Lecture Notes in Computer Science 2880
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Hans L. Bodlaender (Ed.)

Graph-Theoretic Concepts in Computer Science

29th International Workshop, WG 2003
Elspeet, The Netherlands, June 19-21, 2003
Revised Papers

Springer

Volume Editor

Hans L. Bodlaender
Utrecht University
Institute of Information and Computing Sciences
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
E-mail: hansb@cs.uu.nl

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

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

CR Subject Classification (1998): F.2, G.2, G.1.6, G.1.2, E.1, I.3.5

ISSN 0302-9743
ISBN 3-540-20452-0 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
is a part of Springer Science+Business Media GmbH

http://www.springeronline.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP Berlin GmbH
Printed on acid-free paper SPIN: 10966402 06/3142 5 4 3 2 1 0

Preface

The 29th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2003) was held in the Mennorode conference Center in Elspeet, The Netherlands. The workshop was organized by the Center for Algorithmic Systems of the Institute of Information and Computing Sciences of Utrecht University. The workshop took place June 19–21, 2003. The 72 participants of WG 2003 came from universities and research institutes from 18 different countries and five different continents.

The workshop looks back at a long tradition. It was first held in 1975, and has been held 20 times in Germany, twice in Austria, and once in Italy, Slovakia, Switzerland, and the Czech Republic, and has now been held for the third time in The Netherlands. The workshop aims at uniting theory and practice by demonstrating how graph-theoretic concepts can be applied to various areas in computer science, or by extracting new problems from applications. It is devoted to the theoretical and practical aspects of graph concepts in computer science. The goal is to present recent research results and to identify and explore directions of future research. The talks given at the workshop showed how recent research results from algorithmic graph theory can be used in computer science and which graph-theoretic questions arise from new developments in computer science.

A total of 78 papers were submitted to the workshop. The program committee represented a wide scientific spectrum. In a careful reviewing process, with four reports per submission, the program committee selected 30 papers for presentation at the workshop. A number of very good submissions had to be rejected due to a lack of space in the program schedule. The referees' comments and the numerous fruitful discussions during the workshop were taken into account by the authors of these conference proceedings. In addition, there were two fascinating invited talks by A. Schrijver and M. Fellows. The social program of the workshop included a visit to the National Park 'De Hoge Veluwe,' with a visit to the Kröller-Müller museum.

With much pleasure, I thank all those who contributed to the success of WG 2003: the authors who submitted their work to the workshop, the speakers, the program committee members, and all referees. Thanks are also due to Jan van Leeuwen, for support and advice, to the members of the administrative staff of the Institute of Information and Computing Sciences of Utrecht University for their help, to magician Willem Tel who amazed the participants during the conference dinner, and to the helpful personnel of the Mennorode conference Center . Last, but certainly not least, special thanks go to the organizational committee, Gerard Tel, Marinus Veldhorst, and Thomas Wolle, for their many efforts.

Utrecht, August 2003 Hans Bodlaender

The Tradition of WG

1975 U. Pape – Berlin
1976 H. Noltemeier – Göttingen
1977 J. Mühlbacher – Linz
1978 M. Nagl, H.J. Schneider – Schloß Feuerstein, near Erlangen
1979 U. Pape – Berlin
1980 H. Noltemeier – Bad Honnef
1981 J. Mühlbacher – Linz
1982 H.J. Schneider, H. Güttler – Neunkirchen, near Erlangen
1983 M. Nagl, J. Perl – Haus Ohrbeck, near Osnabrück
1984 U. Pape – Berlin
1985 H. Noltemeier – Schloß Schwanenberg, near Würzburg
1986 G. Tinhofer, G. Schmidt – Stift Bernreid, near München
1987 H. Göttler, H.J. Schneider – Schloß Banz, near Bamberg
1988 J. van Leeuwen – Amsterdam
1989 M. Nagl – Schloß Rolduc, near Aachen
1990 R.H. Möhring – Johannesstift Berlin
1991 G. Schmidt, R. Berghammer – Richterheim Fishbackau, München
1992 E.W. Mayr – Wilhelm-Kempf-Haus, Wiesbaden-Naurod
1993 J. van Leeuwen – Sports Center Papendal, near Utrecht
1994 G. Tinhofer, E.W. Mayr, G. Schmidt – Herrsching, near München
1995 M. Nagl – Haus Eich, Aachen
1996 G. Ausiello, A. Marchetti-Spaccamela – Cadenabbia
1997 R.H. Möhring – Bildungszentrum am Müggelsee, Berlin
1998 J. Hromkovič, O. Sýkora – Castle Smolenice, near Bratislava
1999 P. Widmayer – Centro Stefano Franscini, Monte Verità, Ascona
2000 D. Wagner – Waldhaus Jacob, Konstanz
2001 A. Brandstädt – Boltenhagen, near Rostock
2002 L. Kučera – Český Krumlov
2003 H.L. Bodlaender – Mennorode, Elspeet, near Utrecht

Program Committee

Hans Bodlaender	Utrecht University, The Netherlands, Chair
Andreas Brandstädt	University of Rostock, Germany
Fedor Fomin	University of Bergen, Norway
Michel Habib	LIRMM Montpellier, France
Juraj Hromkovič	RWTH Aachen Germany
Luděk Kučera	Charles University, Prague, Czech Republic
Alberto Marchetti-Spaccamela	Università di Roma, Italy
Ernst Mayr	TU München, Germany
Rolf Möhring	TU Berlin, Germany
Manfred Nagl	RWTH Aachen, Germany
Takao Nishizeki	Tohoku University, Japan
Hartmut Noltemeier	Universität Würzburg, Germany
David Peleg	Weizmann Institute, Rehovot, Israel
Ondrej Sýkora	Loughborough University, UK
Gottfried Tinhofer	TU München, Germany
Dorothea Wagner	Universität Karlsruhe, Germany
Peter Widmayer	ETH Zürich, Switzerland
Christos Zaroliagis	CTI Patras, Greece

Additional Reviewers

Luzi Anderegg, Thomas Bayer, Jean R.S. Blair, Hans-Joachim Böckenhauer, Dirk Bongartz, Paul Bonsma, Hajo J. Broersma, Arno Buchner, Bogdan Chlebus, Sabine Cornelsen, Bruno Courcelle, Ingo Demgensky, Jörg Derungs, Miriam Di Ianni, Krzysztof Diks, Stefan Dobrev, Frederic Dorn, Feodor F. Dragan, Markus Eiglsperger, Jens Ernst, Jiri Fiala, Michele Flammini, Dimitris Fotakis, Leszek Gąsieniec, Michael Gatto, Stefanie Gerke, Petr Golovach, Jan Griebsch, Yubao Guo, MohammadTaghi Hajiaghayi, Pinar Heggernes, Fabian Hennecke, Klaus Holzapfel, Lefteris Kirousis, Ton Kloks, Rastislav Královič, Jan Kratochvil, Dieter Kratsch, Hans-Jörg Kreowski, Joachim Kupke, Van Bang Le, Moritz Maaß, Maria Flavia Mammana, Fredrik Manne, J'an Maňuch, Csaba Megyeri, Kazuyuki Miura, Takaaki Mizuki, Michal Mnuk, Shin-ichi Nakano, Alfredo Navarra, Rolf Niedermeier, Marc Nunkesser, Vicky Papadopoulou, Christophe Paul, Leon Peeters, Stefan Pfingstl, Petrica Pop, Md. Saidur Rahman, Udi Rotics, Thomas Schank, Ingo Schiermeyer, Konrad Schlude, Frank Schulz, Thomas Seidl, Otto Spaniol, Jerry Spinrad, Ladislav Stacho, Yannis C. Stamatiou, Daniel Stefankovic, Lorna Stewart, Gabor Szabo, Hanjo Täubig, Gerard Tel, Jan Arne Telle, Wolfgang Thomas, Ioan Todinca, Peter Ullrich, Walter Unger, Ugo Vaccaro, Margit Voigt, Imrich Vrto, Birgitta Weber, Bernhard Westfechtel, Thomas Willhalm, Hans-Christoph Wirth, Gerhard Woeginger, Alexander Wolff, Xiao Zhou.

Table of Contents

Invited Lecture

Regular Papers

Erratum

Blow-Ups, Win/Win's, and Crown Rules: Some New Directions in *FPT*

Michael R. Fellows

School of Electrical Engineering and Computer Science
University of Newcastle, University Drive, Callaghan NSW 2308, Australia
mfellows@cs.newcastle.edu.au

Abstract. This survey reviews the basic notions of parameterized complexity, and describes some new approaches to designing *FPT* algorithms and problem reductions for graph problems.

1 Introduction – The Basic Ideas

The collection of methods for classifying problems as fixed-parameter tractable, for designing *FPT* algorithms, for designing *better* FPT algorithms and transfering these results to practical implementations, and for describing *FPT* problem transformations to prove lower bound and intractability results, has developed with surprising vigor — yet it still seems we are far from having the "basic vocabulary" of effective FPT algorithm design techniques worked out.

Most parameterized problems in *FPT* have seen a trajectory of improvements through many approaches. An example is the following parameterization of the Max Leaf Spanning Tree problem, defined by declaring: (1) the input to the problem, (2) the parameter, and (3) the question.

Max Leaf Spanning Tree I
Input: A graph G and a positive integer k; **Parameter:** k; **Question:** Does G have a spanning tree with at least k leaves?

The parameter can be *anything*. A different parameterization is:

Max Leaf Spanning Tree II
Input: A graph G and a positive integer r; **Parameter:** The treewidth of G; **Question:** Does G have a spanning tree with at least r leaves?

The parameter doesn't have to be small to be interesting:

Max Leaf Spanning Tree III
Input: A graph G and a positive integer r; **Parameter:** The number of vertices of G; **Question:** Does G have a spanning tree with at least r leaves?

The parameter may be geared to any kind of mathematical insight:

Max Leaf Spanning Tree IV
Input: A graph G and a positive integer k; **Parameter:** k; **Output:** A spanning tree for G having a number of leaves that is within a factor of $(1 + 1/k)$ of the maximum number possible.

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 1–12, 2003.

Parameterizing on the number of leaves can be done in more than one way:

MAX LEAF SPANNING TREE V
Input: A graph G on n vertices and a positive integer k; **Parameter:** k; **Question:** Does G have a spanning tree with at least $n - k$ leaves?

The trajectory for MAX LEAF SPANNING TREE I includes:
• A quadratic *FPT* algorithm based on the Graph Minor Theorem [FL92] having a ridiculously fast-growing parameter function.
• The first linear time *FPT* algorithm for the problem due to Bodlaender had a parameter function of $f(k) = (17k^4)!$ or thereabouts [Bod93]. This was based on depth-first search and bounded treewidth.
• A linear time *FPT* algorithm by Downey-Fellows with $f(k) = (2k)^{4k}$, based on a quadratic (in k) problem kernelization [DF95a].
• An improved linear time *FPT* algorithm due to Fellows, McCartin, Rosamond and Stege [FMRS00] with parameter function $f(k) = (14.23)^k$, based on linear kernelization and catalytic branching.
• The current best *FPT* algorithm for the problem, due to Bonsma, Brüggemann and Woeginger, running in time $O(n^3 + 9.4815^k k^3)$, based on a reduction to a problem kernel of size bounded by $4k$, using a substantial result from extremal graph theory [BBW03].

The parameterized complexity framework is a two-dimensional framework for complexity analysis, where the first dimension is the overall imput size n. The parameter k represents some other aspect(s) of the computational problem.

Definition 1. *A parameterized language L is a subset $L \subseteq \Sigma^* \times \Sigma^*$. For $(x, k) \in L$ we refer to k as the parameter.*

Definition 2. *A parameterized language L is fixed-parameter tractable (FPT) if it can be determined in time $f(k)q(n)$ whether $(x, k) \in L$, where $n = |(x, k)|$, $q(n)$ is a polynomial in n, and f is a computable function (otherwise unrestricted).*

These central ideas recently have been reintroduced by Impagliazzo, Paturi and Zane [IPZ98,Woe03], who propose a new notation for *FPT* results. If a parameterized problem is in *FPT* via an algorithm with a running time of $f(k)q(n)$, they write this as $O^*(f(k))$, focusing attention on the exponential complexity contribution of the parameter.

Definition 3. *A parameterized language L is many:1 parametrically reducible to a parameterized language L' if there is an FPT algorithm that transforms (x, k) into (x', k') so that: (1) $(x, k) \in L$ if and only if $(x', k') \in L'$, and (2) $k' = g(k)$ (for some recursive function g).*

The main parameterized classes are organized in the hierarchy:

$$FPT \subseteq M[1] \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[P] \subseteq AW[P] \subseteq XP$$

XP is the class of parameterized problems solvable in time $f(k)n^{g(k)}$, where f and g are unrestricted functions.

The CLIQUE problem (parameterized by the size of the clique) is $W[1]$-complete [DF99], and is a frequent source problem for $W[1]$-hardness reductions. $W[1]$ is strongly analogous to NP because the k-STEP HALTING PROBLEM FOR NONDETERMINISTIC TURING MACHINES (with unlimited nondeterminism) is complete for $W[1]$ ([DF95b] + [CCDF97]).

What is known about the five different parameterizations of the classic MAX LEAF SPANNING TREE problem?

About MAX LEAF I and Related Problems. This has an *FPT* algorithm with running time $O^*(9.4815^k)$ [BBW03]. Where will the cascade of progress on this problem stop?

There are *two main games* being played in *FPT* algorithms research:
(1) **The $f(k)$ game.** This is the game emphasized by the $O^*(f(k))$ notation. Improvements generally depend on three things, in some mix:
• More powerful preprocessing/kernelization data reduction rules.
• Better concrete branching strategies for the exhaustive analysis of the kernel.
• More sophisticated methods of analysis.
(2) **The kernelization game.** The goal is to produce smaller problem kernels. Success depends on a mix of the first and third of the above.

The PLANAR DOMINATING SET problem provides some examples. An $O^*(11^k)$ *FPT* algorithm was described in [DF99]. This was improved to $O^*(8^k)$ by Alber, et al. [AFFFNRS01]. The algorithm did not change, only the analysis. By more powerful reduction rules, a linear problem kernel was proved in [AFN02] — but this does not yield any improvement in the $f(k)$ game over the $O^*(8^k)$ algorithm.

Both games are worth playing — both lead to better understanding of the *problem structure* afforded by the fixed parameter value — both may lead to new strategies for practical implementations [ALSS03,DRST02].

About Max Leaf Spanning Tree II and Related Problems. Here the parameter is the treewidth of G. We know by Bodlaender's $O^*(2^{35k^3})$ algorithm for structural parsing of graphs of treewidth at most k [Bod96], together with dynamic programming, that this parameterization is *FPT*. Treewidth is an almost universal parameter leading to *FPT* algorithms.

About Max Leaf Spanning Tree III and Related Problems. Here the parameter k is the number of vertices of G. This is clearly in *FPT* by brute force! and might not seem interesting. But it is, because there are new "lower bound" techniques for proving qualitative optimality results for *FPT* algorithms. Combining results of [CJ01] and [DFPR03], it is now known that while VERTEX COVER admits an $O^*(2^{O(k)})$ *FPT* algorithm (where the parameter k, as here, is the number of vertices), this cannot be improved to $O^*(2^{o(k)})$ unless $FPT = M[1]$. A similar result holds for MAX LEAF SPANNING TREE III. By parameterizing on such things as the number of vertices, these new methods allow us to prove lower bound results in the area of "worst-case exponential" algorithms.

About Max Leaf Spanning Tree IV and Related Problems. Here we are parameterizing by the goodness of approximation. If we had an *FPT* result for

this problem, then we would have a special kind of PTAS (an EPTAS [CT97]) where the exponent of the polynomial does not depend on the goodness of the approximation.

Recent PTAS results include:

- The PTAS for the EUCLIDEAN TSP due to Arora [Ar96] has a running time of around $O(n^{300/\epsilon})$. Thus for a 20% error, we have a "polynomial-time" algorithm that runs in time $O(n^{1500})$.
- The PTAS for the MULTIPLE KNAPSACK problem due to Chekuri and Khanna [CK00] has a running time of $O(n^{12(\log(1/\epsilon)/\epsilon^8)})$. Thus for a 20% error we have a "polynomial-time" algorithm that runs in time $O(n^{9375000})$.
- The PTAS for the MAXIMUM SUBFOREST PROBLEM due to Shamir and Tsur [ST98] has a running time of $O(n^{2^{2^{1/\epsilon}}-1})$. For a 20% error we thus have a "polynomial" running time of $O(n^{958267391})$.
- The PTAS for the MAXIMUM INDENDENT SET problem on geometric graphs due to Erlebach, Jansen and Seidel [EJS01] has a running time of $O(n^{(4/\pi)(1/\epsilon^2+2)^2(1/\epsilon^2+1)^2})$. Thus for a 20% error, $O(n^{532804})$.

Viewed in the parameterized framework, these results only establish that the analogs of MAX LEAF SPANNING TREE IV are in *XP*. In the case of the EUCLIDEAN TSP problem, Arora later found an *FPT* algorithm (an EPTAS) [Ar97]. Whether the other problems listed above admit *FPT* algorithms, or are $W[1]$-hard, is a significant open problem. MAX LEAF SPANNING TREE IV turns out to be hard for $W[P]$.

About Max Leaf Spanning Tree V and Related Problems. This is an example of a *dual parameterization*, a notion introduced by Khot and Raman [KR00]. Note that a graph has a spanning tree with at least $n - k$ leaves if and only if has a connected dominating set having at most k vertices. This problem turns out to be complete for $W[2]$. Khot and Raman noticed that quite frequently "dual" parameterized problems behave oppositely, with one being *FPT* and the other hard for $W[1]$. Exceptions are now known, but this seems to be a strong tendency for natural problems in NP. CLIQUE and VERTEX COVER (parameterized by the number of vertices in the solution), are dual, with the former being $W[1]$-complete and the latter being *FPT*. DOMINATING SET is $W[2]$-complete, while its parametric dual is in *FPT*.

