) *every vertex $x \in W_1$ is adjacent to every vertex $y \in W_2$, or*
(b) *no vertex $x \in W_1$ is adjacent to any $y \in W_2$.*

The $\mathcal{G}$-width has been investigated for blockgraphs, threshold graphs, and for trivially perfect graphs [4,5]. In this paper we investigate the width-parameter for the class $\mathfrak{C}$ of cographs, henceforth called the *cograph-width*, or $\mathfrak{C}$-width. If a graph G has cograph-width k then we call G also a *k-probe cograph*. We refer to the *partitioned case* of the problem when a collection of, possibly overlapping, independent sets $\mathbb{N}_i$, $i = 1, \ldots, k$ is a part of the input. We call such a collection of independent sets a *witness*.

* This author is supported by the National Science Council of Taiwan under grants NSC 97–2811–E–194–001 and NSC 98–2218–E–194–004.

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 204–215, 2010.

Theorem 2. *A graph $G = (V, E)$ is a k-probe cograph if and only if there exist independent sets $\mathbb{N}_i$, $i = 1, \ldots, k$, such that for every $W \subseteq V$ with $|W| \geq 2$ there exists a partition $\{W_1, W_2\}$ with $W_i \neq \varnothing$ $(i = 1, 2)$, such that either*

i. *for every $x \in W_1$ and $y \in W_2$ either $(x, y) \in E(G)$ or $x, y \in \mathbb{N}_i$ for some $i \in \{1, \ldots, k\}$, or*
ii. *no $x \in W_1$ is adjacent to any $y \in W_2$.*

Proof. Assume that G is a k-probe cograph. If G has at most 3 vertices then, by definition, G is a cograph and thus also a k-probe cograph for any $k \geq 0$. We prove the claim by induction on the number of vertices. By definition there exist independent sets $\mathbb{N}_i$, $i = 1, \ldots, k$ and an embedding $H \in \mathfrak{C}$ such that, for every edge $e = (x, y) \in E(H)$ which is not an edge in G there exists an $i \in \{1, \ldots, k\}$ with $\{x, y\} \subseteq \mathbb{N}_i$.

Let $W \subseteq V$ with $|W| \geq 2$. There exists a partition $\{W_1, W_2\}$ of W, with $W_i \neq \varnothing$, such that either (a) or (b) in Theorem 1 holds for the graph H. It follows that either i. or ii. holds for G with that same partition of W.

For the converse, let $W \subseteq V$ with $|W| \geq 2$. If i. holds, then we may add all edges between W_1 and W_2. Also, ii. implies (b) in Theorem 1. By induction, there exist embeddings of $G[W_1]$ and $G[W_2]$ into cographs, which implies by Theorem 1 that $G[W]$, and thus also G can be embedded. Thus G is a k-probe cograph. □

Theorem 2 is a characterization which can be formulated in monadic second-order logic. We now prove that k-probe cographs have *rankwidth* at most 2^k. It is known that problems which can be formulated in C_2MS-logic can be solved in $O(n^3)$ time on graphs with bounded rankwidth [6]. Thus k-probe cographs can be recognized in $O(n^3)$ time. Alternatively, we can draw the same conclusion from Theorem 6. An explicit description of a recognition algorithm is described in [7].

Theorem 3. *k-Probe cographs have rankwidth at most 2^k.*

Proof. Consider a rank-decomposition (T, τ) with width 1 for an embedding H of G. Consider an edge e in T and assume that M_e, minus the zero rows and columns is an all 1s-matrix. Each independent set $\mathbb{N}_i$ 'creates' a 0-submatrix in M_e. Note that there are at most 2^k different neighborhoods from the set of leaves in one component of $T - e$ to the set of leaves in the other component of $T - e$. It follows that the rank of M_e is at most 2^k. □

2 Partitioned k-Probe Cographs

Cographs are called 'complement-reducible graphs.' In the same vein, we show that partitioned probe cographs are 'dual reducible.'

Let $(G, \mathcal{N})$ be a partitioned graph with a witness

$$\mathcal{N} = \{\mathbb{N}_i \mid i = 1, \ldots, k\}$$

of k, possibly overlapping, independent sets.

Definition 3. *The* dual $(G^d, \mathcal{N})$ *is the partitioned graph obtained from* $\bar{G}$ *by deleting all edges* (x, y) *of* $\bar{G}$ *for which there is an* $i \in \{1, \ldots, k\}$ *such that* $x, y \in \mathbb{N}_i$.

Proposition 1. *A partitioned graph* $(G, \mathcal{N})$ *is a partitioned* k*-probe cograph if and only if its dual* $(G^d, \mathcal{N})$ *is likewise (with 'the same' witness* $\mathcal{N}$ *of* k *independent sets).*

Theorem 4. *There exists a polynomial-time algorithm to check whether a partitioned graph* $(G, \mathcal{N})$ *with a witness* $\mathcal{N}$ *of* k *independent sets, is a partitioned* k*-probe cograph.*

Proof. If G is disconnected then we can check each component individually. Assume G is connected. Assume also that G is a partitioned k-probe cograph and let H be an embedding of G. Then H is also connected and thus $\bar{H}$ is disconnected, since H is a cograph. Since the dual G^d is obtained from $\bar{H}$ by deleting some of the edges, also G^d is disconnected. By Proposition 1, G^d is a partitioned k-probe cograph and we can check that by inspecting the components of G^d. □

Remark 1. Note that the algorithm described in Theorem 4 is *fully* polynomial. The algorithm can be implemented to run in $O(kn^3)$ time.

There exists a list of forbidden induced subgraphs for the class of partitioned 1-probe cographs [8]. We extend this as follows. First we show that partitioned k-probe cographs are well-quasi-ordered by the induced subgraph relation.

Theorem 5. *Let* k *be a natural number. Partitioned* k*-probe cographs are well-quasi-ordered by the induced subgraph relation.*

Proof. A cotree is a binary tree with a bijection from the leaves to the vertices of the graph and internal nodes labeled as join- or union-operators [3]. Two vertices are adjacent in the graph if and only if their lowest common ancestor is a join-node. For partitioned k-probe cographs we equip each leaf with a label which is a 0/1-vector of length k. Two vertices are adjacent if their lowest common ancestor is a join-node and their labels have no common 1. Kruskal's theorem [9] states that the class of finite trees labeled by a well-quasi-ordering, is well-quasi-ordered with respect to this lowest-common-ancestor embedding. This proves the claim. (See [10] for a detailed discussion and extensions to the countable case.) □

Theorem 6. *Let* k *be a natural number. The class of* k*-probe cographs can be characterized by a finite set of forbidden induced subgraphs.*

Proof. Consider a sequence $[G_1, G_2, \ldots]$ of graphs such that no G_i is a k-probe cograph. Assume that for each vertex x in $G_i = (V_i, E_i)$ the subgraph induced by $V_i - x$ is a k-probe cograph ($i = 1, 2, \ldots$). Equip each G_i with a 'root' r_i which is a vertex of G_i.

For $i = 1, 2, \ldots$, consider labeled cotrees of $G_i - r_i$ as in Theorem 5. Extend the labels at the leaves with an additional entry 0 or 1 that indicates whether

the vertex that is mapped to the leaf is adjacent to r_i or not. Consider the well-quasi-ordering of these labeled trees by the lowest-common-ancestor ordering. Kruskal's theorem implies that there must exist $i < j$ such that G_i is an induced subgraph of G_j. This proves the theorem. □

Remark 2. Note that a similar proof shows that partitioned k-probe cographs can be characterized by a finite set of forbidden induced partitioned subgraphs. This extends the result of [8] for 1-probe cographs.

We introduce simple-width after the next intermezzo.

3 $\mathfrak{C}$-width Is NP-Complete

Let $\mathcal{T}$ be the class of complete graphs (cliques). We first note that $\mathcal{T}$-width is NP-complete.

Theorem 7 ([5]). *$\mathcal{T}$-Width is NP-complete.*

Theorem 8. *$\mathfrak{C}$-width is NP-complete.*

Proof. Let $G = (V, E)$ be a graph with n vertices and m edges. Label the vertices $1, \ldots, n$. For each vertex i of G add a clique C_i with n^2 vertices and make every vertex of C_i adjacent to each vertex ℓ with $\ell \leq i$. Let G' be the graph constructed in this way. Note that, when we add edges between nonadjacent vertices of V we obtain an embedding of G' into a cograph; namely then every connected induced subgraph has a universal vertex, and this property implies that. We show that this is the only feasible embedding.

For each nonedge $\{i, j\}$ with $i < j$ in G we now have a collection of P_4's using i, j, and the cliques C_i and C_j. Assume that there is an embedding of G' into a cograph with i and j not adjacent. Then each vertex of C_i is adjacent to each vertex of C_j or to j. Thus each vertex of C_i must be in one of the independent sets $\mathbb{N}_s$, $s = 1, \ldots, k$, and no two are in the same since C_i is a clique. Then $k \geq n^2$ which is a contradiction, because making a clique of G creates a cograph embedding which demands at most $\binom{n}{2} - m$ independent sets. Thus the only feasible embedding makes a clique of G. That is, the $\mathfrak{C}$-width of G' is the same as the $\mathcal{T}$-width of G, and by Theorem 7 this is hard to compute. □

4 Simple-Width

Inspired by results of Oum *et al.* [6] on rankwidth of graphs, we introduce the concept of simple-width.

We begin with the definition of a tree-decomposition of a graph.

Definition 4. *A* tree-decomposition *of a graph $G = (V, E)$ is a pair (T, f) where T is a ternary tree and f a bijection from the leaves of T to the vertices of G.*

Definition 5. *Let (T, f) be a tree-decomposition of a graph $G = (V, E)$. Let e be a line in T and consider the two sets A and B of leaves of the two subtrees of $T - e$. The* cutmatrix *M_e is the submatrix of the adjacency matrix of G with rows indexed by the vertices of A and columns indexed by the vertices of B.*

Note that we identify the cutmatrix with its transpose. Oum *et al.* define the rankwidth via the rank of the cutmatrix over $GF(2)$. Instead, we count the maximum of the number of different rows and the number of different columns.

Definition 6. *The* simple-width of a line *e in a tree-decomposition (T, f) is the maximum of the number of different rows of the cutmatrix M_e and the number of different columns of M_e. The* simple-width of *(T, f) is the maximum simple-width over all lines e in T. The* simple-width of *G is the minimum simple-width over all tree-decompositions of G.*

Remark 3. Simple-width is not clearly defined for graphs that consist of a single vertex. We now define the simple-width of a graph with a single vertex as zero. Any graph with at least two vertices has simple-width at least 1.

Example 1. Consider a 4-cycle C. Since there is only one ternary tree T with 4 leaves, there are only two tree-decompositions for C. Consider the central line e of T. One tree-decomposition maps an edge of C to one part of $T - e$ and the edge disjoint from it to the other part. The central line gets a cutmatrix $M_e = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$. The other tree-decomposition maps one diagonal of C to one part of $T-e$ and the other diagonal to the other part of $T-e$. This tree-decomposition has cutmatrix $M_e = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$. However, every vertex of C has at least one neighbor and at least one nonneighbor. Thus, if e' is incident with a leaf of T, then e' has simple-width two. Thus the simple-width of C is two.

The consensus is that a tree-decomposition with a smaller width is more desirable.

Obviously, if a graph G has a tree-decomposition with simple-width k, then this is also a tree-decomposition with rankwidth at most k. Thus the rankwidth of a graph is at most the simple-width. Conversely, if the rankwidth of a graph is k, then the vector space over $GF(2)$ spanned by the columns of a cutmatrix, has a basis with k elements. Consequently, the number of different columns is at most 2^k because there are only 2^k different linear combinations of the k basisvectors with coefficients in $GF(2)$. Since row-rank equals column-rank, we have also at most 2^k different rows.

Theorem 9. *For any graph G*

$$r(G) \leq s(G) \leq 2^{r(G)}$$

where $s(G)$ is the simple-width and $r(G)$ is the rankwidth of G.

It is easy to see that these bounds are tight.

The graphs of rankwidth at most one are exactly the distance-hereditary graphs [6]; this is the smallest, and only recognized case for rankwidth. To get a feeling for simple-width we study the class of graphs with simple-width at most two.

Basics

Definition 7. *Consider a* $0,1$*-matrix* M*. Let* M' *be the maximal submatrix of* M *with no two rows equal and no two columns equal. The* shape of M *is a class of matrices equivalent to* M' *under permuting rows, permuting columns, and taking the transpose.*

One basic concept that will be useful throughout the rest of this paper is that of a twin. A twin is a module with two vertices. A module is a set M of vertices such that

$$x, y \in M \quad \Rightarrow \quad N(x) - M = N(y) - M.$$

Definition 8. *A* twin *is a pair of vertices* x *and* y *such that* $\{x, y\}$ *is a module. The pair is a* true twin *if* x *and* y *are adjacent and a* false twin *otherwise.*

Cographs can be characterized as those graphs for which every nontrivial induced subgraph has a twin [11]. Tibor Gallai proved that a (finite) graph G is a cograph if and only if G has a tree-decomposition (T, f) rooted at an edge, such that the leaves of every subtree induce a module in G. Consider a subtree B rooted at some edge e and consider the cutmatrix M_e with rows indexed by the vertices of B. Since the leaves of B induce a module, all the rows of M_e are the same. Thus M_e has a shape equivalent to $\begin{pmatrix} 1 & 0 \end{pmatrix}$ or a submatrix thereof. This implies the following:

Theorem 10. *If* G *is a cograph then the simple-width of* G *is at most 2.*

Apropos, a more precise characterization is the following.

Lemma 1. *A graph* G *is a cograph if and only if* G *has a tree-decomposition* (T, f) *such that every cutmatrix, or its transpose, can be reduced to* $\begin{pmatrix} 1 & 0 \end{pmatrix}$*, or a submatrix of that, by removing copies of the same row or column and by permuting columns.*

Proof. The direction $\Rightarrow$ follows from Gallai's characterization.
Assume G has a tree-decomposition (T, f) as stated. We show that every nontrivial induced subgraph H has a twin. If H has only two or three vertices then the claim is obvious: H is a cograph since it has no induced P_4. Otherwise, notice that a tree-decomposition for H can be obtained as follows. Remove branches from T if the leaves are mapped to vertices which are not in H. This gives a tree T' with possibly some internal vertices of degree 2; we could easily get rid of those but we may as well leave them in T' since they harm nobody. Consider a line e in T' such that both branches have at least 2 leaves; since H has at least 4 vertices, such a line exists. The cutmatrix M'_e is a submatrix of the corresponding cutmatrix M_e for G, since it is obtained from M_e by the deletion of some rows and columns. Thus M'_e or its transpose has a shape $\begin{pmatrix} 1 & 0 \end{pmatrix}$ or $\begin{pmatrix} 0 \end{pmatrix}$ or $\begin{pmatrix} 1 \end{pmatrix}$. In other words, at least one of the branches at e, say B induces a module. We may assume, by induction on the number of vertices of H, that the leaves of B induce a cograph, and because B contains at least 2 vertices, it contains a twin. Since B is a module, this twin is a twin in H. □

Howorka defines distance-hereditary graphs $G = (V, E)$ as follows [12]:

> A graph is distance hereditary if for each nonadjacent pair of vertices x and y in any connected subgraph, all induced x, y-paths in the graph have the same length.

We have the following characterization.

Theorem 11 ([13]). *A graph G is distance hereditary if and only if every induced subgraph has a twin, or a pendant vertex, or an isolated vertex.*

Equivalently, a graph is distance hereditary if and only if it is *split decomposable* and this is actually the same statement as the corollary below.

Corollary 1 ([6]). *A graph $G = (V, E)$ is distance hereditary if and only if G has a tree-decomposition (T, f) such that every cutmatrix M_e has a shape which is a submatrix of $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$, where the 1 stands for an all ones block and the zeros stand for all zero blocks. Thus distance-hereditary graphs have simple-width at most 2.*

Remark 4. 'Having the same shape' is an equivalence relation. Also notice this monotonicity property: if a matrix M' is a submatrix of a matrix M, then the number of different rows (columns) of M' cannot be more than the number of different rows (columns) of M; thus, loosely speaking, the shape of M' is a submatrix of the shape of M.

Lemma 2. *The complement of a distance-hereditary graph has simple-width at most 2.*

Proof. Let $G = (V, E)$ be distance hereditary. Let (T, f) be a tree-decomposition of G such that for every line e in T the cutmatrix M_e has rank ≤ 1; *i.e.* after the deletion of zero-rows and zero-columns, all remaining entries are 1. In other words M_e is equivalent to a submatrix of $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$.

Now consider the complement $\bar{G}$ of G. Any cutmatrix M_e of (T, f) changes into the complement $\bar{M}_e$. Thus $\bar{M}_e$ is equivalent to a submatrix of $\left(\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right)$. This proves the claim. □

In general we have that for any graph G

$$s(G) = s\left(\bar{G}\right).$$

Remark 5. Notice that the rank of $\left(\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ is 2 while the rank of $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$ is one; the rankwidth of the complement of a distance-hereditary graph is (in general) 2.

5 Graphs with Simple-Width 2

Lemma 3. *Let G be a graph with simple-width 2. Let (T, f) be a tree-decomposition realizing this width. Then every cutmatrix is equivalent to one of the following:*

$$(0), \quad (1), \quad (1\ 0), \quad \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \quad \text{or} \quad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Proof. A simple case-analysis proves this. □

Lemma 4. *If H is an induced subgraph of G then*

$$s(H) \leq s(G).$$

Proof. If H is only a single vertex then there is nothing to prove. Otherwise, consider a tree-decomposition (T, f) for G and take the minimal subtree T' that spans all leaves mapped to vertices of H. Take the induced bijection f' from the leaves of T' to the vertices of H. Some points of T' may have only two neighbors. We may repeatedly delete a point like that and connect its two neighbors by a line. For simplicity we call this new tree, which is homeomorphic to the old one, again T'. Any cutmatrix of (T', f') is a submatrix of a corresponding cutmatrix of (T, f), thus the simple-width of H is no larger than the simple-width of G. □

Lemma 5. *Let G be nontrivial. Then $s(G) = 1$ if and only if G is a clique or an independent set. 'Creating a twin' of a vertex x in a graph G is the operation of adding a new vertex x' and adding edges incident with x' such that x' becomes a twin of x. Assume G is nontrivial, and not an independent set nor a clique and let G' be obtained from G by creating a twin, then*

$$s(G') = s(G) \geq 2.$$

Proof. Since G is an induced subgraph of G', $s(G) \leq s(G')$.
If a graph G is not a clique and not an independent set then it has a vertex ω which has some neighbors as well as some nonneighbors. Consider the leaf in a tree-decomposition that is mapped to ω and let e be the line incident with this leaf. This cutmatrix has one row (or one column) and some entries are zero and some are one. Thus the simple-width of this cutmatrix is two. This proves that

$$s(G) \geq 2.$$

Consider a tree-decomposition (T, f) of G. We may add x' as a leaf in this decomposition, by subdividing the edge from x to its neighbor in T, and making x' adjacent to the subdivision vertex. The cutmatrix of any edge in T that is not incident with x or with x' is obtained from the corresponding matrix in (T, f) by making a copy of the row (or column) that represents x. The cutmatrix of an edge incident with x or with x' contains only one row (or only one column), thus the simple-width of this cutmatrix is at most two. This proves the claim. □

Lemma 6. *Assume that G is nontrivial and that $s(G) \leq 2$. Assume furthermore that G has no twin, no pendant vertex, and no isolated vertex. Then G has two vertices x and y such that $N[x] \cup N[y] = V$ and either*

$$N(x) \cap N(y) = \varnothing \quad \text{or} \quad N(y) \subset N[x].$$

Proof. Since G is nontrivial and has no twin, G is not a clique nor an independent set. Thus we may assume that $s(G) = 2$. Also, G has at least 4 vertices, otherwise it has a twin.

Consider a tree-decomposition (T, f) of G. It is well-known that any ternary tree T with at least three leaves has two leaves that have the same neighbor. Let x and y be the vertices mapped to two such leaves. Then the tree T has a line e with a cutmatrix M_e which has two rows x and y. Since x and y are not twins, and neither x nor y is isolated or pendant, M_e is equivalent to

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \text{or to} \quad \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

and this proves the claim. □

Remark 6. The second case occurs when $\bar{G}$ is distance hereditary. The first case sprouts a 'new' class of graphs that we baptize as *the class of 2-cographs*.

2-Cographs

Lemma 7. *Let $G = (V, E)$ be a graph and assume that the vertices of G can be colored black and white such that for every $W \subseteq V$ with at least two vertices there is a partition into nonempty subsets W_1 and W_2 such that either*

(a) *$w_1 \in W_1$ and $w_2 \in W_2$ are adjacent if and only if they have the same color, or*
(b) *$w_1 \in W_1$ and $w_2 \in W_2$ are adjacent if and only if they have different colors.*

Then $s(G) \leq 2$.

Proof. We claim that there is a rooted binary tree-decomposition with the following property π: Let T' be any subtree and let e be the line connecting the root of T' with its parent. Let M_e be the cutmatrix with rows indexed by the vertices in the leaves of T'. Then M_e is equivalent to a submatrix of $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ such that the row $\left(\begin{smallmatrix} 1 & 0 \end{smallmatrix}\right)$ represents the rows of the black vertices in T' and the row $\left(\begin{smallmatrix} 0 & 1 \end{smallmatrix}\right)$ represents the rows of the white vertices in T'.

Let $\{V_1, V_2\}$ be a partition of V into nonempty subsets V_1 and V_2 as stated. We may assume that there exist rooted tree-decompositions (T_1, f_1) and (T_2, f_2) for $G[V_1]$ and $G[V_2]$ as claimed. Create a new root r , and make the roots of T_1 and T_2 the two children of r. This creates the tree-decomposition (T, f) for G. We claim that this tree-decomposition satisfies the stated property π and that it has simple-width 2.

Any of the two lines incident with r has a shape which is equivalent to a submatrix of $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ by the assumption on the partition. (It is a proper submatrix if one of V_1 and V_2 has only black or only white vertices.) Now consider a line e of T_1 and let B_1 and B_2 be the two branches of $T - e$ such that B_2 contains r. Consider the cutmatrix M_e of (T_1, f_1) with rows indexed by the vertices of B_1. According to the induction assumption M_e has a shape as above. The vertices of V_2 are added to this matrix as columns. By the assumption on the partition $\{V_1, V_2\}$ the shape of the new matrix M_e is as above, also. □

Remark 7. Note that a cograph has a coloring like that; just color all vertices black. Also note that in a graph with a coloring like that, the black vertices, and also the white vertices, induce a cograph.

Let us call graphs that have a 2-coloring as in Lemma 7 2-cographs.

Lemma 8. *There exists an $O(n^3)$ algorithm to recognize 2-cographs.*

Proof. The characterization can be formulated in monadic-second order logic. The claim follows since 2-cographs have bounded rankwidth [6]. □

Lemma 9. *Assume G has a tree-decomposition (T, f) with simple-width 2. Let x and y be two vertices mapped to leaves that have a common neighbor in T. Assume the cutmatrix with rows x and y has a shape $\left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right)$. Color vertices of $N(x)-y$ black and color vertices of $N(y)-x$ white. Then x and y can be colored differently black and white, such that this coloring of G satisfies the property of Lemma 7.*

Proof. Consider (T, f) and root this at the common neighbor r of the leaves mapped to x and y. Let e be the line that is incident with r but not with x or y and let V_1 and V_2 be the two sets of leaves in the two subtrees rooted at the other endpoint r' of e.

Let e_1 be the line incident with r' and the root of V_1. Consider the matrix M_{e_1} with rows indexed by vertices of V_1. Two of the columns are x and y. Thus they partition the rows into black and white (possibly one set is empty). In any case, x and y provide two different columns and since $s(G) \leq 2$ every other column, corresponding to a vertex of V_2, is equal to the column of x or to the column of y. It follows that all black vertices of V_1 have the same neighbors in V_2 and that all white vertices of V_1 have the same neighbors in V_2. By symmetry, the same holds for the vertices of V_2.

Consider the submatrix with rows V_1 and columns V_2. We consider two cases;

(a) either the black vertices of V_1 are adjacent to the black vertices of V_2 and the white vertices of V_1 are adjacent to the white vertices of V_2, or
(b) the black vertices of V_1 are adjacent to the white vertices of V_2 and the white vertices of V_1 are adjacent to the the black vertices of V_2.

In the first case we color x black and y white and in the second case we color x white and y black. Then in both cases any of the two partitions $\{V_1 + \{x, y\}, V_2\}$ and $\{V_1, V_2 + \{x, y\}\}$ is 'good;' that is, they satisfy the conditions of Lemma 7. We prove that, with this coloring, every induced subgraph has a good partition.

Let $W \subseteq V$. If W has vertices in both V_1 and V_2, and possibly also in $\{x, y\}$, then we can take the partition induced by one of the two partitions above.

Assume W has no vertices in V_2. If x and/or y is in W, then we can take one part of the partition equal to $W \cap \{x, y\} \neq \varnothing$. This is a good partition as long as $W - \{x, y\} \neq \varnothing$.

Assume that W contains only vertices of V_1. Take the root t of the minimal subtree that spans all vertices of W. The two subtrees at t partition W into W_1

and W_2. Consider the edge e^* incident with t and the subtree of W_1. Consider the cutmatrix M_{e^*} with the vertices of W_1 as rows. Then all black vertices of W_1 have the same neighbors in the rest of the graph and all white vertices of W_1 have the same neighbors in the rest of the graph, since this is dictated by the two different columns of x and y. The same holds for the vertices of W_2 and this proves that the edges between W_1 and W_2 satisfy the condition of Lemma 7. □

Theorem 12. *Assume G has simple-width at most 2. Then either*

(i) *G is distance hereditary, or*
(ii) *$\bar{G}$ is distance hereditary, or*
(iii) *G is a 2-cograph.*

Proof. Consider a tree-decomposition (T, f) of G with simple-width 2. Assume T has a line e such that the cutmatrix M_e has a shape equivalent to $\left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right)$. Let $\{V_1, V_2\}$ be the induced partition of V and let the rows of M_e be indexed by V_1. Obviously, V_1 and V_2 both have at least two vertices. Consider a pair of leaves x and y in V_1 with a common neighbor in T. Consider the cutmatrix $M_{e'}$ with rows x and y. Neither of the rows can consist of only zeros, since this would imply a zero-row in M_e as well. It can also not contain a row with only ones, since this would imply a similar row in M_e. It follows that either x and y are twins, or that $M_{e'} = \left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right)$. If there is a pair of leaves such that the cutmatrix has this shape, then the graph is a 2-cograph, by Lemma 9. This proves that the graph is obtained from a 2-cograph by creating twins. It is easy to see that the class of 2-cographs is closed under the operation of creating twins.

We may now assume that (T, f) has no cutmatrix with shape $\left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right)$. Assume that there is a line e such that the cutmatrix M_e has a shape $\left(\begin{smallmatrix}1&1\\1&0\end{smallmatrix}\right)$. Let $\{V_1, V_2\}$ be the partition induced by the rows and columns and assume that the rows of M_e are indexed by V_1. Consider any other line e' of T, say in the branch containing V_1, and assume that the cutmatrix $M_{e'}$ is equivalent to $\left(\begin{smallmatrix}1&0\\0&0\end{smallmatrix}\right)$. We may assume that V_2 is a subset of the columns of $M_{e'}$. But this is a contradiction since V_2 has a vertex adjacent to all vertices of V_1 so $M_{e'}$ must have an all 1s column.

It follows that every cutmatrix is a submatrix of $\left(\begin{smallmatrix}1&1\\1&0\end{smallmatrix}\right)$, in other words, $\bar{G}$ is distance hereditary.

The only remaining case occurs when G is distance hereditary. □

Corollary 2. *There exists an efficient algorithm for the recognition of graphs with simple-width at most 2.*

A graph is a 2-cograph if and only if it has no induced C_5, bull, gem, or co-gem, *i.e.*, the switching class of C_5; see [14] for an alternative description of this class of graphs. Note that the class is closed under switching [15]. If we let G_x be the graph in the switching class of G for which the vertex x is isolated, then G is a 2-cograph if and only if every, or also if some, G_x is a cograph. Alternatively, we have that a graph is a 2-cograph if and only if every nontrivial induced subgraph has a twin or an antitwin. We conclude that 2-cographs are perfect because minimal imperfect graphs have no twins nor antitwins [16,17]. Thus graphs with simple-width at most 2 are perfect.

References

1. Chandler, D.B., Chang, M.S., Kloks, T., Peng, S.L.: Probe graphs (manuscript, 2009), http://www.cs.ccu.edu.tw/~hunglc/ProbeGraphs.pdf
2. Gallai, T.: Transitiv orientierbare graphen. Acta Math. Acad. Sci. Hungar. 18, 25–66 (1967)
3. Corneil, D.G., Lerchs, L.H.: Stewart-Burlingham: Complement reducible graphs. Discrete Applied Mathematics 3, 163–174 (1981)
4. Chang, M.S., Hung, L.J., Kloks, T., Peng, S.L.: Block-graph width. In: Chen, J., Cooper, S.B. (eds.) TAMC 2009. LNCS, vol. 5532, pp. 150–157. Springer, Heidelberg (2009)
5. Hung, L.J., Kloks, T., Lee, C.M.: Trivially-perfect width. In: Fiala, J., Kratochvíl, J., Miller, M. (eds.) IWOCA 2009. LNCS, vol. 5874, pp. 301–311. Springer, Heidelberg (2009)
6. Oum, S.: Graphs of bounded rank-width. PhD thesis, Princeton University (2005)
7. Hung, L.J., Kloks, T.: On the cograph-width of graphs (manuscript, 2009)
8. Le, V.B., de Ridder, H.N.: Characterizations and linear-time recognition of probe cographs. In: Brandstädt, A., Kratsch, D., Müller, H. (eds.) WG 2007. LNCS, vol. 4769, pp. 226–237. Springer, Heidelberg (2007)
9. Kruskal, J.: Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture. Transactions of the American Mathematical Society 95, 210–225 (1960)
10. Thomassé, S.: On better-quasi-ordering countable series-parallel orders. Transactions of the American Mathematical Society 352, 2491–2505 (2000)
11. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A survey. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia (1999)
12. Howorka, E.: A characterization of distance-hereditary graphs. Quarterly Journal of Mathematics 28, 417–420 (1977)
13. Chang, M.S., Hsieh, S.Y., Chen, G.H.: Dynamic programming on distance-hereditary graphs. In: Leong, H.-V., Jain, S., Imai, H. (eds.) ISAAC 1997. LNCS, vol. 1350, pp. 344–353. Springer, Heidelberg (1997)
14. Cameron, P.J.: Two-graphs and trees. Discrete Mathematics 127(1-3), 63–74 (1994)
15. Seidel, J.J.: Geometry and combinatorics. In: Corneil, D.G., Mathon, R. (eds.) Selected works of J. J. Seidel. Academic Press, London (1991)
16. Lovász, L.: Normal hypergraphs and the weak perfect graph conjecture. Discrete Mathematics 2, 253–267 (1972)
17. Olariu, S.: No antitwins in minimal imperfect graphs. Journal of Combinatorial Theory, Series B 45, 255–257 (1988)

A New Model for a Scale-Free Hierarchical Structure of Isolated Cliques*

Takeya Shigezumi[1], Yushi Uno[2], and Osamu Watanabe[1]

[1] Tokyo Institute of Technology, Tokyo 152-8552, Japan
[2] Osaka Prefecture University, Sakai 599-8531, Japan

Abstract. Scale-free networks are usually defined as the ones that have power-law degree distributions. Since many of real world networks such as the World Wide Web, the Internet, citation networks, biological networks, and so on, have this property in common, scale-free networks have attracted interests of researchers so far. They also revealed that such networks have some typical properties such as high cluster coefficient and small diameter as well, and a lot of network models have been proposed to explain them. Recently, some new observations for a real world network are reported [12]. It tries to find a special kind of cliques from a network and introduces observations; 1. the size distributions of cliques show a power-law, 2. the degree distribution of the network after contracting those cliques show a power-law, and 3. by regarding the contracted network as the original, 1 and 2 are observed repeatedly. In this paper, we propose a new network model constructed by a 'clique expansion' procedure, to explain these new hierarchical structure of cliques.

Keywords: scale-free network, isolated cliques, webgraph, web structure modelling.

1 Introduction

Cluster structures have been observed on many real world networks. A community structure that is often seen in large web networks is one of the typical examples of such cluster structures, but it seems to have some specific structural property. In order to analyze this property, Uno et al. [12] adopted "isolated cliques" and investigated the distribution and the structure of isolated cliques in some large web networks. An *isolated clique* (of size k) [5] is a clique consisting of k nodes that does not have more than k edges to its outside (see the next section for the precise definition). That is, an isolated clique is, while it is maximally dense in its inside, sparsely connected to its outside. Furthermore, there is an efficient algorithm [5] that can extract all of isolated cliques from a given graph. Uno et al. used this algorithm to analyze an undirected graph (which we call a "webgraph" here) representing some web network links, and they found some interesting properties that are summarized as follows.

* This research was supported in part by JSPS Global COE program "Computationism as a Foundation for the Sciences".

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 216–227, 2010.

Observation 1. The size distribution of isolated cliques in the webgraph follows a power-law distribution with an exponent that is larger than the exponent for the degree distribution.

Observation 2. Contract each isolated clique to one node and obtain a reduced graph. Then the degree distribution of this reduced graph follows follows almost the same power-law as the degree distribution of the original graph. Furthermore, the reduced graph has again many isolated cliques whose size distribution follows the power-law with almost the same or larger exponent as the isolated clique size distribution of the original graph.

Observation 3. This contraction can be conducted for several times (at least five times) until the number of isolated cliques becomes very small. Then in those reduced graphs, more or less almost the same degree distribution and isolated clique size distribution can be observed (Figure 1).

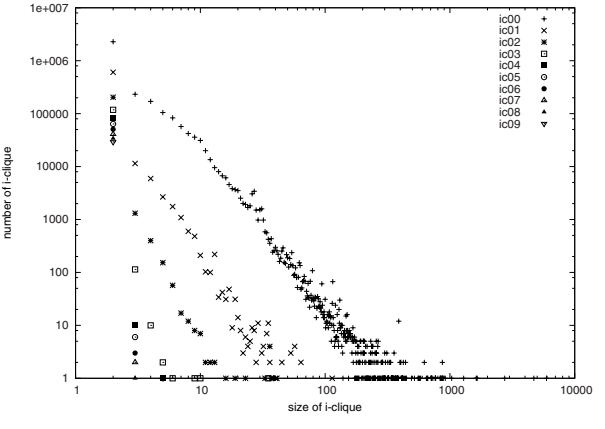


Fig. 1. The size distribution of isolated cliques on the reduced graph

We may call this observed structure as *hierarchical clique structure*. Let us also call the final reduced graph that has no isolated cliques as a *prime network*. Although many scale-free network models have been proposed to explain networks in the real world, e.g., [1,7], most of them can only generate graphs without large cliques (not to mention, isolated cliques), and up until now, no models have been proposed for the hierarchical clique structure. Recently, a different type of some hierarchical structure, called a fractal property, has been also studied by Song et al. [10]. They observed that the power-law degree distribution on the reduced graph obtained by contracting randomly and greedily chosen connected subgraph. They also proposed a model to represent this fractal property [11], however, that model generates a tree so has neither cliques nor hierarchical clique structure. On the other hand, it may be possible that this hierarchical clique structure and the structure of a prime network are independent. The purpose of this paper is to provide some model or method for adding the hierarchical clique structure to any given scale-free network. Thus, for example, we may use the BA model by Barabási and Albert [3] as a prime network model, and based

on it a network with the hierarchical clique structure can be constructed by our method.

For explaining some of the features of our method, we introduce some basic notations (see the next section for their precise definitions). For a given graph W, its reduced graph $\mathcal{C}(W)$ is a graph obtained by contracting all isolated cliques of W into one vertex, where the contraction is made as shown in Figure 2.

Let W^0 denote the original webgraph and define $W^1 = \mathcal{C}(W^0)$, $W^2 = \mathcal{C}(W^1)$, ..., and so on. Uno et al. [12] observed that W^i follows almost the same power-law degree distributions as W^0 for several times (at least five times), and its isolated clique size distributions follows a power-law distribution with slightly larger exponent than W^0.

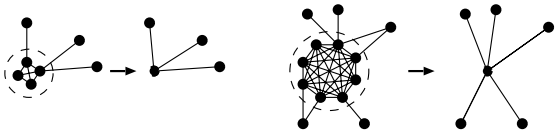


Fig. 2. Examples of the contraction of an isolated clique

Now our method is, roughly speaking, to use some randomized procedure to create $\mathcal{E}(G)$ from a given graph G so that (i) both G and $\mathcal{E}(G)$ follow the same degree distribution, and (ii) $\mathcal{E}(G)$ contains isolated cliques whose size distribution follows the power-law distribution with exponent that is about $+1$ larger than the one for the degree distribution (of G). Consider a graph G^0 that is obtained by any model for scale-free networks (where we may assume that no isolated clique exists in G^0), and define $G^1 = \mathcal{E}(G)$, $G^2 = \mathcal{E}(G^1)$, ..., to G^t for some sufficiently large t. Then we show that the graph $W^0 \triangleq G^t$ has the following property;that is, each W^i that is obtained from this W^0 by the contraction follows the same power-law degree and isolated clique size distributions as W^0.

Technically an interesting point in our analysis is that $\mathcal{C}(\cdot)$ is not necessarily the inverse of $\mathcal{E}(\cdot)$. Thus, the fact that W^i has the desired degree and isolated clique size distributions is not immediate from the above properties (i) and (ii) of $\mathcal{E}(\cdot)$.

The organization of this paper is as follows. We give basic definitions of graphs, scale-free property and basic notations in Section 2. We explain our model precisely in Section 3, and give analysis in Section 4. In Section 5, we give some previous and related works. Finally, we conclude the paper giving some future topics in Section 6.

2 Preliminaries

Throughout this paper, we consider only simple undirected graphs without multiple edges and self loops, and we denote a graph as $G = (V, E)$, where V is a set of vertices and E is a set of unordered pairs $e = \{u, v\}$ of V denoting edges. For any graph $G = (V, E)$, let $V[G] = V$ and $E[G] = E$ denoting the set of vertices and edges respectively. For any vertex $v \in V$, a vertex u is called

adjacent to v if there is an edge $\{u, v\}$ in E. The *neighborhood* of a vertex v is a set $N_G(v) = \{u \in V[G] \mid \{u, v\} \in E[G]\}$, i.e., the set of adjacent vertices of v in G. The *degree* of v is $|N_G(v)|$, which is denoted by $d_G(v)$ and the *maximum degree* of G is $\max_{v \in V} d_G(v)$ and denoted by Δ. A graph $G' = (V', E')$ is called a subgraph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq (V' \times V') \cap E$. A subgraph of G is called a *clique* if every pair of vertices in this subgraph has an edge between them. A clique C is called *c-isolated* if the number of *outgoing* edges from $V(C)$ to $V \backslash V(C)$ is less than $c|V(C)|$. Although finding large cliques in a graph is intractable, finding isolated cliques is not so hard. Furthermore, 1-isolated clique can be enumerated in linear time [5], and it is investigated in [12]. Note that very few overlaps occur among 1-isolated cliques and they are easy to be separated. We consider a process of *contracting an isolated clique* of G into one vertex. We use $\mathcal{C}(G)$ to denote a reduced graph obtained from G by contracting all isolated cliques in G.

The scale-free property is considered as one of the basic properties characterizing real world large graphs. We say that G is 'scale-free' if its degree distribution follows power-law, i.e., a distribution proportional to $k^{-\gamma}$ for some constant γ. Let us make these notions more precise for our discussion. The *degree distribution* of G is a sequence $\{\frac{n_k}{n}\}_{k \geq 1}$, where n_k is the number of vertices with degree k and $\frac{n_k}{n}$ is the ratio of them among n vertices in G. Then we say that G's degree distribution follows a *power-law* if $n_k/n = \Theta(k^{-\gamma})$ for some γ, that is, there are some constants c_1 and c_2 such that $c_1 k^{-\gamma} \leq n_k/n \leq c_2 k^{-\gamma}$ for all $1 \leq k \leq \Delta$. In this paper, we extend this notion to isolated clique size distributions. The *isolated clique size distribution* of G is a sequence $\{\frac{m_s}{m}\}_{s \geq 1}$, where m_s is the number of isolated cliques of s vertices and m is the total number of isolated cliques. We say that G's isolated clique size follows a *power-law* if the sequence $\{\frac{m_s}{m}\}_{s \geq 1}$ satisfies $m_s/m = \Theta(s^{-\gamma})$ for some γ.

It does not make sense for discussing the above properties for any fixed finite graph G. Thus, in this paper, we will consider a family of graphs consisting of infinite number of graphs defined in a certain way and discuss power-law properties with constants c_1 and c_2 that are independent from k and the choice of a graph in the family. Thus, when claiming for example that G's degree distribution follows a power-law with some exponent γ, we formally imply that its degree sequence $\{n_k/n\}_{k \geq 1}$ satisfies $n_k/n = \Theta(k^{-\gamma})$ under some fixed constants c_1 and c_2 for all graphs in our assumed graph family.

3 Model

The main idea of our model is as follows. Let G^0 be a prime scale-free graph generated by a certain scale-free model, e.g., BA model. Consider that a vertex in G^0 is either a "node" that represents an 1-isolated clique or a "(simple) vertex", otherwise. We decide whether a vertex in G^0 is a "node" or a simple vertex randomly. We replace each node by an isolated clique whose size is the same as the degree of the original node as shown in the Figure 3. Then we regard these new vertices in the isolated clique could be "nodes" or vertices, so, we decide

them recursively. In order to technically simplify our analyses and discussions, we here replace 1-isolated cliques by $(1+\epsilon)$-isolated cliques. However, remark that we can obtain almost similar results even if we used 1-isolated clique. From now, we consider $(1+\epsilon)$-isolated cliques and we simply call them *isolated cliques*.

We now explain this idea precisely. Let $G^0 = (V^0, E^0)$ and a parameter p_0 be inputs of our model. Let us assume G^0 is a prime scale-free graph, i.e., any cliques of G^0 must be non-isolated and its degree distribution follows a power-law. From a given graph G^0, we expand it to G^i recursively and randomly. For $G^i = (V^i, E^i)$ $(i \geq 0)$, consider two subsets U^i and A^i of V^i such that $A^i \subseteq U^i \subseteq V^i$, where A^i denotes a set of "nodes" and U^i denotes a set of candidates of being "nodes". At the first step, all of vertices of G^0 are candidate, i.e., $U^0 = V^0$. First, decide a set of contracted isolated cliques $A^i \subseteq U^i$ randomly. Consider a vertex v in U^i with degree k. We choose v into A^i with probability $p_k = \frac{p_0}{k}$. It is independent to the choice of other vertices. Second, for each $v \in A^i$, let C_v be a clique of size $k = d_{G^i}(v)$. Let us define $G^{i+1} = (V^{i+1}, E^{i+1})$ and U^{i+1} as follows.

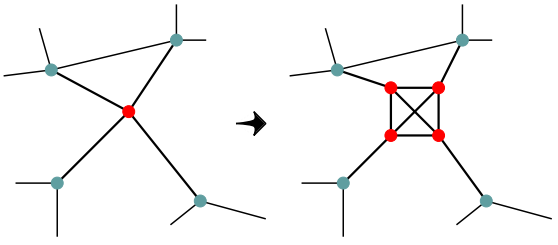


Fig. 3. Replacing a "node" of degree 4 by an isolated clique of size 4

$$
\begin{aligned}
V^{i+1} &= V^i - A^i + \bigcup_{v \in A^i} V[C_v], \\
E^{i+1} &= \left\{ \{u, v\} \in E^i \mid u, v \in (V^i - A^i) \right\} \\
&\quad + \bigcup_{v \in A^i} \left(E[C_v] + \bigcup_{i=1}^{d_{G^i}(v)} \{\{u_i, v_i\} \mid (*)\} \right), \\
&((*) : \{u_1, \ldots, u_{d_{G^i}(v)}\} = N(v) \text{ and } \{v_1, \ldots, v_{d_{G^i}(v)}\} = V[C_v].) \\
U^{i+1} &= \bigcup_{v \in A^i} V[C_v].
\end{aligned}
$$

Let us denote above expansion procedure by a function $\mathcal{E}(\cdot)$, i.e., $(G^{i+1}, U^{i+1}) = \mathcal{E}(G^i, U^i)$ for any $i \geq 0$. In this paper, we always set $U^0 = V^0$, so the obtained $(G^1, U^1), (G^2, U^2), \ldots$ is a sequence of random graphs. We omit U^i and simply write them as $G^i = \mathcal{E}(G^{i-1})$ if no confusion arises.

When $A^t = \emptyset$ for some t, we let $H = G^t$ be an output of our model. We choose the parameter p_0 as $p_0 < 1$, since otherwise, t may become infinite with positive probability. (The recursive procedure will not stop with positive probability.) This can be obtained by the classical analysis of the branching processes. (See

a literature e.g. [2].) Note that throughout our expansion procedure, the degree of vertices expanded from a vertex is the same as the original one.

4 Analysis

Here we consider one vertex in V^0 and investigate the number of vertices in V^i expanded from this vertex. It is easy to see that the process of generating vertices from this vertex obeys the following branching process (as known as Galton-Watson process) starting with one node. (i) start from a single node that is in $U^0 (= V^0)$. (ii) at each step i, on each node in U^i, the decision of "expansion" is made with probability p_k independently, where k is the degree of the starting vertex on G^0 and $p_k = \frac{p_0}{k}$; (iii) those decided not to expand become leaves, and those decided to expand become inner nodes after adding new k children that are regarded as nodes in U^{i+1}; and (iv) repeat (ii) and (iii) until no open node exists. Let T denote a tree generated by this expansion process. Note that we should consider forest $\{T_v\}_{v \in V^0}$, a set of trees starting from each node $v \in V^0$, for the analysis of the number of nodes or the number of isolated cliques. However, we will focus on one tree since each tree is created independently random and the number of total nodes or isolated cliques are the sum of them in each tree.

It is well known that if $p_k k < 1$, then T is finite with probability 1 (see e.g. [4]). We defined $p_0 < 1$ and thus $p_k k = p_0 < 1$ in our model, so our expansion procedure generates a finite tree with probability 1.

The initial node is called a *root* node and a node with no child node is called a *leaf* node. For each node v of T, we define its height $h(v)$ and level $l(v)$ inductively as follows.

$$h(v) = \begin{cases} 0, & \text{if } v \text{ is a root node, and} \\ h(v') + 1 & \text{where } v \text{ is a parent node of } v; \end{cases}$$

$$l(v) = \begin{cases} 0, & \text{if } v \text{ is a leaf node, and} \\ \max\{l(v_1), \ldots, l(v_k)\} + 1 & \text{where } v_1, \ldots, v_k \text{ are a child nodes of } v. \end{cases}$$

The height of a tree is the maximum height of nodes in T and note that the height of a tree equals to the level of the root node of the tree.

An example of a tree representing an expansion procedure and corresponding height and level of nodes are shown in Figure 4. Let $H^0 = H$ and $H^1 = \mathcal{C}(H^0)$, $H^2 = \mathcal{C}(H^1), \ldots$, and so on. As shown in Figure 4, we can easily obtain the following observation.

Observation 4. The number of leaves (which has level 0) in the tree represents the number of nodes in $G^t (= H^0)$ expanded from one vertex; and for $l \geq 1$, the number of nodes in the tree with level l represents the number of isolated cliques in H^{l-1} expanded from the vertex. The number of nodes in the tree with height i represents the number of nodes in G^i expanded from the vertex.

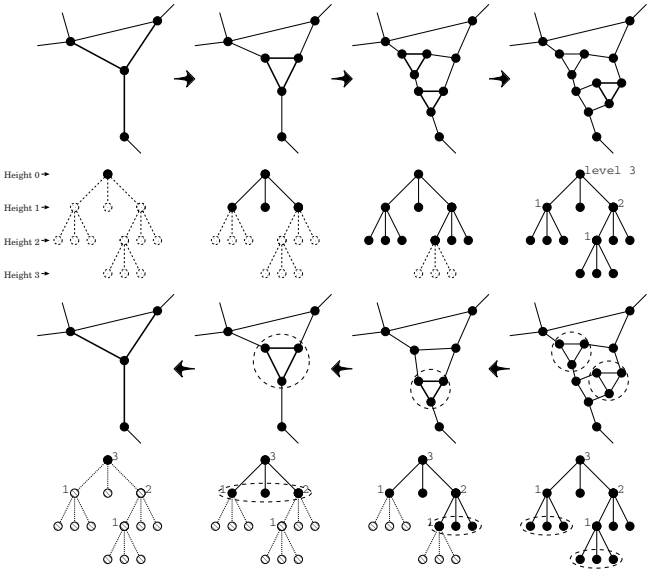


Fig. 4. Expansion and contraction

So, we will analyse the number of nodes with level l for any $l \geq 0$ in this section. For any $l \geq 0$, define the following values:

$$M(l) = \text{the expected number of level } l \text{ nodes in } T,$$
$$q(l) = Pr[T \text{ has a node of level } l] = Pr[\text{the height of } T \geq l],$$
$$P(l) = Pr[\text{the height of } T \text{ is } l] = Pr[\text{the level of the root of } T \text{ is } l].$$

4.1 Degree Distribution of G^t

Let V_k be a set of vertices with degree k in G and let $n_k = |V_k|$. For vertices with degree 1, these vertices does not change throughout our expansion procedure. So, we will consider vertices of degree larger than or equals to two in the rest of the paper. Let us consider random variables N_k for $k \geq 2$ and N_v for $v \in V[G]$. N_k denotes the number of vertices with degree k in $H = (G^t)$ and N_v denotes the number of vertices expanded from v in H. So, we have $N_k = \sum_{v \in V_k} N_v$. In this section, we have the following theorem.

Theorem 1. *The expected number of vertices with degree k in H is;*

$$\mathrm{E}[N_k] = \left(1 + \frac{p_0}{1-p_0}\left(1 - \frac{1}{k}\right)\right) n_k.$$

Proof. By observation 4, the expected number of leaves expanded from v is $M(0)$, i.e. $\mathrm{E}[N_v] = M(0)$. If v_0 is not expanded, then the number of leaves is 1,

and this occurs with probability $1-p_k$. Otherwise, the number of leaf nodes are sum of the number of leaf nodes in subtrees under the k child nodes. Thus, we have

$$M(0) = p_k \cdot kM(0) + (1-p_k) \cdot 1.$$

Hence

$$M(0) = \frac{1-p_k}{1-p_k k} = \left(1 + \frac{p_0}{1-p_0}\left(1-\frac{1}{k}\right)\right).$$

Since $\mathrm{E}[N_k] = \sum_{v \in V_k} \mathrm{E}[N_v] = |V_k| M(0)$, the expected number of vertices with degree k in H is;

$$\mathrm{E}[N_k] = \left(1 + \frac{p_0}{1-p_0}\left(1-\frac{1}{k}\right)\right) n_k.$$

□

Here, let N denote the total number of nodes in H to consider the degree distribution of H. Since N is the sum of the N_k for all k,

$$\mathrm{E}[N] = \sum_{k=1}^{\infty} \mathrm{E}[N_k] = \left(1 + \frac{p_0}{1-p_0}\right)|V| - \frac{p_0}{1-p_0}\sum_{k=1}^{\infty}\frac{n_k}{k} = \left(1 - \frac{p_0 C}{1-p_0}\right)|V|,$$

where C is a constant satisfying $C|V| = |V| - \sum_{k=1}^{\infty} \frac{n_k}{k}$.

So, Theorem 1 gives the following expected degree distribution;

$$\frac{\mathrm{E}[N_k]}{\mathrm{E}[N]} = \frac{\left(1 + \frac{p_0}{1-p_0}\left(1-\frac{1}{k}\right)\right) n_k}{\left(1+\frac{p_0}{1-p_0}\right)|V| - \left\{\frac{p_0}{1-p_0} C\right\}} = \frac{\left(1 + \frac{p_0}{1-p_0}\left(1-\frac{1}{k}\right)\right) n_k}{\left(1 - \frac{p_0 C}{1-p_0}\right) n}.$$

Let c_1 and c_2 be $c_1 = \left(1 + \frac{p_0}{2(1-p_0)}\right)/C'$ and $c_2 = \left(1 + \frac{p_0}{(1-p_0)}\right)/C'$. Then we obtained

$$c_1 \frac{n_k}{n} \leq \frac{\mathrm{E}[N_k]}{\mathrm{E}[N]} \leq c_2 \frac{n_k}{n}.$$

Corollary 1. *If the input graph G has the power-law degree distribution with exponent γ, $n_k/n = \Theta(k^{-\gamma})$, the expected degree distribution of H also follows the power-law distribution, i.e.,* $\mathrm{E}[N_k]/\mathrm{E}[N] = \Theta(k^{-\gamma})$.

4.2 Degree and Isolated Clique Size Distributions of H^i

In this section, we analyze the expected degree distribution and the expected number of isolated cliques in H^i. We must note that the contraction procedure $\mathcal{C}(\cdot)$ is not an inverse procedure of the expansion $\mathcal{E}(\cdot)$. It is easy to observe the fact by an example of the Figure 4.

Let us denote the number of isolated cliques of size k in H^i by $M_k(H^i)$, and the number of vertices with degree k in H^i by $N_k(H^i)$. First, we have the following obvious bound.

Theorem 2. *Let* $C'_1 = 1$ *and* $C'_2 = 1 + \frac{p_0}{1-p_0}$. *Then, for any* i,

$$C'_1 n_k \leq \mathrm{E}[N_k(H^i)] \leq C'_2 n_k.$$

Proof. It is clear that $C'_1 n_k = N_k(G^0) \leq N_k(H^i) \leq N_k(H^0) \leq C'_2 n_k$. □

Corollary 2. *Consider an input graph* G *has a power-law degree distribution with exponent* γ, $\frac{n_k}{n} = \Theta(k^{-\gamma})$, *and* G *has no isolated cliques. Then the expected degree distribution of* H^i *also follows the power-law distribution with exponent* γ.

For the expected number of isolated cliques of size k in H^i, we have the following bounds.

Theorem 3. *Let* C_1 *and* C_2 *be* $C_1 = \left(1 - \frac{p_0}{2(1-p_0)^2}\right)$ *and* $C_2 = \frac{e^{-p_0}}{1-p_0}$. *Then for any* $i \geq 0$,

$$C_1 p_0^{i+1} \frac{n_k}{k} \leq \mathrm{E}[M_k(H^i)] \leq C_2 p_0^{i+1} \frac{n_k}{k}.$$

Note that there is also an obvious lower bound that $\mathrm{E}[M_k(H^i)] \geq 0$ *and if we let* $p_0 < \frac{1}{2}$, *then* $C_1 > 0$.

Proof. As mentioned in Observation 4, we will consider the distribution of the number of nodes which has level i. In the literature, e.g., [4], the distribution of the number of nodes with height i is mentioned. However, the analysis of the distribution of the number of nodes which has level i has not been provided before.

Firstly, we show $\mathrm{E}[M_k(H^i)] = n_k M(i+1)$ and secondly, we give lower and upper bound for $M(l)$.

The expected number of isolated cliques expanded from one vertex and on H^i equals to $M(i+1)$, so the total number of isolated cliques of size k is $\mathrm{E}[M_k(H^i)] = n_k M(i+1)$.

$P(l)$ denotes the probability such that the root has level l. Clearly, this contributes $P(l)$ to $M(l)$. Then consider the other case. Since $M(l)$ is 0 if the root was not expanded; thus, consider the situation that the root was expanded (which occurs with prob. p_k). Let $v_1, \ldots, v_k$ denote the child nodes of the root and let $T_1, \ldots, T_k$ denote the trees rooted by these nodes. Then we may regard that each T_i follows the same probability distribution as T; thus, we may use $M(l)$ for the expected number of level l nodes of T_i. Hence we have $M(l) = pkM(l) + P(l)$. Since the number of nodes on tree T is finite with probability 1, we have;

$$M(l) = \frac{P(l)}{1 - pk}. \tag{1}$$

From now on, we consider $P(l)$.

Lemma 1. *We have* $P(1) = \frac{p_0}{k}(1 - \frac{p_0}{k})^k$ *and for any* $l > 1$

$$P(l) \leq \frac{p_0^l}{k} \left(1 - \frac{p_0}{k}\right)^k$$

Proof. Since $q(1) = p_k$ and $q(2) = p_k(1-\frac{p_0}{k})^k$, $P(1) = q(1) - q(2) = \frac{p_0}{k}(1-\frac{p_0}{k})^k$. By the definition of $q(l)$, we have

$$q(l) = p\left\{1 - (1 - q(l-1))^k\right\} \quad \text{for any } l \geq 1.$$

For any $0 < x < y < 1$, it is easy to show that

$$(1-x)^k - (1-kx) < (1-y)^k - (1-ky)$$

and that $g(l) < g(l-1)$,

$$\begin{aligned} P(l) = q(l) - q(l+1) &= p_k\left[\left\{1-(1-q(l-1))^k\right\} - \left\{1-(1-q(l))^k\right\}\right] \\ &\leq p_k\left[\{1-(1-kq(l-1))\} - \{1-(1-kq(l))\}\right] \\ &= p_k k\left(q(l-1) - q(l)\right) = p_0 P(l-1). \end{aligned}$$

Hence we obtained $P(l) \leq p_0^{l-1}P(1) = \frac{p_0^l}{k}(1-\frac{p_0}{k})^k$. □

To analyse the lower bound of $P(l)$, we need to consider the upper and lower bound of $q(l)$.

Lemma 2. *For $q(l)$, we have*

$$\frac{f(l)}{k} \leq q(l) \leq p_k p_0^{l-1}.$$

where $f(l) = p_0(1 - e^{-f(l-1)})$ and $f(1) = p_0$.

Proof. Proof can be easily done by $e^{-xk} \leq (1-x)^k \leq 1 - kx$ and induction on l, hence omitted. □

Second, we give a lower bound of $f(l)$.

Lemma 3. $f(l) \geq p_0^l - \frac{1}{2}\left(\sum_{j=l+1}^{2l-1} p_0^j\right) > p_0^l\left(1 - \frac{p_0}{2(1-p_0)}\right)$.

Proof. The proof is also easy by $\left(1-\frac{x}{2}\right)x \leq 1 - e^{-x}$ and induction on l. Due to the space limit we omit the proof. For details, see [9]. □

Hence, the lower bound of $P(l)$ is

$$P(l) = q(l) - q(l+1) \geq \frac{f(l)}{k} - p_k p_0^l \geq \frac{1}{k}p_0^l\left(1 - p_0 - \frac{p_0}{2(1-p_0)}\right).$$

By equation (1), We finally obtain

$$\frac{p_0^l}{k}\left(1 - \frac{p_0}{2(1-p_0)^2}\right) \leq M(l) \leq \frac{p_0^l}{k}\frac{(1-\frac{p_0}{k})^k}{1-p_0} < \frac{p_0^l}{k}\frac{e^{-p_0}}{1-p_0}.$$

Now let $C_1 = \left(1 - \frac{p_0}{2(1-p_0)^2}\right)$ and $C_2 = \frac{e^{-p_0}}{1-p_0}$. By $Ex[M_k(H^i)] = n_k M(i+1)$, we obtain Theorem 3.

$$C_1 p_0^{i+1}\frac{n_k}{k} \leq Ex[M_k(H^i)] \leq C_2 p_0^{i+1}\frac{n_k}{k}.$$ □

By Theorem 3, the expected number of isolated cliques in H^i is proportional to $p_0^{i+1}\frac{n_k}{k}$ for any size k. The total number of isolated cliques in H^i is also proportional to $p_0^{i+1}\sum_{k>1}\frac{n_k}{k}$. The ratio of the isolated clique of size k among all isolated cliques in H^i can be written as

$$\frac{Cp_0^{i+1}\frac{n_k}{k}}{p_0^{i+1}\sum_{k>1}\frac{n_k}{k}} = \frac{C}{M}\frac{n_k}{k}$$

for some constant C. Note that $M = \sum_{k>1}\frac{n_k}{k}$ is a constant independent to k. Hence, if $\frac{n_k}{n} = \Theta(k^{-\gamma})$, $\frac{C}{M}\frac{n_k}{k} = \Theta(k^{-\gamma}/k) = \Theta(k^{-(\gamma+1)})$.

Corollary 3. *Consider an input graph G has a power-law degree distribution with exponent γ, $\frac{n_k}{n} = \Theta(k^{-\gamma})$, and G has no isolated cliques. Then the expected size distribution of isolated cliques in H^i also follows the power-law distribution with exponent $\gamma + 1$.*

5 Related Works

In this paper, we investigated a model of the community structure with isolated cliques. Many kinds of other community structure have been introduced and investigated.

Web mining using complete bipartite graph (CBG) has been investigated by Kleinberg [6]. They assumed that web communities contain at least one CBG which is called the core of the community. Reddy and Kitsuregawa [8] relaxed the criteria of existence of a community by defining a dense bipartite graph structure (DBG). They investigated a community hierarchy of the World Wide Web extracting all DBG found in the WWW.

Many other models have been presented, however, there were only few mathematical analysis of the size distribution of communities for these models.

Recently, a different type of some hierarchical structure, called the fractal property, has been also studied by Song et al. [10]. They observed that the power-law degree distribution on the reduced graph obtained by contracting randomly and greedily chosen connected subgraph. They also proposed a model to represent this fractal property [11], however, their minimal model generates a tree so has neither cliques nor hierarchical clique structure.

6 Concluding Remarks

In this paper, we proposed a new model to explain the hierarchical clique structure and its scale-free properties. Our model provides a graph with the similar properties to the ones that are observed in the World Wide Web.

However, our model generates a special kind of isolated cliques such that each member of the clique has exactly one outgoing edge. It is possible to consider some modifications of our model to this problem, randomly connect outgoing

edges of the isolated cliques for example. In our model, we used some other model to generate a prime network (G^0). If we use a single vertex or a clique as a prime network, it generates a regular graph in our current model. We are trying to make more general model which can generate graphs with scale-free property and the hierarchical clique structure from one node or one clique.

Uno et al. also investigates the hierarchical structure of isolated stars [12,13], we also apply our approach to them.

References

1. Albert, R., Barabási, A.-L.: Statistical mechanics of complex networks. Review of Modern Physics 74, 47–97 (2002)
2. Athreya, K.B., Ney, P.E.: Branching Processes. Springer, Heidelberg (1972)
3. Barabási, A.-L., Albert, R.: Emergence of scaling in random networks. Science 286(5439), 509–512 (1999)
4. Feller, W.: An Introduction to Probability Theory and Its Applications, 3rd edn. Wiley, Chichester (1968)
5. Ito, H., Iwama, K., Osumi, T.: Linear-time enumeration of isolated cliques. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 119–130. Springer, Heidelberg (2005)
6. Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. J. ACM 46(5), 604–632 (1999)
7. Newman, M.E.J.: The structure and function of complex networks. SIAM Review 45, 167–256 (2003)
8. Krishna Reddy, P., Kitsuregawa, M.: Building a community hierarchy for the web based on bipartite graphs. In: Proceedings of the 13th IEICE Data Engineering Workshop, pages C4–1 (2002)
9. Shigezumi, T., Uno, Y., Watanabe, O.: A new model for a scale-free hierarchical structure of isolated cliques. Dept. of Math. and Comp. Sciences Tokyo Institute of Technology Research Reports, Series C (2009)
10. Song, C., Havlin, S., Makse, H.A.: Self-similarity of complex networks. Nature 433(7024), 392–395 (2005)
11. Song, C., Havlin, S., Makse, H.A.: Origins of fractality in the growth of complex networks. Nature Physics 2(4), 275–281 (2006)
12. Uno, Y., Kiyotani, T., Oguri, F.: Investigating web structure by cliques and stars. RIMS Kokyuroku 1644, 44–54 (2009)
13. Uno, Y., Ota, Y., Uemichi, A.: Web structure mining by isolated stars. In: Aiello, W., Broder, A., Janssen, J., Milios, E.E. (eds.) WAW 2006. LNCS, vol. 4936, pp. 149–156. Springer, Heidelberg (2008)

The Covert Set-Cover Problem with Application to Network Discovery

Sandeep Sen and V.N. Muralidhara

Department of Computer Science and Engineering,
Indian Institute of Technology, Delhi, India
{ssen,murali}@cse.iitd.ernet.in

Abstract. We address a version of the set-cover problem where we do not know the sets initially (and hence referred to as covert) but we can query an element to find out which sets contain this element as well as query a set to know the elements. We want to find a small set-cover using a minimal number of such queries. We present a Monte Carlo randomized algorithm that approximates an optimal set-cover of size OPT within $O(\log N)$ factor with high probability using $O(OPT \cdot \log^2 N)$ queries where N is the number of elements in the universal set.

We apply this technique to the network discovery problem that involves certifying all the edges and non-edges of an unknown n-vertices graph based on layered-graph queries from a minimal number of vertices. By reducing it to the covert set-cover problem we present an $O(\log^2 n)$-competitive Monte Carlo randomized algorithm for the covert version of network discovery problem. The previously best known algorithm has a competitive ratio of $\Omega(\sqrt{n \log n})$ and therefore our result achieves an exponential improvement.

1 Introduction

Given a ground set S with n' elements and a family of sets $S_1, S_2 \ldots S_{m'}$ where[1] $S_i \subset S$, a *cover* C is a collection of sets from this family whose union is S. It is known that finding a cover consisting of the minimum number of sets is a computationally intractable problem [9]. There are many strategies [11,6,10] to *approximate* the smallest cover within a factor of $O(\log n')$ which is known to be the best possible unless $P = NP$ [7].

In this paper, we consider the following version of the set cover problem. Although we know m', n', we do not know the elements nor the cardinality of any of the sets S_i. We are allowed to query an element $e \in S$ that returns all sets S_i that contain e which we refer to as a hitting-set query; we can also query a set to know its elements. We would like to compute a small set cover of S using a minimal number of such queries. More specifically, if OPT is the minimum size of a set cover for any instance of the problem, we would like to find a set

[1] We have chosen n', m' as notations to keep them distinct from graphs with n vertices and m edges.

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 228–239, 2010.

cover of size $O(OPT \cdot \text{polylog}\, n')$ using only $O(OPT \cdot \text{polylog}\, n')$ queries. Note that by using $\min\{m', n'\}$ queries, we can reduce it to the standard version but the number of queries may not satisfy $O(OPT \cdot \text{polylog}\, n')$. By restricting the number of queries to be close to OPT, an algorithm cannot afford to learn the contents of all the sets, yet it is required to find a cover close to the optimal.

This formalisation is also distinct from the *online* problems addressed in [1,2] where the sets are known but the adversary chooses a set of the ground set for which a minimal cover must be computed. An adversary chooses the elements one after the other and the online algorithm must maintain a cover of the elements revealed upto a given stage. There is no apparent relationship between the two versions. In one case, the initial sets are not known but the algorithm can choose the elements for hitting set queries whereas in the online case, the sets are known but the adversary chooses the elements. Moreover, the number of queries is also a measure of performance in the version considered here.

Our research is motivated by the problem of discovering the topologies of large networks such as the Internet. For large networks such as the Internet which changes frequently, it is very difficult and costly to obtain the topology accurately. Nevertheless, such information about the network is very useful - for example, the robustness properties of the network or studying the routing aspects.

In order to create the topology of the network, one of the techniques used is to obtain local views of the network from various locations and combine them to determine the topology of the network. One can view this technique as an approach for discovering the topology of the network by some queries. Here, a query corresponds to the local view of the network from one specific location. In the real world scenario, the cost of answering a query is usually very high, so the objective of the network discovery problem is to find the map of the network using a minimal number of queries.

Note that in the network discovery problem, we have to confirm the existence and non-existence of an edge between any pair of vertices. So, any query at a vertex should implicitly or explicitly confirm the absence or presence of edges between some pair of vertices. The *Layered Graph Query Model* and *Distance Query Model* are the most widely studied query models.

Layered Graph Query Model: A query at a vertex v yields the set of all edges on shortest paths from the vertex v to any other vertex reachable from v in the graph. More specifically, we obtain information about an edge (x, y), iff $d(v, x)$ and $d(v, y)$ are consecutive where $d(v, x)$ is the level of x (from v, see Fig. 1).

Distance Query Model: A query at a vertex v yields the distances of v to every vertex of the graph, *i.e.* a query at a vertex v returns a vector $\boldsymbol{v}$, where the ith component indicate the distance to ith vertex from vertex v. It is easy to see that it is a weaker query model as compared to *Layered Graph Query Model*. In the *Distance Query Model*, an edge may be discovered by a combination of queries as illustrated in Fig 2. In the example shown in Fig 2, query at vertex 1 discovers the non-edges $\{(1,4),(1,5),(1,6),(2,6),(3,6)\}$ and edges $\{(1,2),(1,3)\}$. A query at vertex 6 discovers the non-edges $\{(1,4),(1,5),(1,6),(2,6),(3,6),(4,2),(5,2),(3,2)\}$

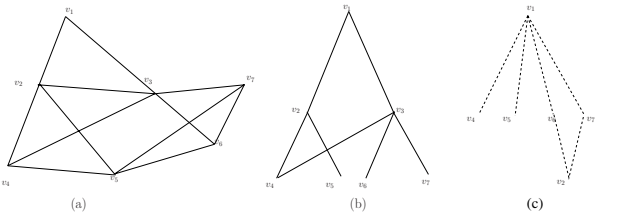


Fig. 1. A query at a vertex v_1 in the layer graph model (a) yields certificate for the edges in (b) and non-edges in (c)

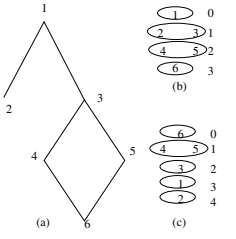


Fig. 2. The edges $(5, 3)$ and $(4, 3)$ of the graph (a) is discovered by the combination of queries at vertex 1 in (b) and at vertex 6 in (c) in the distance query model - the distances are depicted via layers of the graph.

and edges $\{(3, 1), (1, 2), (6, 4), (6, 5)\}$. Combining these two queries, we discover the edges $(5, 3)$ and $(4, 3)$. In the off-line version of network discovery problem, the network is initially known to the algorithm. Unlike the online problem, here the goal is to compute a minimum number of queries that suffice to discover the network. Given a network, we can verify whether what we have been given is the correct information. Thus, we refer to the off-line version of network discovery problem as *network verification*.

1.1 Prior Work in Network Discovery

Bejerano and Rastogi [5] studied the problem of verifying all edges of a graph with as few queries as possible in a model similar to the *Layered Graph Query Model*. For a graph with n vertices, they give a set-cover based $O(\log n)$-approximation algorithm and show that the problem is NP-hard. In contrast to Bejerano and Rastogi, we are interested in verifying (or discovering) both the edges and the non-edges of a graph. It turns out that the network verification problem was considered as a problem of placing landmarks in graphs [8]. The problem was shown to be NP-complete and an $O(\log n)$-approximation algorithm was presented. Beerliova et al. [3] proved an $\Omega(\log n)$ lower bound on the approximation factor for any polynomial

time algorithm for the network verification in the *Layered Graph Query Model* unless $P = NP$.

In the *online* version of the problem, the network (graph) is unknown to the algorithm. To decide the next query, the algorithm can only use the knowledge about the network it has gained from the answers of previously asked queries. Thus, the difficulty in selecting good queries arises from the fact that we only have the partial information about the network.