2 M[1] and Blow-Ups

There is a new class of parameterized problems seemingly intermediate between *FPT* and $W[1]$:

$$FPT \subseteq M[1] \subseteq W[1]$$

$M[1]$ may turn out to be a better primary reference point for intractability than $W[1]$. There are two natural "routes" to $M[1]$.

The Renormalization Route to $M[1]$.
There are $O^*(2^{O(k)})$ *FPT* algorithms for many parameterized problems, such as Vertex Cover. In view of this, we can "renormalize" and define the problem:

$k \log n$ Vertex Cover
Input: A graph G on n vertices and an integer k; **Parameter:** k; **Question:** Does G have a vertex cover of size at most $k \log n$?

The *FPT* algorithm for the original Vertex Cover problem, parameterized by the number of vertices in the vertex cover, allows us to place this new problem in XP. It now makes sense to ask whether the $k \log n$ Vertex Cover problem is also in *FPT* — or is it parametrically intractable? It turns out that $k \log n$ Vertex Cover is $M[1]$-complete.

The Miniaturization Route to $M[1]$.
We certainly know an algorithm to solve n-variable 3SAT in time $O(2^n)$. Consider the following parameterized problem.

Mini-3SAT
Input: Positive integers k and n in unary, and a 3SAT expression E having at most $k \log n$ variables; **Parameter:** k; **Question:** Is E satisfiable?

Using our exponential time algorithm for 3SAT, Mini-3SAT is in XP and we can wonder where it belongs — is it in *FPT* or is it parametrically intractable? This problem also turns out to be complete for $M[1]$.

Dozens of renormalized *FPT* problems and miniaturized arbitrary problems are now known to be $M[1]$-complete [DFPR03]. However, what is known is quite problem-specific. For example, one might expect Mini-Max Leaf to be $M[1]$-complete, but all that is known presently is that it is $M[1]$-hard. It is not known to be $W[1]$-hard, nor is it known to belong to $W[1]$.

The following theorem would be interpreted by most people as indicating that probably $FPT \neq M[1]$. (The theorem is essentially due to Cai and Juedes [CJ01], making use of a result of Impagliazzo, Paturi and Zane [IPZ98].)

Theorem 1. *$FPT = M[1]$ if and only if n-variable 3SAT can be solved in time $2^{o(n)}$.*

$M[1]$ supports convenient although unusual combinatorics. For example, one of the problems that is $M[1]$-complete is the miniature of the Independent Set problem defined as follows.

Mini-Independent Set
Input: Positive integers k and n in unary, a positive integer $r \leq n$, and a graph G having at most $k \log n$ vertices.
Parameter: k
Question: Does G have an independent set of size at least r?

The following example of a **blow-up reduction** (from [DEFPR03]) displays some of the combinatorial possibilities.

Theorem 2. *There is an FPT reduction from* Mini-Independent Set *to ordinary parameterized* Independent Set *(parameterized by the number of vertices in the independent set).*

Proof. Let $G = (V, E)$ be the miniature, for which we wish to determine whether G has an independent set of size r. Here, of course, $|V| \leq k \log n$ and we may regard the vertices of G as organized in k blocks $V_1, ..., V_k$ of size $\log n$. We now employ a simple but useful *counting trick* that can be used when reducing miniatures to "normal" parameterized problems. Our reduction is a Turing reduction, with one branch for each possible way of writing r as a sum of k terms, $r = r_1 + \cdots + r_k$, where each r_i is bounded by $\log n$. The reader can verify that $(\log n)^k$ is an *FPT* function, and thus that there are an allowed number of branches. A branch represents a commitment to choose r_i vertices from block V_i (for each i) to be in the independent set.

We now produce (for a given branch of the Turing reduction) a graph G' that has an independent set of size k if and only if the miniature G has an independent set of size r, distributed as indicated by the commitment made on that branch. The graph G' consists of k cliques, together with some edges between these cliques. The ith clique consists of vertices in 1:1 correspondence with the subsets of V_i of size r_i. An edge connects a vertex x in the ith clique and a vertex y in the jth clique if and only if there is a vertex u in the subset $S_x \subseteq V_i$ represented by x, and a vertex v in the subset $S_y \subseteq V_j$ represented by y, such that $uv \in E$. Verification is straightforward.

The hypothesis that $FPT \neq M[1]$ has many interesting consequences based on similar arguments. The next theorem is due to recent work of Chor, Fellows and Juedes.

Theorem 3. *If $FPT \neq M[1]$ then determining whether a graph has a k-element dominating set (k-element independent set) cannot be solved in time $O(n^{o(k)})$.*

Cai and Juedes [CJ01] proved the following path-breaking results, which established an *FPT* **optimality program** of broad applicability.

Theorem 4. *If $FPT \neq M[1]$ then there cannot be an FPT algorithm for the general* Vertex Cover *problem with a parameter function of the form $f(k) = 2^{o(k)}$, and there cannot be an FPT algorithm for the* Planar Vertex Cover *problem with a parameter function of the form $f(k) = 2^{o(\sqrt{k})}$.*

It has previously been known that Planar Dominating Set, parameterized by the number n of vertices in the graph can be solved optimally in time $O^*(2^{O(\sqrt{n})})$ by using the Lipton-Tarjan Planar Separator Theorem. Combining the lower bound theorem of Cai-Juedes with the linear kernelization result of Alber et al. [AFN02] shows that this cannot be improved to $O^*(2^{o(\sqrt{n})})$ unless $FPT = M[1]$.

3 Win/Win's and Vertex Cover Structure

Early work on parameterized complexity was motivated by the complexity consequences of the Graph Minor Theorem, that povides a powerful way to classify problems as *FPT*. It applies to the following problem:

Max Internal Spanning Tree
Input: A graph G and an integer k; **Parameter:** k; **Question:** Does G have a spanning tree with at least k internal vertices?

This problem is "governed by bounded vertex cover structure" and in this setting there is a more general (in this setting) and much easier to prove classification tool.

Theorem 5. *For every fixed k, the family of graphs having a vertex cover of size at most k is well-quasiordered by ordinary subgraphs. Furthermore, restricting to this family of graphs, the* Subgraph *problem (Is H a subgraph of G?), parameterized by H, is linear time FPT.*

Proof Sketch. Suppose there is an infinite bad sequence of graphs of bounded vertex cover number. Then there is an infinite bad subsequence all having the same vertex cover number, which we can assume to be k. Order the k vertices of the vertex cover arbitrarily in any order for each graph. There are finitely many graphs on k vertices, so there is an infinite bad subsequence where, respecting vertex orderings, the subgraphs induced by the vertex covers are isomorphic. Every other vertex of a graph in this bad subsequence is attached to a subset of the vertex cover, and has no other incident edges. Thus each graph in the infinite bad subsequence can be described by a *census vector* of length 2^k that counts, for each subset S of the vertex cover, the number of vertices u in the complement of the vertex cover such that $N(u) = S$. By Higman's Lemma, these census vectors are well-quasi-ordered by the pointwise ordering of the census numbers. By the well-quasi-ordering of the census vectors, we reach a contradiction. Also, graphs that have a vertex cover of size k have pathwidth at most k.

As an application, we can consider the following problem, the parametric dual of Graph k-Coloring:

Saving k Colors
Input: A graph G and a positive integer k; **Parameter:** k; **Question:** Can G be colored with $n - k$ colors?

Theorem 6. Saving k Colors *is in FPT.*

Proof. For each fixed k, the graph complements of the no-instances constitute a lower ideal in the subgraph order. Theorem 5 therefore applies.

The essence of the **win/win** strategy in *FPT* algorithm design is to play off one kind of structure against another. An old example from [FL92]:

Theorem 7. *In time $O(n)$ we can find either:*
(1) A cycle of length at least k, or,
(2) A tree decomposition of width at most k.

This is just a plain linear time algorithm, there is nothing exponential in k hiding in the big O. Thus if we are interested in determining whether a graph has

a cycle of length at least k, either we are done, or "for free" we have a bounded width tree decomposition.

A Win/Win for Max Internal
A win/win approach to designing an *FPT* algorithm for MAX INTERNAL SPANNING TREE has recently been described by Prieto and Sloper [PS03]. They describe an algorithm that in time $O(n^2)$ produces either: (1) a vertex cover of size at most k, or (2) a k-internal spanning tree. The rest of their *FPT* algorithm is based on a kernelization rule that can be applied in the situation where there is a small vertex cover.

A Win/Win for the Dual of Dominating Set
The complement of a dominating set of size $n - k$ is termed a *nonblocking* set of vertices (of size k). Faisal Abu-Khazm has described a simple algorithm that in time $O(n^2)$ produces either: (1) a vertex cover of size at most k, or (2) a nonblocking set of size k. If the outcome is (2), then we are done. If the outcome is (1), then in $O(n)$ time we can read off a path decomposition of width k as follows. Let C denote the vertex cover, and let v_1, v_2,..., be the remaining vertices in any order. Then we can take the sequence of bags of the decomposition to be: $C \cup \{v_1\}, C \cup \{v_2\}, ...$ Given this path decomposition, we can use the efficient dynamic programming algorithm of Proskurowski and Telle [PT93] for the MINIMUM DOMINATING SET problem, further improved by Alber and Niedermeier [AN02], to get an $O^*(4^k)$ *FPT* algorithm for the problem. (This matches the running time of McCartin's algorithm [McC03], obtained by a completely different route.)

3.1 Crown Rules: A Surprising New Technique for Kernelization

Consider the VERTEX COVER problem, and the reduction rule for this problem for vertices of degree 1. If (G, k) is an instance of the VERTEX COVER problem where G has a vertex v of degree 1 with neighbor $u \neq v$, then we can reduce to the instance (G', k') where $G' = G - u - v$ and $k' = k - 1$. This simple *local* reduction rule admits a global structural generalization.

Definition 4. *A* crown decomposition *of G is a partition of the vertex set of G into three sets, H, C and J such that the following conditions hold:*
(1) C is an independent set,
(2) H is matched into C, and
(3) H is a cutset, i.e., there are no edges between C and J.

If (G, k) is an instance of the VERTEX COVER problem, and G admits a crown decomposition, then we can reduce to (G', k') where $G' = G - H - C$ and $k' = k - |H|$, as a generalization of the degree 1 reduction rule. Presently, it is an open problem whether determining if a graph has a crown decomposition is in P, or is NP-complete. However, if we are given: (1) a graph $G = (V, E)$ without isolated vertices, and (2) a vertex cover V' for G that satifies the inequality $2|V'| < |V|$ (in other words, the vertex cover is less than half the size of G), then we can compute a crown decomposition in polynomial time as follows. Let

$V'' = V - V'$, and compute a maximum matching between V' and V''. For $x \in V'$, let $m(x) \in V''$ denote the vertex of V'' (if any) to which x is matched. For $U \subseteq V'$, define $m(U) = \{v \in V'' : \exists x \in U \text{ with } m(x) = v\}$. For $W \subseteq V''$ let $N(W) = \{x \in V' : \exists y \in W, xy \in E\}$. Let $C_0 = V'' - m(V')$. The size condition on the vertex cover V' insures that C_0 is nonempty. Note that since V' is a vertex cover, V'' is an independent set in G. Now we iteratively compute:
$H_1 = N(C_0)$, $C_1 = m(H_1) \cup C_0$,
$H_2 = N(C_1)$, $C_2 = m(H_2) \cup C_1$,
... until a fixed point is reached in two final sets H and C. Now take $J = V - C - H$. It is easy to verify (using the fact that m is a *maximum* matching) that we have a nontrivial crown decomposition of G — and thus any instance (G, k) of the Vertex Cover problem can be reduced by our *crown reduction rule*. It seems that many parameterized problems that are "subject to vertex cover structure" have reduction rules that are flavors of this approach.

In particular, we can achieve a kernel of size at most $3k$ for Saving k Colors problem by using crown reduction kernelization [CFJ03]. If determining whether a graph has a nontrivial crown decomposition is in P, then crown reductions provide a new way of producing a $2k$ kernel for Vertex Cover that is different from (and orthogonal to) the Nemhauser-Trotter algorithm. (If it is NP-hard, then we can still get a $3k$ kernel from crown reductions.)

Since the WG Workshop, a way has been found to apply crown reductions together with another new and widely applicable technique called **greedy localization**, introduced by Chen et al. [JZC03] to deliver an improved *FPT* algorithm for the Set Splitting problem. Elsewhere in this volume an $O^*(72^k)$ *FPT* algorithm for the Set Splitting problem is described by Dehne, Fellows and Rosamond [DFR03]. Using greedy localization + crown reduction, this can be improved to $O^*(8^k)$. (The details will have to be described elsewhere.)

Data reduction and kernelization rules are one of the primary outcomes of research on parameterized complexity. After all, whether k is small or not — and no matter if you are going to do simulated annealing eventually — it *always* makes sense to simplify and reduce (pre-process) your input! This humble activity, because reduction rules frequently cascade, can sometimes lead to unexpected success against hard problems on real input distributions. A fine example of this has been described by Weihe [Wei98].

This survey has only scratched the surface of a few new ideas and developments in *FPT* algorithmic techniques. The currently best available comprehensive survey is the recent Habilitationschrift of Rolf Niedermeier, which is highly recommended [Nie02].

References

[AFN02] J. Alber, M. Fellows and R. Niedermeier. "Efficient Data Reduction for Dominating Set: A Linear Problem Kernel for the Planar Case." *Proceedings of Scandinavian Workshop on Algorithms and Theory* (SWAT 2002), Springer-Verlag, *Lecture Notes in Computer Science* 2368 (2002), 150–159.

[AFFFNRS01] J. Alber, H. Fan, M. Fellows, H. Fernau, R. Niedermeier, F. Rosamond and U. Stege. "Refined Search Tree Techniques for the Planar Dominating Set Problem," *Proc. 26th International Symposium on Mathematical Foundations of Computer Science (MFCS 2001)*, Springer-Verlag, *Lecture Notes in Computer Science* 2136 (2001), 111–122.

[ALSS03] F. Abu-Khzam, M. A. Langston, P. Shanbhag and C. T. Symons. "High Performance Tools for Fixed-Parameter Tractable Implementations." WADS'03 Workshop Presentation, 2003.

[AN02] J. Alber and R. Niedermeier. "Improved Tree Decomposition Based Algorithms for Domination-Like Problems," *Proceedings of the 5th Latin American Theoretical IN-formatics (LATIN 2002)*, Springer-Verlag LNCS 2286 (2002), 613–627.

[Ar96] S. Arora. "Polynomial Time Approximation Schemes for Euclidean TSP and Other Geometric Problems." In: *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science* (1996), 2–12.

[Ar97] S. Arora. "Nearly Linear Time Approximation Schemes for Euclidean TSP and Other Geometric Problems." *Proc. 38th Annual IEEE Symposium on the Foundations of Computing* (FOCS'97), IEEE Press (1997), 554–563.

[BBW03] P.S. Bonsma, T. Brüggemann and G.J. Woeginger. A faster *FPT* algorithm for finding spanning trees with many leaves. Manuscript, 2003.

[Bod93] H. L. Bodlaender. "On Linear Time Minor Tests and Depth-First Search." *Journal of Algorithms* 14 (1993), 1–23.

[Bod96] H. L. Bodlaender. "A Linear Time Algorithm for Finding Tree-Decompositions of Small Treewidth." *SIAM Journal on Computing* 25 (1996), 1305–1317.

[CCDF97] L. Cai, J. Chen, R. Downey and M. Fellows. "On the Parameterized Complexity of Short Computation and Factorization." *Archive for Mathematical Logic* 36 (1997), 321–337.

[CFJ03] B. Chor, M. Fellows and D. Juedes. "An Efficient *FPT* Algorithm for Saving k Colors." Manuscript, 2003.

[CJ01] L. Cai and D. Juedes. "On the Existence of Subexponential Parameterized Algorithms," manuscript, 2001. Revised version of the paper, "Subexponential Parameterized Algorithms Collapse the W-Hierarchy," in: *Proceedings 28th ICALP*, Springer-Verlag LNCS 2076 (2001), 273–284. (The conference version contains some major flaws.)

[CK00] C. Chekuri and S. Khanna. "A PTAS for the Multiple Knapsack Problem." *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms* (SODA 2000), pp. 213–222.

[CT97] M. Cesati and L. Trevisan. "On the Efficiency of Polynomial Time Approximation Schemes." *Information Processing Letters* 64 (1997), 165–171.

[DEFPR03] R. Downey, V. Estivill-Castro, M. Fellows, E. Prieto-Rodriguez and F. Rosamond. "Cutting Up is Hard to Do: the Parameterized Complexity of k-Cut and Related Problems." *Electronic Notes in Theoretical Computer Science* 78 (2003), 205–218.

[DF95a] R. G. Downey and M. R. Fellows. "Parameterized Computational Feasibility." In: *P. Clote and J. Remmel (eds.), Feasible Mathematics II.* Birkhauser, Boston (1995), 219–244.

[DF95b] R. G. Downey and M. R. Fellows. "Fixed Parameter Tractability and Completeness II: Completeness for $W[1]$." *Theoretical Computer Science* A 141 (1995), 109–131.

[DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer-Verlag, 1999.

[DFPR03] R. Downey, M. Fellows, E. Prieto-Rodriguez and F. Rosamond. "Fixed-parameter Tractability and Completeness V: Parametric Miniatures." Manuscript, 2003.

[DFR03] F. Dehne, M. Fellows and F. Rosamond. "An *FPT* Algorithm for Set Splitting." *Proc. WG'03* (elsewhere in this volume).

[DRST02] F. Dehne, A. Rau-Chaplin, U. Stege and P. Taillon. "Solving Large *FPT* Problems on Coarse Grained Parallel Machines." Manuscript, 2002.

[EJS01] T. Erlebach, K. Jansen and E. Seidel. "Polynomial Time Approximation Schemes for Geometric Graphs." *Proc. ACM Symposium on Discrete Algorithms (SODA'01)*, ACM Press (2001), 671–679.