For the network discovery problem, Beerliova et al.[4] have shown an $\Omega(\sqrt{n})$ lower bound on the competitive ratio of any deterministic online algorithm and an $\Omega(\log n)$ lower bound for any randomized algorithm for the *Distance Query Model*. The best known algorithm in the *Distance Query Model* is a randomized online algorithm which is $O(\sqrt{n \log n})$-competitive [4]. In contrast, for the *Layered Graph Query Model*, Beerliova et al.[4] have shown that no deterministic online algorithm can be $(3 - \varepsilon)$ competitive for any $\varepsilon > 0$. The best known algorithm in this model before this work is an $O(\sqrt{n \log n})$-competitive online randomized algorithm [4] that leaves an exponential gap between the best known lower and upper bounds for the *Layered Graph Query Model*.

In this paper, we present a randomized Monte Carlo online algorithm with a competitive ratio $O(\log^2 n)$ for the *Layered Graph Query Model* thereby nearly closing this exponential gap.

1.2 Our Results and Techniques

The network verification problem can be solved by reducing it to an appropriate instance of the set-cover problem (or hitting set problem). Hence, we obtain an $O(\log n)$ approximation algorithm for the network verification problem which is the best that we can hope to do unless $P = NP$. In the online network discovery problem, we do not know the graph *a priori* and hence the above reduction cannot be used directly. In particular, the sets are not known explicitly, so we first develop an algorithm for solving the covert version of the set-cover problem using queries.

We present an algorithm that computes a set-cover of size at most $O(\log(m' + n') \cdot OPT)$ using at most $O(\log^2(m' + n') \cdot OPT)$ queries with high probability. Using this, we obtain an $O(\log^2 n)$-competitive Monte Carlo randomized algorithm for the network discovery problem in the *Layered Graph Query Model*. This is a significant improvement from the previously best known $O(\sqrt{n \log n})$-competitive algorithm ([3]).

Our algorithm for the set-cover simulates the greedy set-cover algorithm without any information about the contents of any of the sets initially. We use estimation using random sampling to choose the (near) largest cardinality set which is the basis of the greedy algorithm. We have to compensate for the inaccuracies in sampling by using a more careful amortisation argument for proving the approximation factor. The greedy algorithm is modified to run in $O(\log(n' + m')$ rounds instead of the conventional $OPT \cdot \log n'$ stages.

2 Preliminaries

Let $G = (V, E)$ be a connected, undirected, unweighted graph representing a network of n vertices. For two distinct nodes $u, v \in V$, we say that (u, v) is an edge if $(u, v) \in E$ and non-edges if $(u, v) \notin E$. The set of non-edges in G is denoted by $\overline{E}$.

We assume that the set V of nodes is known in advance and it is the presence or absence of edges that need to be discovered or verified. A query at node v is denoted by $query(v)$.

We say that a $query(v)$ *certifies* (u, v) if by using the answers to the $query(v)$, one can confirm the presence or absence of the edge (u, v) in the graph, *i.e.* $query(v)$ implicitly or explicitly confirms whether $(u, v) \in E$ or $(u, v) \in \overline{E}$. We associate two sets with each $query(v)$ as follows. For a given vertex $v \in V$, let Q_v denotes the set of all $(u, v) \in V \times V$ such that $query(v)$ *certifies* (u, v) . For a given $(u, v) \in V \times V$, let $H_{(u,v)}$ denote the set of all vertices v such that $query(v)$*certifies* (u, v) . The two definitions can be considered duals of each other.

$$Q_v = \{(u, v) \in V \times V |\ query(v)\ certifies\ (u, v)\}\ \forall v \in V$$

$$H_{(u,v)} = \{v \in V |\ query(v)\ certifies\ (u, v)\} \forall (u, v) \in V \times V.$$

The above formulation of the network discovery problem can be reduced to the *set-cover* problem in which given a collection of sets Q_v of $E \cup \overline{E}$, the goal is to find a (minimum size) subset $V' \subset V$ such that $\cup_{v \in V'} Q_v = E \cup \overline{E}$. Therefore, querying the vertices of the set-cover will certify all the edges and non-edges that can be used to discover the network.

In the related *hitting-set* problem, given a collection of sets $H_{(u,v)}$ of V, the goal is to find a (minimum size) subset $V' \subset V$ such that for any given set $H_{(u,v)}$, there exists a vertex $v' \in V'$ such that $v' \in H_{(u,v)}$. It may be noted that the (offline) hitting-set problem is often solved by reducing it to the corresponding set-cover problem.

In the *offline* verification problem, given any query model, one can find the above sets exactly as the graph is known. So the network verification problem can be solved by reducing it to the corresponding set-cover problem (or hitting set problem). Hence, we get an $O(\log n)$ competitive algorithm for the network verification problem. As mentioned earlier this is the best that we can hope to do for this problem unless $P = NP$.

In the online network discovery problem, since we do not know the graph *a priori*, we cannot compute the above sets explicitly without querying all the vertices.[2] To circumvent this problem, we develop an algorithm for approximating the set-cover using the related hitting-set queries. It can be easily seen (c.f. Section 6), that $H_{(u,v)}$ can be obtained from Q_u and Q_v in the context of the network discovery problem.

[2] While this may be necessary for some graphs like the complete graphs, in general this will lead to poor competitive ratio.

3 Approximating Set-Cover Sets Using Hitting-Set Queries

In the conventional greedy set-cover algorithm, we choose a set $\mathbf{s_{max}}$ that covers the maximum number of uncovered elements, say n_{max}, and add it to the cover. This leads to a $\log n'$ approximation. Instead, if we choose any set that covers at least half of n_{max} uncovered elements, then it gives a $2 \log n'$ approximation. Recall that m', n' denote the number of elements and the number of sets respectively. More generally, if we choose a set that cover at least $\frac{1}{c'} n_{max}$ elements, then we obtain a $c' \log n'$ approximation. We consider a version of this *Relaxed Greedy-Set-Cover* (*RGSC*) where we repeat the following in stages $1, 2, \ldots \log n'$. At any stage we identify all the sets that contain at least $\frac{1}{2} n_{max}$ uncovered elements. We can consider the sets of in an arbitrary, but fixed ordering O and include those sets that contribute at least $\frac{1}{2} n_{max}$ uncovered elements by deleting elements that have been already covered by sets chosen earlier. Note that the sets that will be included will depend on O - however, at the end of this stage, there will not be any set that contains $n_{max}/2$ or more uncovered elements. Since any such ordering O corresponds to a valid run of *RGSC*, this will yield a $2 \log n'$ approximation guarantee.

Our algorithm is based around simulating this approach, where we try to estimate the value of n_{max} indirectly using random sampling. In round i,[3] we check for $n_{max} \in [\frac{n'}{2^{i-1}}, \frac{n'}{2^{i-2}}]$ by choosing a random set of uncovered elements of an appropriate size. Using hitting set queries, we find the sets containing these randomly chosen elements. We choose an appropriate number of uncovered elements that will hit the sets having $\frac{n'}{2^{i-2}}$ elements with high probability. We consider the sets in a fixed order and if a set contains more than at least a threshold number of randomly picked elements, then we include the set in the set-cover. Because of the estimation using random sampling, we lose a factor $c' > 2$ in the underlying RGSC as we may choose some sets which contain fewer than $n_{max}/2$ uncovered elements (but at least $\frac{n_{max}}{c'}$).

Algorithm **Pseudo Greedy** described below, selects all sets containing at least $n_{max}/2$ uncovered elements and discards the sets containing less than $\frac{1}{c'} n_{max}$ uncovered elements for $4 < c' < 8$ with high probability.

We assume that the sets are numbered in some canonical order. In the specific application of the network discovery problem, this ordering is implicit ($\{v_1, v_2, \ldots v_n\}$, this induces a canonical ordering on the collection Q_v of sets). In the general setting, we assume that such an ordering exits or it can be easily computed.

In Algorithm 1, N denotes the cardinality of the ground set plus the number sets in the given family ($N = n' + m'$). In the case of Network Discovery problem, $N = O(|V|^2)$. In round i, we try to identify the sets containing at least $\frac{n'}{2^{i+1}}$ uncovered elements.

[3] The notation n_{max} will refer to the maximum in the current round i.

Algorithm 1. Pseudo-Greedy

for $i = 0, 1 \ldots$ **do**

1: Let n_i be the number of elements left in this round and $s_i = \min\{\frac{n'}{2^i}, n_i\}$. Choose a random sample R^i of size $(4\alpha n_i/s_i)\log N$.
Comment: α is a constant whose value will be determined in the analysis.

2: If $s_i \leq \alpha \log N$ then solve the hitting set problem directly using at most n_i hitting set queries and run the explicit greedy set-cover algorithm.

3: Else (if $s_i > \alpha \log N$), let S^i be the sets that contain more than $\alpha \log N$ sampled elements.
If S^i is empty, increment i and go to step 1.

4: Process $S^i = \{X_1, X_2, \ldots\}$ in some predefined order until all sets are exhausted.

(i) Let R_j be the union of elements of R^i that are contained in the sets chosen among $X_1, X_2, \ldots X_j$.

(ii) Include X_{j+1} in set-cover if

$$|X_{j+1} \cap (R^i - R_j)| \geq \alpha \log N$$

else discard.

(iii) Update R_j to R_{j+1}. using set queries.

5: Update the elements covered by the sets chosen in this round using set queries.

4 Analysis

We begin with a rough intuition behind the previous algorithm. If the largest set has size n'/t then the minimum number of sets in any set cover is $\Omega(t)$. Therefore we can afford to query a sample of size approximately $O(t \cdot \text{polylog}\, n')$ elements without blowing up the competitive ratio. In this context note that a uniform random sample of size $O(t \cdot \text{polylog}\, n')$ will have $\theta(\text{polylog}\, n')$ elements common with a set of size n'/t with high probability. However, if there are $\Omega(t)$ sets of size $O(n'/t)$, we cannot afford to sample repeatedly for finding these sets. The above observations form the crux of the analysis that are now formalized.

Lemma 1. *In round i, in Step 3, the following holds with high probability*
(i) If a set T contains at least $s_i/2$ elements then with high probability it will have at least $\alpha \log N$ sampled elements.
(ii) Any set T chosen in Step 3 will contain at least $\frac{1}{c'}s_i$ elements for $4 < c' < 8$ with high probability.

Proof. Let T be a set where $m \geq |T| \geq m/2$. Suppose we sample every element independently with probability p. The expected number of sampled elements Y is such that $mp \geq Y \geq mp/2$. From Chernoff bounds,

$$Pr[(1+\varepsilon)mp \geq Y \geq (1-\varepsilon)mp/2] \geq 1 - 2e^{-mp\varepsilon^2/4}$$

Choosing $\varepsilon = 1/2$, we get

$$Pr[3/2mp \geq Y \geq mp/4] \geq 1 - 2e^{-mp/16}$$

In round i, each element is picked independently with probability $(4\alpha/s_i)\log N$, therefore, the expected number of hits in a set of size m is $(4m\alpha/s_i)\log N$. From Chernoff bounds, by substituting $m = s_i$,

$$Pr[6\alpha \log N \geq Y \geq \alpha \log N] \geq 1 - 2e^{-\alpha/4 \log N} = 1 - 2/N^{\alpha/4}$$

Since the number of such T is less than N, the algorithm picks all sets containing at least $s_i/2$ uncovered elements with high probability. On the other hand, T be any set chosen in Step 3 of the algorithm. Then, by applying Chernoff bound, we get,

$$Pr[T < s_i/c'] \leq e^{-(c'-2)\alpha/8 \log N} = 1/N^{(c'-2)\alpha/8}$$

for all $4 < c' < 8$.

Lemma 2. *If round i takes $O(\frac{n_i}{s_i} \cdot f(N))$ steps, then the set-cover can be found using $O(n_g \cdot f(N))$ queries where n_g is the size of the set-cover returned by the underlying RGSC Algorithm.*

Proof. In round i, we include all those sets in the cover that covers at least $s_i/2$ additional elements. In round i, let us distribute the cost uniformly to the remaining elements, i.e., each of the n_i elements is charged $O(f(N)/s_i)$. If an element is covered by a set chosen in round i then it is not charged in the subsequent rounds. So the total cost over all the rounds for element x is $C(x) \leq f(N) \cdot \left(\frac{1}{|s(x)|} + \frac{1}{s_i} + \frac{1}{2s_i} + \frac{1}{4s_i} + \ldots\right)$ where $s(x)$ is the set that *first* covers element x and $s_i/c' \leq |s(x)| \leq s_i$. The constant c' refers to the constant in the previous lemma. Therefore

$$\sum_x C(x) \leq f(N) \sum_x \frac{3c'}{|s(x)|} = 3c' f(N) \cdot \sum_{s(x)} \sum_{x \in s(x)} \frac{1}{|s(x)|}$$

The summation represents the cost of the underlying *RGSC* algorithm and therefore, it is bounded by $3c' f(N) \cdot n_g$ (see Lemma 3 in the Appendix).
Note that the underlying *RGSC* algorithm is a $c' \log N$ approximation to the set-cover.

Theorem 1. *Algorithm 1 returns a set-cover of size at most $O(\log N \cdot OPT)$ using at most $O(\log^2 N \cdot OPT)$ queries with high probability.*

Proof. In our algorithm, $f(N)$ is $O(\log N)$. When $s_i < \alpha \log N$, we solve the problem directly using at most n_i hitting set queries, and explicitly run the greedy set-cover. Since the largest set has size $n'/2^i$, the size of the cover is at least $\Omega(n'/\log N)$ and therefore, the number of queries is $O(\log N \cdot OPT)$. In order to prove the theorem, we will show that the bound on n_g in Lemma 2 is $O(\log N \cdot OPT)$. So, we must establish that the sets *shortlisted* in Step 3 of the Algorithm and finally included in the cover in Step 4 are only those sets (on the basis of their estimates) that covers at least s_i/c' uncovered elements. There is a potential complication if the ordering that we choose is arbitrary - in particular, we must guard against oversampling of the uncovered elements of any set. For

simplicity, let us assume that we consider the sets of S^i in increasing order of their indices.

Let $X_1, X_2 \ldots$ be the sets of S^i in the canonical ordering that contain at least s_i/c' elements. We define X'_i as all the uncovered elements in X_i after $X_1, X_2, \ldots X_{i-1}$ have been considered. We consider X'_i to be *under-sampled* if $|X'_i| \geq s_i$ but the number of sampled elements intersecting X'_i (not including $X_i - X'_i$) is less than $\alpha \log n$. We analogously define *oversampling* for X'_i.

We say that a bad event has occurred in round j, if any of the sets X'_i is under-sampled or oversampled and let the complement of this event be Z_i. From Lemma 1, we can bound the probability of under sampling and over sampling such that $\Pr[Z_i] \geq 1 - 3/N^{(\alpha/4)-1}$ (by choosing $c' > 4$). Let A_i be the event that no under-sampling or oversampling occurs for $X'_1, X'_2 \ldots X'_i$. Then,

$$\Pr[A_i] = \Pr[A_{i-1} \bigcap Z_i] = \Pr[Z_i | A_{i-1}] \cdot \Pr[A_{i-1}]$$

Therefore,

$$\Pr[A_i] \geq \Pr[A_{i-1}] \cdot (1 - 3/N^{(\alpha/4)-1}) \geq \left(1 - 2/N^{\alpha/4-1}\right)^i \text{ for } i \leq N$$

By choosing sufficiently large α this is at least $1 - 1/N^2$. Since this holds for all $j \leq O(\log N)$ rounds, this also bounds the failure probability of our algorithm. We say that the algorithm *fails* if in any of rounds, it does not pick all sets containing at least $s_i/2$ uncovered elements or picks any set containing less than s_i/c' uncovered elements. Since we do not verify this property, we obtain a Monte Carlo algorithm.

Remarks: The sizes of sets that will be chosen will satisfy the the above mentioned bounds with high probability; otherwise, the algorithm will be deemed to have failed. Note that the bound of Lemma 2 also holds with the same probability. A deterministic algorithm picks all the sets of size at least $s_i/2$, and while our randomized algorithm chooses all sets of size at least $s_i/2$, it may pick some sets which are little smaller (but greater than s_i/c').

5 Network Discovery

The off-line problem of network verification can be reduced to a set-cover problem. In the online version, we do not want to compute the sets explicitly since this will lead to a poor competitive ratio in many situations. So we solve the problem by using hitting-set queries as described in the previous section that gives us an estimate of the set sizes. In our setting, the hitting-set problem is defined on the sets $H_{(u,v)}$ and the set-cover problem on the sets Q_v. During any stage, random sampling is done on the set of unresolved edges to obtain estimates of Q_v by querying Q_{xy} where (x, y) is a sampled edge.

Recall that in *Layered Graph Query Model*, a query at a vertex v yields the set of all edges on shortest paths between v and any other vertex. Now, we

observe that this query model is equivalent to the model in which a query at vertex v yields all edges and non-edges between vertices of different distances from v. Note that an edge connects two vertices of different distance from v if and only if it lies on a shortest path between v and one of these two vertices. The shortest path rooted at v implicitly confirms the absence of all edges between vertices of different distance from v. So given an edge or non-edge whose status is not yet resolved, say (v, u), we query both the end points v and u to determine the distances of all nodes to u and v. From this we can deduce the set $H_{(u,v)}$ of nodes from which the edge or non-edge between u and v can be discovered: $H_{(u,v)} = \{x \in V | d(u, x) \neq d(v, x)\ d(s, x) = \text{distance from } s \text{ to } x\}$.

Algorithm **Pseudo Greedy** described in the previous section above translates to the following in the context of the network discovery problem. Randomly pick a undiscovered edge and query the set $H_{(u,v)}$. Let n be the number of vertices in the graph and let Q denote the query set- this is the (approximately minimal) set of vertices which will be used to discover the network. If v is contained in at least $\alpha \log n$ of the queried sets, include v in the set-cover Q. Actually, like the general set-cover problem, it is a two stage process where we first shortlist and then subsequently run through this list in some predefined ordering, say according to the labels of the vertices. As before, we solve the set-cover problem on Q_v using a sequence of $H_{(u,v)}$ hitting set queries. The reader can easily work out the details that we omit to avoid repetition.

In the following algorithm $N = O(n^2)$. The algorithm takes $O(\log n)$ stages and in each stage we make $O(\log n \cdot \texttt{OPT})$ queries, where $\texttt{OPT}$ is the optimum number of queries required to solve the network verification problem. Since this is also optimum for the online problem, Algorithm **Network Discovery** makes $O(\log^2 n \cdot \texttt{OPT})$ queries. The algorithm yields a set $O(\log n \cdot \texttt{OPT})$ Q_v queries that suffices to discover the given network. Therefore the overall number of queries for the online discovery is still $O(\log^2 n \cdot \texttt{OPT})$.

Algorithm 2. Network Discovery

for $i = 0, 1 \ldots$ **do**

1: Let n_i be the number of edges and non-edges which needs to be discovered and $s_i = \min\{\frac{\binom{n}{2}}{2^i}, n_i\}$. Choose a random sample of R^i of size $(4\alpha n_i / s_i) \log N$.
2: If $s_i \leq \alpha \log N$ then find $H_{(u,v)}$ for each of the undiscovered edge/non-edge and solve the network discovery problem by reducing it explicitly to the set-cover problem.
3: If $s_i > \alpha \log N$, for each sampled edge/non-edge (u, v), find the set $H_{(u,v)}$.
4: Consider the vertices $\{v_1, v_2, \ldots\}$ in this order and include v_j in Q (Q_{v_j} is in the set-cover) only if Q_{v_j} contains more than $\alpha \log N$ sampled edge/non-edge. ($v_j \in H_{(u,v)}$ for at least $\alpha \log N$ of the $(u, v) \in R_i$).
(The actual implementation of this is similar to Steps 3-4 of the Algorithm **Pseudo Greedy**.)

From our earlier analysis of the covert set-cover problem it follows that

Theorem 2. *There is a $O(\log^2 n)$-competitive randomized Monte Carlo algorithm for the network discovery problem in the Layered Graph Query Model.*

6 Conclusion and Open Problem

The algorithm described in the last section gave a $O(\log^2 n)$ algorithm for the network discovery problem – Can we improve this to $O(\log n)$? We can consider a weighted version of the network discovery problem, where each query at a vertex costs say w_v, it is not clear whether we can extend our approach to solve the weighted version of the problem.

We note that in the *Distance Query Model*, by querying both v and u, we can discover if u or v is a edge or non-edge. If it is a non-edge, then we can find the set $H_{(u,v)}$ – a vertex w is in this set if $d(u,w) - d(v,w) \geq 2$. But if (u,v) is an edge, then we can not find the set $H_{(u,v)}$. It is not clear how to determine the *partial* witnesses, using set-cover queries as before. Therefore, it remains open if we can we improve the known $O(\sqrt{n \log n})$ bound for network discovery problem to $O(poly(\log n))$ approximation randomized algorithm in the *Distance Query Model*?

Acknowledgement. The first author is thankful to Rajeev Raman and Thomas Erlebach for introducing him to the problem and subsequent technical discussions.

References

1. Alon, N., Awerbuch, B., Azar, Y., Buchbinder, N., Naor, J.: The online set cover problem, pp. 100–105 (2003)
2. Awerbuch, B., Azar, Y., Fiat, A., Leighton, F.T.: Making commitments in the face of uncertainty: How to pick a winner almost every time (extended abstract), pp. 519–530 (1996)
3. Beerliova, Z., Eberhard, F., Erlebach, T., Hall, A., Hoffmann, M., Mihaľák, M., Shankar Ram, L.: Network discovery and verification. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 127–138. Springer, Heidelberg (2005)
4. Beerliova, Z., Eberhard, F., Erlebach, T., Hall, A., Hoffmann, M., Mihalák, M., Shankar Ram, L.: Network discovery and verification. IEEE Journal on Selected Areas in Communications 24(12), 2168–2181 (2006)
5. Bejerano, Y., Rastogi, R.: Robust monitoring of link delays and faults in ip networks. In: INFOCOM (2003)
6. Jonson, D.S.: Approximation algorithms for combinatorial problem. Journal of Computer and System Sciences (9), 256–278 (1974)
7. Feige, U.: A threshold of ln for approximating set cover. J. ACM 45(4), 634–652 (1998)
8. Khuller, S., Raghavachari, B., Rosenfeld, A.: Landmarks in graphs. Discrete Applied Mathematics 70(3), 217–229 (1996)
9. Grey, M.R., Jonson, D.S.: Computers and intractability. Freeman, New York (1979)
10. Vazirani, V.V.: Approximation algorithms. Springer, New York (2001)
11. Chatal, V.: A greedy heuristic for the set-covering problem. Mathematics of Operations Research (4), 233–235 (1979)

Appendix A

Chernoff bounds
If a random variable X is the sum of n iid Bernoulli trials with a success probability of p in each trial, the following equations give us concentration bounds of deviation of X from the expected value of np. These are useful for small deviations from a large expected value.

$$Prob(X \leq (1-\epsilon)pn) \leq exp(-\epsilon^2 np/2) \quad (1)$$

$$Prob(X \geq (1+\epsilon)np) \leq exp(-\epsilon^2 np/4) \quad (2)$$

for all $0 < \epsilon < 1$.

Greedy set-cover
For completeness, we also sketch the proof of approximation factor of $RGSC(\theta)$ for $\theta < 1$, such that at any step, the size of the set chosen is at least $\theta \cdot n_{max}$.

Let us number the elements of S in the order they were covered by the greedy algorithm (wlog, we can renumber such that they are $x_1, x_2 \ldots$). We will apportion the cost of covering an element $e \in S$ as $w(e) = \frac{1}{U-V}$ where e is covered for the first time by U and V is set of elements covered till then. This is also called the *cost-effectiveness* of set U. The total cost of the cover is

$$\sum_U \sum_{e \in U} \frac{1}{n(U)}$$

where $n(U)$ is the number of uncovered elements in U when U was chosen and e is covered for the first time. This can be rewritten as $\sum_i w(x_i)$.

Lemma 3

$$w(x_i) \leq \frac{C_o/\theta}{n-i+1}$$

where C_o is the number of sets in the optimum cover.

In the iteration when x_i is covered for the first time, the number of uncovered elements is $\geq n-i+1$. The pure greedy choice is more cost effective than any left over set of the optimal cover. Suppose $S_{i_1}, S_{i_2} \ldots S_{i_k}$ are the unselected sets of the minimum set-cover. Then, at least one of them has a cost-effectiveness of $\leq \frac{k}{n-i+1} \leq \frac{C_o}{n-i+1}$. It follows that the set chosen by $RGSC(\theta)$ achieves a cost-effectiveness of $\frac{C_o}{(n-i+1)\theta}$. So $w(x_i) \leq \frac{C_o/\theta}{n-i+1}$.

Thus the cost of the greedy cover is $\sum_i \frac{C_o/\theta}{n-i+1}$ which is bounded by $C_o/\theta \cdot H_n$. Here $H_n = \frac{1}{n} + \frac{1}{n-1} + \ldots 1$.

Variants of Spreading Messages

T.V. Thirumala Reddy, D. Sai Krishna, and C. Pandu Rangan

Department of Computer Science and Engineering,
Indian Institute of Technology Madras,
Chennai 600036, India
{tiru114,dsaikris86,prangan55}@gmail.com

Abstract. In a distributed computing environment a faulty node could lead other nodes in the system to behave in a faulty manor. An initial set of faults could make all the nodes in the system become faulty. Such a set is called an irreversible dynamo. This is modelled as spreading a message among individuals V in a community $G = (V, E)$ where E represents the acquaintance relation. A particular individual will believe a message if some of the individual's acquaintances believe the same and forward the believed messages to its neighbours. We are interested in finding the minimum set of initial individuals to be considered as convinced, called the MIN-SEED, such that every individual in the community is finally convinced. We solve for MIN-SEED on some special classes of graphs and then give an upper bound on the cardinality of the MIN-SEED for arbitrary undirected graphs. We consider some interesting variants of the problem and analyse their complexities and give some approximate algorithms.

Keywords: Vertex Cover, Bipartite Graphs, Approximate Algorithms, Fault Tolerance, NP-complete.

1 Introduction

In a distributed computing environment a node could become faulty. A faulty node could make some other nodes in the system behave in a faulty manor. In order to design a fault tolerant system, we need to examine some faulty nodes as well as the cumulative effect of these initial faulty nodes on other nodes of the system. We are interested in the patterns of the initial faults that can occur and then could lead all the other nodes in the system behave in a faulty manor. The initial set of faults that leads all the nodes to become faulty is called a *dynamic monopoly* in the system. Faults can be temporary or permanent. If we consider the faults of the system as permanent then the problem is called as the *irreversible dynamo* [1]. This problem is modelled in graph theory as the SPREADING MESSAGE problem.

In the SPREADING MESSAGE problem we have a set of individuals representing the vertices in a graph and the acquaintance relation of individuals represents the edges of the graph. An individual believes a message when he/she receives it from his acquaintances, who are already convinced by the message. Every vertex

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 240–251, 2010.

v has a threshold $\alpha(v)$. A vertex is considered as *convinced*, if at least $\alpha(v)$ of its neighbours are already convinced. We are interested in finding a minimum cardinality set of individuals to be convinced who can eventually convince all the individuals. We can now observe that in the spreading messages problem, vertices represent nodes in a distributed environment and a convinced vertex represents a faulty node. The threshold function $\alpha(v)$ represents fault tolerance of an individual node. This problem was considered by Peleg [2], where there were white and black nodes corresponding to good and faulty nodes. The problem was largely studied on random graphs [3]. A variant of the problem is to consider the majority scenarios, where a vertex will be convinced if majority of its neighbours are convinced. Majority scenarios like strict majority and week majority were considered in the past on tori[4], butterfly graph[5] and chordal rings[6]. The problem with an arbitrary threshold function was first considered by Ching-Lueh Chang and Yuh-Dauh Lyuu [7,8], where it is shown to be NP-complete on arbitrary undirected graphs.

In our paper, we give an upper bound on MIN-SEED of unbounded spreading messages of an arbitrary undirected graph and shows that the problem is NP-Complete on bipartite graphs. Then we provide polynomial time algorithms for the unbounded spreading messages problem on special classes of graphs like trees, cliques, complete bipartite graphs, threshold graphs and chain graphs. The first variant we consider is spreading messages within one round. For this variant we give a lower bound and an $(H(\alpha_M) + H(\Delta))$ approximation algorithm, where $H(n)$ represents the sum of first n terms in the harmonic series, α_M represents the maximum threshold of vertices in the graph and Δ represents the maximum degree of a graph. We also show that this variant is APX-Complete on bounded degree 3 graphs and on p-claw free graphs we provide an $\frac{\alpha_M \cdot (p-1)}{\alpha_m}$ approximation algorithm, where α_m represents minimum threshold of vertices in the graph. Another variant we consider is spreading messages within k rounds. We show that this variant is NP-complete on arbitrary undirected graphs. Then we introduce spreading messages problem with real thresholds and belief factors. We show that this variant is NP-Complete on cliques and complete bipartite graphs. Finally we consider spreading messages with each individual having r radius of coverage and we give an $(H(\alpha_M) + H(n))$ approximation algorithm for arbitrary undirected graphs.

2 Notation and Definitions

A simple graph is a collection of vertices V and edges E represented as $G = (V, E)$, where each edge is an unordered pair of distinct vertices. In this paper we are considering only simple undirected connected graphs. We also assume that $|V| > 1$ for the graphs we consider. The set of neighbours of a vertex v is denoted by $N(v)$ and $N[v] = N(v) \cup \{v\}$. The distance between two vertices in a graph is the number of edges in a shortest path connecting them. $N^r(v)$ is the set of all vertices whose distance from v is less than or equal to r and $N^r[v] = N^r(v) \cup \{v\}$. For a subset $S \subseteq V$, $N_S(v) = N(v) \cap S$ and $N_S[v] = N_S(v) \cup \{v\}$. The degree

of a vertex v, $d(v)$, is defined as $d(v) = |N(v)|$. The maximum degree of G, $\Delta = \max_{v \in V}\{d(v)\}$. Two vertices u and v are called twin vertices if $N(u) = N(v)$. If $uv \in E$ then they are called true twin vertices otherwise they are called false twin vertices. Let $G = (V, E)$ be a simple undirected graph. Let $\alpha : V \to \mathbb{N}$ be a threshold function such that $1 \leq \alpha(v) \leq d(v)$ for all $v \in V$, where a vertex v is *convinced* if at least $\alpha(v)$ of neighbours of v are already convinced.

Definition 1. *Let $S_0 \subseteq V$ be a vertex subset. Then spreading of a message will happen in rounds. Let $C_0 = S_0 \subseteq V$ the initial set of vertices considered directly convinced.*

$$S_1 = \{x | \alpha(x) \leq |C_0 \cap N(x)|\} \cup S_0 \, , \, C_1 = S_1 \cup C_0$$
$$\cdots$$
$$S_i = \{x | \alpha(x) \leq |C_{i-1} \cap N(x)|\} \cup S_{i-1} \, , \, C_i = S_1 \cup C_{i-1}$$

Unbounded Spreading Messages: *S_0 is called* seed *if and only if $\bigcup_{i=0}^{\infty} S_i = V$.* MIN-SEED (G, α, ∞) *is defined as* $\min_S(|S|)$ *for all possible seeds S. A seed S with $|S| =$* MIN-SEED (G, α, ∞) *is called as an optimum seed.*

Bounded Spreading Messages with in k Rounds: *S_0 is called* seed *if and only if $\bigcup_{i=0}^{k} S_i = V$.* MIN-SEED (G, α, k) *is defined as* $\min_S(|S|)$ *for all seeds S.*

Unbounded Spreading Messages With Real Thresholds and Belief Factors: *Let $\beta : E \to \mathbb{R}^+$ be a mapping such that $0 < \beta(u, v) \leq 1$ and $\alpha : V \to \mathbb{R}^+$ be a mapping such that $1 \leq \alpha(v) \leq \sum_{u \in N(v)} \beta(u, v)$, for all $v \in V$. For an edge (u, v), $\beta(u, v)$ is call belief factor of (u, v). In any round C denotes the set of convinced vertices and $N_c(v)$ denote the set of convinced neighbours of a vertex v. A vertex v is convinced by a message if $\sum_{u \in N_c(v)} \beta(u, v) \geq \alpha(v)$. A set $S_0 \subseteq V$ is called* seed *if and only if $\bigcup_{i=0}^{\infty} S_i = V$.* MIN-SEED$(G, \alpha, \beta, \infty)$ *is defined as* $\min_S(|S|)$ *for all seeds S.*

3 Unbounded Spreading Messages

3.1 Complexity

Theorem 1. MIN-SEED (G, α, ∞) *is NP-Complete, when G is a bipartite graph.*

Proof. We prove that the decision version of the MIN-SEED (G, α, ∞) problem is NP-Complete when G is a bipartite graph, by giving a reduction from the SET COVER problem.

Construction: Given an instance of the SET COVER problem with an universal set $U = \{x_1, x_2, x_3, ..., x_n\}$, a set of subsets $S = \{S_1, S_2, ..., S_m\}$ and an integer s, where $S_i \subseteq U$. Construct a bipartite graph $G = (X, Y, E)$, with $|X| = \{s_1, s_2, s_3, ..., s_m\}$ and $|Y| = \{x_1, x_2, x_3, ..., x_n\}$. That is the vertex set X

contains a vertex for every set of $\mathbb{S}$ and the vertex set Y contains a vertex for every element of U. If the element $x_j \in S_i$ then connect the vertex s_i to the vertex x_j. Set $\alpha(s_i) = d(s_i), \forall s_i \in X$ and $\alpha(x_j) = 1, \forall x_j \in Y$. Now we prove that there is a solution for SET COVER problem of size s if and only if there is a solution of size s to the corresponding MIN-SEED (G, α, ∞) instance.

Let there exist a solution for SET COVER problem with size s. Now the solution for the MIN-SEED(G, α, ∞) problem is: for every set S_i in the SET COVER solution, choose the vertex s_i of X in the seed. These s vertices of X first convince all the vertices of Y. Then the remaining vertices of X get convinced. Because $\forall s_i \in X, \alpha(s_i) = d(s_i)$ and all the neighbours of s_i (vertices of Y) are convinced.

Let there exist a solution for MIN-SEED(G, α, ∞) with size s. If the vertex s_i of X is in the MIN-SEED(G, α, ∞) then include set S_i in set cover solution. If a vertex $x_j \in Y$ is in MIN-SEED(G, α, ∞) then choose any neighbour of the vertex x_j. Let say the vertex s_k of X is chosen, then include the set S_k in set cover solution. Now we prove that the sets chosen cover all the elements of U. Let us assume that some element $x_j \in U$ not covered. Consider the possibilities of how the vertex $x_j \in Y$ is convinced. Definitely the vertex x_j and the neighbours of x_j are not in MIN-SEED(G, α, ∞) solution, so the vertex x_j must be convinced by its neighbours. Neighbours of the vertex x_j are convinced if and only if the vertex x_j is convinced because $\forall s_i \in N(x_j), \alpha(s_i) = d(s_i)$. This implies that the vertex x_j never gets convinced, which is a contradicting statement. Therefore the sets we chosen is a SET COVER solution. □

Theorem 2. VERTEX COVER *is the upper bound for* MIN-SEED(G, α, ∞).

Proof. First we show how to construct an instance of the MIN-SEED(G, α, ∞) problem from VERTEX COVER problem, then we prove that the VERTEX COVER is upper bound for the MIN-SEED(G, α, ∞).

Construction: Given an instance of VERTEX COVER problem with a graph $G = (V, E)$ and a positive integer s, construct an instance of the MIN-SEED problem with the same graph G. Define $\alpha(v) = d(v)$ for all $v \in V$. Now we prove that there is a solution for VERTEX COVER problem of size s if and only if there is a solution of size s to the corresponding MIN-SEED (G, α, ∞) instance.

Let there exist a solution for VERTEX COVER problem with s vertices. These s vertices also give us a solution for MIN-SEED(G, α, ∞), because if a vertex v is not in the VERTEX COVER then all its neighbours must be there in the VERTEX COVER. So all the vertices not in VERTEX COVER get convinced if we convince vertices in VERTEX COVER.

Let there exist a solution for MIN-SEED(G, α, ∞) with size s. Now we prove that these s vertices gives a solution for VERTEX COVER. Let us assume that some edge (u, v) is not covered. Both vertices u and v are not in the MIN-SEED solution. Now consider the possibilities of how the vertices u and v get convinced. In order to convince the vertex u, first the vertex v must be convinced. Similarly, in order to convince the vertex v, first vertex u must be convinced. This leads to a contradiction that neither of the vertex u nor the vertex v gets convinced. So one of the vertices u, v must be there in MIN-SEED(G, α, ∞) solution.

As we are setting $\alpha(v)$ to the maximum possible value $\forall v \in V$, MIN-SEED$(G, \alpha', \infty) \leqslant$ MIN-SEED(G, α, ∞), where α' is any threshold function from $V \to \mathbb{N}$. □

Corollary 1. *Upper Bound for* MIN-SEED$(G, \alpha, 1)$ *is* Vertex Cover *of* G.

3.2 Exact Algorithms

Lemma 1. *Let v be a vertex in $G = (V, E)$ such that $\alpha(v) = 1$ then there is an optimum seed without v.*

Lemma 2. *Let $G = (V, E)$ be a graph, $u, v \in V$ be two false twin(or true twin) vertices such that $\alpha(v) \leq \alpha(u)$. Let S be an optimum seed such that $v \in S$ and $u \notin S$. Then there exists an optimum seed S' such that $u \in S$ and $v \notin S$.*

Lemma 3. *Let $G = (V, E)$ be a graph, $u, v \in V$ be two vertices such that $N(v) \subset N(u)$. Let S be an optimum seed such that $v \in S$ and $u \notin S$. Then there exists an optimum seed S' such that $u \in S'$ and $v \notin S'$.*

Trees

Theorem 3. MIN-SEED (T, α, ∞) *can be calculated in $\mathcal{O}(n)$ time, where T is a tree and n is the number of vertices in T.*

Theorem 4. MIN-SEED (C_n, α, ∞) *where C_n is a cycle can be calculated in $\mathcal{O}(n)$ time.*

Complete Graphs

Let $\langle \alpha_1, \alpha_2 \ldots \alpha_m \rangle$ be distinct threshold values taken by vertices in the complete graph K_n with n vertices in ascending order. We define $V_i = \{x | \alpha(x) = \alpha_i\}, i \leq i \leq m$ as a partition of vertices of K_n.

Definition 2. *A seed S of (K_n, α) is called a greedy seed if and only if $\forall x \in S$ and $1 \leq h \leq m$, if $x \in V_h$ then $V_i \subseteq S, \forall i > h$. That means that all the seed vertices are concentrated in vertices of larger threshold values in a greedy seed.*

Lemma 4. *There exists a greedy seed for (K_n, α) of minimum cardinality k.*

Theorem 5. MIN-SEED (K_n, α, ∞) *can be calculated in $\mathcal{O}\left(n^2\right)$ time.*

Theorem 6. MIN-SEED (K_{mn}, α, ∞) *can be calculated in $\mathcal{O}\left(|V|^2 \cdot \log |V|\right)$ time.*

Working with the sorted sequence of threshold values which we can get in linear time using bucket sort we can compute MIN-SEED (K_n, α, ∞) in linear time.

Corollary 2. MIN-SEED (G, α, ∞)*, where G is a complete graph K_n can be calculated in $\mathcal{O}(n)$ time.*

For proofs and algorithm on Threshold Graphs please refer extended version [9].

4 Spreading Messages within One Round

4.1 Complexity

Theorem 7. *Lower bound for cardinality of* MIN-SEED$(G, \alpha, 1)$ *is the cardinality of minimum* DOMINATING SET *of* G.

Proof. First we show how to reduce the DOMINATING SET problem to the MIN-SEED$(G, \alpha, 1)$ problem, then we prove that DOMINATING SET is lower bound for the MIN-SEED$(G, \alpha, 1)$.

Construction: Given an instance of the DOMINATING SET problem with a graph $G = (V, E)$ and a positive integer s, construct an instance of MIN-SEED$(G, \alpha, 1)$ problem with the same graph G. Define $\alpha(v) = 1$ for all $v \in V$. Now we prove that there is a solution for DOMINATING SET problem of size s if and only if there is a solution of size s to the corresponding MIN-SEED $(G, \alpha, 1)$ instance.

Let there exist a solution for the DOMINATING SET problem with s vertices. These s vertices also give us a seed, because if a vertex v is not in the DOMINATING SET then one of its neighbours must be there in the DOMINATING SET. So all vertices not in DOMINATING SET get convinced if we convince vertices in DOMINATING SET in one round.

Let there exist a solution for the MIN-SEED$(G, \alpha, 1)$ of size s. Now we prove that these s vertices also give a DOMINATING SET. Let us assume that some vertex v is not dominated by any vertex. The vertex v and the vertices in $N(v)$ are not in MIN-SEED$(G, \alpha, 1)$ solution. The vertices belongs to $N(v)$ get convinced in one round but in order to convince v we need two rounds. This leads to a contradiction. So either v or a vertex from $N(v)$ must be there in the MIN-SEED solution.

As we are setting $\alpha(v)$ to the minimum possible value $\forall v \in V$, MIN-SEED$(G, \alpha', 1) \geq$ MIN-SEED$(G, \alpha, 1)$ where α' is any threshold function from $V \to \mathbb{N}$. □

Corollary 3. *Computing* MIN-SEED$(G, \alpha, 1)$ *is NP-Complete.*

4.2 Approximation

Theorem 8. *If there is an* $\epsilon > 0$ *such that a polynomial time algorithm can approximate* MIN-SEED$(G, \alpha, 1)$ *within* $(1 - \epsilon) \ln |V|$, *then* $NP \subseteq TIME(n^{O(\log \log |V|)})$.

Proof. In theorem 7 we have shown that how to reduce every instance of the DOMINATING SET to an instance of MIN-SEED$(G, \alpha, 1)$. We know that every instance of the SET COVER problem can be reduced to the DOMINATING SET problem. Feige proved the threshold of $\ln(n)$ approximation for SET COVER [10]. Therefore the same theorem holds true here. □

We now give an $(H(\alpha_M)+H(\Delta))$ approximation algorithm for MIN-SEED$(G, \alpha, 1)$. Therefore, $(H(\alpha_M) + H(\Delta))$ is the upper bound for MIN-SEED$(G, \alpha, 1)$.

Algorithm 1. Algorithm to compute MIN-SEED$(G, \alpha, 1)$.

Require: A graph $G = (V, E)$, a function $\alpha : V \to \mathbb{N}$.
1: $S \leftarrow \emptyset$ {S is the seed}
2: $C \leftarrow \emptyset$ {C is the set of vertices that are convinced. At the end of the algorithm C must be equal to V}
3: $i \leftarrow 0$
4: **while** $C \neq V$ **do**
5: $i \leftarrow i + 1$
6: Choose a vertex $v \notin S$ which maximizes $|N[v] \cap (V \setminus C)|$
7: $S \leftarrow S \cup \{v\}$
8: **if** $v \notin C_1 \cup C_2 \cup ... \cup C_{i-1}$ **then**
9: $C_i \leftarrow \{v\} \cup$ set of vertices newly convinced by choosing v
10: **else**
11: $C_i \leftarrow$ set of vertices newly convinced by choosing v
12: **end if**
13: $C \leftarrow C \cup C_i$
14: **end while**
15: **return** S {S is the seed}

Lemma 5. *Algorithm 1 runs in polynomial amount of time.*

Proof. The *while* loop at line number 4 of *Algorithm 1* can execute maximum $|V|-1$ times. For each $v \in V$ step 6 can take linear time. In worst case *Algorithm 1* takes $\mathcal{O}(n^3)$ time. □

Let the while loop of Algorithm 1 execute s times and let the vertices chosen as the seed be $v_1, v_2, ..., v_s$. Therefore the size of the seed given by Algorithm 1 is s. Now for $i = 1$ to s and $\forall u \in (N[v_i] \cap (V - C_1 \cup C_2 \cup ... \cup C_{i-1}))$ assign cost $\frac{1}{|N[v_i] \cap (V - C_1 \cup C_2 \cup ... \cup C_{i-1})|}$.

For every $v \in V$, at most $\alpha(v)$ values are assigned. Let the values assigned to v be $d_v^1, d_v^2, ..., d_v^{\alpha(v)}$. Define $c'_v = \sum_{1 \leq i \leq \alpha(v)} d_v^i$ and $c_v = \max_{1 \leq i \leq \alpha(v)} d_v^i$.

Lemma 6. $d_v^1 \leq d_v^2 \leq ... \leq d_v^{\alpha(v)}$ *and* $c'_v \leq \alpha(v) c_v$.

Proof. From Algorithm 1 it is obvious that, for $1 \leq i < s$

$$|N[v_i] \cap (V \setminus C_1 \cup C_2 \cup ... \cup C_{i-1})| \geq |N[v_{i+1}] \cap (V \setminus C_1 \cup C_2 \cup ... \cup C_i)|.$$

Therefore, $\forall v \in V$ $d_v^i \leq d_v^{i+1}$, until d_v^{i+1} is defined and $\forall v \in V \setminus S$, $c_v = d_v^{\alpha(v)}$. Now from the definition of c'_v, we have

$$\forall v \in V,\ c'_v = \sum_{1 \leq i \leq \alpha(v)} d_v^i \leq \alpha(v) c_v.$$ □

Let $|X|$ be the size of the seed given by Algorithm 1 and let $|X^*|$ be the optimal solution.

Lemma 7. $|X| = \sum_{v \in V} c'_v$.

Lemma 8. *For all* $v \in V$, $c'_v \leq H(\alpha(v))$.

Proof. From the definition of c'_v, we know that $c'_v = \sum_{1 \leq i \leq \alpha(v)} d^i_v$. Define α^i_v be the remaining threshold value of v after choosing $v_1, v_2, ..., v_{i-1}$. Therefore,

$$\alpha(v) = \alpha^0_v \geq \alpha^1_v \geq \alpha^2_v \geq ... \geq \alpha^s_v.$$

From the definition of α^i_v, we have

$$\alpha^{i-1}_v \leq |N[v] \cap (V \setminus C_1 \cup C_2 \cup ... \cup C_{i-1})| \leq |N[v_i] \cap (V \setminus C_1 \cup C_2 \cup ... \cup C_{i-1})|.$$

Therefore,

$$\begin{aligned} c'_v &= \sum_{1 \leq i \leq \alpha(v)} d^i_v = \sum_{1 \leq i \leq s} (\alpha^{i-1}_v - \alpha^i_v) \frac{1}{|N[v_i] \cap (V \setminus C_1 \cup C_2 \cup ... \cup C_{i-1})|} \\ &\leq \sum_{1 \leq i \leq s} (\alpha^{i-1}_v - \alpha^i_v) \frac{1}{\alpha^{i-1}_v} \leq H(\alpha(v). \end{aligned}$$

□

Lemma 9. $\sum_{v \in V} c'_v \leq \sum_{v \in X^*} \sum_{u \in N(v)} c_u + \sum_{v \in X^*} c'_v$.

Proof. We know that X^* is the optimal solution for MIN-SEED$(G, \alpha, 1)$. We can divide $\sum_{v \in V} c'_v$ as,

$$\sum_{v \in V} c'_v = \sum_{v \in X^*} c'_v + \sum_{v \in V \setminus X^*} c'_v.$$

From lemma 6, we have $c'_v \leq \alpha(v) c_v$. Therefore,

$$\sum_{v \in V \setminus X^*} c'_v \leq \sum_{v \in V \setminus X^*} \alpha(v) c_v.$$

As X^* is a seed $\forall v \in V \setminus X^*$, $|N(v) \cap X^*| \geq \alpha(v)$. Therefore,

$$\begin{aligned} \sum_{v \in V \setminus X^*} \alpha(v) c_v &\leq \sum_{v \in V \setminus X^*} |N(v) \cap X^*| c_v \\ &= \sum_{v \in V \setminus X^*} \sum_{1 \leq i \leq |N(v) \cap X^*|} c_v \\ &= \sum_{u \in X^*} \sum_{v \in N(u) \cap V \setminus X^*} c_v \\ &\leq \sum_{u \in X^*} \sum_{v \in N(u)} c_v. \end{aligned}$$

Therefore,

$$\begin{aligned}\sum_{v\in V} c'_v &= \sum_{v\in X^*} c'_v + \sum_{v\in V\setminus X^*} c'_v \\ &\leq \sum_{v\in X^*} c'_v + \sum_{v\in V\setminus X^*} \alpha(v)c_v \\ &\leq \sum_{v\in X^*} c'_v + \sum_{u\in X^*}\sum_{v\in N(u)} c_v.\end{aligned}$$

□

Lemma 10. $\sum_{u\in N(v)} c_u \leq H(\Delta)$.

Proof. Let v be a vertex in V. Define z_v^i be the number of unconvinced neighbours in $N(v) \cap (V \setminus C_1 \cup C_2 \cup ... \cup C_i)$, where $1 \leq i \leq s$. Therefore,

$$|N(v)| = z_v^0 \geq z_v^1 \geq z_v^2 \geq ... \geq z_v^s.$$

From Algorithm 1 and from the definition of c_v, we have

$$\sum_{u\in N(v)} c_u = \sum_{1\leq i\leq s} (z_v^{i-1} - z_v^i)\frac{1}{|N[v_i] \cap (V \setminus C_1 \cup C_2 \cup ... \cup C_{i-1})|}.$$

From Algorithm 1, we know that

$$z_v^{i-1} \leq |N[v_i] \cap (V \setminus C_1 \cup C_2 \cup ... \cup C_{i-1})|.$$

Therefore,

$$\begin{aligned}\sum_{u\in N(v)} c_u &= \sum_{1\leq i\leq s} (z_v^{i-1} - z_v^i)\frac{1}{|N[v_i] \cap (V \setminus C_1 \cup C_2 \cup ... \cup C_{i-1})|} \\ &\leq \sum_{1\leq i\leq s} (z_v^{i-1} - z_v^i)\frac{1}{z_v^{i-1}} \leq H(|N(v)|) \leq H(\Delta).\end{aligned}$$

□

Theorem 9. *Algorithm 1 is* $(H(\alpha_M) + H(\Delta))$ *approximation algorithm for* MIN-SEED$(G, \alpha, 1)$.

Proof. From lemma 7 and lemma 9, we have

$$|X| = \sum_{v\in V} c'_v \leq \sum_{v\in X^*}\sum_{u\in N(v)} c_u + \sum_{v\in X^*} c'_v.$$

From lemma 8 and lemma 10, we have

$$c'_v \leq H(\alpha_M) \text{ and } \sum_{u\in N(v)} c_u \leq H(\Delta).$$

Therefore,

$$\begin{aligned}|X| &\leq \sum_{v\in X^*} H(\alpha_M) + \sum_{v\in X^*} H(\Delta) \\ &= (H(\alpha_M) + H(\Delta))|X^*|.\end{aligned}$$

□

4.3 Bounded Degree Graphs

Definition 3. *Given two NP optimization problems P and Q and a polynomial transformation f from instances of P to instances of Q, we say that f is an $L-$reduction if there are positive constants a and b such that for every instance x of P*

1. $opt_Q(f(x)) \leq a \cdot opt_P(x)$,
2. *for every feasible solution y of $f(x)$ with objective value $m_Q(f(x), y) = c_2$ we can in polynomial time find a solution y' of x with $m_P(f(x), y') = c_1$ such that $|opt_P(x) - c_1| \leq b \cdot |opt_Q(f(x)) - c_2|$.*

To show the APX-completeness of a problem $P \in APX$, it is enough to show that there is an L-reduction from some APX-complete problem to P.

Theorem 10. MIN-SEED$(G, \alpha, 1)$ *is $APX - Complete$ for bounded degree graphs.*

Proof. From Theorem 9 if the degree of a graph G is bounded by constant then Algorithm 1 gives a constant approximation ratio. This implies that MIN-SEED$(G, \alpha, 1)$ belongs to APX. For proving $APX - Complete$ we have to give a $L - reduction$ from known $APX - complete$ problem. We know that DOMINATING SET on bounded degree graphs is $APX - Complete$. The reduction used in Theorem 7 to show $NP - complete$ also acts as the reduction to show $APX - Complete$ with a constant $a = 1$ and $b = 1$ (See Definition 3). □

4.4 p-Claw Free Graphs

Definition 4. *A graph $G = (V, E)$ is called a p-claw free graph if for all the vertices v, the subgraph induced by $N(v)$ does not have an independent set of size p.*

Alternately, a graph $G = (V, E)$ is called a p-claw free graph if there is no induced subgraph of G isomorphic to the star graph K_{1p}.

Lemma 11. *Let $G = (V, E)$ be any p-claw free graph and let $\alpha : V \to \mathbb{N}$ be any threshold function. Let $|D^*_\alpha|$ be any optimal solution for* MIN-SEED$(G, \alpha, 1)$ *and S be any Maximal Independent Set of G. Let α_m be the minimum threshold value of S vertices. Then $|S| \leq \frac{p-1}{\alpha_m}|D^*_\alpha|$.*

Proof. For all $u \in S \setminus D^*_\alpha$, define $c_u = |(D^*_\alpha \setminus S) \cap N(v)|$. Let α_m is the minimum threshold of S vertices then

$$\sum_{u \in S \setminus D^*_\alpha} c_u \geq \alpha_m |S \setminus D^*_\alpha|$$

For all $v \in D^*_\alpha \setminus S$, define $d_v = |(S \setminus D^*_\alpha) \cap N(v)|$. We know that $\forall v \in D^*_\alpha$ there are at most $(p-1)$ independent vertices in its neighbourhood and $d_v \leq p - 1$.

$$\sum_{v \in D^*_\alpha \setminus S} d_v \leq (p-1)|D^*_\alpha \setminus S|$$

Now consider the definitions of c_u and d_v,

$$\sum_{u \in S \setminus D^*_\alpha} c_u = \{(u,v) \in E \text{ such that } u \in S \setminus D^*_\alpha \text{ and } v \in D^*_\alpha \setminus S\} = \sum_{v \in D^*_\alpha \setminus S} d_v.$$

Therefore,

$$\alpha_m |S \setminus D^*_\alpha| \leq \sum_{u \in S \setminus D^*_\alpha} c_u = \sum_{v \in D^*_\alpha \setminus S} d_v \leq (p-1)|D^*_\alpha \setminus S|$$

$$\begin{aligned}
\alpha_m |S \setminus D^*_\alpha| &\leq (p-1)|D^*_\alpha \setminus S| \\
\alpha_m (|S| - |S \cap D^*_\alpha|) &\leq (p-1)(|D^*_\alpha| - |D^*_\alpha \cap S|) \\
\alpha_m |S| - \alpha_m |S \cap D^*_\alpha| &\leq (p-1)|D^*_\alpha| - (p-1)|D^*_\alpha \cap S| \\
\alpha_m |S| &\leq (p-1)|D^*_\alpha| - (p + \alpha_m - 1)|D^*_\alpha \cap S| \\
\alpha_m |S| &\leq (p-1)|D^*_\alpha| \\
|S| &\leq \frac{p-1}{\alpha_m} |D^*_\alpha|.
\end{aligned}$$

□

Algorithm 2. Algorithm to compute MIN-SEED$(G, \alpha, 1)$.

Require: A graph $G = (V, E)$, a function $\alpha : V \to \mathbb{N}$.

```
1: D ← ∅                                   {D is the seed}
2: C ← ∅                                   {C is the set of vertices that are convinced.}
3: i ← 0
4: while C ≠ V do
5:     i ← i + 1
6:     Choose a Maximal Independent Set S_i from V \ C
7:     C ← C ∪ S_i ∪ { set of vertices newly convinced by choosing v}
8:     D ← D ∪ S_i
9: end while
10: return D                               {D is the seed}
```

Lemma 12. *Algorithm 2 is a* $\frac{\alpha_M(p-1)}{\alpha_m}$ *approximation algorithm for* MIN-SEED$(G, \alpha, 1)$ *on p-claw free graphs, where* α_M *is the maximum threshold in* G.

Proof. Let D be the solution given by Algorithm 2 and let D^*_α be the optimal solution for MIN-SEED$(G, \alpha, 1)$. Let the while loop of Algorithm 2 execute k times and let the Maximal Independent Sets chosen be $S_1, S_2, ..., S_k$. From lemma 11, we have

$$|S_i| \leq \tfrac{p-1}{\alpha_m} |D^*_\alpha|, \text{ where } 1 \leq i \leq k.$$

By summation of all Maximal Independent Sets, we have

$$|S_1| + |S_2| + ... + |S_k| \leq \tfrac{k(p-1)}{\alpha_m} |D^*_\alpha|.$$

From Algorithm 2, we know that $|D| = |S_1| + |S_2| + ... + |S_k|$.
Therefore,

$$|D| \leq \frac{k(p-1)}{\alpha_m}|D^*_\alpha|.$$

The maximum possible value for k is α_M.
Therefore,

$$|D| \leq \frac{\alpha_M(p-1)}{\alpha_m}|D^*_\alpha|. \qquad \square$$

For definitions and detailed results on other variants please refer extended version [9].

5 Conclusion

In this paper we provided exact algorithms for Spreading Messages problem on several special graph classes and we also provided some complexity results. Then we considered several variants of the problem and provided complexity results and approximation algorithms for the variants of this problem.

No approximation algorithm exists for the unbounded spreading messages problem. We proved that MIN-SEED(G, α, ∞) is NP-Complete on *Bipartite Graphs* and solved in polynomial time on *Complete Bipartite Graphs*. So, it is also interesting to look at this problem on *Bipartite Permutation Graphs*.

References

1. Chang, C.-L., Lyuu, Y.-D.: On irreversible dynamic monopolies in general graphs. CoRR abs/0904.2306 (2009)
2. Peleg, D.: Size bounds for dynamic monopolies. Discrete Applied Mathematics 86(2-3), 263–273 (1998)
3. Watts, D.: A simple model of global cascades on random networks. P. Natl. Acad. Sci. USA 99(9), 5766–5771 (2002)
4. Flocchini, P., Lodi, E., Luccio, F., Santoro, N.: Irreversible dynamos in tori. In: Pritchard, D., Reeve, J.S. (eds.) Euro-Par 1998. LNCS, vol. 1470, pp. 554–562. Springer, Heidelberg (1998)
5. Luccio, F., Pagli, L., Sanossian, H.: Irreversible dynamos in butterflies. In: Gavoille, C., Bermond, J.C., Raspaud, A. (eds.) SIROCCO, pp. 204–218. Carleton Scientific (1999)
6. Flocchini, P., Geurts, F., Santoro, N.: Optimal irreversible dynamos in chordal rings. Discrete Applied Mathematics 113(1), 23–42 (2001)
7. Chang, C.L., Lyuu, Y.D.: Spreading messages. Theor. Comput. Sci. 410(27-29), 2714–2724 (2009)
8. Chang, C.L., Lyuu, Y.D.: Spreading of messages in random graphs. In: Downey, R., Manyem, P. (eds.) Fifteenth Computing: The Australasian Theory Symposium (CATS 2009), Wellington, New Zealand, ACS. CRPIT, vol. 94, pp. 3–7 (2009)
9. Reddy, T., Krishna, S., Rangan, P.: Variants of spreading messages (2010), http://www.cse.iitm.ac.in/~tiru/tiru/Publications_files/var_12page.pdf
10. Feige, U.: A threshold of ln for approximating set cover. J. ACM 45(4), 634–652 (1998)

On Finding a Better Position of a Convex Polygon Inside a Circle to Minimize the Cutting Cost

Syed Ishtiaque Ahmed, Md. Mansurul Alam Bhuiyan, Masud Hasan, and Ishita Kamal Khan

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka-1000, Bangladesh
ishtiaque@csebuet.org, {mansurul1985,ishitactg}@yahoo.com,
masudhasan@cse.buet.ac.bd

Abstract. The problem of cutting a convex polygon P out of a planar piece of material Q (P is already drawn on Q) with minimum total cutting cost is a well studied problem in computational geometry that has been studied with several variations such as P and Q are convex or non-convex polygons, Q is a circle, and the cuts are line cuts or ray cuts. In this paper, we address this problem without the restriction that P is fixed inside Q and consider the variation where Q is a circle and the cuts are line cuts. We show that if P can be placed inside Q such that P does not contain the center of Q, then placing P in a most cornered position inside Q gives a cutting cost of 6.48 times the optimal. We also give an $O(n^2)$-time algorithm for finding such a position of P, a problem that may be of independent interest. When any placement of P must contain the center of Q, we show that P can be cut of Q with cost 6.054 times the optimal.

Keywords: Polygon cutting, line cut, cutting cost, most cornered position, cornerable and non-cornerable polygon

1 Introduction

Let P be a convex polygon and Q be a circle where the circumcircle of P is no bigger than Q. We consider the problem of finding a position of P inside Q so that the total cost of cutting P out of Q by using line cuts is minimum. A *line cut*, or simply a *cut*, is a line segment with two endpoints that lie on the boundary of Q and does not intersect P. A cut always divides Q into two portions, lying to the left and right side of the cut. After a cut is applied, we update Q as the portion that contains P. The *cost* of a cut is its length.

Related Works. The problem has its origin in the problem of cutting a convex polygon P out of another polygon Q. Overmars and Welzl [10] first introduced this problem by cutting P out of Q with line cuts where the cuts are allowed

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 252–262, 2010.

only along the edges of P. Given P and Q with n and m edges respectively, they proposed an $O(n^3+m)$-time algorithm for this problem [10]. They also proposed another version of this problem where the line cuts are not restricted to touch the polygonal edges [10]. Bhaduri and Chandrasekaran [4] indicated that one can give only an approximation algorithm for this problem and presented an approximation scheme with pseudo-polynomial running time. The first polynomial-time approximation algorithm for this problem was given by Dumitrescu [7], who presented an $O(\log n)$-approximation algorithm with $O(mn+n\log n)$ running time. Introducing a constant factor approximation algorithm for this problem, Daescu and Luo [5] gave an algorithm with ratio $2.5 + ||Q||/||P||$, where $||P||$ is the perimeter of P and $||Q||$ is the perimeter of a minimum bounding rectangle of P. This algorithm has running time $O(n^3+(n+m)\log(n+m))$. Later, Tan [11] gave a 7.9-approximation algorithm for this problem. As the best known result so far, Bereg et. al. [3] gave a polynomial time approximation scheme (PTAS) for this problem with running time $O(m+n^6/\epsilon^{12})$. Recently, Ahmed et.al. [1] have given similar constant factor and $O(\log n)$-factor approximation algorithms where Q is a circle. As observed in [1], algorithms for Q being a convex polygon are not easily transfered for Q being a circle, as the running time of the formers depend upon the number of edges of Q.

For ray cuts, Demaine, Demaine and Kaplan [6] gave a linear time algorithm to decide whether a given polygon P is *ray-cuttable* or not. For optimally cutting P out of Q by ray cuts, if Q is convex, and P is non-convex but ray-cuttable, Daescu and Luo [5] gave an almost linear time $O(\log^2 n)$-approximation algorithm. If P is convex, then they gave a linear time 18-approximation algorithm. Tan [11] improved the approximation ratio for both cases to $O(\log n)$ and 6, respectively, but with much higher running time of $O(n^3+m)$.

There also exist a few results on generalization of this problem to 3D. Jaromczyk and Kowaluk [9] have studied the problem of deciding whether a polyhedral object can be cut out from a larger block using continuous hot wire cuts and they have given an $O(n^5)$-time algorithm that can decide the existence of such a cutting sequence. Ahmed et. al. [2] have studied a more relevant problem in 3D. Given a convex polyhedron P fixed inside a sphere Q, they have given an $O(\log^2 n)$-approximation algorithm for cutting P out of Q by using guillotine cuts. Their algorithm runs in $O(n^3)$ time. See Table 1 for a summary of these results.

Our Results. In this paper, we allow moving P arbitrarily inside Q. Let o be the center of Q. We call P to be *cornerable* if it is possible to place P inside Q such that P does not contain o. If no such placement of P is possible, then P is called *non-cornerable*. A *cornered position* of P inside Q is such that at least one vertex of P is on the boundary of Q. Note that a *cornered* position is applied for both cornerable and non-cornerable P. When P is cornerable, a *most cornered position* of P inside Q is such that P does not contain o and the distance of o from the closest point of P is maximum.

Table 1. Comparison of the results

Dim., movement	Cut Type	Q	P	Approx. Ratio	Running Time	Reference
2D, P fixed	Ray	-	Non-convex	Ray-cuttable?	$O(n)$	[6]
		Convex	Convex	18	$O(n)$	[5]
		Convex	Non-convex	$O(\log^2 n)$	$O(n)$	[5]
		Convex	Convex	6	$O(n^3+m)$	[11]
		Convex	Non-convex	$O(\log n)$	$O(n^3+m)$	[11]
	Line	Convex	Convex	$O(\log n)$	$O(mn+n\log n)$	[7,8]
		Convex	Convex	$2.5+\|\|Q\|\|/\|\|P\|\|$	$O(n^3+(n+m)\log(n+m))$	[5]
		Convex	Convex	7.9	$O(n^3+m)$	[11]
		Convex	Convex	$(1+\epsilon)$	$O(m+\frac{n^6}{\epsilon^{12}})$	[3]
		Circle	Cornered convex	$O(\log n)$	$O(n)$	[1]
		Circle	Cornered convex	6.48	$O(n^3)$	[1]
3D, P fixed	Hot-wire	-	Non-convex	Cuttable?	$O(n^5)$	[9]
	Guillotine	Sphere	Convex	$O(\log^2 n)$	$O(n^3)$	[2]
2D, P moves	**Line**	**Circle**	**Convex**	**Most cornered position**	$\boldsymbol{O(n^2)}$	**This paper**
	line	**Circle**	**Convex cornerable**	**6.48**	$\boldsymbol{O(n^3)}$	**This paper**
	line	**Circle**	**Convex non-cornerable**	**6.054**	$\boldsymbol{O(n^3)}$	**This paper**

Let C^* be the optimal cost of cutting P out of Q over all position of P. We show that when P is cornerable and is placed in a most cornered position, P can be cut out of Q in maximum cutting cost of $6.48C^*$. We also give an $O(n^2)$-time algorithm for finding a most cornered position of P. For non-cornerable P, we find a cornered position of P from where it can be cut out in maximum cutting cost of $6.054C^*$. (See Table 1).

Outline. First we will give for cornerable and non-cornerable P lower bounds on optimal cutting cost over all possible placements of P. Then for a cornerable P we will first find a most cornered position of P and will separate the minimum circular segment that contains P. Then we will cut a minimum bounding rectangle R_b of P from that circular segment. Finally, we will use known constant factor approximation algorithm for cutting P out of R_b. For a non-cornerable P we will find the minimum bounding rectangle R_b of P and will cut it from Q. Then the remaining procedure is same as that for a cornerable P.

We organize the rest of the paper as follows. Section 2 gives some preliminaries and lower bounds on C^*. Section 3 deals with cornerable P, where we give algorithms for finding the most cornered position of P and cutting P out of Q from that position and derive the approximation ratio. Section 4 deals with non-cornerable P. Finally, Section 5 concludes with directions to future works.

2 Lower Bounds

We start with some preliminaries. Consider a fixed position of P inside Q. Let x be the closest point on the boundary of P from o. x is called the *critical* point of P. If x is a vertex v of P, then v is called a *critical vertex* and if x is a point of an edge e, then e is called a *critical edge*. Observe that for cornerable P, since P is convex, P has either only one critical vertex or only one critical edge. We define the *D-separation* of P (w.r.t its fixed position) to be a single cut that creates the smallest possible circular segment of Q containing P. A crucial observation is that the D-separation is a tangent to P that passes through the critical vertex or the critical edge and is perpendicular to the line segment ox at x, since otherwise x would not be closest to o.

Let C^* be the cost of an optimal cutting sequence over all positions of P. Let $|P|$ be the perimeter of P. An easy lower bound on C^* for both cornerable and non-cornerable P is that $C^* \geq |P|$. This lower bound can be improved for both cornerable and non-cornerable P, which we do in the next two lemmas. The first lemma gives a lower bound that works for a non-cornerable P as well as a cornerable P when it contains o inside it. Let R be the radius of Q.

Lemma 1. *If P contains o, then $C^* \geq 2R$.*

Proof. An optimal cutting sequence must cut along the boundary of P. Moreover, if P contains o, then it must have two disjoint subsequences, each starting from a point of Q and ending at a point of P where it first touches P (possibly one subsequence having its second end point in the other subsequence). Let a

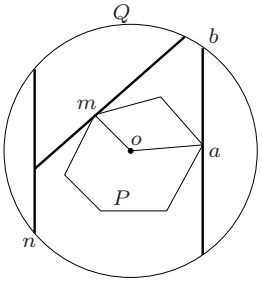


Fig. 1. Lower bound on C^* for non-cornerable P

and m be two such points of P and let b and n be the two corresponding points on Q. (See Fig. 1, where the two subsequences are shown in bold.) Now for one subsequence, let l_1 be the length of the shortest polygonal chain from a to b that is created by the cuts in the subsequence. Then clearly, $|ao| + l_1 \geq R$. Similarly, for the other subsequence let l_2 be the length of the shortest polygonal chain from m to n. Then $|mo| + l_2 \geq R$. Furthermore, $|ao| + |om| \leq |P|$. So, $C^* \geq |P| + l_1 + l_2 \geq |ao| + |om| + l_1 + l_2 \geq 2R$. □

Imagine a most cornered position of a cornerable P inside Q. Let D be the corresponding circular segment and D^* be the cost of the corresponding single cut that creates D. D^* is the minimum cost of a D-separation over all possible positions of P. Now we give a lower bound for cornerable P when it does not contain o.

Lemma 2. *When a cornerable P does not contain o, $C^* \geq D^*$.*

Proof. The authors in [1] proved that when P is placed such that it does not contain o, the optimal cutting cost for that position is at least the D-separation of P from o. Since D^* is the minimum of all possible D-separations of P, the lemma follows. □

3 Cornerable *P*

3.1 Most Cornered Position of *P*

Assume that P is cornerable. Our idea of finding the most cornered position of P is as follows. Remember that in a most cornered position, P must have at least one vertex incident on the boundary of Q. For each vertex v of P we will first position P, if possible, such that v is incident on the boundary of Q. We call this position of P *a base position* w.r.t v and we call v the *anchor* of this base position. From a particular base position of P with anchor v, we rotate P around v clockwise until another vertex hits the boundary of Q. We call this position the *right most base position* of P w.r.t v. Similarly, if we rotate v counter

clockwise until another vertex hits the boundary of Q, then we get the *left most base position* of P w.r.t v. To find the most cornered position of P w.r.t v, we need to rotate P around v from one of its extreme base positions, say the right one, in counter clockwise to its left most base position. During this rotation, we keep track of when a vertex or an edge becomes critical. A vertex or an edge can be critical for infinitely many positions. However, observe that all of them are consecutive in the rotation, because once a vertex or an edge has lost its critical property, it never becomes critical again. Moreover, remember that at any time either only one vertex or only one edge can be critical. Over the period when a vertex or an edge remains critical, we will find its furthest position from o. Maximum among all such position will give the most cornered position of P w.r.t that critical vertex or critical edge and w.r.t v. Considering all vertices (other than v) and edges one by one w.r.t v, we can find the most cornered position of P w.r.t v. Then we repeat the same procedure with other vertices as an anchor and take the most cornered position of P among them.