[FL92] M.R. Fellows and M.A. Langston. "On Well-Partial-Order Theory and its Applications to Combinatorial Problems of VLSI Design." *SIAM Journal on Discrete Mathematics* 5, 117–126.

[FMRS00] M. Fellows, C. McCartin, F. Rosamond and U. Stege. "Coordinatized Kernels and Catalytic Reductions: An Improved *FPT* Algorithm for Max Leaf Spanning Tree and Other Problems." *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'2000)*, Springer, Lecture Notes in Computer Science 1974 (2000), 240–251.

[IPZ98] R. Impagliazzo, R. Paturi and F. Zane. "Which Problems Have Strongly Exponential Complexity?" *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS'1998)*, 653–663.

[JZC03] W. Jia, C. Zhang and J. Chen. "An Efficient Parameterized Algorithm for Set Splitting." Manuscript, 2003, to appear in *Journal of Algorithms.*

[KR00] S. Khot and V. Raman. "Parameterized Complexity of Finding Subgraphs with Hereditary properties." *Proceedings of the Sixth Annual International Computing and Combinatorics Conference (COCOON 2000)* July 2000, Sydney, Australia, Lecture Notes in Computer Science, Springer-Verlag 1858 (2000), 137–147.

[McC03] C. McCartin. Ph.D. dissertation in Computer Science, Victoria University, Wellington, New Zealand 2003.

[Nie02] R. Niedermeier. "Invitation to Fixed-Parameter Algorithms." Habilitationschrift, University of Tübingen, 2002.

[PS03] E. Prieto and C. Sloper. "Either/Or: Using Vertex Cover Structure in Designing *FPT* Algorithms — the Case of k-Internal Spanning Tree." *Proc. WADS'03*, Springer-Verlag LNCS 2748 (2003), 474–483.

[PT93] J.A. Telle and A. Proskurowski. "Practical Algorithms on Partial k-Trees with an Application to Domination-Like Problems." *Proceedings WADS'93 – The Third Workshop on Algorithms and Data Structures*, Springer-Verlag LNCS 709 (1993), 610–621.

[ST98] R. Shamir and D. Tzur. "The Maximum Subforest Problem: Approximation and Exact Algorithms." *Proc. ACM Symposium on Discrete Algorithms* (SODA'98), ACM Press (1998), 394–399.

[Wei98] K. Weihe, "Covering Trains by Stations, or the Power of Data Reduction," *Proc. ALEX'98* (1998), 1–8.

[Woe03] G. J. Woeginger, "Exact Algorithms for NP-hard Problems: A Survey," manuscript, 2003.

Matching, Edge-Colouring, Dimers

Alexander Schrijver[1,2]

[1] CWI, Kruislaan 413,
1098 SJ Amsterdam, The Netherlands,
[2] Department of Mathematics,
University of Amsterdam, Plantage Muidergracht 24,
1018 TV Amsterdam, The Netherlands.

Abstract. We survey some recent results on finding and counting perfect matchings in regular bipartite graphs, with applications to bipartite edge-colouring and the dimer constant. Main results are improved complexity bounds for finding a perfect matching in a regular bipartite graph and for edge-colouring bipartite graphs, the solution of a problem of Erdős and Rényi concerning lower bounds for the number of perfect matchings, and an improved lower bound for s dimer constant.

1 Finding a Perfect Matching in a Regular Bipartite Graph

The fastest known algorithms for finding a perfect matching in a general bipartite graph have running time of order about $O(\sqrt{n}\,m)$ (Hopcroft and Karp [12], Feder and Motwani [8]) or $O(n^{2.376})$ (Ibarra and Moran [13]). For *regular* bipartite graphs, however, faster algorithms are known: Cole and Hopcroft [4] gave an $O(m \log n)$ algorithm, while Cole [3] gave an $O(2^{2^{O(k)}} n)$ algorithm, where k is the degree of the vertices. So the latter algorithm is linear-time for any fixed k. We now describe an easy $O(k^2 n)$ $(= O(km))$ algorithm ([17]). Here is the idea for $k = 3$.

Let G be a 3-regular bipartite graph. Find a circuit C in G, by finding a path $Q = v_0, e_1, v_1, \ldots$, till we arrive at a vertex v_k where we have been before (that is, $v_k = v_i$ for some $i < k$). Next delete from G every second edge of C. The remaining edges of C form the middle edges of paths of length 3 in the remaining graph G'. Replace each such path P by an edge e_P connecting the ends of P. The resulting graph G'' is 3-regular and bipartite. Find recursively a perfect matching M in G''. Replace any edge e_P that occurs in M by the two end edges of P. For each of the other paths P, add its middle edge to M. This gives a perfect matching in G', hence in G, as required.

To obtain a linear-time algorithm, one should use in the recursion the tail $v_0, e_1, v_1, \ldots, v_i$ of the path Q to find the next circuit in G''. Then the time spent on running through the tail when finding the successive circuits will not be lost, and any recursive step takes amortized time $|VC|$. Since in any recursive step, the size of the graph reduces also by $|VC|$, the algorithm is linear-time.

This gives the theorem of Cole [3]:

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 13–22, 2003.

Theorem 1. *A perfect matching in a 3-regular bipartite graph can be found in linear time.*

We next describe the extension to k-regular bipartite graphs. This uses a weighting of the edges.

Let $G = (V, E)$ be a k-regular bipartite graph. Initially, set $w(e) := 1$ for each edge e. Next, iteratively, find a circuit C in G, split the edge set EC of C into two matchings M and N, in such a way that

$$\sum_{e \in M} w(e) \geq \sum_{e \in N} w(e), \tag{1}$$

reset $w(e) := w(e) + 1$ if $e \in M$ and $w(e) := w(e) - 1$ if $e \in N$, delete the edges e with $w(e) = 0$, and iterate.

Again we find C by following a path, and we keep its tail (if nontrivial) for the next iteration. Note that the resetting maintains the property

$$\sum_{e \ni v} w(e) = k \text{ for each vertex } v. \tag{2}$$

So as long as there exist edges e with $w(e) < k$, we can find a circuit. Hence, the iterations stop if $w(e) = k$ for each edge e. In that case, the edges form a perfect matching, and we are done.

The key to estimating the running time is considering

$$\sum_{e \in E} w(e)^2. \tag{3}$$

This sum is bounded by $\frac{1}{2}k^2|V|$. Moreover, in any iteration, this sum increases by

$$\begin{aligned} &\sum_{e \in M} ((w(e)+1)^2 - w(e)^2) + \sum_{e \in N} ((w(e)-1)^2 - w(e)^2) \\ &= \sum_{e \in M} (2w(e)+1) + \sum_{e \in N} (-2w(e)+1) \geq |M| + |N| = |EC| \end{aligned} \tag{4}$$

(by (1)). Since the amortized time of any iteration is proportional to $|EC|$, this gives an $O(k^2 n) = O(km)$ running time bound ([17]):

Theorem 2. *A perfect matching in a k-regular bipartite graph can be found in $O(km)$ time.*

2 Edge-Colouring

The latter result implies an $O(km)$ algorithm for finding an optimum edge-colouring of a bipartite graph G, where k denotes the maximum degree. (An optimum edge-colouring colours the edges with k colours such that each colour forms a matching.)

First observe that one trivially obtains an $O(k^2m)$ algorithm. Indeed, we can assume that the bipartite graph G is k-regular (as we can extend G to a k-regular bipartite graph, in linear time). Then iteratively find a perfect matching in G and delete it from G. The successive perfect matchings form the colours. This can be done by applying k times the $O(km)$ algorithm, yielding $O(k^2m)$.

However, with a method of Gabow [10], one may speed up this. If k is odd, find a perfect matching in G and delete it from G. If k is even, find an Eulerian orientation of G (that is, an orientation such that the indegree in each vertex is equal to its outdegree). This can be done in linear time. Next split the edge set E of G into the set E_1 of edges oriented from one colour class of G to the other, and the set E_2 of edges oriented in the opposite direction. Then (V, E_1) and (V, E_2) are $\frac{1}{2}k$-regular bipartite graphs, in which we can find optimum edge-colourings recursively. Combining them gives an optimum edge-colouring of G. The running time is

$$O(km + 2(\tfrac{1}{2}k\tfrac{1}{2}m) + 4(\tfrac{1}{4}k\tfrac{1}{4}m) + \cdots) = O(km). \tag{5}$$

Hence:

Theorem 3. *An optimum edge-colouring of a bipartite graph can be found in $O(km)$ time, where k is the maximum degree.*

3 Speed-Up of Cole, Ost, and Schirra

The above $O(km)$ algorithm for perfect matching in k-regular bipartite graphs raises the question if there is a linear-time algorithm, independent of k. This was resolved positively by Cole, Ost, and Schirra [5], by a refinement of the method above, utilizing the data-structure of 'self-adjusting binary trees'. We outline their method.

A first improvement is not to replace $w(e)$ by $w(e) \pm 1$ for the edges in C, but by $w(e) \pm \alpha$, where α is the minimum weight of the edges in N. So at least one edge in N gets weight 0.

A second improvement is to store the paths ('chains') left in the circuit C (after removing the edges of weight 0), so that these chains can be used to speed up later circuit searches. This requires that if in a later circuit search we hit any such chain, then relatively fast we should be able to identify the ends of the chain. (If we have to follow the chain vertex by vertex till its end, no gain in running time is obtained.) This can be done by supplying these chains with the data structure of self-adjusting binary trees (cf. Tarjan [19]). To get the required running time, it turns out that these chains should have length at most k^2 — in the case that they are longer, split them into chains of length about k^2.

A third improvement is a preprocessing that reduces the number of edges of the graph from $\frac{1}{2}kn$ to at most $n \log_2 k$. This is obtained as follows. Start with setting $w(e) := 1$ for each edge e. Next, successively, for $i = 0, 1, \ldots, \lfloor \log_2 k \rfloor$, do the following. Consider the set E_i of edges of weight 2^i. Iteratively (as above) find a circuit C in E_i, split EC arbitrarily into matchings M and N, and reset

$w(e) := w(e) + 2^i = 2^{i+1}$ if $e \in M$ and $w(e) := w(e) - 2^i = 0$ if $e \in N$. (So in each iteration, the set E_i changes.) In linear time we arrive at the situation that E_i contains no circuits, implying $|E_i| \leq n-1$. Then we go over to the case $i+1$. So we end up with at most (about) $n \log_2 k$ edges, together with a weighting w satisfying (2).

For each i, the preprocessing takes time linear in the size of the initial E_i, which is at most $2^{-i}m$. Hence the preprocessing takes $O(m)$ time in total. It turns out that, using the first two improvements, the rest of the algorithm takes $O(n \log^3 k)$ only, which is faster than $O(m)$.

This gives the theorem of Cole, Ost, and Schirra [5]:

Theorem 4. *A perfect matching in a regular bipartite graph can be found in linear time.*

With the method described in Section 2, it has as consequence:

Corollary 1. *An optimum edge-colouring of a bipartite graph can be found in $O(m \log k)$ time, where k is the maximum degree.*

4 From Finding to Counting Perfect Matchings

We now go over to the problem of counting perfect matchings, or rather giving a lower bound for their number. We first relate the algorithm described in Section 1, for finding a perfect matching in a 3-regular bipartite graph, to a lower bound of Voorhoeve [21] on the number of such perfect matchings.

To this end, we modify the algorithm slightly. We may note that when following the path Q in finding the circuit, we can start immediately from the beginning with removing edges. We don't have to wait till we have a circuit. This can be made more precise as follows.

Call a bipartite graph *almost 3-regular* if all vertices have degree 3, except for two vertices of degree 2 (automatically belonging to different colour classes). So an almost 3-regular bipartite graph arises by deleting one edge from a 3-regular bipartite graph. Hence a linear-time algorithm for finding a perfect matching in an almost 3-regular bipartite graph yields the same for 3-regular bipartite graphs. We describe such an algorithm.

Let G be an almost 3-regular bipartite graph, and let u be any of the two vertices of degree 2. To find a perfect matching, we can assume that u is not incident with the other vertex of degree 2, and that it has two distinct neighbours, x and y say. (Otherwise, there is an easy reduction.)

Let u, s, t be the neighbours of x. Delete edge xs. Then edge ux becomes the middle edge of a path $P = (y, u, x, t)$. Replace it by a new edge e_P connecting y and t. Find recursively a perfect matching M in the new graph G'. If e_P is in M, replace it by yu and xt. If e_P is not in M, add ux to M. We end up with a perfect matching in G.

As each iteration takes constant time, and as it reduces the number of vertices by 2, this gives a linear-time algorithm. This might be easier to implement than the algorithm described earlier, since only local operations are performed.

This method is in fact inspired by the method of Voorhoeve [21] to prove that any 3-regular bipartite graph has at least

$$(\tfrac{4}{3})^n \tag{6}$$

perfect matchings, where, for convenience, n denotes *half* of the number of vertices. To prove this bound, it suffices to show that each almost 3-regular bipartite graph has at least $(\frac{4}{3})^n$ perfect matchings. Again, choose a vertex u of degree 2, and we may assume that it has two distinct neighbours of degree 3. (Otherwise, there is an easy induction.) Let $e_1, \ldots, e_4$ be the edges incident with a neighbour of u but not with u. For $i = 1, \ldots, 4$, let G_i be the graph obtained from G by deleting edge e_i. Denote the number of perfect matchings in any graph H by $\pi(H)$. Then, by induction,

$$\pi(G_i) \geq (\tfrac{4}{3})^{n-1} \tag{7}$$

for $i = 1, \ldots, 4$, since replacing the path of length 3 through u by a new edge, gives an almost 3-regular bipartite graph H_i with $2(n-1)$ vertices and with $\pi(H_i) = \pi(G_i)$. Moreover,

$$\pi(G_1) + \cdots + \pi(G_4) = 3\pi(G), \tag{8}$$

since each perfect matching M in G is maintained in precisely three of the G_i (as M contains precisely one of $e_1, \ldots, e_4$). Combining (7) and (8) gives $\pi(G) \geq (\frac{4}{3})^n$, as required.

Incidentally, this may look like an exact inductive calculation of $\pi(G)$, but strict inequality is obtained in the reduction if u has no two distinct neighbours of degree 3.

So we have proved the theorem of Voorhoeve [21]:

Theorem 5. *Any 3-regular bipartite graph on $2n$ vertices has at least $(\frac{4}{3})^n$ perfect matchings.*

With this, Voorhoeve answered a question posed by Erdős and Rényi [6] whether there exists an exponential lower bound on the number of perfect matchings in 3-regular bipartite graphs. (The best bound proved before is only linear in n.)

Erdős and Rényi formulated their question in terms of permanents, which relates to the Van der Waerden conjecture (which was not yet proved when Voorhoeve gave his bound). The *permanent* of an $n \times n$ matrix $A = (a_{i,j})$ is

$$\mathrm{per}A := \sum_{\pi} \prod_{i=1}^{n} a_{i,\pi(i)}, \tag{9}$$

where the sum ranges over all permutations π of $\{1, \ldots, n\}$. So if A is nonnegative and integer, and we make the bipartite graph G with colour classes $\{u_1, \ldots, u_n\}$ and $\{v_1, \ldots, v_n\}$ and with $a_{i,j}$ edges connecting u_i and v_j (for $i, j = 1, \ldots, n$), then per$A$ is equal to the number of perfect matchings in G.

Call a matrix *k-regular* if it is nonnegative and integer and if each row sum and each column sum is equal to k. Then Erdős and Rényi asked for an exponential lower bound for the permanents of 3-regular matrices.

The Van der Waerden conjecture (van der Waerden [22]) asserts that the permanents of any $n \times n$ doubly stochastic matrix is at least

$$\frac{n!}{n^n}. \tag{10}$$

(A matrix is *doubly stochastic* if it is nonnegative and each row sum and each column sum is equal to 1.) The value (10) is attained if all entries of the matrix are equal to $\frac{1}{n}$. Van der Waerden's conjecture remained open for more than half a century, despite considerable research efforts, and was finally proved by Falikman [7].

For each k-regular matrix A, the matrix $\frac{1}{k}A$ is doubly stochastic and satisfies $\mathrm{per}\frac{1}{k}A = k^{-n}\mathrm{per}A$. So Van der Waerden's conjecture implies that the permanent of any k-regular matrix is at least

$$\frac{k^n n!}{n^n} \approx (\frac{k}{e})^n. \tag{11}$$

(This consequence in fact can be seen to be equivalent to Van der Waerden's conjecture.) Hence also Falikman's theorem implies an exponential lower bound on the number of perfect matchings in 3-regular bipartite graphs. The lower bound $(\frac{k}{e})^n$ was proved by Bang [1] and Friedland [9], thus also providing a solution of Erdős and Rényi's question.

It can be proved that the ground number $\frac{4}{3}$ in Voorhoeve's bound is best possible ([18]). To this end, let μ_3 be the largest real such that each 3-regular bipartite graph on $2n$ vertices has at least μ_3^n perfect matchings. So $\mu_3 \geq \frac{4}{3}$.

To prove the reverse inequality, fix n, and consider the collection $\mathcal{G}$ of 3-regular bipartite graphs with colour classes $\{u_1, \ldots, u_n\}$ and $\{v_1, \ldots, v_n\}$ and with (labeled) edges $e_1, \ldots, e_{3n}$. Then

$$|\mathcal{G}| = \left(\frac{(3n)!}{3!^n}\right)^2. \tag{12}$$

Indeed, it is equal to the square of the number of ordered partitions of $\{1, \ldots, 3n\}$ into n classes of size 3.