Now we come to exact detail. We give some more preliminaries. Let $u \neq v$ be a vertex with two supporting lines l_1 and l_2 passing through its two adjacent edges. Consider the pair of two opposite cones that are created by l_1 and l_2 and neither of which contains P. Let that pair of cone be C_u. Rotate this pair by $\pi/2$, i.e., rotate each of l_1 and l_2 around u. By this rotation, let l_1, l_2 and C_u become $l_1{}'$, $l_2{}'$ and $C_u{}'$ respectively. See Fig. 2(a).

First we determine exactly when a vertex or an edge can become critical.

Lemma 3. *For a particular position of P during its rotation around v, the vertex u of P will be critical iff $C_u{}'$ contains o.*

Proof. If $C_u{}'$ contains o, i.e., contains the line segment ou, then the line perpendicular to ou is contained within C_u and is also a tangent to P at u. On the other hand, for a particular position of P, if u is critical then the tangent of P at u corresponding to the D-separation must be within C_u. Therefore, its perpendicular line segment ou is in $C_u{}'$. □

Let us denote the range of rotation in which u remains critical by $\mathcal{R}_u$. Observe that $\mathcal{R}_u$ may not exist for some u.

Corollary 1. *$\mathcal{R}_u$ can be found in $O(1)$ time.*

Proof. All we need to do is to find the two positions of P (which may not exist) when $l_1{}'$ and $l_2{}'$ intersect o. This can be easily done by computing and then comparing the angles of o with $l_1{}'$ and $l_2{}'$ at u. □

Lemma 4. *Within $\mathcal{R}_u$, the minimum D-separation passing through u can be found in $O(1)$ time.*

Proof. Consider the circle c_u with center v and passing through u. We need the position of u where the length of ou is the maximum. Since u moves along a circle, clearly that position of u can be found in constant time from c_u, $\mathcal{R}_u$ and o. □

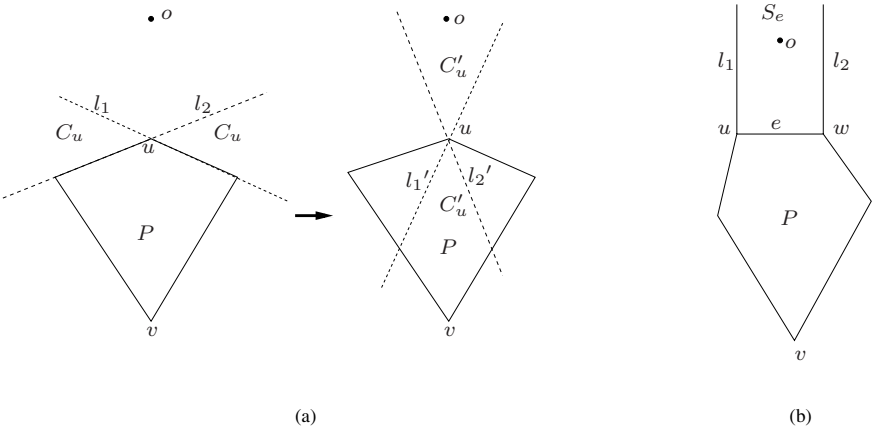


Fig. 2. Critical positions for a vertex and an edge

Now we find similar results for a critical edge $e = uw$. Let l_1 and l_2 be the two lines that are perpendicular to e at u and w respectively. Let S_e be the infinite half strip which is bounded by e, l_1, and l_2 and contains the outward normal of e. See Fig. 2(b).

Lemma 5. *For a particular position of P during its rotation around v, an edge e is critical iff S_e contains o.*

Proof. Assume that S_e contains o. If op is the line segment that is perpendicular to the supporting line of e at p, then p is a point of e. So, e is critical. On the other hand, if e is critical, then let x be the point of e that is closest from o. The line segment ox is perpendicular to e at x. So, ox is in S_e. □

Let us denote the range of rotation in which e remains critical by $\mathcal{R}_e$. Again, $\mathcal{R}_e$ may not exist for some e.

Corollary 2. *$\mathcal{R}_e$ can be found in $O(1)$ time.*

Lemma 6. *Within $\mathcal{R}_e$, the minimum D-separation passing through e can be found in $O(1)$ time.*

Proof. Let t be the line passing through e and let p be the point on t such that vp is perpendicular to t. p may or may not be a point on e. Let c_p be the circle with center v and passing through p. Then t is the tangent of c_p at p and let h be the perpendicular distance of o from t. While rotating P within $\mathcal{R}_e$, since p moves along c_p, the function of h is unimodal in the rotation angle. More precisely, from the boundary position $\mathcal{R}_e$, where o is on l_1, if we rotate P counter clockwise, then h gradually increases to the maximum and then gradually decreaes until it hits l_2. Moreover, some elementary geometry proves that the function of h is elliptical (see Appendix for a complete proof.) So, within $\mathcal{R}_e$, the position of e at which h is maximum can be found in constant time. □

Theorem 1. *A most cornered position of P inside Q can be found in $O(n^2)$ time.*

Proof. By the above lemmas and corollaries, for a fixed anchor v, as long as a vertex u or an edge e remains critical the corresponding minimum D-separation passing through u or e can be found in constant time, whose maximum over all $O(n)$ vertices and edges can be found in $O(n)$ time. Since any vertex can be an anchor, total time for computing D^* is $O(n^2)$. □

Corollary 3. *Deciding whether P is cornerable or not can be done in $O(n^2)$ time.*

Proof. P is cornerable iff a most cornered position of P does not contain o inside it. By Theorem 1, a most cornered position of P can be found in $O(n^2)$ time. Then, in that position of P, finding the containment of o by P can be done in $O(n)$ time. □

3.2 Cutting Cornerable P

After we have placed P in a most-cornered position, we execute the first cut along D^* to find the minimum circular segment D containing P. We use the rotating calipers technique [12] to determine the minimum area bounding rectangle R_b of P. We introduce four more cuts along the edges of R_b (in any sequence) to cut R_b out of D. Next we apply the dynamic programming technique of Tan [11] for cutting P out of R_b.

Theorem 2. *A cornerable convex polygon P can be cut out of a circle Q in $O(n^3)$ time with cutting cost $6.48C^*$.*

Proof. After getting the smallest D from the most cornered position of P, getting R_b by using rotating calipers takes $O(n)$ time [12]. The dynamic programming technique of [11] takes $O(n^3)$ time. Hence the overall running time is $O(n^3)$.

After placing P at a most cornered position, our first cut is along D^*, which is smaller than C^* by Lemma 2. At this moment the perimeter of D is at most $\frac{\pi}{2}D^* + D^* = 2.57D^*$. Now, one crucial observation is that while cutting R_b from D, each of the four cuts throws away some portion of the perimeter of D, whose length is larger than the corresponding cut. So, total cost of those four cuts is smaller than the perimeter of D, which is at most $2.57C^*$. Finally, Tan's algorithm [11] has a cost of $(1.5 + \sqrt{2})C^*$. Hence the overall cutting cost of our algorithm is bounded by $6.48C^*$. □

Observe that some portion of R_b may remain outside of D. But in that case, the cost of cutting R_b is still bounded by $2.57C^*$. Moreover, in that case the resulting shape, which may not be a rectangle, will be smaller then R_b. Since Tan's algorithm [11] cuts P out of a rectangle, applying this algorithm on this smaller shape will give a cutting cost even smaller than $(1.5 + \sqrt{2})C^*$.

Remember from Lemma 1 that if a cornerable P is placed with the center o of Q inside of it, then any cutting sequence must have cost at least $2R$. It suggests

that the most cornered position is, possibly, the best position for a cornerable P, since moving P anywhere else brings it closer to o. Moreover, for such a placement of P, an algorithm similar to that of our algorithm given above would give an inferior approximation ratio. Based on these observations we have the following conjecture.

Conjecture 1. Any algorithm for cutting a cornerable P with better approximation ratio must start with a most cornered position of P.

4 Non-cornerable P

Assume that P is non-cornerable. First we compute R_b of P by rotating calipers [12]. We place R_b along with P arbitrarily inside Q. We apply four cuts along the edges of R_b (in any sequence). Then we apply Tan's algorithm [11] to cut P out of R_b.

Theorem 3. *A non-cornerable convex polygon P can be cut out of a circle Q in $O(n^3)$ time with a cutting cost of $6.054C^*$.*

Proof. Computing R_b takes $O(n)$ time [12]. Then Tan's algorithm takes $O(n^3)$ time.

We now derive the ratio. Since P must contain the center o, by Lemma 1 $C^* \geq 2R$. First we see the cost of cutting R_b. Let the four cuts applied to cut R_b be $c_1, \ldots, c_4$. Now remember that each of c_1, c_2, c_3 and c_4 throws a portion of the boundary of Q whose length is no smaller than the corresponding cut. So, total cost of c_1 c_2, c_3 and c_4 is no more than the perimeter of Q, i.e, $\leq 2\pi R$. (This argument is similar to that in the proof of Theorem 2.)

Thus the total cost of cutting R_b, which we denote by C_{R_b}, is

$$\begin{aligned} C_{R_b} &= |c_1| + |c_2| + |c_3| + |c_4| \\ &\leq 2\pi R \\ &\leq \pi C^*, \text{by Lemma 1} \end{aligned}$$

With Tan's cutting cost of $(1.5 + \sqrt{2})C^*$ for cutting P out of R_b, total cutting cost becomes,

$$\begin{aligned} & C_{R_b} + (1.5 + \sqrt{2})C^* \\ \leq\ & (3.14 + (1.5 + \sqrt{2}))C^* \\ =\ & 6.054C^* \end{aligned}$$

□

5 Conclusion

In this paper we address the problem of cutting a convex polygon P from a circle Q, where P can move freely inside of P, by using line cuts with minimum total cutting cost. We first find the most cornered position of P towards the boundary

of Q in $O(n)$ time. Then we give a constant factor approximation algorithm that cuts P out of Q. If it is possible to position P without having the center of Q inside of P, then our algorithm has approximation ratio 6.48; otherwise it has the ratio of 5.65. In both cases our algorithm runs in $O(n^3)$ time.

There remains several directions for future research.

1. An immediate direction for research is to work with Q as a convex polygon, which we think would be more difficult due to non-regularity of the boundary of Q.
2. Another immediate direction is to generalize the problem in 3D, where P would be a convex polyhedron and Q would be a sphere.
3. It might be possible to improve the running times of our algorithm. In particular, improving $O(n^2)$ for finding a most cornered position of P would be interesting.

References

1. Ahmed, S.I., Hasan, M., Islam, M.A.: Cutting a cornered convex polygon out of a circle. Journal of Computers (to appear), http://203.208.166.84/masudhasan/cut.pdf
2. Ahmed, S.I., Hasan, M., Islam, M.A.: Cutting a convex polyhedron out of a sphere. In: 7th Japan Conference on Computational Geometry and Graphs, JCCGG 2009 (2009); Also in 4th Annual Workshop on Algorithms and Computation (WALCOM 2010), Dhaka, Bangladesh, February 10-12, 2010, http://arxiv.org/abs/0907.4068
3. Bereg, S., Daescu, O., Jiang, M.: A PTAS for cutting out polygons with lines. In: Chen, D.Z., Lee, D.T. (eds.) COCOON 2006. LNCS, vol. 4112, pp. 176–185. Springer, Heidelberg (2006)
4. Bhadury, J., Chandrasekaran, R.: Stock cutting to minimize cutting length. Euro. J. Oper. Res. 88, 69–87 (1996)
5. Daescu, O., Luo, J.: Cutting out polygons with lines and rays. International Journal of Computational Geometry and Applications 16, 227–248 (2006)
6. Demaine, E.D., Demaine, M.L., Kaplan, C.S.: Polygons cuttable by a circular saw. Computational Geometry: Theory and Algorithms 20, 69–84 (2001)
7. Dumitrescu, A.: An approximation algorithm for cutting out convex polygons. Computational Geometry: Theory and Algorithms 29, 223–231 (2004)
8. Dumitrescu, A.: The cost of cutting out convex n-gons. Discrete Applied Mathematics 143, 353–358 (2004)
9. Jaromczyk, J.W., Kowaluk, M.: Sets of lines and cutting out polyhedral objects. Computational Geometry: Theory and Algorithms 25, 67–95 (2003)
10. Overmars, M.H., Welzl, E.: The complexity of cutting paper. In: SoCG 1985, pp. 316–321 (1985)
11. Tan, X.: Approximation algorithms for cutting out polygons with lines and rays. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 534–543. Springer, Heidelberg (2005)
12. Toussaint, G.: Solving geometric problems with the rotating calipers. In: MELECON 1983, Athens, Greece (1983)

Appendix

Claim. Within $\mathcal{R}_u$, h is elliptical.

Proof. W.l.o.g after necessary translation assume that the equation of c_p is

$$c_p : x^2 + y^2 = r^2,$$

where $r = |vp|$ is the radius of the c_p. Let, (x_1, y_1) and (x_2, y_2) be the coordinates of p and o, respectively. Then the equation of t is

$$t : m_1 x_1 + y_1 - c_1 = 0,$$

where $m_1 = x_1/y_1$ and $c_1 = r^2/y_1$. The distance function h from (x_2, y_2) to the tangent t passing through x_1, y_1 is given by

$$h = \|m_1 x_2 + y_2 - c_1\|/\sqrt{((m_1)^2 + 1)}$$

Now, $\sqrt{((m_1)^2 + 1)} = r/y_1$ (as (x_1, y_1) is a point on C). Hence we get

$$h = \|x_1 x_2 + y_1 y_2 - r^2\|/r$$

After replacing y_1 by $\sqrt{(r^2 - (x_1)^2)}$ we get,

$$k_1 h^2 + k_2 (x_1)^2 + k_3 h x_1 + k_4 x_1 + k_5 h + k_6 = 0,$$

where $k_1, k_2, k_3, k_4, k_5, k_6$ are constants in terms of x_2, y_2 and r, and this is an elliptical function. □

Real Root Isolation of Multi-Exponential Polynomials with Application*

Ming Xu[1,2], Liangyu Chen[1], Zhenbing Zeng[1], and Zhi-bin Li[1,2]

[1] Shanghai Key Laboratory of Trustworthy Computing,
[2] Computer Science and Technology Department,
East China Normal University, Shanghai 200062, China
lizb@cs.ecnu.edu.cn

Abstract. Real root isolation problem is to compute a list of disjoint intervals, each containing a distinct real root and together containing all. Traditional methods and tools often attack the root isolation for ordinary polynomials. However many other complex systems in engineering are modeling with non-ordinary polynomials. In this paper, we extend the pseudo-derivative sequences and Budan–Fourier theorem for multi-exponential polynomials to estimate the bounds and counts of all real roots. Furthermore we present an efficient algorithm for isolating all real roots under given minimum root separation. As a proof of serviceability, the reachability of linear systems with real eigenvalues only is approximately computable by this algorithm.

1 Introduction

Real root isolation is fundamental and critical to modern algorithms. Specially the real root isolation for polynomials has been implemented and integrated into many existing tools [1–5]. However there still exist many systems in engineering modeled with non-ordinary polynomials, so that those traditional methods for polynomials are hardly applied to these complex systems. A multi-exponential polynomial, i.e. $p^*(x) = \sum_{i=0}^{n} p_i(x)\lambda_i^x$ where $p_i(x)$s are polynomials and λ_is are positive numbers, is a typical one. Applications of real root isolation for multi-exponential polynomials are numerous. For instance, computing the reachable region of linear systems $\xi' = \mathbf{A}\xi + \mathbf{u}$ is an essential key to safety analysis in control theory. G. Lafferriere presented the reachable region is computable if $\mathbf{A}$ is a diagonalizable matrix with rational eigenvalues only and the input $\mathbf{u}$ is proper in [6]. For a larger class of linear systems with real eigenvalues only, the reachability computation can be immediately reduced to the real root isolation problem of multi-exponential polynomials.

In this paper, we develop the pseudo-derivative sequences for multi-exponential polynomials inspired by M. Achatz's work [7]. Based on them, we generalize Budan–Fourier theorem, which is no longer suitable for polynomials only [8], to

* Supported by NSFC (No. 90718041), Shanghai Leading Academic Discipline Project (No. B412) and PhD Program Scholarship Fund of ECNU 2009 (No. 2009056).

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 263–268, 2010.

estimate the bounds and counts of real roots for given multi-exponential polynomials. Besides a factorization for multi-exponential polynomials is introduced by $\mathbb{Q}$-linear independence to reduce the complexity of isolation. Finally we present an algorithm for isolating all real roots of multi-exponential polynomial under given root separation. Our algorithm mixed of symbolic and numeric computation can efficiently tackle with vast problems from engineering practice.

The rest of this paper is organized as follows. Section 2 defines the multi-exponential polynomials and their factorization. Section 3 describes the real root isolation algorithm for them. Section 4 applies this algorithm to the reachability problem of linear systems with real eigenvalues only. Section 5 is the conclusion.

2 Multi-Exponential Polynomials

Definition 1. *Let* $p^*(x) = \sum_{i=0}^{n} p_i(x)\lambda_i^x$ $(0 < \lambda_0 < \lambda_1 < \cdots < \lambda_n \wedge p_i(x) \in \mathbb{R}[x] \setminus \{0\})$ *be a multi-exponential polynomial (MEP). Then the degree of* $p^*(x)$ *is* $n + \sum_{i=0}^{n} \deg(p_i)$ *and the tail base is* λ_0*, denoted by* $\deg(p^*)$ *and* $\mathrm{tbase}(p^*)$.

A pseudo-derivative sequence of p^* can be constructed recursively as follows:

$$\begin{cases} F_0 = \dfrac{p^*}{(\mathrm{tbase}(p^*))^x} \\ F_{i+1} = \dfrac{F_i'}{(\mathrm{tbase}(F_i'))^x} \end{cases} \tag{1}$$

until $\deg(F_{i+1}) = 0$. Let $\mathrm{PDS}(p^*) = [F_0, F_1, \cdots, F_{\deg(p^*)}]$ denote the whole pseudo-derivative sequence since $\deg(F_{i+1}) - \deg(F_i) = 1$ and $\mathrm{PDS}_M(p^*) = [F_0, F_1, \cdots, F_M]$ $(M \leq \deg(p^*))$ denote the partial pseudo-derivative sequence.

Theorem 1. *If* α *is an* M*-multiple root of* F_i*, then* $F_{i+j}(x)$ *and* $F_i^{(j)}(x)$ *share the same sign in an* ϵ*-neighborhood of* α *for each* j $(0 \leq j \leq M)$.

Proof. On one hand, we can construct a lower triangular transition matrix $(t_{i;j,k}(x))_{M\times M}$ on induction such that

$$\begin{pmatrix} F_{i+1}(x) \\ F_{i+2}(x) \\ \vdots \\ F_{i+M-1}(x) \\ F_{i+M}(x) \end{pmatrix} = (t_{i;j,k}(x))_{M\times M} \begin{pmatrix} F_i'(x) \\ F_i^{(2)}(x) \\ \vdots \\ F_i^{(M-1)}(x) \\ F_i^{(M)}(x) \end{pmatrix}, \tag{2}$$

where

$$\begin{cases} t_{i;1,1}(x) = \dfrac{1}{(\mathrm{tbase}(F_i'))^x} \\ t_{i;j+1,1}(x) = \dfrac{t'_{i;j,1}(x)}{(\mathrm{tbase}(F'_{i+j}))^x} \\ t_{i;j+1,k}(x) = \dfrac{t'_{i;j,k}(x) + t_{i;j,k-1}(x)}{(\mathrm{tbase}(F'_{i+j}))^x} \quad \text{for } 1 < k < j+1 \\ t_{i;j+1,j+1}(x) = \dfrac{t_{i;j,j}(x)}{(\mathrm{tbase}(F'_{i+j}))^x} \end{cases}.$$

Since $(t_{i;j,k})_{M\times M}$ is nonsingular, $F_i(\alpha) = F_i'(\alpha) = \cdots = F_i^{(M-1)}(\alpha) = 0 \neq F_i^{(M)}(\alpha)$ if and only if $F_i(\alpha) = F_{i+1}(\alpha) = \cdots = F_{i+M-1}(\alpha) = 0 \neq F_{i+M}(\alpha)$.

On the other hand, we will prove inductively on all $0 \leq j \leq M$ that $F_i^{(j)}(x) > 0$ if and only if $F_{i+j}(x) > 0$ for arbitrary x in $(\alpha-\epsilon, \alpha) \cup (\alpha, \alpha+\epsilon)$. If $j = 0$, it holds trivially; otherwise the following statements are mutually equivalent:

1. $F_i^{(j+1)}(x) > 0$.
2. If $x < \alpha$, then $F_j^{(i)}(x) < 0$; else $F_j^{(i)}(x) > 0$ by $F_i^{(j)}(\alpha) = 0$.
3. If $x < \alpha$, then $F_{j+i}(x) < 0$; else $F_{j+i}(x) > 0$ by inductive assumption.
4. $F_{i+j}'(x) > 0$ by $F_{i+j}(\alpha) = 0$.
5. $F_{i+j+1}(x) > 0$ by the definition of pseudo-derivative sequence.

Therefore $\operatorname{sign}(F_{i+j}(x)) = \operatorname{sign}(F_i^{(j)}(x))$ in $(\alpha-\epsilon, \alpha+\epsilon)$. □

Definition 2. *A set of numbers $\{\lambda_0, \lambda_1, \cdots, \lambda_n\}$ is $\mathbb{Q}$-linearly independent if no linear relation $a_0\lambda_0 + a_1\lambda_1 + \cdots + a_n\lambda_n = 0$ with rational coefficients a_is, not all zero, holds between them.*

Given a MEP $p^*(x) = \sum_{i=0}^{n} p_i(x)\lambda_i^x$, it can be factorized as follows:

1. Choose a $\mathbb{Q}$-linearly independent subset $\{\lambda_0, \mu_1, \cdots, \mu_k\}$ of $\{\lambda_0, \lambda_1, \cdots, \lambda_n\}$.
2. Define the reverse mapping $p^*(x) \to p^{**}(x, z_0, z_1, \cdots, z_k)$, i.e. the back-substitution of each λ_i^x with $\prod_{j=0}^{k} z_j^{a_{ij}}$ as $\lambda_i = a_{i0}\lambda_0 + \sum_{j=1}^{k} a_{ij}\mu_j$.
3. The factorization of $p^*(x)$ corresponds to that of $p^{**}(x, z_0, z_1, \cdots, z_k)$, a multivariate "polynomial" with rational coefficients and exponentials.

3 Real Root Isolation Algorithm

Definition 3. *Given a finite sequence $S = [s_0, s_1, \cdots, s_m]$, the number of the sign variations $\mathrm{V}(S)$ is the number of pairs (i, j) with $(0 \leq i < j \leq m)$ satisfying:*

$$(s_i s_j < 0) \wedge (\bigwedge_{i<j'<j} s_{j'} = 0).$$

Theorem 2 (Budan–Fourier theorem for MEPs). *The number of MEP p^*'s real roots (multiplicities counted) during an interval (a, b) is less a nonnegative even number than $\mathrm{V}([\mathrm{PDS}(p^*)]_b^a) = \mathrm{V}([\mathrm{PDS}(p^*)]_{x=a}) - \mathrm{V}([\mathrm{PDS}(p^*)]_{x=b})$.*

Proof. Assume $\alpha \in (a, b)$ is an M-multiple root of $F_i(x)$, by Taylor's theorem, $F_i(x) = \sum_{k=0}^{+\infty} \frac{(x-\alpha)^k}{k!} F_i^{(k)}(\alpha) = \sum_{k=M}^{+\infty} \frac{(x-\alpha)^k}{k!} F_i^{(k)}(\alpha) \approx \frac{(x-\alpha)^M}{M!} F_i^{(M)}(\alpha)$ for arbitrary $x \in (\alpha-\epsilon, \alpha) \cup (\alpha, \alpha+\epsilon)$. Then we will discuss two distinct cases.

1. One is $i = 0$. Then $\mathrm{V}([\mathrm{PDS}_M(F_0)]_{\alpha+\epsilon}^{\alpha-\epsilon}) = M$ since $\operatorname{sign}(F_i(x)) = \operatorname{sign}(F_0^{(i)}(x))$ by Theorem 1.
2. The other is $i > 0$ and $F_{i-1}(\alpha) \neq 0$, we have $\mathrm{V}([\mathrm{PDS}_M(F_i)]_{\alpha+\epsilon}^{\alpha-\epsilon}) = M$ similarly. If M is even, $\mathrm{V}([\mathrm{PDS}_1(F_{i-1})]_{\alpha+\epsilon}^{\alpha-\epsilon}) = 0$; otherwise $\mathrm{V}([\mathrm{PDS}_1(F_{i-1})]_{\alpha+\epsilon}^{\alpha-\epsilon}) = \pm 1$. Hence $\mathrm{V}([\mathrm{PDS}_{M+1}(F_{i-1})]_{\alpha+\epsilon}^{\alpha-\epsilon})$ is a nonnegative even number $2N_k$.

Therefore $\mathrm{V}([\mathrm{PDS}(p^*)]_b^a)$ is more a nonnegative even number $\sum_k 2N_k$ than the number of real roots. □

Corollary 1. *A pair of real numbers (l, u) is the real root (lower and upper) bounds of $p^*(x)$ if $\mathrm{V}([\mathrm{PDS}(p^*)]_l^{-\infty}) = 0 = \mathrm{V}([\mathrm{PDS}(p^*)]_{+\infty}^u)$.*

Definition 4. *Given a function f, let $\alpha_1, \alpha_2, \cdots, \alpha_n$ be all roots of f, the minimum root separation is*

$$\mathrm{sep}(f) = \min_{1 \le i < j \le n} |\alpha_i - \alpha_j|$$

with the convention that $\mathrm{sep}(f) = +\infty$ in case f has only one root.

Algorithm 1. Assuming $\mathrm{sep}(p^*) > \epsilon$,

$$L \leftarrow \mathrm{ISOL}(p^*, \epsilon).$$

Input: $p^*(x)$ is a MEP and $\epsilon \in \mathbb{Q}^+$.
Output: $L = \{(a_1, b_1), \cdots, (a_k, b_k)\}$ is a list of disjoint open intervals with rational endpoints, satisfying:
(a) k is the number of distinct real roots of p^*;
(b) each (a_i, b_i) contains exact one distinct real root of p^*.

S1 (Initialization) Compute $\mathrm{PDS}(p^*) := [F_0, F_1, \cdots, F_{\deg(p^*)}]$.
S2 (Bound) Compute the upper and lower bounds $u, l \in \mathbb{Q}^+$, containing all real roots of p^*. Let $L' := \{(l, u)\}$ and $L, L'' := \emptyset$.
S3 (Refinement) For each $I = (a, b) \in L'$:
(a) If $\mathrm{V}([\mathrm{PDS}(p^*)]_b^a) = 1$, then set $L := L \cup \{I\}$.
(b) If $\mathrm{V}([\mathrm{PDS}(p^*)]_b^a) > 1$, then set the average $c := \frac{a+b}{2}$.
  i. If $p^*(c) = 0$, then $L := L \cup \{(\max\{a, c - \epsilon\}, \min\{c + \epsilon, b\})\}$ and $L'' := L'' \cup \{(a, \max\{a, c - \epsilon\}), (\min\{c + \epsilon, b\}, b)\}$.
  ii. Otherwise $L'' := L'' \cup \{(a, c), (c, b)\}$.
Finally set $L' := L' \setminus \{I\}$.
S4 (Reduction) For each $I = (a, b) \in L''$:
(a) If $\| I \| \le \epsilon$ and $p^*(a)p^*(b) < 0$, then $L := L \cup \{I\}$.
(b) If $\| I \| > \epsilon$, then $L' := L' \cup \{I\}$.
Finally set $L'' := L'' \setminus \{I\}$.
S5 (Recursion) If $L' = \emptyset$, rearrange L and RETURN it; else GOTO S3. □

This algorithm has the complexity of $\frac{\mathrm{V}([\mathrm{PDS}(p^*)]_u^l)}{2} \lg_2(\frac{u-l}{\epsilon})$ in the worst case.

4 Application of Approximate Reachability Analysis

The linear system we concern is of the form

$$\xi'(t) = \mathbf{A}\xi(t) + \mathbf{u}(t) \tag{3}$$

where

- $\xi(t) \in \mathbb{R}^n$ is the system state at time t;
- $\mathbf{A} \in \mathbb{R}^{n\times n}$ is a matrix with real eigenvalues only;
- $\mathbf{u}(t) : \mathbb{R} \to \mathbb{R}^n$ is a vector of multi-exponential polynomials.

Assuming $\xi(0)$ is the initial state, the whole solution of (3) is

$$\xi(t) = \mathrm{e}^{\mathbf{A}t}\xi(0) + \int_0^t \mathrm{e}^{\mathbf{A}(\tau-t)}\mathbf{u}(\tau)d\tau \ . \tag{4}$$

In particular, given a matrix $\mathbf{A}$ with real eigenvalues only, with the Jordan form $\mathbf{J} = \mathbf{T}^{-1}\mathbf{A}\mathbf{T}$, all entries in $\mathrm{e}^{\mathbf{A}t} = \mathbf{T}\mathrm{e}^{\mathbf{J}t}\mathbf{T}^{-1}$ are MEPs in t. Hence all ξ_is are MEPs in t too since MEPs are closed under addition, multiplication, differential and integral calculus. For (3), a cell Ω is reachable from $\xi(0)$ if there exists a time t_0 such that $\xi(t_0) \in \Omega$.

Algorithm 2. Assuming the tolerance is δ,

$$S \leftarrow \mathrm{REACH}(\xi, \Omega, \delta).$$

Input: $\xi(t)$ is a vector of MEPs; $\Omega \subseteq \mathbb{R}^n$ is a target cell characterized by polynomial equations and inequalities; and $\delta \in \mathbb{Q}^+$.
Output: $S = [I_1, I_2, \cdots, I_s]$, a list of sample intervals, satisfies that each I_j contains at least one point in Ω that is reachable from $\xi(0)$.

Without loss of generality, it is enough to decide whether $\Omega \equiv \wedge_i(f_i(\xi) = 0) \bigwedge \wedge_j(g_j(\xi) > 0)$ is reachable. $f_i(t)$s and $g_j(t)$s are MEPs after substituting $\xi(t)$ into f_is and g_js. Set $S, S' := \emptyset$. We will discuss two distinct cases.

C1 One is that $\Omega \equiv \wedge_{i=1}^{l}(f_i = 0) \bigwedge \wedge_j(g_j > 0)$ $(l > 0)$.
   (a) Assuming $\mathrm{sep}(f_i) > \delta$, isolate each f_i. in its isolation list $L_i = [I_{i,1}, \cdots, I_{i,s_i}]$ by calling Algorithm 1 and bisecting each I_{i,j_i} ensuring $\| I_{i,j_i} \| < \delta$.
   (b) For each $[I_{1,j_1}, I_{2,j_2}, \cdots, I_{l,j_l}] \in L_1 \times L_2 \times \cdots \times L_l$, if there exists an intersection $(a, b) = \cap_{1\le i\le l} I_{i,j_i} \neq \emptyset$ and $f_i(a)f_i(b) < 0$ for all i, then (a, b) contains a common root of all f_is and set $S' := S' \cup \{(a, b)\}$.
   (c) For each $(a, b) \in S'$, if $g_j(a) > 0$ and $g_j(b) > 0$ for all j, (a, b) is a sample interval satisfying all Ω's constraints under the tolerance δ, set $S := S \cup \{(a, b)\}$.

C2 Otherwise $\Omega \equiv \wedge_j(g_j > 0)$ depicted by inequalities only, then it is one n-dimension cell. The cell Ω is reachable if and only if its boundary $\partial\Omega$ is passable, i.e. $\partial\Omega$ is reachable in an open interval I with one endpoint in Ω. Let $\mathrm{PS}(G) := 2^G \setminus \{\emptyset\}$. Construct the boundary $\partial\Omega \equiv \bigcup_{G'\in \mathrm{PS}(G)} \Omega_{G'}$ where $\Omega_{G'} \equiv \wedge_{g_j\in G'}(g_j = 0) \bigwedge \wedge_{g_{j'}\notin G'}(g_{j'} > 0)$. Now it is necessary to just check the reachability of each $\Omega_{G'}$, which is similar to C1.

Let (a, b) be a reachable sample interval for $\Omega_{G'}$. If $g_j(a) > 0$ for all $g_j \in G'$ or $g_j(b) > 0$ for all $g_j \in G'$, then (a, b) would be a passable sample interval for $\Omega_{G'}$ and set $S := S \cup \{(a, b)\}$. □

Example 1. Consider a linear system with real eigenvalues only as follows:

$$\begin{cases} x' = 2x + y + \mathrm{e}^{\sqrt{2}t} \\ y' = -x + \mathrm{e}^{t} \ . \end{cases} \tag{5}$$

Let the initial state be the origin $(0,0)^{\mathrm{T}}$ and the target region be depicted by $f = x - y - 1 = 0$. Its unique solution is

$$\begin{cases} x = (-4 - 3\sqrt{2} - t - \sqrt{2}t + \frac{1}{2}t^2)\mathrm{e}^{t} + (4 + 3\sqrt{2})\mathrm{e}^{\sqrt{2}t} \\ y = (3 + 2\sqrt{2} + 2t + \sqrt{2}t - \frac{1}{2}t^2)\mathrm{e}^{t} + (-3 - 2\sqrt{2})\mathrm{e}^{\sqrt{2}t} \ . \end{cases} \tag{6}$$

Then, by substitution, $f = -1 + (-7 - 5\sqrt{2} - 3t - 2\sqrt{2}t + t^2)\mathrm{e}^{t} + (7 + 5\sqrt{2})\mathrm{e}^{\sqrt{2}t}$.

Since $\mathrm{V}([\mathrm{PDS}(f)]_{t=-\infty}) = \mathrm{V}([\mathrm{PDS}(f)]_{x=-3}) = 3$ and $\mathrm{V}([\mathrm{PDS}(f)]_{t=+\infty}) = \mathrm{V}([\mathrm{PDS}(f)]_{x=1}) = 0$, all real roots of f lie in $(-3, 1)$. Furthermore $\mathrm{V}([\mathrm{PDS}(f)]_{t=-1}) = 1$ and $\mathrm{V}([\mathrm{PDS}(f)]_{t=-2}) = 2$. Hence the system reaches the target region thrice at times $(-3,-2), (-2,-1), (-1,1)$. □

5 Conclusion

Complex systems modeled by non-ordinary polynomials are ubiquitous and widely-used but have few research on them. In this paper, based on pseudo-derivative sequences, we generalize Budan–Fourier theorem and present a practical algorithm for isolating all real roots of given multi-exponential polynomial. An application of linear system reachability shows the efficiency of our approach.

References

1. Uspensky, J.V.: Theory of Equations. McGraw-Hill, New York (1948)
2. Collins, G.E., Loos, R.: Real zeros of polynomials. In: Buchberger, B., Collins, G.E., Loos, R. (eds.) Computer Algebra: Symbolic and Algebraic Computation, pp. 83–94. Springer, Heidelberg (1983)
3. Collins, G.E., Hong, H.: Partial cylindrical algebraic decomposition for quantifier elimination. J. Symb. Comput. 12(3), 299–328 (1991)
4. Yang, L., Hou, X., Zeng, Z.: A complete discrimination system for polynomials. Science in China (Ser. E) 39, 628–646 (1996)
5. Tsigaridas, E.P., Emiris, I.Z.: Univariate polynomial real root isolation: Continued fractions revisited. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 817–828. Springer, Heidelberg (2006)
6. Lafferriere, G., Pappas, G.J., Yovine, S.: Symbolic reachability computaion for families of linear vector fields. J. Symb. Comput. 32(3), 231–253 (2001)
7. Achatz, M., McCallum, S., Weispfenning, V.: Deciding polynomial-exponential problems. In: ISSAC 2008, pp. 215–222. ACM Press, New York (2008)
8. Akritas, A.G.: Elements of Computer Algebra with Applications. Wiley, New York (1989)

FPT Algorithms for Connected Feedback Vertex Set

Neeldhara Misra, Geevarghese Philip, Venkatesh Raman,
Saket Saurabh*, and Somnath Sikdar

The Institute of Mathematical Sciences, India
{neeldhara,gphilip,vraman,saket,somnath}@imsc.res.in

Abstract. We study the recently introduced Connected Feedback Vertex Set (CFVS) problem from the view-point of parameterized algorithms. CFVS is the connected variant of the classical Feedback Vertex Set problem and is defined as follows: given a graph $G = (V, E)$ and an integer k, decide whether there exists $F \subseteq V$, $|F| \leq k$, such that $G[V \setminus F]$ is a forest and $G[F]$ is connected. We show that Connected Feedback Vertex Set can be solved in time $O(2^{O(k)}n^{O(1)})$ on general graphs and in time $O(2^{O(\sqrt{k}\log k)}n^{O(1)})$ on graphs excluding a fixed graph H as a minor. Our result on general undirected graphs uses, as a subroutine, a parameterized algorithm for Group Steiner Tree, a well studied variant of Steiner Tree. We find the algorithm for Group Steiner Tree of independent interest and believe that it could be useful for obtaining parameterized algorithms for other connectivity problems.

1 Introduction

Feedback Vertex Set (FVS) is a classical NP-complete problem and has been extensively studied in all subfields of algorithms and complexity. In this problem we are given an undirected graph $G = (V, E)$ and a positive integer k as input, and the goal is to check whether there exists a subset $F \subseteq V$ of size at most k such that $G[V \setminus F]$ is a forest. This problem originated in combinatorial circuit design and found its way into diverse applications such as deadlock prevention in operating systems, constraint satisfaction and Bayesian inference in artificial intelligence. We refer to the survey by Festa, Pardalos and Resende [12] for further details on the algorithmic study of feedback set problems in a variety of areas like approximation algorithms, linear programming and polyhedral combinatorics.

In this paper we focus on the recently introduced connected variant of Feedback Vertex Set, namely, Connected Feedback Vertex Set (CFVS). Here, given a graph $G = (V, E)$ and a positive integer k, the objective is to check whether there exists a vertex-subset F of size at most k such that $G[V \setminus F]$ is a forest and $G[F]$ is connected. Sitters and Grigoriev [21] recently introduced this problem and obtained a polynomial time approximation scheme (PTAS) for

* This work was done while the author was at the University of Bergen, Norway.

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 269–280, 2010.

CFVS on planar graphs. We find it a bit surprising that the connected version of FVS has not been studied in the literature until now. This is in complete contrast to the fact that the connected variants of other problems, like Vertex Cover—Connected Vertex Cover, and Dominating Set—Connected Dominating Set are extremely well-studied in the literature (See, e.g, [17], [14], respectively.). In this paper, we initiate the algorithmic study of CFVS from the view-point of parameterized algorithms.

Parameterized complexity is a two-dimensional generalization of "P vs. NP" where, in addition to the overall input size n, one studies how a secondary measurement (called the *parameter*), that captures additional relevant information, affects the computational complexity of the problem in question. Parameterized decision problems are defined by specifying the input, the parameter, and the question to be answered. The two-dimensional analogue of the class P is decidability within a time bound of $f(k)n^c$, where n is the total input size, k is the parameter, f is some computable function and c is a constant that does not depend on k or n. A parameterized problem that can be decided in such a time-bound is termed *fixed-parameter tractable* (FPT). For general background on the theory of fixed-parameter tractability, see, e.g, the textbook by Flum and Grohe [13].

FVS has been extensively studied in the context of parameterized algorithms. The earliest known FPT algorithms for FVS go back to the early 90's (e.g, [2]). After several rounds of improvements, the current best FPT algorithm for FVS runs in time $O(5^k kn^2)$ [5].

In this paper, we show that CFVS can be solved in time $O(2^{O(k)}n^{O(1)})$ on general graphs and in time $O(2^{O(\sqrt{k}\log k)}n^{O(1)})$ on graphs excluding a fixed graph H as a minor. Most of the known FPT algorithms for connectivity problems enumerate *all* minimal solutions and then try to connect each solution using an algorithm for the Steiner Tree problem. For instance, this is the case with the existing FPT algorithms for Connected Vertex Cover(e.g, [17]). The crucial observation which the algorithms for Connected Vertex Cover rely on is that there are at most 2^k *minimal* vertex covers of size at most k. However, this approach fails for CFVS as the number of minimal feedback vertex sets of size at most k is $\Omega(n^k)$ (consider a graph that is a collection of k vertex-disjoint cycles each of length approximately n/k). To circumvent this problem, we make use of "compact representations" of feedback vertex sets. A compact representation is simply a collection of families of mutually disjoint sets, where each family represents a number of different feedback vertex sets. This notion was defined by Guo et al. [16] who showed that the set of all minimal feedback vertex sets of size at most k can be represented by a collection of set-families of size $O(2^{O(k)})$.

We use compact representations to obtain an FPT algorithm for CFVS in Section 3. In order to do this we need an FPT algorithm for a general version of Steiner Tree, namely Group Steiner Tree (GST), which is defined as follows: Given a graph $G = (V, E); |V| = n, |E| = m$, subsets $T_i \subseteq V$, $1 \leq i \leq l$, and an integer p, does there exist a subgraph of G on p vertices that is a tree T and includes at least one vertex from each T_i? Observe that when the T_i's are

each of size one, then GST is the Steiner Tree problem. Our FPT algorithm for GST runs in polynomial space and uses a Turing-reduction to a directed version of Steiner Tree, called Directed Steiner Out-Tree, which we show to be fixed-parameter tractable. We note that GST is known to be of interest to database theorists, and that it has been studied in [10], where an algorithm with running time $O(3^l \cdot n + 2^l \cdot (n+m))$ (that uses exponential space) is discussed.

We also show that CFVS does not admit a polynomial kernel (See Section 2) on general graphs but has a quadratic kernel on the class of graphs that exclude a fixed graph H as minor. Finally, in Section 4 we design a subexponential-time algorithm for CFVS on graphs excluding some fixed graph H as a minor using the theory of bidimensionality. This algorithm is obtained using an $O^*(w^{O(w)})$-time algorithm that computes an optimal connected feedback vertex set in graphs of treewidth at most w.

2 Preliminaries

In this section we state some basic definitions related to parameterized complexity and graph theory, and give an overview of the notation used in this paper. To describe running times of algorithms we sometimes use the O^* notation. Given $f : \mathbb{N} \to \mathbb{N}$, we define $O^*(f(n))$ to be $O(f(n) \cdot p(n))$, where $p(\cdot)$ is some polynomial function. That is, the O^* notation suppresses polynomial factors in the running-time expression.

A parameterized problem Π is a subset of $\Gamma^* \times \mathbb{N}$, where Γ is a finite alphabet. An instance of a parameterized problem is a tuple (x, k), where k is called the parameter. A central notion in parameterized complexity is *fixed-parameter tractability (FPT)* which means, for a given instance (x, k), decidability in time $f(k) \cdot p(|x|)$, where f is an arbitrary function of k and p is a polynomial in the input size. The notion of *kernelization* is formally defined as follows.

Definition 1. [**Kernelization**] *[13,20]*
A kernelization algorithm for a parameterized problem $\Pi \subseteq \Gamma^* \times \mathbb{N}$ is an algorithm that, given $(x, k) \in \Gamma^* \times \mathbb{N}$, outputs, in time polynomial in $|x| + k$, a pair $(x', k') \in \Gamma^* \times \mathbb{N}$ such that (a) $(x, k) \in \Pi$ if and only if $(x', k') \in \Pi$ and (b) $|x'|, k' \leq g(k)$, where g is some computable function. The output instance x' is called the kernel, and the function g is referred to as the size of the kernel. If $g(k) = k^{O(1)}$ (resp. $g(k) = O(k)$) then we say that Π admits a polynomial (resp. linear) kernel.

We say that a graph G (undirected or directed) *contains* a graph H if H is a subgraph of G. Given a directed graph (digraph) $D = (V, A)$, we let $V(D)$ and $A(D)$ denote the vertex and arc set of D, respectively. A vertex $u \in V(D)$ is an *in-neighbor* (*out-neighbor*) of $v \in V(D)$ if $uv \in A$ ($vu \in A$, respectively). The in- and out-neighborhood of a vertex v are denoted by $N^-(v)$ and $N^+(v)$, respectively. The *in-degree* $d^-(v)$ (resp. *out-degree* $d^+(v)$) of a vertex v is $|N^-(v)|$ (resp. $|N^+(v)|$). We say that a subdigraph T of D with vertex set $V_T \subseteq V(D)$ is an *out-tree* if T is an oriented tree (see [1]) with only one vertex r of in-degree

zero (called the *root*). The vertices of T of out-degree zero are called *leaves* and every other vertex is called an *internal vertex*.

3 Connected Feedback Vertex in General Graphs

In this section we give an FPT algorithm for CFVS on general graphs. We start by describing an FPT algorithm for the Group Steiner Tree problem which is crucially used in our algorithm for CFVS.

3.1 Group Steiner Tree

The Group Steiner Tree (GST) problem is defined as follows:

Input:	An undirected graph $G = (V, E)$; vertex-disjoint subsets $S_1, \ldots, S_l \subseteq V$; and an integer p.
Parameter:	The integer l.
Question:	Does G contain a tree on at most p vertices that includes at least one vertex from each S_i?

Our fixed-parameter algorithm for GST first reduces it to Directed Steiner Out-Tree (defined below) which we then show to be fixed-parameter tractable.

Input:	A directed graph $D = (V, A)$; a distinguished vertex $r \in V$; a set of terminals $S \subseteq V$; and an integer p.
Parameter:	The integer $l = \|S\|$.
Question:	Does D contain an out-tree on at most p vertices that is rooted at r and that contains all the vertices of S?

Lemma 1. *The GST problem Turing-reduces to the* Directed Steiner Out-Tree *problem.*

Proof. Given an instance $(G = (V, E), S_1, \ldots, S_l, p)$ of GST, construct an instance of Directed Steiner Out-Tree as follows. Let $S = \{s_1, s_2, \ldots, s_l\}$ be a set of l new vertices, that is, $s_i \notin V$ for $1 \leq i \leq l$. Let $V' = V \cup S$ and $A = \{uv, vu : \{u, v\} \in E\} \cup \bigcup_{i=1}^{l} \{xs_i : x \in S_i\}$. Finally, let $D = (V', A)$. It is easy to see that G contains a tree on at most p vertices that includes at least one vertex from each S_i if and only if there exists a vertex $r \in V'$ and an out-tree in D rooted at r on at most $p + l$ vertices containing all vertices of S. □

Lemma 2. Directed Steiner Out-Tree *can be solved in* $O(2^l \cdot n^{O(1)})$ *time using polynomial space.*

Nederlof [19] uses the Inclusion-Exclusion Principle and a notion of *branching walks* to give an algorithm for the Steiner Tree problem that runs in $O(2^l \cdot n^{O(1)})$ time using polynomial space, where l is the number of terminals. Essentially the same algorithm works for Directed Steiner Out-Tree, with the same resource bounds; we omit the details due to space constraints.

Lemmas 1 and 2 together imply:

Lemma 3. *The* Group Steiner Tree *problem can be solved in* $O(2^l \cdot n^{O(1)})$ *time using polynomial space.*

3.2 An FPT Algorithm for CFVS

Our FPT algorithm for CFVS uses as a subroutine an algorithm (due to Guo et al. [16]) for enumerating an efficient representation of minimal feedback vertex sets of size at most k. Strictly speaking, the subroutine enumerates all compact representations of minimal feedback sets. A *compact representation* for a set of minimal feedback sets of a graph $G = (V, E)$ is a set $\mathcal{C}$ of pairwise disjoint subsets of V such that choosing exactly one vertex from every set in $\mathcal{C}$ results in a minimal feedback set for G. Call a compact representation a *k-compact representation* if the number of sets in the representation is at most k. Clearly, any connected feedback set of size at most k must necessarily pick vertices from the sets of *some* k-compact representation. Given a graph $G = (V, E)$ and a k-compact representation $S_1, \ldots, S_r$, where $r \leq k$, the problem of deciding whether there exists a connected feedback vertex set that contains at least one vertex from each set S_i reduces to the Group Steiner Tree problem where the Steiner groups are the sets of the compact representation.

Our algorithm therefore cycles through all k-compact representations and for each such representation uses the algorithm for Group Steiner Tree to check if there is a tree on at most k vertices that includes one vertex from each set S_i of the compact representation. If the answer is no for all k-compact representations, the algorithm reports that the given instance is a no-instance. If the answer is yes for some compact representation, the algorithm returns the tree found. Since one can enumerate all compact representations in time $O(c^k \cdot m)$ [16], we have:

Theorem 1. *Given a graph $G = (V, E)$ and an integer k, one can decide whether G has a connected feedback set of size at most k in time $O(c^k \cdot n^{O(1)})$, for some constant c.*

Although CFVS is fixed-parameter tractable, it is unlikely to admit a polynomial kernel as the following theorem shows. This is in contrast to Feedback Vertex Set which admits a quadratic kernel [22].

Theorem 2. *The* CFVS *problem does not admit a polynomial kernel unless the Polynomial Hierarchy collapses to Σ_3.*

Proof. The proof follows from a polynomial-time parameter-preserving reduction from Connected Vertex Cover, which does not admit a polynomial kernel unless the Polynomial Hierarchy collapses to the third level [11]. This would prove that CFVS too does not admit a polynomial kernel [4]. Given an instance $(G = (V, E), k)$ of the Connected Vertex Cover problem, construct a new graph G' as follows: $V(G') = V(G) \cup \{x_{uv} \notin V(G) : \{u, v\} \in E(G)\}$; if $\{u, v\} \in E(G)$ then add the edges $\{u, v\}, \{u, x_{uv}\}, \{x_{uv}, v\}$ to $E(G')$. This completes the construction of G'. It is easy to see that G has a connected vertex cover of size

at most k if and only if G' has a connected feedback vertex set of size at most k. This completes the proof of the theorem. □

Interestingly, the results from [15] imply that CFVS has polynomial kernel on a graph class $\mathcal{C}$ which excludes a fixed graph H as a minor(See Section 4.1).

We note in passing that the algorithm for enumerating compact representations can be improved using results from [6]. The authors of [6] describe a set of reduction rules such that if a YES-instance of the FOREST BIPARTITION problem (defined below) is reduced with respect to this set of rules then the instance has size at most $5k + 1$.

FOREST BIPARTITION

Input:	An undirected graph $G = (V, E)$, possibly with multiple edges and loops and a set $S \subseteq V$ such that $\|S\| = k + 1$ and $G \setminus S$ is acyclic.
Parameter:	The integer k.
Question:	Does G have a feedback vertex set of size at most k contained in $V \setminus S$?

Thus in a YES-instance of FOREST BIPARTITION that is reduced with respect to the rules in [6], we have $|V \setminus S| \leq 4k$. Using this bound in the algorithm described by Guo et al. [16], one obtains a $O^*(c^k)$-time algorithm for enumerating compact representations of minimal feedback vertex sets of size at most k, where $c = 52$. The constant c in [16] is more than 160.

Theorem 3. [6,16] *Given a graph $G = (V, E)$ and an integer k, the compact representations of all minimal feedback vertex sets of G of size at most k can be enumerated in time $O(52^k \cdot |E|)$.*

4 A Subexponential FPT Algorithm for CFVS on *H*-Minor-Free Graphs

In the last section, we obtained an $O^*(c^k)$ algorithm for CFVS on general graphs. In this section we show that CFVS on the class of H-minor-free graphs admits a sub-exponential time algorithm with running time $O(2^{O(\sqrt{k}\log k)}n^{O(1)})$. This section is divided into three parts. In the first part we give essential definitions from topological graph theory, and in the second part we show that CFVS can be solved in time $O(w^{O(w)}n^{O(1)})$ on graphs with treewidth bounded by w. In the last part we present an algorithm with the stated running time for CFVS on H-minor-free graphs, by bounding the treewidth of the input graph using the known "grid theorems".

4.1 Definitions and Terminology

We use terminology from [9]. Given an edge e in a graph G, the *contraction* of e is the result of identifying its endpoints in G and then removing all loops and

duplicate edges. A *minor* of a graph G is a graph H that can be obtained from a subgraph of G by contracting edges. A graph class $\mathcal{C}$ is *minor-closed* if any minor of any graph in $\mathcal{C}$ is also an element of $\mathcal{C}$. A minor-closed graph class $\mathcal{C}$ is *H-minor-free* or simply *H-free* if $H \notin \mathcal{C}$.

A *tree decomposition* of a graph $G = (V, E)$ is a pair $(T = (V_T, E_T), \mathcal{X} = \{X_t\}_{t \in V_T})$ where T is a tree and the X_t are subsets of V such that:

1. $\bigcup_{u \in V_T} X_t = V$;
2. for each edge $e = \{u, v\} \in E$ there exists $t \in V_T$ such that $u, v \in X_t$; and
3. for each vertex $v \in V$, the subgraph $T[\{t \mid v \in X_t\}]$ is connected.

The *width* of a tree decomposition is $\max_{t \in V_T} |X_t| - 1$ and the *treewidth* of $G = (V, E)$, denoted $tw(G)$, is the minimum width over all tree decompositions of G.

A tree decomposition is called a *nice tree decomposition* [3] if the following conditions are satisfied:

- Every node of the tree T has at most two children. A node that has no children is called a *leaf* node. The non-leaf nodes are of three kinds:
  - If a node t has two children t_1 and t_2, then $X_t = X_{t_1} = X_{t_2}$, and t is called a *join* node.
  - if a node t has one child t_1, then either $|X_t| = |X_{t_1}| + 1$ and $X_{t_1} \subset X_t$ (t is called an *introduce* node), or $|X_t| = |X_{t_1}| - 1$ and $X_t \subset X_{t_1}$ (t is called a *forget* node).

It is possible to transform a given tree decomposition into a nice tree decomposition in time $O(|V| + |E|)$ [3].

4.2 Connected FVS and Treewidth

In this section we show that the Connected Feedback Vertex Set problem is FPT with the treewidth of the input graph as the parameter. That is, we show that the following problem is FPT:

Input:	An undirected graph $G = (V, E)$; an integer k; and a nice tree decomposition of G of width w.
Parameter:	The treewidth w of the graph G.
Question:	Does there exist $S \subseteq V$ such that $G \setminus S$ is acyclic, $G[S]$ is connected, and $\lvert S\rvert \leq k$?

We design a dynamic programming algorithm on the nice tree decomposition with running time $O(w^{O(w)} \cdot n^{O(1)})$ for this problem. See, e.g, Moser [18] for a detailed exposition of this paradigm; in particular, our algorithm is similar in spirit to the algorithm given in [18] for the Connected Vertex Cover problem.

Let $(T = (I, F), \{X_i | i \in I\})$ be a nice tree decomposition of the input graph G of width w and rooted at $r \in I$. We let T_i denote the subtree of T rooted at

$i \in I$, and $G_i = (V_i, E_i)$ denote the subgraph of G induced on all the vertices of G in the subtree T_i, that is, $G_i = G[\bigcup_{j \in V(T_i)} X_j]$.

For each node $i \in I$ we compute a table A_i, the rows of which are 4-tuples $[S, P, Y, val]$. Table A_i contains one row for each combination of the first three components which denote the following:

- S is a subset of X_i.
- P is a partition of S into at most $|S|$ labelled pieces.
- Y is a partition of $X_i \setminus S$ into at most $|X_i \setminus S|$ labelled pieces.

We use $P(v)$ (resp. $Y(v)$) to denote the piece of the partition P (resp. Y) that contains the vertex v. We let $|P|$ (resp. $|Y|$) denote the number of pieces in the partition P (resp. Y). The last component *val*, also denoted as $A_i[S, P, Y]$, is the size of a smallest feedback vertex set $F_i \subseteq V(G_i)$ of G_i which satisfies the following properties:

- If $S = \emptyset$, then F_i is *connected* in G_i.
- If $S \neq \emptyset$, then
  - $F_i \cap X_i = S$.
  - All vertices of S that are in any one piece of P are in a single connected component of $G_i[F_i]$. Moreover $G_i[F_i]$ has exactly $|P|$ connected components.
  - All vertices of $X_i \setminus S$ that are in the same piece of Y are in a single connected component (a tree) of $G_i[V_i \setminus F_i]$. Moreover $G_i[V_i \setminus F_i]$ has at least $|Y|$ connected components.

If there is no such set F_i, then the last component of the row is set to ∞.

We fix an arbitrary ordering of the vertices of X_i, and compute the table A_i for each node $i \in I$ of the tree decomposition. Since there are at most $w + 1$ vertices in each bag X_i, there are no more than

$$\sum_{i=0}^{w+1} \binom{w+1}{i} i^i \cdot (w+1-i)^{w+1-i} \leq (2w+2)^{2w+2}$$

rows in any table A_i. We compute the tables A_i starting from the leaf nodes of the tree decomposition and going up to the root.

Leaf Nodes. Let i be a leaf node of the tree decomposition. We compute the table A_i as follows. For each triple (S, P, Y) where S is a subset of X_i, P a partition of S, and Y a partition of $X_i \setminus S$:

- Set $A_i[S, P, Y] = \infty$ if at least one of the following holds:
  - $G_i \setminus S$ contains a cycle (i.e., S is *not* an FVS of G_i).
  - At least one piece of P is *not* connected in $G_i[S]$ or if $G_i[S]$ has less than $|S|$ connected components.
  - At least one piece of Y is *not* connected in $G_i[V_i \setminus S]$ or if $G_i[V_i \setminus S]$ has less than $|Y|$ connected components.
- In all other cases, set $A_i[S, P, Y] = |S|$.

It is easy to see that this computation correctly determines the last component of each row of A_i for a leaf node i of the tree decomposition.

Introduce Nodes. Let i be an introduce node and j its unique child. Let $x \in X_i \setminus X_j$ be the introduced vertex. For each triple (S, P, Y), we compute the entry $A_i[S, P, Y]$ as follows.

Case 1. $x \in S$. Check whether $N(x) \cap S \subseteq P(x)$; if not, set $A_i[S, P, Y] = \infty$.

- Subcase 1. $P(x) = \{x\}$. Set $A_i[S, P, Y] = A_j[S \setminus \{x\}, P \setminus P(x), Y] + 1$.
- Subcase 2: $|P(x)| \geq 2$ and $N(x) \cap P(x) = \emptyset$. Set $A_i[S, P, Y] = \infty$, as no extension of S to an fvs for G_i can make $P(x)$ connected.
- Subcase 3: $|P(x)| \geq 2$ and $N(x) \cap P(x) \neq \emptyset$. Let $\mathcal{A}$ be the set of all rows $[S', P', Y]$ of the table A_j that satisfy the following conditions:
  - $S' = S \setminus \{x\}$.
  - $P' = (P \setminus P(x)) \cup Q$, where Q is a partition of $P(x) \setminus \{x\}$ such that each piece of Q contains an element of $N(x) \cap P(x)$.

  Set $A_i[S, P, Y] = \min_{[S', P', Y] \in \mathcal{A}} \{A_j[S', P', Y]\} + 1$.

Case 2. $x \notin S$. Check whether $N(x) \cap (X_i \setminus S) \subseteq Y(x)$; if not, set $A_i[S, P, Y] = \infty$.

- Subcase 1: $Y(x) = \{x\}$. Set $A_i[S, P, Y] = A_j[S, P, Y \setminus Y(x)]$.
- Subcase 2: $|Y(x)| \geq 2$ and $N(x) \cap Y(x) = \emptyset$. Set $A_i[S, P, Y] = \infty$, as no extension of S to an fvs F_i for G_i can make $Y(x)$ a connected component in $G_i[V_i \setminus F_i]$.
- Subcase 3: $|Y(x)| \geq 2$ and $N(x) \cap Y(x) \neq \emptyset$. Let $\mathcal{A}$ be the set of all rows $[S, P, Y']$ of the table A_j where $Y' = (Y \setminus Y(x)) \cup Q$, and Q is a partition of $Y(x) \setminus \{x\}$ such that each piece of Q contains *exactly* one element of $N(x) \cap Y(x)$. Set $A_i[S, P, Y] = \min_{[S, P, Y'] \in \mathcal{A}} \{A_j[S, P, Y']\}$.

Forget Nodes. Let i be a forget node and j its unique child node. Let $x \in X_j \setminus X_i$ be the forgotten vertex. For each triple (S, P, Y) in the table A_i, let $\mathcal{A}$ be the set of all rows $[S', P', Y]$ of the table A_j that satisfy the following conditions:

- $S' = S \cup \{x\}$, and
- $P'(x) = P(y) \cup \{x\}$ for some $y \in S$.

Let $\mathcal{B}$ be the set of all rows $[S, P, Y']$ of the table A_j such that $Y'(x) = Y(z) \cup \{x\}$ for some $z \in S$. Set

$$A_i[S, P, Y] = \min \left\{ \min_{[S', P', Y] \in \mathcal{A}} A_j[S', P', Y], \min_{[S, P, Y'] \in \mathcal{B}} A_j[S, P, Y'] \right\}.$$

Join Nodes. Let i be a join node and j and l its children. For each triple (S, P, Y) we compute $A_i[S, P, Y]$ as follows.

- Case 1. $S = \emptyset$. If both $A_j[\emptyset, P, Y]$ and $A_l[\emptyset, P, Y]$ are positive finite, then set $A_i[\emptyset, P, Y] = \infty$. Otherwise, set $A_i[\emptyset, P, Y] = \max\{A_j[\emptyset, P, Y], A_l[\emptyset, P, Y]\}$.

– Case 2. $S \neq \emptyset$. Let $\mathcal{A}$ denote the set of all pairs of triples $\langle (S, P_1, Y_1), (S, P_2, Y_2) \rangle$, where $(S, P_1, Y_1) \in A_j$ and $(S, P_2, Y_2) \in A_l$ with the following property: Starting with the partitions $Q_p = P_1$ and $Q_y = Y_1$ and repeatedly applying the following set of operations, we reach stable partitions that are identical to P and Y. The first operation that we apply is:

  If there exist vertices $u, v \in S$ such that they are in different pieces of Q_p but are in the same piece of P_2, delete $Q_p(u)$ and $Q_p(v)$ from Q_p and add $Q_p(u) \cup Q_p(v)$.

  To describe the second set of operations, we need some notation. Let $Z = X_i \setminus S$ and let the connected components of $G_i[Z]$ be $C_1, \ldots, C_q$. First contract each connected component C_i to a vertex c_i, the *representative* of that component, and let $\mathcal{C} = \{c_1, \ldots, c_q\}$. Note that for each $1 \leq i \leq q$, the component C_i is not split across pieces in either Y_1 or Y_2. Denote by Y_1' and Y_2' the partitions obtained from Y_1 and Y_2, respectively, be replacing each connected component C_i by its representative vertex c_i. Let $Q_y = Y_1'$. Repeat until no longer possible:

  If there exist $c_a, c_b \in \mathcal{C}$ that are in different pieces of Q_y but in the same piece of Y_2 then delete $Q_y(c_a), Q_y(c_b)$ from Q_y and add $Q_y(c_a) \cup Q_y(c_b)$ provided the following condition holds: for all $c_e \in \mathcal{C} \setminus \{c_a, c_b\}$ either $Y_2(c_e) \cap Q_y(c_a) = \emptyset$ or $Y_2(c_e) \cap Q_y(c_b) = \emptyset$.

  If this latter condition does not hold, move on to the next pair of triples. Finally expand each c_i to the connected component it represents.

  Set

$$A_i[S, P, Y] = \min_{\langle (S,P_1,Y_1),(S,P_2,Y_2) \rangle \in \mathcal{A}} \{A_j[S, P_1, Y_1] + A_l[S, P_2, Y_2] - |S|\}.$$

  The stated conditions ensure that $u, v \in S$ are in the same piece of P if and only if for each $\langle (S, P_1, Y_1), (S, P_2, Y_2) \rangle \in \mathcal{A}$, they are in the same piece of P_1 or of P_2 (or both). Similarly, the stated conditions ensure that merging solutions at join nodes do not create new cycles. Given this, it is easy to verify that the above computation correctly determines $A_i[S, P, Y]$.

Root Node. We compute the size of a smallest CFVS of G from the table A_r for the root node r as follows. Find the minimum of $A_r[S, P, Y]$ over all triples (S, P, Y), where $S \subseteq X_r$, P a partition of S such that P consists of a single (possibly empty) piece and Y is a partition of $X_r \setminus S$. This minimum is the size of a smallest CFVS of G.

This concludes the description of the dynamic programming algorithm for CFVS when the treewidth of the input graph is bounded by w. From the above description and the size of tables being bounded by $(2w + 2)^{2w+2}$, we obtain the following result.

Lemma 4. *Given a graph $G = (V, E)$, a tree-decomposition of G of width w, one can compute the size of an optimum connected feedback vertex set of G (if it exists) in time $O((2w + 2)^{2w+2} \cdot n^{O(1)})$.*

4.3 FPT Algorithms for H-Minor Free Graphs

We first bound the treewidth of the yes instance of input graphs by $O(\sqrt{k})$.

Lemma 5. *If (G, k) is a yes-instance of CFVS where G excludes a fixed graph H as a minor, then $\mathbf{tw}(G) \leq c_H\sqrt{k}$, where c_H is a constant that depends only on the graph H.*

Proof. By [7], for any fixed graph H, every H-minor-free graph G that does not contain a $(w \times w)$-grid as a minor has treewidth at most $c'_H w$, where c'_H is a constant that depends only on the graph H. Clearly a $(w \times w)$-grid has a feedback vertex set of size at least $c_1 w^2$, where c_1 is a constant independent of w. Therefore if G has a connected feedback vertex set of size at most k, it cannot have a $(w \times w)$-grid minor, where $w > \sqrt{k/c_1}$. Therefore $\mathbf{tw}(G) \leq c'_H w \leq c'_H \cdot (\sqrt{k/c_1} + 1) \leq c_H\sqrt{k}$, where $c_H = (c'_H + 1)/\sqrt{c_1}$. □

Theorem 4. *CFVS can be solved in time $O(2^{O(\sqrt{k}\log k)} + n^{O(1)})$ on H-minor-free graphs.*

Proof. Given an instance (G, k) of CFVS, we first find a tree-decomposition of G using the polynomial-time constant-factor approximation algorithm of Demaine et al. [8]. If $\mathbf{tw}(G) > c_H\sqrt{k}$, then the given instance is a no-instance; else, use Lemma 4 to find an optimal CFVS for G. All this can be done in $O(2^{O(\sqrt{k}\log k)} \cdot n^{O(1)})$. To obtain the claimed running time bound we first apply the results from [15] and obtain an $O(k^2)$ kernel for the problem in polynomial time and then apply the algorithm described. □

5 Conclusion

We conclude with some open problems. The obvious question is to obtain an $O^*(c^k)$ algorithm for CFVS in general graphs with a smaller value of c. Also the approximability of CFVS in general graphs is unknown. Is there a constant-factor approximation algorithm for CFVS? If not, what is the limit of approximation? Is there an $O^*(c^w)$ algorithm for CFVS, for a constant c, for graphs of treewidth at most w? Note that this question is open even in the context of finding a (unconnected) feedback vertex set in graphs of treewidth at most w.

References

1. Bang-Jensen, J., Gutin, G.Z.: Digraphs: Theory, Algorithms and Applications, 2nd edn. Springer, Heidelberg (2009)
2. Bodlaender, H.L.: On disjoint cycles. In: Schmidt, G., Berghammer, R. (eds.) WG 1991. LNCS, vol. 570, pp. 230–238. Springer, Heidelberg (1992)
3. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM Journal on Computing 25(6), 1305–1317 (1996)

4. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels (extended abstract). In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 563–574. Springer, Heidelberg (2008)
5. Chen, J., Fomin, F.V., Liu, Y., Lu, S., Villanger, Y.: Improved algorithms for the feedback vertex set problems. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 422–433. Springer, Heidelberg (2007)
6. Dehne, F., Fellows, M., Langston, M.A., Rosamond, F., Stevens, K.: An $O(2^{O(k)}n^3)$ FPT-Algorithm for the Undirected Feedback Vertex Set problem. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 859–869. Springer, Heidelberg (2005)
7. Demaine, E.D., Hajiaghayi, M.: Linearity of grid minors in treewidth with applications through bidimensionality. Combinatorica 28(1), 19–36 (2008)
8. Demaine, E.D., Hajiaghayi, M., ichi Kawarabayashi, K.: Algorithmic graph minor theory: Decomposition, approximation, and coloring. In: Proceedings of FOCS 2005, pp. 637–646. IEEE Computer Society, Los Alamitos (2005)
9. Diestel, R.: Graph Theory, 3rd edn. Springer, Heidelberg (2005)
10. Ding, B., Yu, J.X., Wang, S., Qin, L., Zhang, X., Lin, X.: Finding top-k min-cost connected trees in databases. In: ICDE, pp. 836–845. IEEE, Los Alamitos (2007)
11. Dom, M., Lokshtanov, D., Saurabh, S.: Incompressibility through Colors and IDs. In: Albers, S., et al. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 378–389. Springer, Heidelberg (2009)
12. Festa, P., Pardalos, P.M., Resende, M.G.: Feedback set problems. In: Handbook of Combinatorial Optimization, pp. 209–258. Kluwer Academic Publishers, Dordrecht (1999)
13. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin (2006)
14. Fomin, F.V., Grandoni, F., Kratsch, D.: Solving connected dominating set faster than 2^n. Algorithmica 52(2), 153–166 (2008)
15. Fomin, F.V., Lokshtanov, D., Saurabh, S., Thilikos, D.M.: Bidimensionality and kernels. In: Proceedings of SODA 2010 (2010) (to appear)
16. Guo, J., Gramm, J., Hüffner, F., Niedermeier, R., Wernicke, S.: Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. Journal of Computer and System Sciences 72(8), 1386–1396 (2006)
17. Mölle, D., Richter, S., Rossmanith, P.: Enumerate and expand: Improved algorithms for connected vertex cover and tree cover. Theory of Computing Systems 43(2), 234–253 (2008)
18. Moser, H.: Exact algorithms for generalizations of vertex cover. Master's thesis, Institut für Informatik, Friedrich-Schiller-Universität (2005)
19. Nederlof, J.: Fast polynomial-space algorithms using möbius inversion: Improving on steiner tree and related problems. In: Albers, S., et al. (eds.) ICALP 2009, pp. 713–725. Springer, Heidelberg (2009)
20. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics and its Applications, vol. 31. Oxford University Press, Oxford (2006)
21. Sitters, R., Grigoriev, A.: Connected feedback vertex set in planar graphs. In: Paul, C., Habib, M. (eds.) WG 2009. LNCS, vol. 5911. Springer, Heidelberg (2009)
22. Thomassé, S.: A quadratic kernel for feedback vertex set. In: Proceedings of SODA 2009, pp. 115–119. Society for Industrial and Applied Mathematics (2009)

A Simple and Fast Algorithm for Maximum Independent Set in 3-Degree Graphs
(Extended Abstract)

Mingyu Xiao*

School of Computer Science and Engineering
University of Electronic Science and Technology of China
Chengdu, China
myxiao@gmail.com

Abstract. We present a simple $O^*(1.0885^n)$-time algorithm for finding a maximum independent set in an n-vertex graph with degree bounded by 3, which improves most previous running time bounds obtained with far more complicated algorithms. In this paper, we use a nontraditional measure to analyze the problem size and some uniform branching rules to avoid tedious case analysis. Those techniques help us to design simple and fast algorithms with moderately complicated analysis.