We can also precisely count for how many graphs G in $\mathcal{G}$, a given subset M of $\{1, \ldots, 3n\}$ of size n forms a perfect matching in G:

$$\left(n!\frac{(2n)!}{2^n}\right)^2. \tag{13}$$

Since M can be chosen in $\binom{3n}{n}$ ways, this implies that the number of pairs G, M with $G \in \mathcal{G}$ and M is a perfect matching in G is equal to

$$\binom{3n}{n}\left(n!\frac{(2n)!}{2^n}\right)^2. \tag{14}$$

By (12) and by definition of μ_3, (14) has as lower bound:

$$\left(\frac{(3n)!}{3!^n}\right)^2 \mu_3^n. \tag{15}$$

Therefore,

$$\mu_3 \leq \left(\binom{3n}{n}\left(n!\frac{(2n)!}{2^n}\right)^2\left(\frac{3!^n}{(3n)!}\right)^2\right)^{1/n} \stackrel{n\to\infty}{\longrightarrow} \frac{4}{3}. \tag{16}$$

(The latter limit uses Stirling's formula.) So $\mu_3 = \frac{4}{3}$.

5 General *k*

Erdős and Rényi also asked for the value, for any k, of the largest real μ_k such that each k-regular bipartite graph G on $2n$ vertices has at least μ_k^n perfect matchings. So by Falikman's theorem (in fact, already by the results of Bang and Friedland), $\mu_k \geq \frac{k}{e}$. On the other hand, the same method as just described gives ([18])

$$\mu_k \leq \frac{(k-1)^{k-1}}{k^{k-2}}. \tag{17}$$

In [18] it was also conjectured that equality holds:

$$\mu_k = \frac{(k-1)^{k-1}}{k^{k-2}}. \tag{18}$$

This in fact was be proved in [16]. Hence:

Theorem 6. *Each k-regular bipartite graph with $2n$ vertices has at least*

$$\left(\frac{(k-1)^{k-1}}{k^{k-2}}\right)^n \tag{19}$$

perfect matchings.

In contrast with the simplicity of Voorhoeve's method for the case $k = 3$, the proof for general k is highly complicated. It is based on a technique of assigning weights to the edges of the graph similar to the algorithm for finding a perfect matching in a k-regular bipartite graph described in Section 1.

Let us briefly relate this bound to Falikman's bound. Both bounds are asymptotically best possible, in different asymptotic directions. Let $\mu(k, n)$ denote the minimum permanent of k-regular $n \times n$ matrices. (Equivalently, of the minimum number of perfect matchings, taken over all k-regular bipartite graphs with $2n$ vertices.) So

$$\mu_k = \inf_{n\in\mathbb{N}} \mu(k, n)^{1/n}. \tag{20}$$

Then in one asymptotic direction one has by (18):

$$\inf_{n\in\mathbb{N}} \frac{\mu(k,n)^{1/n}}{k} = \frac{1}{k}\mu_k = \left(\frac{k-1}{k}\right)^{k-1}. \tag{21}$$

In another direction, by Falikman's theorem:

$$\inf_{k\in\mathbb{N}} \frac{\mu(k,n)^{1/n}}{k} = \frac{n!^{1/n}}{n}. \tag{22}$$

Note that both in (21) and in (22), the right-hand term converges to $1/e$, if k or n tends to infinity.

6 Application to the 3D Dimer Constant

We finally apply the lower bound described in Theorem 6 to obtain a better lower bound for the 3-dimensional dimer problem. This is one of the classical unsolved problems in solid-state chemistry. For integers d, n, consider the 'block' $H_{d,n}$, which is the graph with vertex set $\{1,\dots,n\}^d$, two vertices being adjacent if and only if their Euclidean distance is 1. In this context, an edge is called a *dimer*, and a perfect matching a *dimer tiling*. Let $t_{d,n}$ denote the number of dimer tilings of $H_{d,n}$. So $t_{d,n} > 0$ if and only if n is even.

Hammersley [11] showed that

$$\lambda_d := \lim_{n\to\infty} \frac{1}{(2n)^d} \log t_{d,2n} \tag{23}$$

exists. In fact

$$\lim_{n\to\infty} \frac{1}{(2n)^d} \log t_{d,2n} = \sup_n \frac{1}{(2n)^d} \log t_{d,2n}. \tag{24}$$

Otherwise, there exists a k such that

$$\liminf_{n\to\infty} \frac{1}{(2n)^d} \log t_{d,2n} < \frac{1}{(2k)^d} \log t_{d,2k}. \tag{25}$$

However,

$$t_{d,2n} \geq (t_{d,2k})^{\lfloor \frac{n}{k} \rfloor^d}, \tag{26}$$

since $H_{d,2n}$ contains $\lfloor \frac{n}{k} \rfloor^d$ disjoint copies of $H_{d,2k}$ such that the rest has a perfect matching. This implies that the left-hand side in (25) is at least

$$\liminf_{n\to\infty} \frac{\lfloor \frac{n}{k} \rfloor^d}{(2n)^d} \log t_{d,2k}, \tag{27}$$

which is equal to the right-hand side of (25) — a contradiction.

So λ_d is defined. For $d = 2$, the value of λ_d was determined precisely by Kasteleyn [14] and Temperley and Fisher [20]:

$$\lambda_2 = \frac{1}{\pi} \sum_{i=0}^{\infty} \frac{(-1)^i}{(2i+1)^2} = 0.29156090\dots. \tag{28}$$

The proof uses the fact that $H_{2,n}$ is planar, and that the graph therefore has a 'Pfaffian' orientation, making it possible to count dimer tilings by calculating a determinant.

For dimensions larger than two, no such orientation exists, and no exact formula for λ_d is known. Since $H_{d,n}$ is bipartite and 'almost' $2d$-regular, one could try to apply the results obtained earlier. In fact one has:

Theorem 7. $\lambda_d \geq \frac{1}{2}\log \mu_{2d}$. To see this, for each $i \in \{1,\ldots,d\}$ and each $j \in \{1,2n\}$, let $M_{i,j}$ be a perfect matching in the subgraph of $H_{d,2n}$ spanned by

$$\{x \in \{1,\ldots,2n\}^d \mid x_i = j\}. \tag{29}$$

(So this set represents a 'face' of $H_{d,2n}$.) Let $H'_{d,2n}$ be the $2d$-regular bipartite graph obtained from $H_{d,2n}$ by adding parallel edges for the edges in the $M_{i,j}$. Then $H'_{d,2n}$ has more perfect matching than $H_{d,2n}$ has, but not too much more:

$$\pi(H'_{d,2n}) \leq 2^{d(2n)^{d-1}} \pi(H_{d,2n}). \tag{30}$$

This follows from the facts that we have added $d(2n)^{d-1}$ parallel edges, and that adding any such edge at most doubles the number of perfect matchings.

Since $\pi(H'_{d,2n}) \geq \mu_{2d}^{(2n)^d/2}$ (by definition of μ_{2d}), we have

$$\pi(H_{d,2n}) \geq 2^{-d(2n)^{d-1}} \mu_{2d}^{(2n)^d/2}. \tag{31}$$

Therefore,

$$\begin{aligned}\lambda_d \geq \sup_n \frac{1}{(2n)^d} \log\left(2^{-d(2n)^{d-1}} \mu_{2d}^{(2n)^d/2}\right) &= \sup_n(\tfrac{1}{2}\log\mu_{2d} - \frac{d\log 2}{2n}) \\ &= \tfrac{1}{2}\log\mu_{2d},\end{aligned} \tag{32}$$

proving Theorem 7.

Evaluation for $d = 3$ by using $\mu_6 = 5^5/6^4$, gives the best known lower bound for λ_3:

$$\lambda_3 \geq 0.44007584\ldots. \tag{33}$$

The best known upper bound is due to Lundow [15]: $0.457547\ldots$. Computational experiments of Beichl and Sullivan [2] suggest that $\lambda_3 = 0.4466 \pm 0.0006$.

References

[1] T. Bang, *On matrix-functions giving a good approximation to the v.d.Waerden permanent conjecture*, Preprint Series 1979 No. 30, Matematisk Institut, Københavns Universitet, Copenhagen, 1979.

[2] I. Beichl, F. Sullivan, Approximating the permanent via importance sampling with application to the dimer covering problem, *Journal of Computational Physics* 149 (1999) 128–147.

[3] R.J. Cole, *Two Problems in Graph Theory*, Ph.D. Thesis, Cornell University, Ithaca, New York, 1982.
[4] R. Cole, J. Hopcroft, On edge coloring bipartite graphs, *SIAM Journal on Computing* 11 (1982) 540–546.
[5] R. Cole, K. Ost, S. Schirra, Edge-coloring bipartite multigraphs in $O(E\log D)$ time, *Combinatorica* 21 (2001) 5–12.
[6] P. Erdős, A. Rényi, On random matrices II, *Studia Scientiarum Mathematicarum Hungarica* 3 (1968) 459–464.
[7] D.I. Falikman, Proof of the van der Waerden conjecture regarding the permanent of a doubly stochastic matrix [in Russian], *Matematicheskie Zametki* 29 (1981) 931–938 [English translation: *Mathematical Notes of the Academy of Sciences of the USSR* 29 (1981) 475–479].
[8] T. Feder, R. Motwani, Clique partitions, graph compression and speeding-up algorithms, *Journal of Computer and System Sciences* 51 (1995) 261–272.
[9] S. Friedland, A lower bound for the permanent of a doubly stochastic matrix, *Annals of Mathematics* 110 (1979) 167–176.
[10] H.N. Gabow, Using Euler partitions to edge color bipartite multigraphs, *International Journal of Computer and Information Sciences* 5 (1976) 345–355.
[11] J.M. Hammersley, Existence theorems and Monte Carlo methods for the monomer-dimer problem, in: *Research Papers in Statistics* [Festschrift for J. Neyman] (F.N. David, ed.), Wiley, London, 1966, pp. 125–146.
[12] J. Hopcroft, R.M. Karp, An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs, *SIAM Journal on Computing* 2 (1973) 225–231.
[13] O.H. Ibarra, S. Moran, Deterministic and probabilistic algorithms for maximum bipartite matching via fast matrix multiplication, *Information Processing Letters* 13 (1981) 12–15.
[14] P.W. Kasteleyn, The statistics of dimers on a lattice, I: the number of dimer arrangements on a quadratic lattice, *Physica* 27 (1961) 1209–1225.
[15] P.H. Lundow, Compression of transfer matrices, *Discrete Mathematics* 231 (2001) 321–329.
[16] A. Schrijver, Counting 1-factors in regular bipartite graphs, *Journal of Combinatorial Theory, Series B* 72 (1998) 122–135.
[17] A. Schrijver, Bipartite edge-colouring in $O(\Delta m)$ time, *SIAM Journal on Computing* 28 (1999) 841–846.
[18] A. Schrijver, W.G. Valiant, On lower bounds for permanents, *Indagationes Mathematicae* 42 (1980) 425–427.
[19] R.E. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1983.
[20] H.N.V. Temperley, M.E. Fisher, Dimer problem in statistical mechanics — an exact result, *Philosophical Magazine* (8) 6 (1961) 1061–1063.
[21] M. Voorhoeve, A lower bound for the permanents of certain $(0,1)$-matrices, *Indagationes Mathematicae* 41 (1979) 83–86.
[22] B.L. van der Waerden, [Aufgabe] 45, *Jahresbericht der Deutschen Mathematiker-Vereinigung* 35 (1926) *117*.

Minimum Flow Time Graph Ordering

Claudio Arbib, Michele Flammini, and Fabrizio Marinelli

Dipartimento di Informatica, Università di L'Aquila,
via Vetoio, Coppito, I-67100 L'Aquila, Italy

Abstract. The graph ordering problem here addressed derives from industrial applications where one can associate vertices with process steps and edges with jobs. A linear layout of the vertices corresponds then to a production schedule, and one wants to find a layout minimizing the average completion time of the jobs. We prove that the problem is NP-hard in general and is polynomial on trees. We then provide a 2-approximate algorithm and investigate necessary conditions for optimality. On this basis, we devised a combinatorial branch-and-bound algorithm and tested it on random graphs with up to 100 nodes.

1 The Problem

Consider the production of m different part types achieved with a process organized into n subsequent steps. Each step produces some part types, and the completion time C_i of the i-th part type ($i = 1, \ldots, m$) is defined as the time when the part type is consolidated, that is when the last step producing that part type is over. A problem then arises of finding an order of the n process steps which optimizes some performance indicators related with work-in-process.

There are areas (for example cutting stock, see [3]) where this kind of problem makes sense even under such simplifying assumptions as:

i) each part type is produced in exactly two process steps;
ii) process steps have all the same duration.

A nice feature of assumptions (*i*) and (*ii*) is that they allow to describe the problem data by means of a symmetric graph $G = (V, E)$ with n vertices and m edges. A vertex (an edge) of G represents a process step (a part type), an edge links two vertices whenever the corresponding part type is produced by the corresponding steps, and a permutation $\sigma : V \to \{1, \ldots, n\}$ defines an order in which the steps can be executed. We can associate with σ such performance indicators as

- the *work-in-process* $\mathbf{w}(\sigma)$, where $w_i(\sigma)$ is the number of in-process part types between step $(i-1)$ and step i, $1 \le i \le n$;
- the *permanence* $\mathbf{p}(\sigma)$, where $p_{uv}(\sigma) = |\sigma(u) - \sigma(v)|$ is the time spent by part type uv in the system.

A general problem is then to (*a*) minimizing $||\mathbf{w}(\sigma)||$, or (*b*) minimizing $||\mathbf{p}(\sigma)||$ over all the permutations σ of V, where $||.||$ is a suitable norm. It is easy to see that:

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 23–33, 2003.

- if $||.||$ is the $L_{\max}$ norm, then (a) defines the MIN CUT LINEAR ARRANGEMENT (or CUTWIDTH) problem and (b) the BANDWIDTH problem;
- if $||.||$ is the L_1 norm, then both (a) and (b) define the MINIMUM LINEAR ARRANGEMENT problem.

All of the three above are well known NP-complete problems, see [4]. In particular, CUTWIDTH (respectively: BANDWIDTH; MINIMUM LINEAR ARRANGEMENT) is NP-complete on planar graphs with degree ≤ 3 (respectively: trees with degree ≤ 3; bipartite graphs), but can be solved in polynomial time on trees (respectively: interval graphs; trees and De Bruijn graphs), and admits a $\log^2(n)$-approximate (respectively: $\log^{11/2}(n)$-approximate; $\log(n)$-approximate) polynomial algorithm. For a survey on the layout problems see Silvestre [8].

Besides work-in-process and permanence, other meaningful indicators for graph orderings can be drawn from the theory of scheduling, in particular from 1-machine scheduling problems (see [7]). For example, one can minimize the makespan $C_{\max}$, or the flow (or mean completion) time $\sum C_i$ of the lots, or again, if individual due dates $d_1, \ldots, d_m$ are associated with the lots, the maximum lateness $L_{\max}$ or the average lateness $\sum L_i$ of the lots, or the total number $\sum h_i$ of tardy lots, etc. To the best of our knowledge, however, none of these indicators has been considered yet in the literature. In this paper, we will focus on a graph ordering problem with the objective of minimizing the flow time $\sum C_i$.

1.1 Formalization

Let $G = (V, E)$ be an undirected graph with n vertices and m edges. Let $\eta(uv) = 1$ if $uv \in E$ and $\eta(uv) = 0$ otherwise, and for any $v \in V$ let $N(v)$ indicate the set of vertices u such that $\eta(uv) = 1$. Further, let $\sigma : V \to \{1, \ldots, n\}$ be a bijection identifying a linear order of the vertices of G. For any $i \in \{1, \ldots, n\}$ and for any σ let $E_i(\sigma) = \{uv \in E : \sigma(u) < \sigma(v) = i\}$ indicate the set of edges with one extreme of order i and the other of order $< i$. The edges in $E_i(\sigma)$ are said to be *entering* the i-th vertex of σ. Define the cost of σ as:

$$c_G(\sigma) = \sum_{i=1}^{n} i \cdot e_i(\sigma) \tag{1}$$

where $e_i(\sigma) = |E_i(\sigma)|$ (from now on, for any given orders σ, σ^*, ... we will write for simplicity $e_i = e_i(\sigma)$, $e_i^* = e_i(\sigma^*)$ and the like; clearly, $\sum_{i=1}^{n} e_i = m$ and $e_i \leq i - 1$, $i = 1, \ldots, n$). Notice that by assumption (ii), if edge (part type) uv terminates on the i-th vertex (process step), then i represents the completion time of uv. Hence $c_G(\sigma)$ correctly represents the flow time of the schedule. The problem is:

Problem 1. Find an order σ^* such that $\sigma^* \leq c_G(\sigma)$ for all $\sigma : V \to \{1, \ldots, n\}$.

In the rest of paper we investigate properties of optimal solutions (Sect. 2), evaluate the problem complexity (Sect. 3), provide an approximation algorithm (Sect. 4), and devise on this basis a branch-and-bound scheme (Sect. 5).

2 Some Properties

In this section we investigate some properties of Problem 1. Some of these properties will be used as preliminary results to establish the problem complexity (Sect. 3), and to derive dominating rules applied in the branch-and-bound algorithm (Sect. 5).

Similarly to the optimal solutions of BANDWIDTH, MIN CUT LINEAR ARRANGEMENT, MINIMUM LINEAR ARRANGEMENT, an optimal solution σ^* of Problem 1 has the following monotonicity property:

Proposition 1. *Let σ^* be optimal for $G = (V, E)$. Then for any $v \in V$, $e \in E$ and optimal orders σ of $G - v$, $\bar{\sigma}$ of $G - e$, it results that $c_{G-v}(\sigma) \leq c_G(\sigma^*)$ and $c_{G-e}(\bar{\sigma}) \leq c_G(\sigma^*)$.*

Proof. The first inequality is trivial. For the second, suppose by contradiction $c_{G-e}(\bar{\sigma}) > c_G(\sigma^*)$. But in turn, obviously $c_G(\sigma^*) > c_{G-e}(\sigma^*)$; hence $\bar{\sigma}$ would be non-optimal on $G - e$. □

Unlike the above classical ordering problems, instead, the reverse of an optimal sequence σ^* for Problem 1 is generally non-optimal. Moreover, it is easy to see that σ^* has the following properties.

Proposition 2. *Let σ^* be optimal for G, and let $\sigma^*(u) = k$, $\sigma^*(v) = k + 1$. Then*

$$e_k^* \geq e_{k+1}^* - \eta(uv) \tag{2}$$

Proof. Write the cost of σ^* as

$$c_G(\sigma^*) = C(\sigma^*) + ke_k^* + (k+1)e_{k+1}^*$$

where $C(\sigma^*)$ denotes the contribution to $c_G(\sigma^*)$ due to the arcs entering the first $k-1$ and the last $n-k-1$ vertices of σ^*. Consider then the order σ obtained from σ^* by swapping u and v. Clearly, the contribution $C(\sigma^*)$ to the cost remains unchanged, and

i) if $uv \notin E$, then $e_i = e_i^*$ for $i = k, k+1$;
ii) if $uv \in E$, then $e_k = e_k^* - 1$ and $e_{k+1} = e_{k+1}^* + 1$

In the former case, one has

$$\begin{aligned} & c_G(\sigma^*) - c_G(\sigma) = \\ = \quad & ke_k^* + (k+1)e_{k+1}^* - ke_{k+1}^* + (k+1)e_k^* \quad = \\ & = e_{k+1}^* - e_k^* \ . \end{aligned}$$

In the latter

$$\begin{aligned} & c_G(\sigma^*) - c_G(\sigma) = \\ = \quad & ke_k^* + (k+1)e_{k+1}^* - [k(e_{k+1}^* - 1) + (k+1)(e_k^* + 1)] \quad = \\ & = e_{k+1}^* - e_k^* - 1 \ . \end{aligned}$$

Since σ^* is optimal, the above quantities must be ≤ 0, which is true if and only if $e^*_k \geq e^*_{k+1} - \eta(uv)$. □

The above proposition has the following, immediate corollary.

Proposition 3. *In an optimal order σ^*, let $\sigma^*(u) = n-1$ and $\sigma^*(v) = n$. Then* $\deg(u) \geq \deg(v)$.

Proof. By Proposition 2, $\deg(u) = e^*_{n-1} \geq e^*_n = \deg(v)$ if $uv \notin E$, and $\deg(u) = e^*_{n-1} + 1 \geq e^*_n = \deg(v)$ if $uv \in E$. □

Proposition 3 can be used in two ways: if v has already been fixed at position n, then we can exclude all the vertices with degree $\leq \deg(v)$ from the $(n-1)$-th position; moreover, we can exclude from the n-th position all the vertices with maximum degree which are not adjacent to any other vertex with maximum degree.

Proposition 4. *Let σ^* be optimal for $G = (V, E)$, and $\sigma^*(v) = n$ for some $v \in V$. Then $N(v) \not\supset N(u) \forall u \in G$.*

Proof. Indirectly suppose that $N(v) \supset N(u)$. Let $\sigma(u) = i$, and define

- $B_1 \cup B_2 = N(u)$ where B_1 (B_2) is the set of vertices adjacent to both u and v, preceding (following) the position i in the sequence σ^*,
- $A_1 \cup A_2 = N(v) \setminus N(u)$ where A_1 (A_2) is the set of vertices only adjacent to v, preceding (following) the position i in the sequence σ^*.

Consider now the sequence $\bar{\sigma}$ where v is swapped with u. Then:

$$\begin{aligned} c(\bar{\sigma}) - c(\sigma^*) &= i(|A_1| + |B_1|) + c(B_2) + c(A_2) + n(|B_1| + |B_2|) + \\ &\quad -n(|B_1| + |B_2|) - n(|A_1| + |A_2|) - i|B_1| - c(B_2) \\ &= |A_1|(i-n) + c(A_2) - n|A_2| < 0 \end{aligned} \tag{3}$$

Then there exists a solution which has u in the last position and is better than σ^*. □

As an immediate consequence of Proposition 4, if G has a vertex v of degree 1, then there exist an optimal solution σ^* with $\sigma^*(v) = n$.

Proposition 5. *Let σ^* be optimal for $G = (V, E)$, and $\sigma^*(v) = n$ for some $v \in V$. If d is the minimum degree of G and $k = \deg(v)$, then*

$$k < \frac{\sqrt{1 + 8n(d+1)} - 1}{2} \tag{4}$$

Proof. Let u be a vertex of G with $\deg(u) = d$, and let $\sigma^*(u) = i$. Consider the sequence $\bar{\sigma}$ where u and v are swapped. Denote as $c(\sigma^*(u))$ the cost due to all the edges incident on u in σ^*: in particular $c(\sigma^*(u)) \geq id$, since in the best case the other extreme of each edge incident on u lies before u in σ^*; similarly,

$c(\bar{\sigma}(v)) \leq \sum_{i=n-k}^{n} i$, since in the worst case the other extremes of all edges incident on v lie consecutively in positions $n, n-1, \ldots, n-k$. Then,

$$c(\bar{\sigma}) - c(\sigma^*) = c(\bar{\sigma}(v)) + c(\bar{\sigma}(u)) - c(\sigma^*(v)) - c(\sigma^*(u)) \tag{5}$$

But

$$c(\bar{\sigma}(u)) - c(\sigma^*(u)) \leq d(n-i)$$
$$c(\bar{\sigma}(v)) - c(\sigma^*(v)) \leq \sum_{i=n-k}^{n} i - nk = n - \frac{k(k+1)}{2}$$

and therefore

$$c(\bar{\sigma}) - c(\sigma^*) \leq nd - id + n - \frac{k(k+1)}{2} \tag{6}$$

Since σ^* is optimal, the swap between the i-th and the n-th vertex of σ^* leads to no improvement, i.e., $c(\bar{\sigma}) - c(\sigma^*) \geq 0$, or equivalently, $i \leq n + \frac{n}{d} - \frac{k(k+1)}{2d}$. In particular, as no assumptions are made on the position i of vertex u, it follows

$$n + \frac{n}{d} - \frac{k(k+1)}{2d} \geq 0 \tag{7}$$

and this happens if (4) holds. □

It is easy to see that the properties in Propositions 3 through 5 do not dominate each other.

Finally, for any order σ, σ^*, ..., denote as V_k, V_k^*, ... the set of the $u \in V$ such that $\sigma(u) \leq k$, $\sigma^*(u) \leq k$, ..., and let G_k, G_k^*, ... denote the subgraph of G induced by such vertices.

Proposition 6. *An order σ^* which is optimal for G is also optimal for G_k^*, $k = 1, \ldots, n$.*

Proof. One has

$$c_G(\sigma^*) = \sum_{i=1}^{k} i e_i^* + C(\sigma^*) \ .$$

where $C(\sigma^*)$ denotes here the contribution to $c_G(\sigma^*)$ due to the last $n-k$ vertices of σ^*. Assume by contradiction σ^* non-optimal for G_k^*, that is, there exists an order $\sigma : V_k^* \to \{1, \ldots, k\}$ such that

$$\sum_{i=1}^{k} i e_i < \sum_{i=1}^{k} i e_i^* \ .$$

But then the extension $\bar{\sigma}$ of σ to V where the last $n-k$ vertices have the same order as in σ^* would cost

$$c_G(\bar{\sigma}) = \sum_{i=1}^{k} i e_i + C(\sigma^*) < c_G(\sigma^*) \ ,$$

contradicting the optimality of σ^*. □

3 Complexity Results

This section is devoted to establish the computational complexity of Problem 1.

Theorem 1. *Problem 1 is NP-hard.*

To prove Theorem 1 we need some preliminary results. The first relates the cost function (1) of Problem 1 to the cost function $\xi_G(\sigma)$ of MINIMUM LINEAR ARRANGEMENT:

Lemma 1. *Let*

$$\delta_G(\sigma) = \sum_{v \in V} \sigma(v) \deg_G(v) \tag{8}$$

where $\deg_G(v)$ *denotes the number of edges of* G *that are incident to* $v \in V$. *Then for any graph* G *and any order* σ *one has*

$$c_G(\sigma) = \frac{1}{2}[\xi_G(\sigma) + \delta_G(\sigma)] \tag{9}$$

Proof. The objective of MINIMUM LINEAR ARRANGEMENT is to minimize the total number $\xi_G(\sigma) = ||\mathbf{w}(\sigma)||_1$ of edges appearing in the $(n-1)$ cuts separating the first i vertices from the remaining $(n-i)$, $i = 1, \ldots, n-1$. Considering that an edge from the i-th to the j-th vertex of σ appears in $(j-i)$ cuts, one can write

$$\xi_G(\sigma) = \sum_{k=1}^{n} k e_k - \sum_{k=1}^{n} k[\deg_G(\sigma^{-1}(k)) - e_k]$$

i.e., the thesis. □

In order to introduce the second result, recall that a *cut* of a connected graph $G = (V, E)$ is a minimal subset C of E whose removal disconnects G. Finding a cut C with maximum number of edges is a well-known NP-hard problem called SIMPLE MAX CUT, see [5]. The second result then states that

Lemma 2. MINIMUM LINEAR ARRANGEMENT *remains NP-hard even when restricted to graphs with all vertices of odd degree.*

Proof. By reduction from SIMPLE MAX CUT. First, we prove that SIMPLE MAX CUT remains NP-hard even when restricted to graphs with all vertices of odd degree. Let $G = (V, E)$ be an instance of SIMPLE MAX CUT, and let $P \subseteq V$ be the set of vertices with even degree. Now, match P with a set P' of $|P|$ new vertices. The resulting graph G' has $|E| + |P|$ edges, the new vertices have all degree 1 and those originally in V have odd degree: thus all vertices of G' have odd degree. Let C be a cut of G, $\{V_1, V_2\}$ be the corresponding bipartition of V, and P'_i be the mates of $P \cap V_i$ in P', $i = 1, 2$. Clearly, the cut of G' associated with the bipartition $\{V_1 \cup P'_2, V_2 \cup P'_1\}$ has $|C| + |P|$ edges and, as it can be easily checked, is maximum for G' if and only if C is maximum for G.

Based on the above and on [5], we next provide a similar restriction to MINIMUM LINEAR ARRANGEMENT. Let $G = (V, E)$ be an instance of SIMPLE MAX

CUT restricted to graphs with all vertices of odd degree. Since all degrees in G are odd, $n = |V|$ is even. Hence n^4 is also even. Construct an instance G' of MINIMUM LINEAR ARRANGEMENT by taking the complement $\bar{G}$ of G, adding to it a complete graph Q with n^4 vertices, and joining each vertex of $\bar{G}$ to $n^4 - 1$ vertices of Q. Let $\bar{u}$ be the vertex of Q which is not joined to any vertex of $\bar{G}$. Since n is even, every $v \in V$ in G has odd degree and $n^4 - 1$ is odd, it follows that all vertices in $\bar{G}$ have odd degree. Also, since both n and n^4 are even, all the vertices of Q have odd degree. Thus, all the vertices of $G' = (V', E')$ have odd degree. Using an argument similar to [5], it is then easy to show that G admits a cut of at least k edges if and only if there exists a linear arrangement $f : V' \to \{1, 2, \ldots, n^4 + n\}$ of G' such that $f(\bar{u}) = 1$ and

$$\sum_{(u,v)\in E'} |f(u) - f(v)| \leq W = \binom{n^4 + n + 1}{3} - \frac{n^5}{2} - kn^4 - \frac{n^2}{4} - \frac{n}{2} \ .$$

□

We are now in a position to prove Theorem 1.

Proof of Theorem 1. By reduction from MINIMUM LINEAR ARRANGEMENT. Let $G = (V, E)$, $|V| = n$, be an instance of such a problem restricted as in Lemma 2. The basic idea of the proof is to construct an instance G' of Problem 1 by

i) adding to G a complete graph Q with q vertices and an empty graph R with r vertices;
ii) choosing q, r and connecting Q, R to G so that in any sequence minimizing $c_{G'}$ all the vertices of Q (of R) precede (follow) the vertices of G.

To this aim, first set $q = 2n + 5$. To compute a suitable value of r, define $\bar{d}_v = \max_{v' \in V}\{\deg_G(v')\} - \deg_G(v)$, for all $v \in V$. Since the degree of all the vertices in G is odd, $\bar{d}_v$ is even. Set then $r = \frac{\sum_{v\in V} \bar{d}_v}{2} + n$. Finally, join each vertex v of G to $\frac{\bar{d}_v}{2} + 1$ arbitrary vertices of Q, and to $\frac{\bar{d}_v}{2} + 1$ vertices of R so as to have every vertex of R with degree 1 in G'. Notice that the degree of every vertex of Q (of R) is greater than (less than or equal to) that of any other vertex in G'. Moreover, the vertices of $G' - (Q \cup R)$ (i.e., those originating from G) have all the same degree $d < n + 2$.

Let us now prove that Problem 1 admits an optimal solution σ^* such that

$$\sigma^*(v) \leq q \quad \text{for all } v \in Q \tag{10}$$

$$q + n < \sigma^*(v) \leq q + n + r \quad \text{for all } v \in R \tag{11}$$

In fact, suppose by contradiction that in an optimal sequence σ some consecutive vertices of Q are preceded by some $u_0 \notin Q$. Denote by $U = \{u_1, \ldots, u_k\}$ the rightmost set of such vertices. Assume $\sigma(u_0) = p$, and let e_{p+i} denote the number of edges entering u_i in σ $(i = 0, 1, \ldots, k)$. Then the contribution of $U \cup \{u_0\}$ to $c_{G'}$ equals $\sum_{i=0}^{k} (p + i)e_{p+i}$. If we move u_0 after u_k, this contribution becomes $\sum_{i=1}^{k} (p + i - 1)(e_{p+i} - a_i) + (p+k)(e_p - \sum_{i=1}^{k} a_i)$ where $a_i = 1$ if u_i is adjacent to

u_0, and 0 otherwise. The difference between the former and the latter expression of the contribution is then non-positive if and only if $\sum_{i=1}^{k}(2p+k+i-1)a_i + \sum_{i=1}^{k} e_{p+i} \leq ke_p$. But since $e_p \leq d < n+2$ and $e_{p+i} \geq q-k+i-1$ for $i = 1, \ldots, k$, when $q > 2n+5$ it results that $\sum_{i=1}^{k} e_{p+i} - ke_p \geq \frac{k}{2}(2q-k-1) - k(n+2) = k(q - \frac{k+2n+5}{2}) > 0$ for any k between 1 and q. Thus σ can be improved by moving u_0 after u_k, and therefore is non-optimal. Finally, inequality (11) directly follows from Proposition 4.

If we now compute the cost of σ^* using equality (9) of Lemma 1, we observe that the second term of the right hand side equals $\delta_{G'}(\sigma^*) = \sum_{v \in Q} \sigma^*(v) \deg_{G'}(v) + d\sum_{i=1}^{n}(q+i) + \sum_{i=1}^{r}(q+n+i) = \sum_{v \in Q} \sigma^*(v) \deg_{G'}(v) + dn(q + \frac{n+1}{2}) + r(q+n+\frac{r+1}{2})$ and is therefore independent on how vertices in G are sequenced. Moreover, since each vertex of G is incident to the same number of vertices toward Q and R by construction, the contribution of the edges with one extreme in G and the other in $Q \cup R$ to $\xi_{G'}$ is also independent of how vertices in G are sequenced. Thus, the restriction of σ^* to these vertices minimizes ξ_G. □

Although Problem 1 is NP-hard in general, it is polynomial on trees.

Proposition 7. *Let $G = (V, E)$ be a tree. Then for any optimal permutation σ^* of V one has $e_k^* = 1$ for $k = 2, \ldots, n$.*

Proof. There always exists an order σ^* with $e_i = 1$ for all $2 \leq i \leq n$. Suppose indirectly σ^* non-optimal, that is, there exists an order σ such that $c(\sigma) < c(\sigma^*)$.

Let $2 \leq k_1, \ldots, k_s \leq n$ be the indexes of σ such that $e_{k_j} > 1$, for all $1 \leq j \leq s$ and some $s \geq 1$. Let $q = \sum_{i=1}^{s} e_{k_i}$. Then, there exist $p \geq q - s + 1$ indexes $j_1 < \ldots < j_p < k_s$ such that $e_{j_1} = \ldots = e_{j_p} = 0$. In fact, if there are less than p vertices preceding $\sigma^{-1}(k_s)$ and with no entering edges, then the forest induced on G by the first k_s vertices has more than $k_s - 1$ edges. Hence, it follows that $c(\sigma) - c(\sigma^*) \geq \sum_{i=1}^{s} k_i(e_{k_i} - 1) - p = \sum_{i=1}^{s} e_{k_i}(k_i - 1) - 1 > 0$ (contradiction). □

Proposition 8. *Let $G = (V, E)$ be a tree with n vertices. Then $c_G(\sigma^*) = \frac{n(n+1)}{2} - 1$ for any optimal permutation σ^* of V.*

Theorem 2. *Let $G = (V, E)$ be a tree with n vertices. Then an optimal solution can be computed in $O(n)$ time.*

Proof. Every depth-first visit provides a solution obeying to Proposition 7. □

4 A 2-Approximation Algorithm

Let $\deg_G(u_1) \geq \deg_G(u_2) \geq \ldots \geq \deg_G(u_n)$. It is easy to see that although every non-increasing degree order $\tilde{\sigma} = (u_1, \ldots, u_n)$ fulfils the necessary conditions for optimality expressed by Propositions. 2-5, such an order can be non-optimal, see Fig. 1. However, since $\tilde{\sigma}$ minimizes expression (8) and $c_G(\sigma) = \frac{\xi_G(\sigma) + \delta_G(\sigma)}{2}$, $\tilde{\sigma}$ might be a good solution for Problem 1. In fact we can prove the following approximation result.