1 Introduction

The *maximum independent set* problem (MIS), to find a maximum set of vertices in a graph such that there is no edge between any two vertices in the set, is one of the basic NP-hard optimization problems and has been well studied in the literature, in particular in the line of research on worst-case analysis of algorithms for NP-hard optimization problems. In 1977, Tarjan and Trojanowski [1] published the first algorithm for this problem, which runs in $O^*(2^{n/3})$ time and polynomial space. Later, the running time was improved to $O^*(2^{0.304n})$ by Jian [2]. Robson [3] obtained an $O^*(2^{0.296n})$-time polynomial-space algorithm and an $O^*(2^{0.276n})$-time exponential-space algorithm. In a technical report [4], Robson also claimed better running times. Recently, Fomin *et al.* [5] got a simple $O^*(2^{0.288n})$-time polynomial-space algorithm by using the "Measure and Conquer" method. There is also a considerable amount of contributions to the maximum independent set problem in sparse graphs, especially in degree-3 graphs [6,7,8,9]. We summarize currently published results on low-degree graphs as well as general graphs in Table 1.

In the literature, there are several methods for designing algorithms for finding maximum independent sets in graphs. One method is to find a minimum

* This work was supported in part by the National Natural Science Foundation of China Grant (No. 60903007) and the UESTC Youth Science Funds (No. JX0843). Part of the work was done when the author was a Ph.D. student in Department of Computer Science and Engineering, the Chinese University of Hong Kong.

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 281–292, 2010.

Table 1. Published exact algorithms for the maximum independent set problem

Authors	Running times	References	Notes
Tarjan & Trojanowski	$O^*(1.2600^n)$ for MIS	1977 [1]	n: number of vertices
Jian	$O^*(1.2346^n)$ for MIS	1986 [2]	
Robson	$O^*(1.2109^n)$ for MIS	1986 [3]	Exponential space
Beigel	$O^*(1.0823^m)$ for MIS $O^*(1.1259^n)$ for 3-MIS	1999 [6]	m: number of edges 3-MIS: MIS in degree-3 graphs
Chen et al.	$O^*(1.1254^n)$ for 3-MIS	2003 [7]	
Xiao et al.	$O^*(1.1034^n)$ for 3-MIS	2005 [8]	Published in Chinese
Fomin et al.	$O^*(1.2210^n)$ for MIS	2006 [5]	
Fomin & Høie	$O^*(1.1225^n)$ for 3-MIS	2006 [10]	
Fürer	$O^*(1.1120^n)$ for 3-MIS	2006 [11]	
Razgon	$O^*(1.1034^n)$ for 3-MIS	2006 [12]	
Bourgeois et al.	$O^*(1.0977^n)$ for 3-MIS	2008 [9]	
Razgon	$O^*(1.0892^n)$ for 3-MIS	2009 [13]	

vertex cover (a set of vertices such that each edge in the graph has at least one endpoint in the set), and then to get a maximum independent set by taking all the remaining vertices, such as the algorithms presented in [7,14]. In this kind of algorithms, the dominating part of the running time is the running time for finding a minimum vertex cover. Another method is based on the search tree method. We will use a branch-and-reduce paradigm. We choose a parameter, such as the number of vertices or edges or others, as a measure of the size of the problem. When the parameter is zero or a negative number, the problem can be solved in polynomial time. We branch on the current graph G into several graphs $G_1, G_2, \cdots, G_l$ such that the parameter r_i of graph G_i is less than the parameter r of graph G ($i = 1, 2, \cdots, l$), and a maximum independent set in G can be found in polynomial time if a maximum independent set in each of the l graphs G_1, G_2, $\cdots, G_l$ is known. With this method, we can build up a search tree, and the exponential part of the running time of the algorithm corresponds to the size of the search tree. The running time analysis leads to a linear recurrence for each node in the search tree that can be solved by using standard techniques. Let $C(r)$ denote the worst-case size of the search tree when the parameter of graph G is r, then we get the recurrence relation $C(r) \leq \sum_{i=1}^{l} C(r_i)$. Solving the recurrence, we get $C(r) = [\alpha(r, r_1, r_2, \cdots, r_l)]^r$, where $\alpha(r, r_1, r_2, \cdots, r_l)$ is the largest root of the function $f(x) = 1 - \sum_{i=1}^{l} x^{-r_i}$. As for the measure (the parameter r), a natural one is the number of vertices or edges in the graph. Most previous algorithms for the maximum independent set problem were analyzed by using the number of vertices as a measure [1,2,3,5]. The number of edges is considered in Beigel's algorithm [6]. There are also some other measures. Xiao et al. [8] used the number of degree-3 vertices as a measure to analyze algorithms and got an $O^*(1.1034^n)$-time algorithm for MIS in degree-3 graphs. Unfortunately, that paper was published in Chinese. Recently, Razgon [12] also got an $O^*(1.1034^n)$-time algorithm for MIS in degree-3 graphs by measuring the number of degree-3 vertices. But the two algorithms are totally different. Fürer [11] designed an algorithm for MIS in degree-3 graphs by tackling $m - n$,

where m is the number of edges and n the number of vertices. Based upon a refined branching with respect to Fürer's algorithm, Bourgeois et al. [9] got an $O^*(1.0977^n)$-time algorithm for MIS in degree-3 graphs. Currently, the best published result on this problem is Razgon's $O^*(1.0892^n)$-time algorithm [13]. In a recent technical report, Bourgeois *et al.* [15] claimed an algorithm with running time $O^*(1.0854^n)$.

Most fast algorithms for the maximum independent set problem are obtained via careful examinations of the structures in the graph. In those algorithms, a long list of reduction and branching rules are used, which is derived from a somewhat tedious and complicated case analysis. In this paper, we use a new measure and some new branching rules to design a quite simple (does not contain many branching rules) and fast algorithm. We will use $r = sum_{v \in V}(d_v)$ as a measure to analyze our algorithm, where $d_v = \max(0, d(v) - 2)$ and $d(v)$ is the degree of vertex v. When the graph is a degree-3 graph, measure r is the number of degree-3 vertices in the graph. Our algorithm runs in $O^*(1.0885^n)$ time, which slightly improves the best published result of $O^*(1.0892^n)$ [13]. Some techniques in this paper can also be used to simplify previous algorithms. Furthermore, our result can be used to solve the *k-vertex cover* problem (to decide if the graph has a vertex cover of size k) in degree-3 graphs in $O^*(1.1849^k)$ time.

2 Preliminaries

We shall try to be consistent in using the following notation. The number of vertices in a graph will be denoted by n and the measure will be denoted by r. For a vertex v in a graph, $d(v)$ is the degree of v, $N(v)$ the set of all neighbors of v, $N[v] = N(v) \cup \{v\}$ the set of vertices with distance at most 1 from v, and $N_2(v)$ the set of vertices with distance exactly 2 from v. We say edge e is incident on a vertex set V', if at least one endpoint of e is in V'. A component of a graph always means a connected component of the graph. In our algorithm, when we remove a set of vertices, we also remove all the edges that are incident on it. Throughout the paper we use a modified O notation that suppresses all polynomially bounded factors. For two functions f and g, we write $f(n) = O^*(g(n))$ if $f(n) = O(g(n)poly(n))$, where $poly(n)$ is a polynomial.

Our algorithms are based on the branch-and-reduce paradigm. We will first apply some reduction rules to reduce the size of instances of the problem. Then we apply some branching rules to branch on the graph by including some vertices in the independent set or excluding some vertices from the independent set. In each branch, we will get a maximum independent set problem in a graph with a smaller measure. Next, we introduce the reduction rules and branching rules that will be used in our algorithms.

2.1 Reduction Rules

There are several standard preprocesses to reduce the size of instances of the problem. *Folding a degree-*1 *or degree-*2 *vertex* and *removing a dominated vertex* [14,5] are frequently used rules. Besides these reduction rules, we still need

to reduce some other local structures called 2-3 *structure*, 3-3 *structure* and 3-4 *structure*.

Folding a degree-1 vertex
Folding a degree-1 vertex v means removing v and u from the graph, where u is the unique neighbor of v.

Folding a degree-2 vertex
Folding a degree-2 vertex v (with two neighbors a and b) means

(a) removing v, a and b from the graph, when a and b are adjacent.
(b) removing v, a and b from the graph and introducing a new vertex s that is adjacent to all neighbors of a and b in G (except the removed vertex v), when a and b are nonadjacent.

Please refer to Figure 1 for an illustration of the operation in case (b) of folding a degree-2 vertex. Let $\alpha(G)$ denote the size of a maximum independent set of graph G and $G^\star(v)$ the graph after folding a degree-1 or degree-2 vertex v in G. Then we have the following lemma.

Lemma 1. *For any degree-1 or degree-2 vertex v in graph G,*

$$\alpha(G) = 1 + \alpha(G^\star(v)).$$

The correctness of folding a degree-1 or degree-2 vertex has been discussed in many previous papers. In fact, general folding rules are known in the literature,

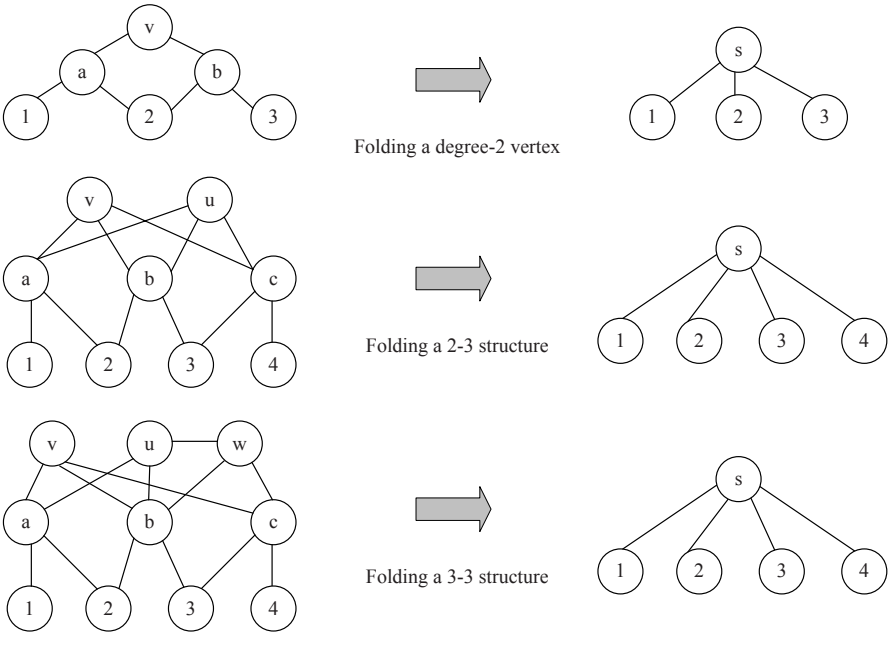


Fig. 1. Illustrations of folding operations of case (b)

which can deal with a vertex of degree ≥ 3 or a set of independent vertices [14,5]. In this paper, we still need to fold the following three local structures called 2-3 structure, 3-3 structure and 3-4 structure.

Let v and u be two independent degree-3 vertices, if they have three common neighbors a, b and c, then we say that the five vertices compose a 2-3 *structure* (see Figure 1), and denote it by $\{v, u\}$-$\{a, b, c\}$. Let v be a degree-3 vertex, and u and w two adjacent vertices of degree ≥ 3. If $N(u) \cup N(w) - \{u, w\} = N(v)$, then we say that the six vertices $\{v, u, w\} \cup N(v)$ compose a 3-3 *structure* (see Figure 1), and denote it by $\{v, u, w\}$-$\{a, b, c\}$, where $\{a, b, c\} = N(v)$. Let u, v and w be three independent vertices of degree ≥ 3. Let $N(u, v, w) = N(u) \cup N(v) \cup N(w) - \{u, v, w\}$. If $|N(u, v, w)| = 4$, then the seven vertices $\{u, v, w\} \cup N\{u, v, w\}$ compose a 3-4 *structure*. It is denoted by $\{v, u, w\}$-$\{a, b, c, d\}$, where $\{a, b, c, d\} = N\{u, v, w\}$.

Folding a 2-3 structure, 3-3 structure or 3-4 structure
Let A-B be a 2-3 *structure or* 3-3 *structure or* 3-4 *structure. Folding A-B means*

(*a*) *removing* $A \cup B$ *from the graph, when* B *is not an independent set.*
(*b*) *removing* $A \cup B$ *from the graph and introducing a new vertex* s *that is adjacent to all neighbors of vertices in* B (*except the removed vertices*), *when* B *is an independent set.*

Lemma 2. *If graph* G *has a* 2-3 *structure or* 3-3 *structure, then*

$$\alpha(G) = 2 + \alpha(G_2^\star),$$

where $G_2^\star$ *is the graph after folding a* 2-3 *structure or* 3-3 *structure in* G.
If graph G *has a* 3-4 *structure, then*

$$\alpha(G) = 3 + \alpha(G_3^\star),$$

where $G_3^\star$ *is the graph after folding a* 3-4 *structure in* G.

A degree-2 vertex can be regarded as a 1-2 structure. In fact, degree-2 vertex, 2-3 structure and 3-4 structure are special cases described in Lemma 2.4 in [14]. The 3-3 structure is introduced for the first time. The correctness of folding A-B (a 1-2 structure, 2-3 structure, 3-3 structure or 3-4 structure) follows from the observation: When B is not an independent set, there is a maximum independent set that contains A (two independent vertices in A, when A-B is a 3-3 structure). When B is an independent set, there is a maximum independent set that contains either B or A (B or two independent vertices in A, when A-B is a 3-3 structure). We omit the detailed proofs here.

Dominance
If there are two vertices v *and* u *such that* $N[u] \subseteq N[v]$, *we say* u *dominates* v.

Lemma 3. *If vertex* v *is dominated by any other vertices in graph* G, *then*

$$\alpha(G) = \alpha(G - \{v\}).$$

Definition 1. *A graph is called a* reduced graph, *if it has no degree-*1 *vertex, degree-*2 *vertex, dominated vertex,* 2-3 *structure,* 3-3 *structure or* 3-4 *structure.*

2.2 Branching Rules

Next we introduce two branching techniques, *branching on a bottle* and *branching on a* 4*-cycle*, which are simple and obvious, but can be used to avoid tedious branching rules in the algorithms.

Let a be a degree-3 vertex, and b, c, d the three neighbors of a. If two neighbors of a, say c and d, are adjacent, then we say that the four vertices compose a *bottle* and denote it by b-a-$\{c, d\}$.

Lemma 4. *Let b-a-$\{c, d\}$ be a bottle in graph G, then there is a maximum independent set S in G such that either $a \in S$ or $b \in S$.*

Proof. If b is not in a maximum independent set, we can directly remove b from the graph. In the remaining graph a becomes a degree-2 vertex and the two neighbors of it are adjacent. In this case, there is a maximum independent set that contains a. ∎

Based on Lemma 4, we get the following branching rule.

Branching on a bottle
Branching on a bottle b-a-$\{c, d\}$ means branching by either including a in the independent set or including b in the independent set.

Note. In fact, we can fold a bottle by using the general folding rule mentioned in [5] (also in [6]), but that folding rule is helpless for our analysis, especially when the three neighbors of the degree-3 vertex are high-degree vertices.

Let a, b, c and d be four vertices in graph G, if G has four edges ab, bc, cd and da, then we say that $abcd$ is a 4*-cycle* in G.

Lemma 5. *Let $abcd$ be a* 4*-cycle in graph G, then for any independent set S in G, either $a, c \notin S$ or $b, d \notin S$.*

Proof. Since any independent set contains at most 2 vertices in a 4-cycle and the two vertices cannot be adjacent, we know the lemma holds. ∎

Based on Lemma 5, we get the following branching rule.

Branching on a 4-cycle
Branching on a 4*-cycle $abcd$ means branching by either excluding a and c from the independent set or excluding b and d from the independent set.*

3 A Simple Algorithm

Our algorithm for the maximum independent set problem is described in Figure 2. It works as follows. When the graph is not a reduced graph, we apply our reduction rules to reduce the graph in **Step** $2 \sim 5$. When the graph cannot be reduced, we apply our branching rules. If there is a bottle, we branch on a bottle in **Step** 6. Else if there is a 4-cycle, we branch on a 4-cycle in **Step** 7. Else in **Step** 8, we greedily select a vertex of maximum degree and branch on it by including it in the independent set or excluding it from the independent set.

Input: A graph G.
Output: The size of a maximum independent set in G.

1. **If** $\{G$ has a component P of at most 15 vertices$\}$, **return** $t+MIS(G-P)$, where t is the size of a maximum independent set in P.
2. **Else if** $\{\exists v \in V$: $d(v) = 1$ or $2\}$, **return** $1 + MIS(G^\star(v))$.
3. **Else if** $\{\exists v, u \in V$: $N[u] \subseteq N[v]\}$, **return** $MIS(G - \{v\})$.
4. **Else if** $\{$there is a 2-3 structure or 3-3 structure$\}$, **return** $2 + MIS(G_2^\star)$.
5. **Else if** $\{$there is a 3-4 structure$\}$, **return** $3 + MIS(G_3^\star)$.
6. **Else if** $\{$there is a bottle b-a-$\{c, d\}\}$, **return** $\max\{1+MIS(G-N[a]), 1+MIS(G-N[b])\}$.
7. **Else if**$\{$there is a 4-cycle $abcd\}$, **return** $\max\{MIS(G-\{a, c\}), MIS(G-\{b, d\})\}$.
8. **Else,** pick up a vertex v of maximum degree, and **return** $\max\{MIS(G-\{v\}), 1 + MIS(G - N[v])\}$.

Note: With a few modifications, the algorithm can provide a maximum independent itself.

Fig. 2. The Algorithm $MIS(G)$

4 The Analysis

To analyze the time complexity of our algorithm, we will consider recurrence relations related to measure $r = sum_{v \in V}(d_v)$ in the corresponding graph, where $d_v = \max(0, d(v) - 2)$ and $d(v)$ is the degree of vertex v. When measure $r = 0$, the graph has only degree-0, degree-1 and degree-2 vertices and the maximum independent set problem can be solved in linear time. We use $C(r)$ to denote the worst-case size of the search tree in our algorithm when the measure of the graph is r, and consider how much the measure can be reduced in each branch of our search tree. To make the measure reduction clearer, we adopt a notation to indicate how much r is reduced from a vertex or a set of vertices in an operation. For example, when we remove a degree-d ($d \geq 3$) vertex v from the graph, we have $v \to d - 2$. Furthermore, if all the neighbors of v are vertices of degree > 2, then in this operation we still have $N(v) \to d$. Totally, we will reduce r by at least $d - 2 + d = 2d - 2$. Next, we analyze how much r can be reduced in each step of our algorithm.

Lemma 6. *After folding a degree-*1 *or degree-*2 *vertex, measure* r *will not increase.*

Lemma 7. *Let* G *be a graph having no degree-*1 *or degree-*2 *vertex, then after folding a* 2-3 *structure or* 3-3 *structure or* 3-4 *structure, or removing a dominated vertex from* G*, measure* r *will be reduced by at least* 4.

Proof. In each case, a degree-3 vertex is removed (or an even better case occurs), then r will be reduced by at least 4. ∎

Lemma 8. *Let G be a connected graph. If G has at least x degree-1 vertices and the measure of G is at least x, then after iteratively folding degree-1 vertices until the graph has no degree-1 vertex, measure r will be reduced by at least x.*

Proof. Let $V' \neq \emptyset$ be the set of vertices of degree ≥ 2 in the remaining graph after iteratively folding degree-1 vertices (the lemma obviously holds, when $V' = \emptyset$). Assume there are y edges between $V'' = V - V'$ and V'. After removing V'', we get $V' \to y$. We will prove that $V'' \to x - y$. To prove that, we first construct a new graph G' from G by contracting V' into a single vertex v and removing all self-loops incident on it (keeping parallel edges).

Since all the x degree-1 vertices of G are in V'', G' has at least x' degree-1 vertices, where $x' = x + 1$ when v is a degree-1 vertex and $x' = x$ when v is not a degree-1 vertex. Note that the measure of a tree with x' degree-1 vertices is at least $x' - 2$. The measure of G' is also at least $x' - 2$ (G' is a connected graph). We consider the following three cases. **Case** 1: $y = 1$. For this case, v is a degree-1 vertex and $x' = x + 1$. The measure of G' is at least $x' - 2 = x - 1$, and then we will get $V'' \to x - 1$. **Case** 2: $y = 2$. For this case, v is a degree-2 vertex and $x' = x$, and the measure of G' is at least $x - 2$. We will get $V'' \to x - 2$. **Case** 3: $y \geq 3$. For this case, v is a degree-y vertex and $x' = x$. The measure of G' is at least $x - 2$. Excepting $y - 2$ counted from v, there are still $x - 2 - (y - 2) = x - y$ left, which implies $V'' \to x - y$.

Therefore, after removing V'', r will be reduced by at least x. ∎

Corollary 1. *Let G be a graph having no connected path component. If G has any degree-1 vertex, then we can reduce r by at least 1 by iteratively folding degree-1 vertices. If G has exactly 2 degree-1 vertices, then we can reduce r by at least 2 by iteratively folding degree-1 vertices.*

Lemma 9. *Let G be a reduced graph and v a degree-3 vertex in G. Then no degree-0 vertex or component of a 1-path or component of a 2-path is created after removing $N[v]$.*

Proof. If a degree-0 vertex u is created, then G has a 2-3 structure $\{v, u\}$-$N(v)$. If a 1-path ab is created, then there is a 3-3 structure $\{v, a, b\}$-$N(v)$. If a 2-path abc is created, then there is a 3-4 structure $\{a, c, v\}$-$N(v) \cup \{b\}$. ∎

Lemma 10. *Let G be a connected reduced graph of more than 8 vertices and v a degree-3 vertex in G. Then after removing $N[v]$, measure r will be reduced by at least 8. Furthermore, if each 3-cycle in G contains at least one vertex of degree ≥ 4, then after removing $N[v]$, measure r will be reduced by at least 10.*

Proof. There is at most one edge with both endpoints in $N(v)$, otherwise v will dominate a neighbor of it. Therefore, there are at least four edges between $N(v)$ and $N_2(v)$. If $|N_2(v)| \geq 4$, r will be reduced by $4 + 4 = 8$ directly after removing $N[v]$ ($v \to 4$ and $N(v) \to 4$). If $|N_2(v)| \leq 3$, it is impossible to create a component of a l-path ($l \geq 3$) after removing $N[v]$. By Lemma 8 and Corollary 1 and Lemma 9 we know that eventually r will be reduced by at least 8.

Next, we assume that in each 3-cycle in G there is a vertex of degree ≥ 4. We distinguish the following two cases. **Case** 1: All vertices in $N(v)$ are degree-3 vertices. In this case, no pair of vertices in $N(v)$ are adjacent and there are exactly six edges between $N(v)$ and $N_2(v)$, which means at most 3 degree-1 vertices will be created after removing $N[v]$. It is impossible to create a component of a path after removing $N[v]$ (Obviously, no path of length ≥ 4 will be created. Lemma 9 shows no path of length ≤ 2 will be created. If a 3-path is created, then the graph G is an 8-vertex graph). So by Corollary 1, if a component with 1 or 2 degree-1 vertices is created after removing $N[v]$, we can further reduce r by 1 or 2 by further reducing degree-1 vertices in the component. If a component with 3 degree-1 vertices is created, then the component also contains at least 3 degree-3 vertices, otherwise the only possibility of the component is that it has 4 vertices: a degree-3 vertex adjacent to three degree-1 vertices, which also implies a contradiction — the graph G has only 8 vertices. By Lemma 8, we still can further reduce r by least 3. In any case, totally we can reduce r by at least $4 + 6 = 10$. **Case** 2: There is a vertex of degree ≥ 4 in $N(v)$. Then there are at least five edges between $N(v)$ and $N_2(v)$ (note that there is at most one edge with both endpoints in $N(v)$). By Lemma 8 and Lemma 9 we know that r will be reduced by at least $5 + 5 = 10$. ∎

Lemma 11. *Let G be a connected reduced graph of more than 8 vertices and v a vertex of degree ≥ 4 in G. Then after removing $N[v]$, measure r will be reduced by at least 10.*

The detailed proof of this lemma can be found in the full version of this paper. We remove it from this version due to space limited.

Lemma 12. *Let G be a connected reduced graph of more than 8 vertices. If G has a bottle, then algorithm $MIS(G)$ will branch on a bottle with recurrence relation*

$$C(r) \leq 2C(r-8), \tag{1}$$

where $C(r)$ is the worst-case size of the search tree in our algorithm.

Moreover, if each 3-cycle in G contains at least one vertex of degree ≥ 4, then $MIS(G)$ will branch on a bottle with recurrence relation

$$C(r) \leq 2C(r-10). \tag{2}$$

Proof. Let the bottle called by our algorithm be b-a-$\{c, d\}$. Our algorithm will branch by either removing $N[a]$ or $N[b]$. By Lemma 10 and Lemma 11, we get (1) and (2) directly. ∎

Lemma 13. *Let G be a connected bottle-free reduced graph of more than 8 vertices. If G has a 4-cycle, then algorithm $MIS(G)$ will branch on a 4-cycle with recurrence relation*

$$C(r) \leq 2C(r-8). \tag{3}$$

Moreover, if each 3*-cycle or* 4*-cycle in* G *contains at least one vertex of degree* ≥ 4*, then* $MIS(G)$ *will branch on a* 4*-cycle with recurrence relation*

$$C(r) \leq 2C(r-10). \tag{4}$$

The detailed proof of this lemma can be found in the full version of this paper. We remove it from this version due to space limited.

Lemma 14. *Let* G *be a reduced graph that has no bottle or* 4*-cycle. If* G *has a vertex of degree* ≥ 4*, then algorithm* $MIS(G)$ *will branch on a vertex of maximum degree with recurrence relation*

$$C(r) \leq C(r-6) + C(r-14). \tag{5}$$

Proof. Our algorithm will select a vertex v of maximum degree and branch on it by excluding it from the independent set or including it in the independent set. In the former branch, v is removed and r decreases by at least $2+4=6$. In the latter branch, $N[v]$ is removed. Since G has no bottle or 4-cycle, there are at least 8 vertices in $N_2(v)$. Then in this branch, r will be reduced by at least $6+8=14$. Therefore, we get (5). ∎

Lemma 15. *Let* G *be a connected reduced graph of more than* 15 *vertices that has no bottle or* 4*-cycle. If* G *is also a* 3*-regular graph, then algorithm* $MIS(G)$ *can branch with recurrence relation*

$$C(r) \leq C(r-10) + C(r-16) + C(r-20) + C(r-24). \tag{6}$$

Proof. Our algorithm will select a degree-3 vertex and branch on it. Since G is a 3-regular graph that has no 3-cycle or 4-cycle, there are exactly 6 vertices in $N_2(v)$. In the branch where $N[v]$ is removed, 10 degree-3 vertices are reduced. So we can branch with recurrence relation

$$C(r) \leq Q_1(r-4) + Q_2(r-10), \tag{7}$$

where $Q_1 \leq C$ is some function corresponding to the size of the branch where v is removed and $Q_2 \leq C$ some function corresponding to the size of the branch where $N(v)$ is removed. Next, we focus on refining analysis of Q_1 and Q_2.

In the branch where v is removed, 3 nonadjacent degree-2 vertices are created. Our algorithm will fold the three degree-2 vertices in the next step. Let G' be the resulted graph. Then G' has exactly 3 degree-4 vertices (note that the original graph has no 3-cycle or 4-cycle, and then it is impossible to create a degree-3 vertex after folding a degree-2 vertex), and each 3-cycle and 4-cycle in the current graph contains at least one degree-4 vertex. If G' has a bottle or 4-cycle, we can branch with $Q_1(r) \leq 2C(r-10)$ by Lemma 12 and Lemma 13. If G' has no bottle or 4-cycle, we will branch on a degree-4 vertex v'. We further distinguish three different cases. **Case** 1: The other two degree-4 vertices are adjacent to v'. In this case, we have $|N_2(v')| \geq 8$ (the three degree-4 vertices may form a triangle). In the branch where v' is removed, r is reduced by at least 6, and in the branch

where $N[v']$ is removed, r is reduced by at least $8+8=16$. Furthermore, in the branch where v' is removed, two nonadjacent degree-2 vertices are created, and then we can further branch with (5) at least. In total, we get

$$Q_1(r) \leq C(r-6-6)+C(r-6-14)+C(r-16) \quad (8)$$
$$= C(r-12)+C(r-16)+C(r-20). \quad (9)$$

Case 2: There is only one degree-4 vertex adjacent to v'. Since there is no bottle and 4-cycle, we get $|N_2(v')| \geq 9$. In the branch where $N[v']$ is removed, r is reduced by $7+9=16$. In the branch where v' is removed, we also can further branch with (5) at least. Therefore, we get (8). **Case** 3: There is no degree-4 vertex adjacent to v'. We will branch on v' with (5) directly, and in the branch where v' is removed, some other degree-4 vertices are left. We can further branch with (5) at least. Then we get $Q_1(r) \leq C(r-6-6)+C(r-6-14)+C(r-14) = C(r-12)+C(r-14)+C(r-20)$.

In the branch where $N[v]$ is removed, six degree-2 vertices are created. We distinguish the following two cases. **Case 1**: there are some degree-4 vertices created after folding degree-2 vertices. **Case 2**: the graph is a 3-regular graph after folding degree-2 vertices. For Case 1, we can further branch with (5), i.e. $Q_2(r) \leq C(r-6)+C(r-14)$. For Case 2, we only get $Q_2(r) \leq C(r)$. But in the branch where v is removed, we will get a triangle with three vertices being degree-4 vertices after folding degree-2 vertices. Then for this case, we will branch with (8).

Among all the cases the worst one is that we branch with $Q_2(r) \leq C(r)$ and (8). Therefore, we get $C(r) \leq Q_1(r-4)+Q_2(r-10) \leq C(r-4-12)+C(r-4-16)+C(r-4-20)+C(r-10) = C(r-10)+C(r-16)+C(r-20)+C(r-24)$, as claimed in the lemma. ∎

Among all the cases in our algorithm, the worst running time corresponds to recurrence relation (6). The only positive root of function $f(x) = 1-x^{-10}-x^{-16}-x^{-20}-x^{-24}$ is $1.0884\cdots$. As mentioned in the introduction, $C(r) = O(1.0885^r)$ will satisfy (6). Therefore, we get

Theorem 1. *Algorithm $MIS(G)$ can find a maximum independent set in a degree-3 graph in $O^*(1.0885^n)$ time.*

5 Concluding Remarks

In this paper, we have presented a simple $O^*(1.0885^n)$-time algorithm for the maximum independent set problem in degree-3 graphs. This algorithm also implies that we can decide if a graph with degree bounded by 3 has a vertex cover of size at most k in $O^*(1.0885^{2k}) = O^*(1.1849^k)$ time.

Unlike most previous algorithms, our algorithms do not contain many branching rules. We use two new branching techniques, called branching on a bottle and branching on a 4-cycle, to avoid tedious examinations of the local structures. The branching rules catch the structural properties of small cycles in graphs, which

make our algorithms simple and practical. Many previous algorithms can apply these two new branching rules to simplify the description and analysis.

Our algorithm for the maximum independent set problem is analyzed by measuring $r = sum_{v \in V}(d_v)$. The idea comes from the observation that when the graph has no vertex of degree ≥ 3, i.e. $r = 0$, the problem can be solved in polynomial time. We have checked that our algorithm $MIS(G)$ can also be analyzed by measuring parameter $m - n + t$ to get the same running time bound, where m is the number of edges, n the number of vertices, and t the number of tree components in the graph.

References

1. Tarjan, R., Trojanowski, A.: Finding a maximum independent set. SIAM J. on Computing 6(3), 537–546 (1977)
2. Jian, T.: An $O(2^{0.304n})$ algorithm for solving maximum independent set problem. IEEE Transactions on Computers 35(9), 847–851 (1986)
3. Robson, J.: Algorithms for maximum independent sets. J. of Algorithms 7(3), 425–440 (1986)
4. Robson, J.: Finding a maximum independent set in time $O(2^{n/4})$. Technical Report 1251-01, LaBRI, Université Bordeaux I (2001)
5. Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and conquer: a simple $O(2^{0.288n})$ independent set algorithm. In: SODA, pp. 18–25. ACM Press, New York (2006)
6. Beigel, R.: Finding maximum independent sets in sparse and general graphs. In: SODA, pp. 856–857. ACM Press, New York (1999)
7. Chen, J., Kanj, I.A., Xia, G.: Labeled search trees and amortized analysis: Improved upper bounds for NP-hard problems. Algorithmica 43(4), 245–273 (2005)
8. Xiao, M.Y., Chen, J.E., Han, X.L.: Improvement on vertex cover and independent set problems for low-degree graphs. Chinese Journal of Computers 28(2), 153–160 (2005)
9. Bourgeois, N., Escoffier, B., Paschos, V.T.: An $O^*(1.0977^n)$ exact algorithm for max independent set in sparse graphs. In: Grohe, M., Niedermeier, R. (eds.) IWPEC 2008. LNCS, vol. 5018, pp. 55–65. Springer, Heidelberg (2008)
10. Fomin, F.V., Høie, K.: Pathwidth of cubic graphs and exact algorithms. Inf. Process. Lett. 97(5), 191–196 (2006)
11. Fürer, M.: A faster algorithm for finding maximum independent sets in sparse graphs. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 491–501. Springer, Heidelberg (2006)
12. Razgon, I.: A faster solving of the maximum independent set problem for graphs with maximal degree 3. In: Broersma, H., Dantchev, S.S., Johnson, M., Szeider, S. (eds.) ACiD. Texts in Algorithmics, vol. 7, pp. 131–142. King's College, London (2006)
13. Razgon, I.: Faster computation of maximum independent set and parameterized vertex cover for graphs with maximum degree 3. J. of Discrete Algorithms 7(2), 191–212 (2009)
14. Chen, J., Kanj, I., Xia, G.: Simplicity is beauty: Improved upper bounds for vertex cover. Technical Report TR05-008, School of CTI, DePaul University (2005)
15. Bourgeois, N., Escoffier, B., Paschos, V.T., van Rooij, J.M.M.: Fast algorithms for max independent set in graphs of small average degree. CoRR abs/0901.1563 (2009)

Pathwidth and Searching in Parameterized Threshold Graphs

D. Sai Krishna[1], T.V. Thirumala Reddy[1], B. Sai Shashank[2], and C. Pandu Rangan[1]

[1] Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai 600036, India
[2] Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, Guwahati 781039, India
{dsaikris86,tiru114,shashank920045,prangan55}@gmail.com

Abstract. Treewidth and pathwidth are important graph parameters that represent how close the graph is to trees and paths respectively. We calculate treewidth and pathwidth on parameterized chordal and threshold graphs. We define a *chordal* $+ 1v$ graph as a graph that can be made into a chordal graph by removing a vertex. We give polynomial time algorithms for computing the treewidth of a *chordal* $+ 1v$ graph, pathwidth of a *threshold* $+ 1v$ graph and a *threshold* $+ 2e$ graph. The mixed search number of a graph is the minimum number of cops required to capture a single robber, who is hiding in the graph. We apply the algorithm to compute the pathwidth in order to compute the mixed search number of a *threshold* $+ 1v$ graph.

Keywords: Graph searching, treewidth, pathwidth, parameterization, threshold graphs.