Theorem 3. *Any algorithm ranking the vertices of G by non-increasing degree is a 2-approximation algorithm.*

Proof. Take a sequence σ^* minimizing c_G and a non-increasing degree order $\tilde{\sigma}$. On the one hand we have

$$c_G(\tilde{\sigma}) \leq \delta_G(\tilde{\sigma}) \tag{12}$$

since $e_i \leq \deg_G(\sigma^{-1}(i))$ for all i. On the other hand, by Lemma 1

$$c_G(\sigma^*) \geq \frac{\delta_G(\sigma^*)}{2} \geq \frac{\delta_G(\tilde{\sigma})}{2} \tag{13}$$

since $\delta_G(\tilde{\sigma}) \leq \delta_G(\sigma)$ for any sequence σ. Then, by inequalities (12)-(13)

$$\frac{c_G(\tilde{\sigma})}{c_G(\sigma^*)} \leq \frac{2\delta_G(\tilde{\sigma})}{\delta_G(\tilde{\sigma})} = 2 \ .$$

□

Observe however that in a non-increasing degree order the vertex position is decided regardless of the number of entering arcs, which yet directly affect the solution cost. In fact, despite the above approximation result, computational tests show that non-increasing degree orders are usually not very good on general graphs. To take entering arcs into account one can apply the following algorithm, which performs significantly better in practice:

Algorithm *MiniDEO* (**input**: G; **output**: $\tilde{\sigma}$)
$G_n := G$;
for $k := n$ **down to** 1
  $u_k :=$ a minimum degree vertex of G_k;
  $G_{k-1} := G_k - u_k$
end for;
$\tilde{\sigma} := (u_1, u_2, \ldots, u_n)$
end algorithm.

The (generally not unique) output sequence $\tilde{\sigma} = (u_1, \ldots, u_n)$ of Algorithm *MiniDEO* is called a *minimum degree elimination order*. Such sequences have been studied in [1] with respect to the problem of finding a so-called *weak k-visit* of a graph. Also minimum degree elimination orders can be non-optimal (see Fig. 1), although they fulfil the necessary conditions for optimality 2-5.

5 An Exact Algorithm

Linear ordering problems admit several formulations in terms of 0-1 linear programming. Among them, *tournaments formulations* [6] can be enforced by several known facets and valid inequalities, see [2]. However, some manipulations are needed to use tournaments variables in Problem 1, since the objective function is quadratic, and the weakness of the lower bounds obtained by the resulting formulation discourages the application of an LP-based branch-and-bound algorithm. For this reason we here focus on a combinatorial enumeration scheme.

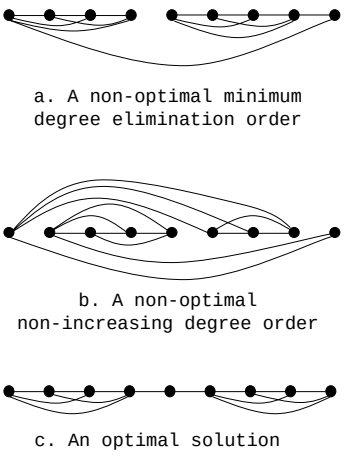


Fig. 1. Both non-increasing degree and minimum degree elimination orders can be non-optimal.

5.1 A Combinatorial Branch-and-Bound Scheme

The implicit enumeration of the search space is done on a tree, where the i-th level ($i = 1, \ldots, n-1$) identifies all the partial solutions σ_i assigning i vertices of G to the last i positions. Branching corresponds to deciding the $(n-i)$-th vertex in the order. Hence, a partial solution has in general $(n-i)$ branches, that is as many as the order of the subgraph G_{n-i} induced on G by the vertices whose order is still undetermined. The search space is reduced both by upper and lower bounds, and by applying dominating rules directly deriving from Propositions. 3-5. The exploitation of the above rules allows us to determine optimal solution on sparse (dense) random graphs with up to 100 (50) nodes.

Upper and Lower Bounds. Algorithm MiniDEO yields the upper bound. Lower bounds are computed by applying Lemma 1, and relaxing the topology of G:

– **MinLA Lower Bound:** Let σ^* be an optimal solution, $\hat{\sigma}$ a lower bound for MINIMUM LINEAR ARRANGEMENT, and $\tilde{\sigma}$ a minimum degree elimination order of G. Since $\min_\sigma\{\delta_G(\sigma)\} = \delta_G(\tilde{\sigma})$, by Lemma 1 a lower bound is given by

$$\mathrm{LB}_{\mathrm{LA}}(G) = \frac{1}{2}[\xi_G(\hat{\sigma}) + \delta_G(\tilde{\sigma})] \tag{14}$$

For a survey about lower bounds for MINIMUM LINEAR ARRANGEMENT problem see Silvestre [8].

– **Clique Lower Bound:** If $d > 0$ is the minimum degree of G, by Proposition 2 the number of entering arcs on the last $n - d$ vertices is at least

$$e_n = d, e_{n-1} = d - 1, \ldots, e_{n-d} = 0 \tag{15}$$

A lower bound is then obtained by compacting the $\bar{m} = m - \sum_{i=1}^{d} i$ remaining arcs as far as possible toward the order. Let K be the clique with $\bar{n} = \lfloor \frac{1+\sqrt{1+8\bar{m}}}{2} \rfloor$ nodes. The lower bound is given by the optimal cost of K, plus the cost of the remaining arcs computed on node $\bar{n}+1$, plus the cost of (15):

$$\mathrm{LB_C}(G) = \sum_{i=1}^{\bar{n}} [i \cdot (i-1)] + (\bar{n}+1) \cdot \left(\bar{m} - \sum_{i=1}^{\bar{n}-1} i \right) + \sum_{i=1}^{d} [i \cdot (n-d+i)] \tag{16}$$

Summarizing, the lower bound is $\mathrm{LB} = c(\sigma_i) + \max\{\mathrm{LB_{LA}}(G_{n-i}), \mathrm{LB_C}(G_{n-i})\}$.

Dominating Rules. Propositions 3 through 5 can be exploited to reduce the search space of the enumeration scheme. At level i, let v be the vertex fixed at the $(n-i+1)$-th position and denote with G_{n-i} the subgraph induced on G by the vertices whose order is still undetermined. Then

– by Proposition 3, the search for the $(n-i)$-th vertex is restricted to the vertices of $G_{n-i+1} = G_{n-i} \cup \{v\}$ having degree greater than or equal to the degree of v in G_{n-i+1}.
– By Proposition 4, if $N(u) \supset N(w)$, then u can be cut off, for each pair of vertices u, w of G_{n-i}. If $N(u) = N(w)$, then u can also be cut off if $label(u) < label(w)$ by applying a lexicographic rule.
– By Proposition 5, if d is the minimum degree of G_{n-i} then we can restrict the search to all the vertices of G_{n-i} with degree $< \frac{\sqrt{1+8(n-i)(d+1)}-1}{2}$.

A further dominating rule is to maintaining, among all the partial solutions with the same set of fixed vertices, only the one with minimum cost.

References

1. Arbib, C., M. Flammini, and E. Nardelli, "How to Survive while Visiting a Graph", *Discrete Applied Mathematics* **99** (2000) 279–293.
2. Borndörfer, R., and R. Weismantel, "Set Packing Relaxations of Some Integer Programs", *Mathematical Programming* **88**:3 (2000) 425–450.
3. Dyckhoff, H., G. Scheithauer, and J. Terno, "Cutting and Packing: an Annotated Bibliography", `ibl@ibl.RWTH-aachen.de` (1996).
4. Garey, M.R., and D.S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-completeness*, Freeman & Co., S. Francisco (1979).
5. Garey, M.R., D.S. Johnson, and L. Stockmeyer, "Some Simplified NP-complete Problems", *Theoretical Computer Science*, **1** (1976) 237–267.
6. Grötschel, M., M. Jünger and G. Reinelt, "A Cutting Plane Algorithm for the Linear Ordering Problem", *Operations Research*, **32**:6 (1984) 1195–1220.
7. Pinedo, M., *Scheduling: Theory Algorithms and Systems*, Prentice Hall, Englewood Cliffs, New Jersey (1995).
8. Silvestre, J.P., "Layout Problems", Ph.D. Thesis (2001).

Searching Is Not Jumping

Lali Barrière[1], Pierre Fraigniaud[2], Nicola Santoro[3], and Dimitrios M. Thilikos[4]

[1] Dept. de Matemàtica Aplicada IV, UPC, Barcelona, Spain,
lali@mat.upc.es
http://www-mat.upc.es/~{}lali
[2] CNRS, Laboratoire de Recherche en Informatique, Université Paris-Sud, France,
pierre@lri.fr
http://www.lri.fr/~{}pierre
[3] School of Computer Science, Carleton University, Canada,
santoro@scs.carleton.ca
[4] Dept. de Llenguatges i Sistemes Informàtics, UPC, Barcelona, Spain,
sedthilk@lsi.upc.es,
http://www.lsi.upc.es/~{}sedthilk

Abstract. This paper is concerned with the *graph searching* game: we are given a graph containing a fugitive (or lost) entity or item; the goal is to clear the edges of the graph, using searchers; an edge is clear if it cannot contain the searched entity, contaminated otherwise. The *search number* $s(G)$ of a graph G is the smallest number of searchers required to "clear" G. A search strategy is *monotone* (m) if no recontamination ever occurs. It is *connected* (c) if the set of clear edges always forms a connected subgraph. It is *internal* (i) if the removal of searchers is not allowed (i.e., searchers can not jump but only move along the edges). The difficulty of the "connected" version and of the "monotone internal" version of the graph searching problem comes from the fact that none of these problems is minor closed for arbitrary graphs, as opposed to all known variants of graph searching. We prove that there is a unique chain of inequalities linking all the search numbers above. More precisely, for any graph G, $s(G) = is(G) = ms(G) \leq mis(G) \leq cs(G) = ics(G) \leq mcs(G) = mics(G)$. The first two inequalities can be strict. Motivated by the fact that connected graph searching and monotone internal graph searching are both minor closed *in trees*, we provide a complete characterization of the set of trees that can be cleared by a given number of searchers. In fact, we show that, in trees, there is *exactly one* obstruction for monotone internal search, as well as for connected search, and this obstruction is the same for the two problems. This allows us to prove that, for any tree T, $mis(T) = cs(T)$ and $cs(T) \leq 2\ s(T) - 2$, using that $ics(T) = mcs(T)$. This implies that there are only two different search numbers, and these search numbers differ by a factor of 2 at most.

1 Introduction

Imagine a group of $k+1$ friends visiting a large bookstore, and assume that one of them gets separated from the k others, who are now looking for him in the store.

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 34–45, 2003.

The search for the lost friend is made difficult by the height and placement of the shelves, and by the complex topology of the store occupying several buildings, on several floors, connected by many stairs and bridges. It is also made difficult by the behavior of the lost friend who, as opposed to what is recommended in this situation, starts moving sporadically in the store, also looking for his friends. The question that arises among the group of k "searchers" is whether they are enough to eventually find their "fugitive" friend. For instance, if the bookstore would be displayed as a path, then $k = 1$ searchers would be enough. But if the bookstore is displayed as a ring, then 2 searchers are required, otherwise the fugitive could perpetually escape from the unique searcher. Let G be the graph corresponding to the map of the bookstore. We address the problem of computing the minimum number of searchers required to find the fugitive in G, and determining the strategy they have to follow to achieve the goal. This problem is known as *graph searching*. However, the search strategy developed by our group of friends must satisfy an additional crucial property: it must be "internal", in the sense that searchers must follow the corridors of the bookstore, as they cannot jump over the shelves, nor pass through the walls. The strategy should have also other desirable properties: it should be "monotone", in the sense that searchers don't want to check several times the same part of the bookstore; and it should be "connected" as searchers certainly prefer not to split in several groups that could lose contact from each other.

The searchers consult the literature available at the bookstore. They find that, according to Lapaugh's theorem [10], monotonicity can be assumed for free. However they observe that this monotone strategy is not internal nor necessarily connected. Hence, the group of searchers start doubting whether the classic definition of graph searching is realistic. Subsequently they start wondering how much does it cost (in term of number of searchers) to impose internality. In this paper, we show that it may cost *more* than the classic search but *not more* than imposing connectedness: if k searchers can find the fugitive according to a connected strategy, then they can also find it according to a monotone internal strategy. This paper studies more thoroughly the relationships between monotone, internal, and connected search strategies.

1.1 Statement of the Problem

More formally, in the *graph searching* problem we are given a graph whose edges are all "contaminated", and a set of "searchers". The goal is to obtain a state of the graph in which all edges are simultaneously "clear". To clear an edge $e = \{u, v\}$, a searcher must traverse the edge from one end-point u to the other end-point v. A clear edge is preserved from recontamination if either another searcher remains in u, or all other edges incident to u are clear. In other words, a clear edge e is recontaminated if there exists a path between e and a contaminated edge, with no searcher on any node of the path. In the standard setting of the graph searching problem, the basic operations, called *search steps*, are the following: (1) place a searcher on a node, (2) move a searcher along an edge, and (3) remove a searcher from a node.

Graph searching is the problem of developing a *search strategy*, that is a sequence of search steps that results in all edges being simultaneously clear. A search strategy for a graph G is *minimal* if it uses the smallest number of searchers, which is called the *search number* of G, denoted by $s(G)$. As far as practical applications are concerned (e.g., decontaminating a set of tunnels, capturing an intruder in a network, rescuing a speleologist in a maze of caves, etc.), the line of investigation is the determination of efficient search strategies satisfying additional properties, which are desirable or even necessary for some applications. Three properties are of particular interest.

Internal Search. A search strategy is *internal* if, once placed, searchers can only move along the graph edges (i.e., they cannot be removed and placed somewhere else). It is easy to see that this is equivalent to the case where operation (3) is not allowed. The minimum number of searchers for which an internal search strategy exists is denoted by $is(G)$.

Monotone search. A search strategy is *monotone* if no recontamination ever occurs. Hence each edge should be cleared only once. The minimum number of searchers for which a monotone search strategy exists is denoted by $ms(G)$.

Connected search. A search strategy is *connected* if the set of clear edges is always connected. The minimum number of searchers for which a connected search strategy exists is denoted by $cs(G)$.

Obviously, for any G, $is(G) = s(G)$ because the removal of a searcher (operation (3)) from a node u, and its placement (operation (1)) later at node v, can be replaced by a sequence of moves (operation (2)) from u to v. Lapaugh's theorem [10] says that, for any G, $ms(G) = s(G)$, that is recontamination does not help. Interestingly, for internal search, recontamination does help, i.e., there are graphs G for which $is(G) < mis(G)$, for instance the 22-node tree T^* obtained from three copies of the complete binary tree of depth 2, by joining their roots to a new vertex. Similarly, it is easy to check that $s(T^*) < cs(T^*)$, that is non-connectedness helps too. Now, it can be desirable to mix the three properties, monotonicity, internality, and connectedness, resulting in the search numbers mis, mcs, ics, and $mics$. Obviously, for any G, $cs(G) = ics(G)$, and $mcs(G) = mics(G)$ because once a strategy is connected, it is easy to make it internal as well. It is known [1] that, for trees, all the four numbers cs, mcs, ics, and $mics$ collapse into one because $mcs(T) = cs(T)$ for any tree T, i.e., recontamination does not help for connected search *in trees*. Surprisingly, unlike the case of monotone strategies for which there exist detailed studies and characterizations, very little is known about connected search strategies and monotone internal strategies. Unfortunately, the existing techniques and results for the many variants of the problem (cf. Section 1.3) not only cannot be employed but do not even provide any direct insight on these two important properties.

1.2 Our Results

In this paper, we prove a strong difference between traditional search and both connected and monotone internal searches: *connected searches and monotone internal searches are not minor closed*. Nevertheless, we show that there is a *unique*

chain of inequalities linking all the search numbers above. More precisely, for any graph G, $s(G) = is(G) = ms(G) \leq mis(G) \leq cs(G) = ics(G) \leq mcs(G) = mics(G)$. Some of this equations are obvious or known (see Section 1.1). The main result in this paper is the proof of $mis(G) \leq cs(G)$. To obtain this result, we extend the notion of crusades defined by Bienstock and Seymour [4], and use it in a novel way. In fact, we employ it not to prove monotonicity, but to transform a connected strategy into a monotone internal one with the same number of searchers. We also show that the first two inequalities can be strict. In particular, $mis \neq cs$. The main open problem is whether the last inequality can be strict or not.

On the other hand, it is easy to see that for all mentioned search problems, the class of *trees* that can be cleared with up to k searchers *is minor closed*. Therefore, the figure can be more precisely stated. We prove that, for any tree T, $cs(T) \leq 2\ s(T) - 2$, that is, for any tree T there exists a *monotone connected internal* search strategy for T using at most $2\ s(T) - 2$ searchers, and the upper bound is tight. We also show that $mis(T) = cs(T)$ for any tree T. This and the result in [1] imply that there are only two different search numbers in total for trees. These search numbers differ by a factor of 2 at most. We summarize the situation for trees by the equalities $s(T) = is(T) = ms(T)$ and $mis(T) = cs(T) = ics(T) = mcs(T) = mics(T)$, and the inequalities $s(T) \leq cs(T) \leq 2\ s(T) - 2$. We provide a complete characterization of the set of trees that can be cleared by k searchers. This characterization is given both explicitly, in terms of *k-caterpillars* (related to the notion of caterpillar dimension of [12]), and implicitly in terms of minimal forbidden minors. In fact, we show that, in trees, there is *only one obstruction* for monotone internal search, as well as for connected search, and the obstructions for the two problems are identical. This must be contrasted with the fact that, for traditional search, the number of obstructions in trees is *super-exponential* in the number of searchers [15,20].

1.3 Previous Works

Graph searching refers to a problem that has been throughly and extensively investigated in the literature, and that describes a variety of application scenarios [5,15,16]. In particular, it arises in VLSI design, (see, e.g., [7]), it is also related to network security for its relation with the capture of an intruder by software agents [1], and protection from mobile eavesdroppers [9]. Moreover, the problem and its variants, i.e., *node-search*, *mixed-search*, *inert-search*, etc., are closely related to standard graph parameters and concepts, including treewidth, cutwidth, pathwidth, and linear-width [3,18,21]. For instance, $s(G)$ is equal to the cutwidth of G for all graphs of maximum degree 3 (see [13]), and is equal to the vertex separation of the 2-expansion of G for all graphs (see [6]). For more information on graph searching, we refer the reader to the references in the full version of the paper [2].

Determining whether $s(G) \leq k$ for arbitrary G and k, is NP-complete [14]. Not surprisingly, the research has focused on restricted classes of graphs (e.g., [13, 19]), and on bounded search numbers (e.g., see [15,20,21]). In particular, for any

fixed k, the class of graphs that can be cleared with up to k searchers is minor closed. Therefore, there is a finite number of *obstructions* for this class [17]. Hence, there is a polynomial-time algorithm for testing whether an arbitrary graph G satisfies $s(G) \leq k$, for a fixed k. Of course, the algorithm requires the knowledge of the whole set of obstructions. Unfortunately, the number of obstructions for search grows super-exponentially with k, even for trees [15,20]. More precisely, for any k, there are at least $(k!)^2$ obstructions for the class of trees T such that $s(T) \leq k$.

The importance of monotone searching arises in applications where the cost of clearing an edge by far exceeds the cost of traversing an edge. Lapaugh [10] has proved that for every G there is always a monotone search strategy that uses $s(G)$ searchers. A similar positive result exists also for node-search and mixed-search [3,4]. The necessity for connectedness arises, e.g., in applications where communication between the searchers can occur only within completely clear areas of the network. Hence connectedness is required for their coordination. Safety is another motivation for connectedness, as it would always ensure the presence of secure routes between all the searchers. The problem of determining minimal search strategies under the connectedness and/or the internality constraint is still NP-complete in general (it follows from the reduction in [14], as observed in [1]). It has been shown in [1] that minimal connected strategies can however be computed in linear time for trees. The removal of a searcher from a node x, and the placement of this searcher in another node y, might be difficult or impossible to implement. In fact, it assumes that a searcher is able to go "out of the system" and to reenter the system elsewhere. This assumption is clearly unrealistic e.g., in the case of software mobile agents. In this case the searchers can only move in the network from site to neighboring site. Actually, it does not hold even in the original setting of a maze of caves [15]. Hence the importance of internal search strategies. There are trees for which minimal internal search strategies require $\Omega(n \log n)$ moves (i.e., edge traversal) [14], whereas, if the removal of searchers (and their arbitrary placement somewhere else) is allowed, then, for any graph G, there exists a search strategy that requires at most $O(m)$ moves in G [10], where m is the size of G.

2 Connected vs. Monotone Internal Graph Searching

In this section, we show the following.

Theorem 1. *For every graph G, $mis(G) \leq cs(G)$.*

To prove this result, we use the concept of *crusade* introduced by Bienstock and Seymour in a novel way, through the new concepts of skeleton and consistency.

According to [4], given a graph $G = (V, E)$, a sequence $(X_0, X_1, \ldots, X_r)$ of subsets of edges is a *crusade* if $X_0 = \varnothing$, $X_r = E$, and $|X_i \setminus X_{i-1}| \leq 1$ for any $1 \leq i \leq r$. For a set X of edges in a graph G, denote by $\delta(X)$ the set of nodes in G having at least one incident edge in X, and at least one incident edge not

in X. The *frontier* of a crusade $(X_0, X_1, \ldots, X_r)$ is $\max_{1 \leq i \leq r} |\delta(X_i)|$. A crusade is *progressive* if $X_0 \subseteq X_1 \subseteq \ldots \subseteq X_r$ and $|X_i \setminus X_{i-1}| = 1$ for $1 \leq i \leq r$.

We say that a crusade is *connected* if the subgraph induced by X_i is connected for any $1 \leq i \leq r$. One can easily check that the sequence $(X_0, X_1, \ldots, X_r)$ of subsets of edges such that $X_0 = \varnothing$, and X_i is the set of clear edges after step i of a connected search strategy using $\leq k$ searchers is a connected crusade of frontier at most k. Therefore, we have:

Lemma 1. *If $cs(G) \leq k$ then there exists a connected crusade of frontier at most k in G.*

Given a crusade $C = (X_0, X_1, \ldots, X_r)$, we define the *skeleton* $\mathcal{S}$ of C as the directed graph of $r+1$ levels L_i, $i = 0, \ldots, r$ such that L_i consists of as many nodes as the number of connected components of X_i. There are edges only between levels of consecutive indices in $\mathcal{S}$ (i.e., the nodes in each L_i are independent). More precisely, there is an edge from the node $a \in L_i$, representing a connected component A of X_i, to the node $b \in L_{i+1}$, representing a connected component B of X_{i+1}, if and only if one of the three following properties holds: (1) $X_{i+1} \setminus X_i = \varnothing$, and $B \subseteq A$; (2) $X_{i+1} \setminus X_i = e_i \notin B$ and $B \subseteq A$; (3) $X_{i+1} \setminus X_i = e_i \in B$ and one of the (at most two) connected component(s) of $B \setminus \{e_i\}$ is included in A.

Note that the out-degree of a node in a skeleton $\mathcal{S}$ can be greater than 1 because a connected component of X_i can split in several connected components of X_{i+1} due to recontamination. On the other hand, the in-degree of a node is at most 2 because $|X_{i+1} \setminus X_i| \leq 1$ (i.e., there is at most one new clear edge in X_{i+1}). Hence at most two distinct connected components of level i can merge into a unique component at level $i+1$. More precisely, there is at most one node of in-degree 2 at every level of $\mathcal{S}$, and all the other nodes have in-degree ≤ 1. Note also that all nodes in a skeleton of a progressive crusade have out-degree 1 because $X_i \subseteq X_{i+1}$, and hence a connected component never splits.

We denote by $\Gamma^+(u)$ (resp., $\Gamma^-(u)$) the set of edges in $\mathcal{S}$ out-going from (resp., incoming to) node $u \in \mathcal{S}$. A node $u \in \mathcal{S}$ represents a set of edges X in G. Hence, by extension, we denote by $\delta(u)$ the set of nodes in $\delta(X)$. A crusade C is *k-consistent* if its frontier is at most k, and every node u (resp., edge e) of its skeleton $\mathcal{S}$ can be labeled by a positive integer k_u (resp., k_e) satisfying the three following conditions: (1) $k_u \geq |\delta(u)|$ for every $u \in \mathcal{S}$; (2) $\sum_{u \in L_i} k_u \leq k$ for every level L_i; and (3) $\sum_{e \in \Gamma^+(u)} k_e = k_u \geq \sum_{e \in \Gamma^-(u)} k_e$.

Observe that the skeleton $\mathcal{S}$ of a connected crusade C is a path. Labeling every node and edge of $\mathcal{S}$ by k makes it k-consistent. Therefore, a connected crusade of frontier at most k is k-consistent. The main reason for introducing consistent crusades is actually the following lemma.

Lemma 2. *If there exists a k-consistent crusade in G, then there exists a progressive k-consistent crusade in G.*

Proof. The proof is inspired by (2.2) in [4]. Among all k-consistent crusades, choose a crusade $C = (X_0, X_1, \ldots, X_r)$ satisfying: (C1) $\sum_{i=0}^{r} (|\delta(X_i)| + 1)$ is

minimum, and (C2) $\sum_{i=0}^{r} |X_i|$ is minimum subject to (C1). Let us show that this crusade is progressive. First, we show that $|X_i \setminus X_{i-1}| = 1$ for every $i \geq 1$. For the purpose of contradiction, let i be such that $|X_i \setminus X_{i-1}| = 0$, i.e., $X_i \subseteq X_{i-1}$. Then

$$C' = (X_0, X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_r)$$

is a crusade of frontier $\leq k$. Let us show that C' is k-consistent. In a skeleton, the out-neighbors of a node u are called the children of u, and the out-neighbors of the children of u are called its grandchildren. We define similarly the notion of parents and grandparents. The skeleton $\mathcal{S}'$ of C' can be obtained from the skeleton $\mathcal{S}$ of C by removing level i, and connecting every node of L_{i-1} to its grandchildren in $\mathcal{S}$ (see Figure 1). We show that we can label $\mathcal{S}'$ so that the three conditions for a crusade to be consistent are satisfied. The node-labeling of $\mathcal{S}'$ is the node-labeling of $\mathcal{S}$. The edge-labeling of $\mathcal{S}'$ is the edge-labeling of $\mathcal{S}$, but between L_{i-1} and L_{i+1}. Edges from L_{i-1} to L_{i+1} are labeled as follows. Let $v \in L_{i+1}$. Let u be a grandparent of v in $\mathcal{S}$. There can be at most two distinct paths from u to v in $\mathcal{S}$ because the in-degree of v is at most 2. The edge (u, v) of $\mathcal{S}'$ receives the label of (w, v) of $\mathcal{S}$ if there is a unique path (u, w, v) from u to v in $\mathcal{S}$. It receives the sum of the labels of (w, v) and (w', v) if there are two paths (u, w, v) and (u, w', v) from u to v in $\mathcal{S}$. Since $|X_i \setminus X_{i-1}| = 0$, there is no node of in-degree 2 in L_i of $\mathcal{S}$, and thus this labeling gives k-consistency to $\mathcal{S}'$. Hence C' is a k-consistent crusade contradicting (C1). Therefore $|X_i \setminus X_{i-1}| = 1$ for every $i \geq 1$.

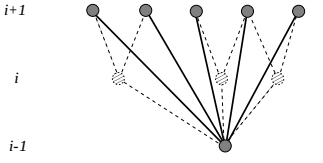


Fig. 1. Skeleton $\mathcal{S}'$.

Next, we show that $X_{i-1} \subseteq X_i$ for every $i \geq 1$. For that purpose, we first show that $|\delta(X_{i-1} \cup X_i)| \geq |\delta(X_i)|$ for every $i \geq 1$. For the purpose of contradiction, assume that there exists i such that $|\delta(X_{i-1} \cup X_i)| < |\delta(X_i)|$, and let $C'' = (X_0, X_1, \ldots, X_{i-1}, X_{i-1} \cup X_i, X_{i+1}, \ldots, X_r)$.

C'' is a crusade of frontier $\leq k$. Let us show that it is k-consistent. As for the skeleton $\mathcal{S}'$ of C', the skeleton $\mathcal{S}''$ of C'' can be obtained from the skeleton $\mathcal{S}$ of C. Replace L_i by a copy L'_i of L_{i-1}, and place edges from each node in L_{i-1} to its copy L'_i (see Figure 2). If the edge $X_i \setminus X_{i-1}$ merges two components of X_{i-1}, then the corresponding two nodes of L'_i are merged into one node. Finally, there is an edge from node $u' \in L'_i$ to all the grandchildren (in $\mathcal{S}$) of its copy $u \in L_{i-1}$. In Figure 2, there are three nodes displayed at level L_{i-1}. There are six nodes in L_i which are replaced by three copies of the three nodes of L_{i-1}.

The two copies of u' and u'' are merged into a single node u because the two components corresponding to u' and u'' are connected by $X_i \setminus X_{i-1}$.

We show that we can label $\mathcal{S}''$ so that the three conditions for a crusade to be consistent are satisfied. The node-labeling of $\mathcal{S}''$ is the node-labeling of $\mathcal{S}$ for all nodes of levels $j \neq i$. If a node $u \in L'_i$ does not result from the merging of two nodes u' and u'', then u receives the label of its original in L_{i-1}. Otherwise u receives the sum of the labels of the originals of u' and u'' in L_{i-1}. The edge-labeling of $\mathcal{S}'$ is the edge-labeling of $\mathcal{S}$ except between levels $i-1$, i, and $i+1$. The edge out-going from any node u of L_{i-1} receives label k_u. The setting of the edge-labeling between levels i and $i+1$ is slightly more complex. (Recall that there is at most one node of in-degree 2 at every level.) At levels i and $i+1$, there is a one-to-one correspondence between edges incoming to nodes with in-degree 1 in $\mathcal{S}''$ and edges incoming to nodes with in-degree 1 in $\mathcal{S}$. Thus, the edge incoming to a node at level $i+1$, with in-degree 1 in $\mathcal{S}''$, receives the label of its corresponding edge in $\mathcal{S}$. Let v be a node at level $i+1$ of $\mathcal{S}$, with in-degree 2. If v is still of in-degree 2 in $\mathcal{S}''$ (like in Figure 2), then the two incoming edges of $\mathcal{S}''$ take the same labels as the corresponding edges in $\mathcal{S}$. Otherwise, the unique edge incoming to node v in $\mathcal{S}''$ takes the sum of the labels of the two edges incoming to node v in $\mathcal{S}$. One can easily check that this labeling gives k-consistency to $\mathcal{S}''$. Hence, C'' is a k-consistent crusade, in contradiction with (C1). Therefore, for every $i \geq 1$, $|\delta(X_{i-1} \cup X_i)| \geq |\delta(X_i)|$.

Now, any pair of edge-sets A and B satisfies submodularity, i.e., $|\delta(A \cap B)| + |\delta(A \cup B)| \leq |\delta(A)| + |\delta(B)|$. Hence, $|\delta(X_{i-1} \cap X_i)| \leq |\delta(X_{i-1})|$ for any $i \geq 1$. Let $C''' = (X_0, X_1, \ldots, X_{i-2}, X_{i-1} \cap X_i, X_i, \ldots, X_r)$.
C''' is a crusade of frontier at most k. We show that it is k-consistent using the same arguments as for C' and C'' (see [2] for more details). From (C2), we then get $|X_{i-1} \cap X_i| \geq |X_{i-1}|$, that is $X_{i-1} \subseteq X_i$. Therefore C is a progressive k-consistent crusade, which completes the proof.

Lemma 3. *Let G be a graph such that every edge has one of its extremities incident to only one other edge. If there is a progressive k-consistent crusade in G, then $mis(G) \leq k$.*

The proof proceeds by transformation of a progressive k-consistent crusade $C = (X_0, X_1, \ldots, X_r)$, with skeleton $\mathcal{S}$ consistently labeled, into a monotone internal search strategy that successively clears the edges $e_1, e_2, \ldots, e_r$, where $e_i = X_i \setminus X_{i-1}$. Intuitively, k_u represents the number of searchers in the connected component represented by u. The labels k_e, $e \in \Gamma^-(u)$, represent how the searchers are distributed among the possibly two connected components whose merging results in the component represented by u. Due to lack of space, the proof is omitted. It can be found in [2].

Proof of Theorem 1. Let G be any graph with $cs(G) \leq k$. The 1-expansion of G is the graph H obtained from G by replacing every edge e by two consecutive edges e' and e''. We have $cs(H) \leq cs(G)$. Therefore, thanks to Lemma 1, there exists a connected, and hence consistent, crusade of frontier $\leq k$ in H. Applying

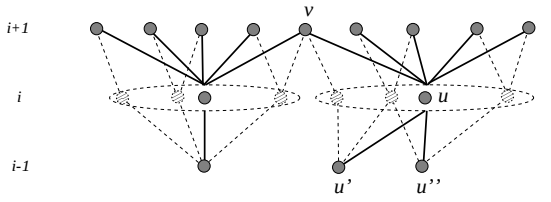


Fig. 2. Skeleton $\mathcal{S}''$.

Lemma 2, we get that there exists a progressive k-consistent crusade in H, and thus, by Lemma 3, $mis(H) \leq k$. We complete the proof by observing that $mis(G) \leq mis(H)$. □

We conclude the section by noticing that cs and mis can differ significantly for some graphs, and thus that the inequality of Theorem 1 can be strict. Figure 3 displays a subgraph G_0 of the $p \times q$ mesh, with $q = 2k$, and $p = 6l$, $l \gg k$. One can check that $mis(G_0 - e - f)/s(G_0 - e - f) \simeq 3/2$, and $cs(G_0 - e)/mis(G_0 - e) \simeq 3/2$. The graph G_0 can also be used to show that the class of graphs that can be cleared by a connected (resp., monotone internal) search strategy using at most k searchers is *not* minor closed (see [2] for more details). However, one can easily check that it is minor closed when restricted to the class of trees.

3 The Case of Trees

In this section, we show that there is a unique obstruction for the class of trees T such that $cs(T) \leq k$. Since, for any tree T, $mcs(T) = cs(T)$ [1], we consider only the monotone case. Our proof is based on the notion of *k-caterpillar* and *spine*. A spine is a path. A 0-caterpillar is also a path, and it is its own spine. For $k > 0$, a tree T is a k-caterpillar with spine P if, for every connected component T' of $T \setminus P$, the two following properties hold: (1) there is a path P' such that T' is a $(k-1)$-caterpillar with spine P', and (2) one of the two extremities of P' is adjacent to P. A 1-caterpillar is hence a subdivision of a caterpillar in the usual sense, i.e., a path $x_1, \ldots, x_k$ with $k_i \geq 0$ paths pending from every x_i. Notice that any tree is a k-caterpillar for k large enough. The notion of k-caterpillar is related to the notion of *caterpillar dimension* introduced in [12] (see also [11]). We establish an equivalence between connected search numbers and k-caterpillars.

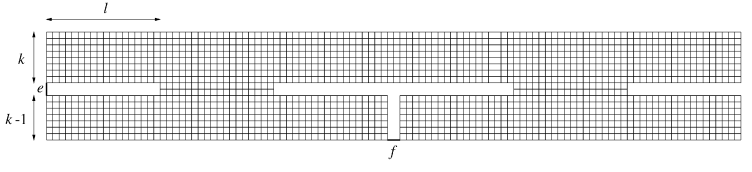


Fig. 3. The graph G_0

Given a tree T and two vertices v, w of T, we denotes by T_v the tree T rooted at v, and by $T_v[w]$ the subtree of T_v rooted at w. Recall that the depth of a rooted tree T is the maximum distance from its root to the leaves. We denote by B_k the complete binary tree of depth k, and by D_k the tree obtained by connecting the three roots of three copies of B_{k-1} to a unique new vertex. Finally, we denote by $T_1 \preceq T_2$ the relation "T_1 is a minor of T_2".