1 Introduction

Treewidth is an important graph parameter which measures the tree-like nature of a graph. It is closely related to *chordal* graphs. Similarly, pathwidth measures path-like nature of a graph. Pathwidth is closely related to *interval* graphs. Computing treewidth and pathwidth of a general graph is known to be NP-complete [1]. Dynamic programming techniques on tree decompositions have allowed significant improvements on some graph algorithms like Hamiltonian Circuit, Independent Set, Vertex Cover, etc. Each problem that can be formulated in Monadic Second Order Logic (MSOL) can be solved in linear time on graphs of bounded treewidth. See [3] for a survey. Computing these decompositions and widths effectively for the required class of graphs is the key subproblem in these algorithms [2]. This makes it interesting to solve treewidth and pathwidth of special classes of graphs. Some instances of an NP-complete problem are polynomially solvable and some instances are difficult. Parameterization provides a

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 293–304, 2010.

formal way to study instances that are polynomially solvable and instances that are difficult. See [7] for a complete analysis. A *parameterization* of an alphabet Σ^*, is a mapping $k : \Sigma^* \to \mathbb{N}$ that is a polynomial time computable function. Graphs that become *chordal* by removing some k vertices are called *chordal*$+kv$ graphs. This number k is considered as a *parameterization* of the input graph. Graph theoretic problems on graphs like *split*$+kv$ graphs, which are also called parameterized graph classes have been studied in the past [4,13,16].

Graph searching is a scenario where cops need to capture one robber who hides in the vertices of the graph. Both the robber and the cops can move with infinite speed along the edges of the graph. The robber is invisible to the cops, but the cops are visible to the robber. However, the cops can jump between vertices in the graph and obstruct the movement of the robber. Mixed search number of a graph G, $miams\,(G)$ is the minimum number of cops required to capture this robber hiding in G. It has been studied extensively [12,11,9]. In a computer network setting, the graph searching problem serves as a mathematical model for protecting networks against viruses and other unwanted agents, like spyware or eavesdroppers. A practical example is the problem of finding a successful strategy for a group of collaborating software programs that are designed to clean the network from a virus [6].

In our paper, we prove in section 3 that the polynomial time algorithm for treewidth of *split* $+ 1v$ graphs [15] given by Federico Mancini, can be used for computing treewidth of *chordal* $+ 1v$ graphs. In section 4 we give a polynomial time algorithm which computes pathwidth of *threshold*$+1v$ graphs by modifying the interval ordering of the threshold part of the graph. Threshold graphs have many important mathematical properties and several applications like psychology, scheduling and synchronization of parallel processes [14]. Similarly we give a polynomial time algorithm for pathwidth of *threshold* $+ 2e$ graphs. Finally, we calculate $miams\,(G)$ for *threshold* $+ 1v$ graphs in section 6 by searching for a special kind of path decomposition called *good path decomposition*, which was introduced by Pinar Heggernes and Rodica Mihai [11].

2 Notations and Definitions

A simple graph is a collection of vertices V and edges E represented as $G = (V, E)$, where each edge is an unordered pair of distinct vertices. The set of neighbors of a vertex v is denoted by $N\,(v)$. The *induced subgraph* of G by a vertex set $S \subseteq V$ is $G\,(S) = (S, E')$ where $E' = \{(u, v) \in E | u \in S \text{ and } v \in S \text{ and } u \neq v\}$. A class of graphs $\mathbb{F}$ is called *h ereditary* if $G = (V, E) \in \mathbb{F} \Rightarrow \forall S \subseteq V, G\,(S) \in \mathbb{F}$. Number $\omega\,(G)$ represents maximum clique size of G. A vertex v is called *simplicial* if $N\,(v)$ forms a clique in the graph. A graph $G = (V, E)$ is called *chordal* or *triangulated* if every cycle in G with more than 3 vertices has a chord. A minimum set of edges F to be added to a graph G to make it a chordal graph G' is called a set of *fill edges*. The graph G' is called a *minimum chordal completion* of G. A graph $G = (V, E)$ is called *split* if and only if $V = K \cup I$ such that $K \cap I = \phi$ where K is a clique in G and I is an independent set in G. A graph

$G = (V, E)$ is called *interval* if and only if G is connected and $\exists \Pi, \sigma$ where $\Pi = \{I_1, I_2, \ldots\}$ is a set of intervals and $\sigma : V \rightarrow \Pi$ is a bijection such that $\forall u, v \in V$ *and* $u \neq v \mid (u, v) \in E \Leftrightarrow \sigma(u) \bigcap \sigma(v) \neq \phi$. A minimum set of edges F to be added to a graph G to make its interval is called a *minimum interval completion* of G. A graph $G = (V, E)$ is called a *threshold* graph iff it is a split graph and there exists a split partition $V = K \cup I$ where $K = \{a_1, a_2, \ldots\}$ is a maximum clique and $I = \{b_1, b_2, \ldots\}$ is an independent set such that $N(b_1) \subseteq N(b_2) \subseteq \ldots$ and $N(a_1) \supseteq N(a_2) \supseteq \ldots$

Tree decomposition of a graph $G = (V, E)$ is an ordered pair $D = (X, T)$ where X is a collection of some subsets of V and $T = (I, E_t)$ is a tree with the following properties. Let $X = \{X_1, X_2, ..X_m\}$ be some subsets of V also called *bags* in tree decomposition D and $I = \{1, 2, ...m\}$ vertex set of tree T. $\bigcup_{i \in I} X_i = V$, $\forall (u, v) \in E : \exists i \in I$ such that $u, v \in X_i$ and $\forall v \in V : T(\{i | v \in X_i\})$ is a connected subtree of T. $Width(D) = max_{i=1}^{m}\{|X_i|\} - 1$. $Treewidth$ of $G = (V, E)$ is defined as $tw(G) = min_D \{width(D)\}$. A tree decomposition with width as the treewidth is called an optimal tree decomposition. $Treewidth$ is alternately defined as $min_H\{\omega(H)\} - 1$ where H is a minimal chordal completion of G. Pathwidth also can be defined in a similar way. $Pathwidth$ is alternately defined as $min\{\omega(H)\} - 1$ where H is a minimal interval completion of G.

Proposition 1. *We have the following properties for treewidth and pathwidth.*

1. *Treewidth of a chordal graph G is $\omega(G) - 1$. An optimum tree decomposition of a chordal graph G is called a clique tree if all bags in the decomposition are maximal cliques in G.*
2. *By removing all vertices in any of the internal bags of a clique tree, the graph would be disconnected [10].*
3. *Fill edges are not added to any simplicial vertex in any chordal completion of G [15].*
4. *Pathwidth of an interval graph G is $\omega(G) - 1$. An optimum path decomposition of an interval graph G exists, called a clique path, if all bags in the decomposition are maximal cliques in G [10].*

Let $\mathbb{F}$ be a hereditary graph class. We define $\mathbb{F} + kv$ as a class of graphs obtained by adding at most k vertices to a graph in $\mathbb{F}$, i.e $G \in \mathbb{F} + kv$ iff $\exists V_k$, where $|V_k| = k$ vertices such that $G \setminus V_k \in \mathbb{F}$. V_k is called a *modulator* of G. We can also define the parameterized graph class, $\mathbb{F} + kv$ graphs, in the same way. Let $G = (V, E) \in \mathbb{F} + 1v$ with modulator $V_k = \{x\}$ and let $\omega(G - x) = \alpha$. We use $N(v)$ as neighborhood of v in $G - x$ for a vertex v.

Proposition 2. [15] *We observe that*

1. *If $G \in chordal + 1v$ then $tw(G) \in \{\alpha - 1, \alpha\}$.*
   If $\omega(G) = \alpha + 1$ then $tw(G) = \alpha$.
2. *If $G \in interval + 1v$ then $pw(G) \in \{\alpha - 1, \alpha\}$.*
   If $\omega(G) = \alpha + 1$ then $pw(G) = \alpha$.

3 Treewidth of *Chordal* + 1*v* Graphs

Let $G = (V, E) \in chordal + 1v$ with modulator $V_k = \{x\}$ and let $\omega(G - x) = \alpha$. We use $N(v)$ as the neighborhood of v in $G - x$ for a vertex v. If $\omega(G) = \alpha$, then we construct a simpler graph G' from G. If a vertex v is simplicial and not adjacent to x then we remove v from the graph. We repeat the above process for all simplicial vertices not adjacent to x until no such vertex v exists. We call the resultant graph G'. Observe that degree of all the vertices removed is at-most $\alpha - 1$. In order to calculate the treewidth of G' we analyze the properties of the graph G'. Observe that G' is also a *chordal+1v* graph. Treewidths of G' and G are related by the following lemma $tw(G') < \alpha \iff tw(G) < \alpha$ in [15].

Lemma 1. *Let $D(G' - x) = (\{X_1, X_2 \ldots X_m\}, T)$ be an optimal clique-tree decomposition of $G' - x$ with maximal cliques $\{X_1, X_2 \ldots X_m\}$.*

- *$N_{G'}(x) \bigcap X_i \neq \phi$ for all leaf bags X_i of D.*
- *$G' - X_j$ is connected for all internal bags X_j of D.*

For proof refer extended version[5].

Lemma 2. *$tw(G') = \omega(G' - x) = \alpha'$.*

For proof refer extended version[5].

Theorem 1. *Treewidth for chordal+1v graph $G = (V, E)$ can be computed in polynomial time.*

Proof is similar to [15]. For detailed proof refer extended version[5].

4 Pathwidth of *Threshold* + 1*v* Graphs

Let $G = (V, E) \in threshold{+}1v$ with modulator $V_k = \{x\}$ and let $\omega(G - x) = \alpha$. We use $N(v)$ as the neighborhood of v in $G - x$ for a vertex v. If $\omega(G) = \alpha + 1$ then $pw(G) = \alpha$. We can see that $\{x\}$, α, $\omega(G)$ can be found in $O(|V| \cdot |E|)$ time since all maximal cliques of a chordal graph can be listed in that time. From now on we assume $\omega(G) = \alpha$ otherwise $pw(G) = \alpha$. Consider a split partition $V - x = K \cup I$ where $K = \{a_1, a_2, \ldots\}$ is a maximum clique and $I = \{b_1, b_2, \ldots\}$ such that $N(b_1) \subseteq N(b_2) \subseteq \ldots$ and $N(a_1) \supseteq N(a_2) \supseteq \ldots$. We arrange the intervals of K in $\langle a_1, a_2, \ldots \rangle$ order from bottom to top and left to right in α levels to form a stair case pattern and intervals of I also in $\langle b_1, b_2, \ldots \rangle$ order from bottom to top and left to right in the corresponding level to get an interval ordering σ of $G - x$. We number the bottom level as $level_1$ and topmost level as $level_\alpha$. Observe that σ represents an optimal path decomposition of $G - x$ with width $\alpha - 1$ and can be constructed in $O(|V| + |E|)$. Let n_i be the number of vertices in $level_i$ and m_i be the number of vertices in $level_i$ adjacent to x. We have a_i is the K vertex in $level_i$. For example in Figure 1, 2, we have $\alpha = 6$, $K = \{0, 1, 2, 3, 4, 5\}$ and $I = \{6, 7, 8, 9\}$. We have $n_2 = 2$, $m_2 = 1$, a_1 is vertex 5 and a_2 is vertex 4.

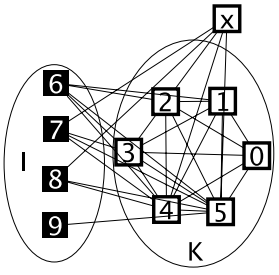


Fig. 1. A *threshold* + $1v$ Graph G

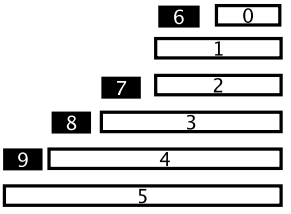


Fig. 2. Interval ordering σ of $G - x$

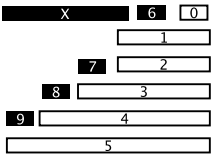


Fig. 3. $m_\alpha = 0$

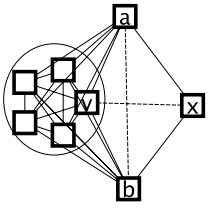


Fig. 4. $m_\alpha \geq 2$

Lemma 3. *If* $m_\alpha = 0$ *then* $pw\,(G) = \alpha - 1$.

Proof. We construct interval ordering of G from σ. We put interval x in $level_\alpha$ to the left of $level_\alpha$ vertices. We make the interval of x intersect with all the intervals except the intervals in $level_\alpha$. We get an interval completion with maximum clique size α since x and the vertices in $level_{(\alpha-1)}$ form a clique of size at most α and no other larger clique is formed. See Figure 3. □

Lemma 4. *If* $m_\alpha \geq 2$ *then* $pw\,(G) = \alpha$.

Proof. Consider any two vertices a and b among m_α vertices adjacent to x in $level_\alpha$. We can see that $N\,(a) = N\,(b)$ and $N\,(a)$ forms a clique of size $\alpha - 1$. For all $y \in N\,(a)$, if $xy \notin E$ then vertices $\{x, a, y, b\}$ form an induced subgraph C_4. At-least one of these cycles exist. If not, x is adjacent to all $N\,(a)$ and a forms a clique of size $\alpha + 1$. To make G interval we need to add ab or $xy, \forall y, xy \notin E$ as chords. In either of the cases a clique of size $\alpha + 1$ is formed. See Figure 4. □

If $m_\alpha = 1$ then without loss of generality we assume a_α to be leftmost interval in $level_\alpha$ vertices in σ.

Lemma 5. *If* $m_\alpha = 1$, $xa_{\alpha-1} \notin E$ *then* $pw\,(G) = \alpha - 1$.

Proof. We construct interval ordering σ' of G from σ. We put interval x in $level_0$ to the left of the $level_\alpha$ vertices. We make the interval of x intersect with all the intervals except the intervals in $level_\alpha$ and $a_{\alpha-1}$. Make the interval of x

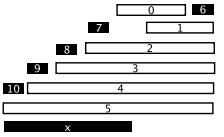


Fig. 5. $xa_{\alpha-1} \notin E$

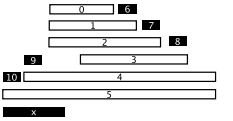


Fig. 6. $xa_{\alpha-i} \notin E$

intersect with a_α in $level_\alpha$ by extending the interval of a_α to the left in σ'. We get an interval completion with maximum clique size α since x and vertices in $level_{(\alpha-1)}$ form a clique of size at most α and no other larger clique is formed. See Figure 5. □

If $m_\alpha = 1$, $xa_{\alpha-1} \in E$ then we proceed incrementally checking next levels.

Lemma 6. *If* $m_\alpha = 1$, $xa_{\alpha-1} \in E$ *and* $m_{\alpha-1} \geq 2$ *then* $pw(G) = \alpha$.

Proof. Since $xa_{\alpha-1} \in E$ and $m_{\alpha-1} \geq 2$, x is adjacent to at-least one I vertex in $level_{(\alpha-1)}$ say $z_{\alpha-1}$. $N(z_{\alpha-1}) \subset N(a_{\alpha-1})$ and $|N(z_{\alpha-1})| = \alpha - 2$. Assume for the sake of contradiction $pw(G) = \alpha - 1$ and let H be a corresponding interval completion. In H, if $z_{\alpha-1}$ and $a_{\alpha-1}$ are adjacent then consider adding $z_{\alpha-1}a_{\alpha-1}$ edge to G to get G'. We get interval representation of G' by putting $z_{\alpha-1}$ in $level_\alpha$ above $a_{\alpha-1}$. Therefore $m_\alpha = 2$ in G' and so by Lemma 4, $pw(G') = \alpha = pw(G')$, giving rise to a contradiction. Hence $z_{\alpha-1}$ and $a_{\alpha-1}$ are not intersecting in any optimal interval completion of G. Therefore the intervals of $N(z_{\alpha-1}) \cup \{x\}$ should come in between intervals of $z_{\alpha-1}$ and $a_{\alpha-1}$ forming a clique $\{x\} \cup K$ of size $\alpha + 1$. □

Lemma 7. *If* $m_{\alpha-j} = 1$ *for all* $0 \leq j \leq i-1$, $xa_{\alpha-i} \notin E$ *then* $pw(G) = \alpha - 1$.

Proof. We construct interval ordering σ' of G from σ. Without loss of generality we assume $a_{\alpha-j}$ to be leftmost interval in $level_{(\alpha-j)}$ vertices in σ, $0 \leq j \leq i-1$. We make the interval of x intersect with all the intervals except the intervals in $level_\alpha$ to $level_{\alpha-i}$. For $0 \leq j \leq i-1$, we make the interval of x intersect with $a_{\alpha-j}$ in $level_{(\alpha-j)}$ by extending the intervals of $a_{\alpha-j}$ to the left. We get an interval completion with maximum clique size α since x and the vertices in $level_{(\alpha-i)}$ forms a clique of size at most α and no other larger clique is formed. See figure 6. □

Lemma 8. *If* $m_{\alpha-j} = 1$ *for all* $0 \leq j \leq i-1$, $xa_{\alpha-i} \in E$ *and* $m_{\alpha-i} \geq 2$ *then* $pw(G) = \alpha - 1$.

Proof. We prove by induction on i. For $i = 1$ th claim follows by Lemma 6. We assume the claim to be true for all $0 \leq j \leq i-1$. Since $xa_{\alpha-i} \in E$ and $m_{\alpha-i} \geq 2$, x is adjacent to at-least one I vertex in $level_{(\alpha-i)}$ say $z_{\alpha-i}$. $N(z_{\alpha-i}) \subset N(a_{\alpha-i})$ and $|N(z_{\alpha-i})| = \alpha - i - 1$. Assume for the sake of contradiction $pw(G) = \alpha - 1$ and let H be a corresponding interval completion. In H if $z_{\alpha-i}$, $a_{\alpha-i}$ are adjacent then consider adding $z_{\alpha-i}a_{\alpha-i}$ edge to G to get G'. We get the interval

Table 1. A sketch of the algorithm for pathwidth of $threshold + 1v$ graph

Condition										$pw(G)$	Lemma
$m_\alpha = 0$										$\alpha - 1$	3
$m_\alpha \geq 2$										α	4
$m_\alpha = 1$	$xa_{\alpha-1} \notin E$									$\alpha - 1$	5
	$xa_{\alpha-1} \in E$	$m_{\alpha-1} \geq 2$								α	6
		$m_{\alpha-1} = 1$	$xa_{\alpha-2} \notin E$							$\alpha - 1$	7
			$xa_{\alpha-2} \in E$	$m_{\alpha-2} \geq 2$						α	8
				$\ddots$						$\vdots$	7, 8
								$xa_1 \notin E$		$\alpha - 1$	7
Now, x is adjacent to all K vertices, $\omega(G) = \alpha + 1$								$xa_1 \in E$		α	

representation of G' by putting $z_{\alpha-i}$ in level $level_{\alpha-i-1}$ above $a_{\alpha-i}$ and therefore $m_{\alpha-i-1} = 2$ in G'. So, by the induction hypothesis $pw(G') = \alpha = pw(G')$ giving rise to a contradiction. Hence $z_{\alpha-i}$ and $a_{\alpha-i}$ are not intersecting in any optimal interval completion of G. Therefore the intervals of $N(z_{\alpha-i}) \cup \{x\}$ should come in between the intervals of $z_{\alpha-i}$ and $a_{\alpha-i}$ forming a clique $\{x\} \cup K$ of size $\alpha + 1$, which is a contradiction. □

Now, we give our algorithm for pathwidth of $threshold + 1v$ graphs. We first check the value of m_α. If it is 0 we return $\alpha - 1$ by Lemma 3. If $m_\alpha \geq 2$ we return α by Lemma 4. If $m_\alpha = 1$ then we further split into two cases $xa_{\alpha-1} \notin E$ and $xa_{\alpha-1} \in E$ and proceed to lower levels. A sketch of the algorithm is presented in table 1. We take the meaning of row 6 as we report the pathwidth as α if $m_\alpha = 1$, $xa_{\alpha-1} \in E$, $m_{\alpha-1} = 1$, $xa_{\alpha-1} \in E$ and $m_{\alpha-2} \geq 2$ holds, likewise for all the other rows.

Theorem 2. *Pathwidth for threshold+1v graph $G = (V, E)$ can be solved in quadratic time.*

Proof. By the above lemmas we can verify correctness of algorithm presented in Table 1. The interval ordering σ of $G - x$ can be constructed in $O(|V| + |E|)$ time. Values m_i and n_i for $1 \leq i \leq \alpha$ can be calculated in $O(|V| \cdot |E|)$ time. There is a loop which runs for each level, checking conditions in constant time. We have $\alpha \leq |V|$ the number of levels. Hence we can check all conditions in $O(|V| \cdot |E|)$ time □

5 Pathwidth of $Threshold + 2e$ Graphs

Let $G = (V, E) \in threshold + 2e$ with modulator $E_k = \{e_1, e_2\}$ and let $\omega(G \setminus E_k) = \alpha$. If both edges $\{e_1, e_2\}$ have a common vertex then this graph can be treated as $threshold + 1v$ graph and we can find its pathwidth as given in section 3.

Theorem 3. *Pathwidth for threshold+2e graph $G = (V, E)$ can be calculated in $O(|E|^3)$ time.*

For proof refer extended version [5].

6 Mixed Search Number of Threshold+1v Graphs

Graph searching is a scenario where cops need to capture one robber who hides in the vertices of the graph. The robber and the cops can move with infinite speed along the edges of the graph, the robber is invisible to the cops, but the cops are visible to the robber. However the cops can jump between vertices in the graph and obstruct the movement of the robber. A robber is said to be captured if he is not allowed to move and his location is known to the cops. The mixed search number of a graph G, $miams\,(G)$, is the minimum number of cops required to capture this robber hiding in G. Since the robber is invisible, the strategy of the cops does not depend on the position of the robber. Therefore, whatever may be the strategy of the robber, the cops strategy must be the same. Hence cops strategy is a *pre-determined* strategy. If the cops ensure that the robber is not hiding in a vertex v then they need to protect the vertex v, called a cleared vertex, from the robber re-entering it. If the robber can enter the vertex v then v is said to be a *re-contaminated* vertex.

One of the pre-determined strategies is to corner the robber to a vertex and capture him. Such a strategy is called a *monotone strategy*. In such a strategy the cops must be careful about recontamination so they may use some cops to avoid recontamination. In [12], Andrea and LaPaugh showed that avoiding recontamination will not cost cops, i.e for any graph there is monotone search strategy with optimum number of cops where cops restrict the space where the robber can move to smaller and smaller parts and finally capture the robber. This problem was considered on interval graphs, split graphs and on permutation graphs in [8,11]. Mixed search number is closely related to path decompositions. One can view a search strategy as placing cops on vertices of a bag in a path decomposition and proceeding to next bag upto the last bag to monotonically capture the robber. Not all path decompositions will give a valid search strategy and only some of those valid ones will use optimum number of cops. These kinds of path decompositions are characterized by Pinar Heggernes in [11] and are called good path decompositions.

Definition 1. [15] *A path decomposition $P_G = \langle Y_1, Y_2, \ldots Y_m \rangle$ of a graph $G = (V, E)$ with width $pw\,(G)$ and no redundant bags is called a good path decomposition if $|Y_{i-1} \cap Y_i \cap Y_{i+1}| < pw\,(G)$ and $|Y_{i-1}| = |Y_i| = |Y_{i+1}| = pw\,(G) + 1$, for all $1 \leq i \leq m-1$. If $|Y_{i-1} \cap Y_i \cap Y_{i+1}| = pw\,(G)$ then they are called conflicting bags. If P_G has redundant bags then the path decomposition obtained from P_G by removing all redundant bags should be good.*

Proposition 3. [15] *Let G be an arbitrary undirected simple graph then*

1. *$miams\,(G) \in \{pw\,(G), pw\,(G) + 1\}$.*
2. *Monotone invisible active mixed search number, $miams\,(G) = pw\,(G)$ if and only if there exists a good path decomposition for G.*

We compute mixed search number of $threshold + 1v$ graphs by constructing a good path decomposition (if it exists) where no three bags conflict. Let

$G = (V, E) \in threshold + 1v$ with modulator $V_k = \{x\}$ and let $\omega(G - x) = \alpha$, We use $N(v)$ as neighborhood of v in $G - x$ for a vertex v. We have $miams(G) \in \{\alpha - 1, \alpha, \alpha + 1\}$.

Lemma 9. *If* $n_\alpha \geq 3$ *then* $miams(G - x) = \alpha$ *otherwise* $\alpha - 1$.

For proof refer extended version[5].

Lemma 10. *If* $m_\alpha \geq 3$ *then* $miams(G) = \alpha + 1$.

For proof refer extended version[5].

Lemma 11. *If* $m_\alpha = 2$ *then* $miams(G) = \alpha$.

For proof refer extended version[5].

If $m_\alpha = 1$ then without the loss of generality let a_α be the vertex in $level_\alpha$, which is adjacent to x, and a_α to be the leftmost interval among $level_\alpha$ vertices.

Lemma 12. *If* $m_\alpha = 1$ *and* $pw(G) = \alpha$ *then* $miams(G) = \alpha$.

For proof refer extended version[5].

If $m_\alpha = 1$ and $pw(G) = \alpha - 1$ then we have stopped in some $level_{\alpha-i}$, $i \neq \alpha$ such that $xa_{\alpha-i} \notin E$ during the calculation of $pw(G)$. We call it i_{pw}. If $i_{pw} \neq 1$. Then we construct a path decomposition where the interval of x is made adjacent to all K vertices and all the vertices from $level_0$ upto $level_{\alpha-i_{pw}}$ except the interval of $a_{\alpha-i_{pw}}$ giving a good path decomposition since the newly formed α bag cannot conflict with the bag at the left or the right. If $i_{pw} = 1$, then by the following lemma we calculate the mixed search number.

Lemma 13. *Let* $m_\alpha = 1$, $pw(G) = \alpha - 1$ *and* $i_{pw} = 1$. *If* $m_{\alpha-1} \geq 3$ *then* $miams(G) = \alpha$, *otherwise,* $miams(G) = \alpha - 1$.

Proof. If $m_{\alpha-1} < 3$ then the path decomposition that we get is good where the interval of x is made to intersect with the intervals of all K vertices and all the vertices from $level_0$ upto $level_{\alpha-1}$ except the interval of $a_{\alpha-1}$. Otherwise, consider any three vertices a, b and c among $m_{\alpha-1}$ vertices adjacent to x in $level_{\alpha-1}$. We can see that $N(a) = N(b) = N(c)$ and $N(a)$ forms a clique of size $\alpha - 2$. Let σ' be an interval completion of G which represents a good path decomposition with width $\alpha - 1$ and let I_a, I_b, I_c be intervals of a, b and c in it. In σ' we have three cases

Case A: I_a, I_b and I_c intersect then the intervals of $N(a)$ should also intersect with them forming a clique of size $\alpha + 1$ which contradicts the fact that σ' has width $\alpha - 1$.

Case B: I_a, I_b intersect and I_a and I_c do not intersect then $N(a) \cup \{x\}$ should come in the middle of I_a and I_b. Therefore $N(a) \cup \{x, a, b\}$ forms a clique of size $\alpha + 1$ which contradicts the fact that σ' has width $\alpha - 1$.

Case C: I_a, I_b and I_c do not intersect then intervals of $N(a) \cup \{x\}$ should intersect with I_a, I_b and I_c forming three cliques of size α, which are conflicting. □

Lemma 14. *Let $m_\alpha = 0$ and $n_\alpha = 1$. Now, if $m_{\alpha-1} \geq 3$ then $miams\,(G) = \alpha$ otherwise, $miams\,(G) = \alpha - 1$.*

For proof refer extended version[5].

Lemma 15. *If $m_\alpha = 0$ and $n_\alpha \geq 2$ then $miams\,(G) = \alpha$.*

For proof refer extended version[5].

Lemma 16. *Let $m_\alpha = 0$, $n_\alpha = 2$ and $xa_{\alpha-1} \notin E$. If $m_{\alpha-1} \geq 3$ then $miams\,(G) = \alpha$ otherwise $miams\,(G) = \alpha - 1$.*

Proof. If $m_{\alpha-1} < 3$, then the path decomposition that we get is good where the interval of x is made to intersect with the intervals of all the vertices from $level_0$ upto $level_{\alpha-1}$ except the interval of $a_{\alpha-1}$. Otherwise, consider any three vertices a, b and c among $m_{\alpha-1}$ vertices adjacent to x in $level_{\alpha-1}$. By a proof similar to Lemma 13 $miams\,(G) = \alpha$. □

From now on we assume $m_\alpha = 0$, $n_\alpha = 2$, $xa_{\alpha-1} \in E$ and $m_{\alpha-1} < 3$. If $m_{\alpha-1} \geq 3$ then $miams\,(G) = \alpha$ by a proof similar to Lemma 16.

Lemma 17. *Let G' be the graph formed by adding an edge between two $level_{\alpha-1}$ vertices a and b, which are adjacent to x. Now, if $m_{\alpha-1} = 2$ then $miams\,(G) = miams\,(G')$.*

For proof refer extended version[5].

If $m_{\alpha-1} = 1$ then we go to next lower levels $level_{\alpha-i}$ and check for $xa_{\alpha-i} \in E$ and $m_{\alpha-i} \geq 2$.

Lemma 18. *If $m_\alpha = 0$, $m_{\alpha-j} = 1$ for all $0 < j \leq i-1$, $xa_{\alpha-i} \notin E$ then $miams\,(G) = \alpha - 1$.*

Proof. The proof is similar to the proof of Lemma 16 where $m_{\alpha-1} < 3$. □

Lemma 19. *Let G' be the graph formed by adding an edge between two $level_{\alpha-i}$ vertices vertices a and b, which are adjacent to x. Now, if $m_\alpha = 0$, $m_{\alpha-j} = 1$ for all $0 < j \leq i-1$, $xa_{\alpha-i} \in E$, $m_{\alpha-i} \geq 2$ then $miams\,(G) = miams\,(G')$.*

Proof. The proof is similar to the proof of Lemma 17. □

Now, to calculate the mixed search number of $threshold + 1v$ graphs we first check the value of m_α. If $m_\alpha \geq 3$ we return $\alpha + 1$ (by Lemma 10). If $m_\alpha = 2$ we return α (by Lemma 11) and so on. A sketch of the algorithm is presented in table 2. We take the meaning of row 6 as the algorithm reports $miams\,(G)$ as $\alpha - 1$ if $m_\alpha = 1$, $pw\,(G) = \alpha - 1$, $i_{pw} = 1$ and $m_{\alpha-1} < 3$ holds, likewise for all the other rows.

Theorem 4. *Mixed search number for a threshold+1v graph $G = (V, E)$ can be calculated in $O(|V| \cdot |E|)$ time.*

Table 2. A sketch of the algorithm for $miams\,(G)$ of $threshold + 1v$ graph G

Condition						$miams\,(G)$	Lemma
$m_\alpha \geq 3$						$\alpha + 1$	10
$m_\alpha = 2$						α	11
$m_\alpha = 1$	$pw\,(G) = \alpha$					α	12
	$pw\,(G) = \alpha - 1$	$i_{pw} \neq 1$				$\alpha - 1$	
		$i_{pw} = 1$	$m_{\alpha-1} \geq 3$			α	13
			$m_{\alpha-1} < 3$			$\alpha - 1$	13
Loop							
$m_\alpha = 0$	$n_\alpha = 1$	$m_{\alpha-1} \geq 3$				α	14
$m_\alpha = 0$	$n_\alpha = 1$	$m_{\alpha-1} \leq 2$				$\alpha - 1$	14
	$n_\alpha \geq 3$					α	15
	$n_\alpha = 2$	$xa_{\alpha-1} \notin E$	$m_{\alpha-1} \geq 3$			α	16
			$m_{\alpha-1} < 3$			$\alpha - 1$	16
		$xa_{\alpha-1} \in E$	$m_{\alpha-1} \geq 3$			α	16
			$m_{\alpha-1} = 2$			$miams\,(G')$	17
			$m_{\alpha-1} = 1$	$xa_{\alpha-2} \notin E$		$\alpha - 1$	18
				$\ddots$		$\vdots$	18, 19
Now, x is adjacent to all K vertices, Contradiction					$xa_1 \in E$	α	

Proof. By above lemmas and table 2 we can verify the correctness of our algorithm. The interval ordering σ of $G - x$ can be constructed in $O(|V| + |E|)$ time. Values m_i, n_i $i \leq i \leq \alpha$, $pw\,(G)$ and i_{pw} can be calculated in $O(|V| \cdot |E|)$ time. There is a loop which runs for each level checking conditions in constant time. We have $\alpha \leq |V|$ number of levels hence we can check all conditions in $O(|V| \cdot |E|)$ time. Observe that if for any instance of the problem, we had to call $miams\,(G')$ recursively for tha first time at $level_{\alpha-i}$. Then we know it will be called recursively upto $level_{\alpha-1}$ which will return α always. Hence $miams\,(G')$ call can be replaced by reporting α in our algorithm. □

7 Conclusion

In this paper we have shown that the treewidth of $chordal + kv$ graphs is polynomial for $k = 1$, although it may seem hard for higher values of k. Also, it interests one to examine the treewidth of $split + kv$ graphs for $k \geq 2$. We have also shown thet the pathwidth of a $threshold + 1v$ graph and a $threshold + 2e$ graph can be solved in polynomial time. It would be interesting to see if FPT algorithms exists for $threshold + kv$ and $threshold + ke$ graphs. Computing the pathwidth and the mixed search number of an $interval + kv$ graph is also interesting because the class of threshold graphs is a proper subset of the class of interval graphs and also because on the class of interval graphs these problems are polynomial.

References

1. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM J. Algebraic Discrete Methods 8(2), 277–284 (1987)
2. Bodlaender, H.L.: A tourist guide through treewidth. Acta Cybernetica 11, 1–21 (1993)
3. Bodlaender, H.L.: Treewidth: Characterizations, applications, and computations. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 1–14. Springer, Heidelberg (2006)
4. Cai, L.: Parameterized complexity of vertex colouring. Discrete Appl. Math. 127(3), 415–429 (2003)
5. Sai Krishna, D., Thirumala Reddy, T.V., Sai Shashank, B., Pandu Rangan, C.: Pathwidth and searching in parameterized threshold graphs (to appear in LNCS) (2010), http://www.cse.iitm.ac.in/~dsaikris/Site/Research_files/psptg_full.pdf
6. Flocchini, P., Huang, M.J., Luccio, F.L.: Contiguous search in the hypercube for capturing an intruder. IPDPA 01, 62 (2005)
7. Flum, J., Grohe, M.: Parameterized Complexity Theory, 1st edn. Texts in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2006)
8. Fomin, F.V., Heggernes, P., Mihai, R.: Mixed search number and linear-width of interval and split graphs. In: Brandstädt, A., Kratsch, D., Müller, H. (eds.) WG 2007. LNCS, vol. 4769, pp. 304–315. Springer, Heidelberg (2007)
9. Fomin, F.V., Thilikos, D.M.: An annotated bibliography on guaranteed graph searching. Theor. Comput. Sci. 399(3), 236–245 (2008)
10. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Annals of Discrete Mathematics, vol. 57. North-Holland Publishing Co., Amsterdam (2004)
11. Heggernes, P., Mihai, R.: Mixed search number of permutation graphs. In: Preparata, F.P., Wu, X., Yin, J. (eds.) FAW 2008. LNCS, vol. 5059, pp. 196–207. Springer, Heidelberg (2008)
12. LaPaugh, A.S.: Recontamination does not help to search a graph. J. ACM 40(2), 224–245 (1993)
13. Lozin, V.V., Milanic, M.: Tree-width and optimization in bounded degree graphs. In: Graph-Theoretic Concepts in Computer Science, 32nd International Workshop, WG, Bergen, Norway, June 22-24, Revised Papers, pp. 45–54 (2007)
14. Mahadev, N.V.R., Peled, U.N.: Threshold graphs and related topics. Annals of Discrete Mathematics, vol. 56. Elsevier Science Publishers B.V., North Holland, Amsterdam (1995)
15. Mancini, F.: Minimum fill-in and treewidth of split+ke and split+kv graphs. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 881–892. Springer, Heidelberg (2007)
16. Marx, D.: Parameterized coloring problems on chordal graphs. Theor. Comput. Sci. 351(3), 407–424 (2006)

Author Index