Theorem 2. *For any tree T, the following three properties are equivalent:*
(1) T is not a $(k-1)$-caterpillar;
(2) $D_k \preceq T$;
(3) $cs(T) \geq k+1$.

Proof. We first prove (1)⇒(2). Let T_1 and T_2 be two trees, rooted at x_1 and x_2 respectively. We denote by $T_1 \preceq_{x_2} T_2$ the relation "T_1 is a x_2-rooted minor of T_2", that is node x_1 is either x_2 or the result of contracting a series of edges, some of them containing x_2. Now, let T be a tree and v be a vertex of T such that $B_k \not\preceq_v T$, $k \geq 1$. We claim that T is a $(k-1)$-caterpillar and v is an extremity of its spine. The proof of this claim is by induction on k. If $B_1 \not\preceq_v T$ then T is a path with extremity v. If $k > 1$ and there is a vertex v such that $B_k \not\preceq_v T$, then there are two cases. If $B_{k-1} \not\preceq_v T$, then by induction hypothesis, T is a $(k-2)$-caterpillar with v as the first vertex of the spine. If $B_{k-1} \preceq_v T$, then let S be the set of vertices w such that $B_{k-1} \preceq_w T_v[w]$. S is a path starting at v, and all the connected components of $T - S$ are $(k-2)$-caterpillars, in which the corresponding spine starts at the vertex adjacent to one of the vertices of S in T. Indeed, if $z \notin S$ and z is adjacent to $w \in S$, then $T_v[z]$ is one of the connected components of $T - S$ and $B_{k-1} \not\preceq T_v[z]$. It hence just remains to show that there is a vertex v such that $B_k \not\preceq_v T$. By contradiction, assume that $D_k \not\preceq T$ and for every v vertex of T, $B_k \preceq_v T$. There is a vertex z with two neighbors, z_1 and z_2, such that $B_{k-1} \preceq_{z_1} T_z[z_1]$ and $B_{k-1} \preceq_{z_2} T_z[z_2]$. This implies that either $B_k \preceq_{z_i} T_z[z_i]$ or $B_k \preceq_z T_{z_i}[z]$. In both cases, we get $D_k \preceq T$, a contradiction.

Next, we prove (2)⇒(3) by showing that, for any connected search strategy in D_k, there is a step in which at least $k+1$ searchers are required to avoid recontamination. Finally, we prove (3) ⇒(1). We show by induction on k, that if T is a k-caterpillar with spine P, then there is a connected search strategy using $k+1$ searchers starting at one extremity of P. (See [2] for more details.)

Corollary 1. *For a tree T, $cs(T) \leq k$ if and only if T is a $(k-1)$-caterpillar. Moreover, the set of obstructions of the class of trees T with $cs(T) \leq k$ contains D_k as unique element.*

Corollary 2. *For any tree T, if $s(T) \geq 2$, then $s(T) \leq cs(T) \leq 2s(T) - 2$. Moreover, for $k \geq 1$, $cs(D_{2k-1}) = 2s(D_{2k-1}) - 2$.*

Proof. Let M_k be the tree obtained from a complete ternary tree of depth k by removing one leaf from every set of three sibling leaves. It is shown in [15] that M_k is an obstruction of the class of graphs G with search number $\leq k$. We

show that, for any $k \geq 1$, $M_k \preceq D_{2k-2}$. By Theorem 2, this implies that, for every T, $cs(T) \leq 2s(T) - 2$. To prove that the bound is tight, we show that $cs(D_{2k-1}) = 2s(D_{2k-1}) - 2$ (see [2]).

Theorem 3. *For any tree T, $mis(T) = cs(T)$.*

To prove this theorem, we only need to prove that $mis(D_k) = k + 1$ for all $k \geq 1$ because *mis* and *cs* are both minor closed for trees. This follows from the fact that, for every k, $mis(B_k) = k$, and in any search strategy for B_k using k searchers, there is a step in which (1) k searchers are involved, (2) none of these searchers occupies the root, and (3) all edges incident to the root of B_k are clear. (See [2] for more details.)

Remark. Distinct values for s and cs can be achieved for graphs of arbitrary connectivity. For instance, for a fixed r, the ratio $cs(D_k \times K_r)/s(D_k \times K_r)$ approaches 2 when k goes to infinity.

4 Concluding Remarks and Open Problems

The main open problem is whether recontamination helps for connected search; that is, whether, for any graph G with $cs(G) \leq k$, there exists a monotone connected search strategy using at most k searchers. As observed in [8], all the standard techniques for proving monotonicity *fail* for connected search. This is mainly because all the monotonicity proofs (in any search variant) use as a kernel argument the fact that the cost of the search can be expressed by a *connectivity function*; i.e., a nonnegative valued function on a set $S \subseteq \mathcal{P}(M)$ that is invariant over complement and satisfies the submodular property. However, the intersection of two connected sets is not necessarily connected; hence their failure for connected search.

Another important open problem is whether the ratio $cs(G)/s(G)$ can be bounded. We proved that for trees $cs(T)/s(T) < 2$ (cf. Corollary 2). A similar bound for general graphs, say $cs(G)/s(G) \leq b$, would imply that: $H \prec G \Rightarrow cs(H) \leq b \cdot cs(G)$. That way, we could derive approximation algorithms for *cs* from algorithms for the usual search number s.

Acknowledgements. The authors are thankful to Fedor Fomin for his valuable help and comments.

References

1. L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In 14th *ACM Symp. on Parallel Algorithms and Architectures* (SPAA '02), Winnipeg, August 10–13, 2002.
2. L. Barrière, P. Fraigniaud, N. Santoro, and D. M. Thilikos. Connected and Internal Graph Searching. Technical Report LSI-02-58-R, Universitat Politècnica de Catalunya, Barcelona, Spain, 2002. *http://www.lsi.upc.es/~sedthilk.*

3. D. Bienstock. Graph searching, path-width, tree-width and related problems. *DIMACS Series in Disc. Maths. and Theo. Comp. Sc.*, Vol. 5, 33–49, 1991.
4. D. Bienstock and P. Seymour. Monotonicity in graph searching. *Journal of Algorithms* 12, 239–245, 1991.
5. R. Breisch. An intuitive approach to speleotopology. *Southwestern Cavers* VI(5):72–78, 1967.
6. J. Ellis, H. Sudborough, and J. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–79, 1994.
7. M. Fellows and M. Langston. On search, decision and the efficiency of polynomial time algorithm. In 21st *ACM Symp. on Theory of Computing* (STOC '89), pp. 501–512, 1989.
8. F. Fomin and D.M. Thilikos. On the monotonicity of games generated by symmetric submodular functions. *Discrete Applied Mathematics*, to appear.
9. M. Franklin, Z. Galil, and M. Yung. Eavesdropping games: a graph theoretic approach to privacy in distributed systems. *Journal of the ACM*, 47(2):225–243, 2000.
10. A. Lapaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40(2):224–245, 1993.
11. N. Linial, A. Magen and M. Saks. Trees and Euclidian metrics. In 30st *ACM Symp. on Theory of Computing* (STOC '98), pages 169–175, 1998.
12. J. Matousek. On embedding trees into uniformly convex Banach spaces. *Israelian Journal of Mathematics*, 114:221–237, 1999.
13. F. Makedon and H. Sudborough. Minimizing width in linear layouts. In 10th *Int. Colloquium on Automata, Languages, and Programming* (ICALP '83), LNCS 154, Springer-Verlag, 478–490, 1983.
14. N. Megiddo, S. Hakimi, M. Garey, D. Johnson and C. Papadimitriou. The complexity of searching a graph. *Journal of the ACM*, 35(1):18–44, 1988.
15. T. Parsons. Pursuit-evasion in a graph. *Theory and Applications of Graphs*, Lecture Notes in Mathematics, Springer-Verlag, pages 426–441, 1976.
16. T. Parsons. The search number of a connected graph. In 9th *Southeastern Conference on Combinatorics, Graph Theory and Computing*, Utilitas Mathematica, pages 549–554, 1978.
17. N. Robertson and P. Seymour. Graph minors — A survey. *Surveys in Combinatorics*, Cambridge University Press, I. Anderson (ed.), pages 153–171, 1985.
18. P. Seymour and R. Thomas. Graph searching, and a min-max theorem for treewidth. *Jour. Combin. Theory, Ser. B*, 58:239–257, 1993.
19. K. Skodinis. Computing optimal linear layouts of trees in linear time. In 8th *European Symp. on Algorithms* (ESA '00), Springer, LNCS 1879, pages 403–414, 2000. (To appear in *SIAM J. Computing*.)
20. A. Takahashi, S. Ueno, and Y. Kajitani. Minimal acyclic forbidden minors for the family of graphs with bounded path-width. *Discrete Mathematics*, 127(1–3):293–304, 1994.
21. D. Thilikos. Algorithms and obstructions for linear-width and related search parameters. *Discrete Applied Mathematics* 105, 239–271, 2000.

Incremental Integration Tools for Chemical Engineering: An Industrial Application of Triple Graph Grammars

Simon M. Becker and Bernhard Westfechtel

Lehrstuhl für Informatik III, RWTH Aachen
Ahornstraße 55, D-52074 Aachen
{sbecker,bernhard}@i3.informatik.rwth-aachen.de

Abstract. Triple graph grammars, an extension of pair graph grammars, were introduced for the specification of graph translators. We developed a framework which constitutes an industrial application of triple graph grammars. It solves integration problems in a specific domain, namely design processes in chemical engineering. Here, different design representations of a chemical plant have to be kept consistent with each other. Incremental integration tools assist in propagating changes and performing consistency analysis. The integration tools are driven by triple rules which define relationships between design documents.

1 Introduction

Triple graph grammars, an extension of pair graph grammars [1], were introduced at the WG '94 workshop [2]. Originally, they were motivated by integration problems in software engineering; later, they were applied to other domains as well. In general, triple graph grammars may be used for the specification of graph translations, coupling of graph structures, and consistency maintenance.

This paper reports on an *industrial application* of triple graph grammars. The Collaborative Research Center IMPROVE [3] is concerned with the development of models and tools for chemical engineering design. In IMPROVE, we have realized a framework for building incremental and interactive integration tools [4,5]. The framework has been developed in close cooperation with an industrial partner (innotec, a Germany software company, which offers an engineering database system called COMOS PT).

In chemical engineering design, a chemical plant is described from different perspectives by a set of interrelated *design documents*, including various kinds of flow sheets for describing the chemical process and the components of the chemical plant, simulation models for steady-state and dynamic simulations, etc. Design proceeds incrementally, i.e., the design documents are gradually refined and improved. Throughout the whole design process, interrelated design documents have to be kept consistent with each other. Design documents may be represented as graphs in a natural way. Triple graph grammars are used to define correspondences between graph structures. They serve as specifications for rule-based integration tools.

Section 2 briefly recalls triple graph grammars. Section 3 introduces a motivating example from the chemical engineering domain. Section 4 derives general requirements

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 46–57, 2003.

from the motivating example. Section 5 presents our framework for building incremental and interactive integration tools. Section 6 explains how triple rules are defined in this framework. Section 7 is devoted to implementation issues. Section 8 discusses the way we have adapted and applied the triple graph grammar approach and the experiences we made. Section 9 compares related work. Section 10 presents a short conclusion.

2 Triple Graph Grammars

Pair graph grammars were introduced as early as 1971 by Pratt to specify graph-to-graph translations [1]. A pair grammar defines a set of pair productions which modify the participating graphs and update correspondences between nodes. *Triple graph grammars* [2] are an extension of pair graph grammars. They were motivated by the study of integration problems in software engineering environments [6]. These studies showed the need for a separate *correspondence graph* to be placed in between *source* and *target graph*. The terms "source" and "target" denote distinct ends, but do not imply a direction. A *triple production* consists of productions operating on source, correspondence, and target graph, respectively, as well inter-graph mappings which are used to relate elements of the correspondence graph to elements of the source and the target graph, respectively.

Let us briefly recall some definitions from [2]:

- A *graph* is a quadruple $G = (V, E, s, t)$, where V and E are finite sets of vertices (nodes) and edges, and $s, t : E \to V$ assign source and target nodes to edges.
- A *graph morphism* from G to G' is a pair $h = (h_V, h_E)$, where $h_V : V \to V'$, $h_E : E \to E'$ are defined such that they "preserve" source and target nodes.
- A (monotonic) *graph production* is a pair of graphs $p = (L, R)$, where $L \subset R$.
- A graph production p is *applicable* to a graph G if there is a morphism $h : L \to G$. *Application* of p results in a graph G' which is extended with (copies of) nodes and edges in $R \setminus L$.
- A *triple graph* is a structure $G = (SG \leftarrow h_{SG} - CG - h_{TG} \rightarrow TG)$, where SG, CG, and TG denote source, correspondence, and target graph, respectively, and h_{SG} and h_{TG} are graph morphisms.
- A *triple production* is a structure $p = (sp \leftarrow h_{sp} - cp - h_{tp} \rightarrow tp)$, where sp, cp, and tp denote source, correspondence and target productions, respectively. h_{sp} and h_{tp} are pairs of graph morphisms which map the left-hand and right-hand sides, respectively.
- A triple production p is *applicable* to a triple graph G if its component productions are applicable to the component graphs and the production mappings may be mapped onto the graph mappings. *Application* of p results in a triple graph G' such that the component productions are applied to the component graphs and the graph mappings are updated according to the production mappings.

Based on these definitions, the following propositions were proved in [2]:

- A given triple production
  $p = ((SL, SR) \leftarrow h_{sp} - (CL, CR) - h_{tp} \rightarrow (TL, TR))$
  may be split into *source-local production*

$p_{sl} = ((SL, SR) \leftarrow \epsilon - (\emptyset, \emptyset) - \epsilon \rightarrow (\emptyset, \emptyset))$
and a *source-to-target production*
$p_{st} = ((SR, SR) \leftarrow h_{sp} - (CL, CR) - h_{tp} \rightarrow (TL, TR))$
such that $p = p_{sl}\ p_{st}$.

– A sequence of applications of triple productions $p_1 \ldots p_n$ is equivalent to the application of all source-local productions $p_{1_{sl}} \ldots p_{n_{sl}}$, followed by the application of all source-to-target productions $p_{1_{st}} \ldots p_{n_{st}}$.

Triple graph grammars are used for the specification of graph-based integration tools which may be classified as follows:

Synchronous coupling. Source, correspondence, and target graph are modified synchronously by applying triple productions.

Source-to-target translation. Given a source graph SG and a sequence of source-local productions, apply source-to-target productions, yielding a correspondence graph CG and a target graph TG.

Incremental change propagation. Starting from a triple graph $G = (SG, CG, TG)$, first apply a sequence of source-local productions to SG and then propagate the changes to CG and TG by applying corresponding source-to-target productions.

In the context of this paper, we will focus on incremental change propagation, which in general may be performed bidirectionally.

3 Motivating Example

Incremental change propagation is essential in chemical engineering design, where chemical plants are described in design documents from different perspectives. Below, we focus on a problem which we have been studying in cooperation with an industrial partner. innotec, a Germany software company, offers an engineering database system called COMOS PT [7]. In particular, COMOS PT maintains *flow sheets* describing the chemical process and the composition of the chemical plant to be designed. The problem was to integrate COMOS PT with Aspen Plus [8], a simulation environment provided by another vendor. In Aspen Plus, *simulation models* are created (and executed) which have to be kept consistent with the corresponding flow sheets.

In chemical engineering, the flow sheet acts as a central document for describing the chemical process. The flow sheet is refined iteratively so that it eventually describes the chemical plant to be built. Simulations are performed in order to evaluate design alternatives. Simulation results are fed back to the flow sheet designer, who annotates the flow sheet with flow rates, temperatures, pressures, etc. Thus, information is propagated back and forth between flow sheets and simulation models. Unfortunately, the relationships between them are not always straightforward. To use a simulator such as Aspen Plus, the simulation model has to be composed from pre-defined blocks. Therefore, the composition of the simulation model is specific to the respective simulator and may deviate structurally from the flow sheet.

Figure 1 illustrates how an incremental integration tool assists in maintaining consistency between flow sheets and simulation models. The chemical process taken as

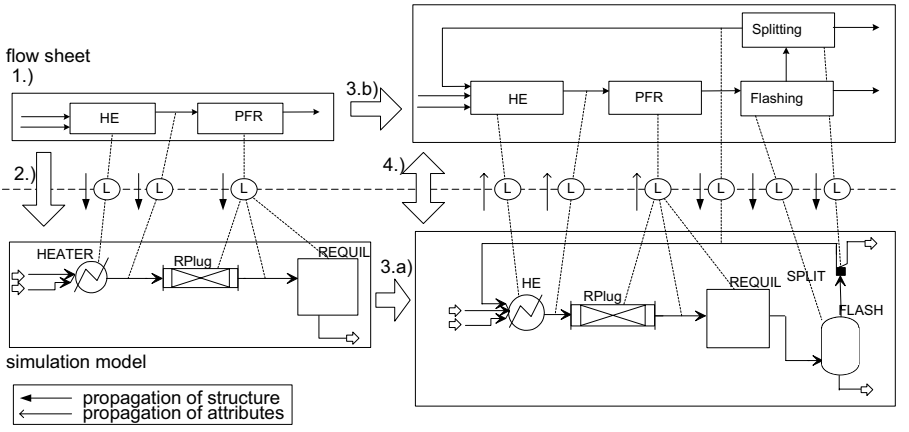


Fig. 1. Integration between flow sheet and simulation model

example produces ethanol from ethen and water. Flow sheet and simulation model are shown above and below the dashed line, respectively. The integration document for connecting them contains links which are drawn on the dashed line. The figure illustrates a design process consisting of four steps:

1. An initial flow sheet is created in COMOS PT. This flow sheet is still incomplete, i.e., it describes only a part of the chemical process (heating of substances and reaction in a plug flow reactor, PFR).
2. The integration tool is used to transform the initial flow sheet into a simulation model for Aspen Plus. Here, the user has to perform two decisions. While the heating step can be mapped structurally 1:1 into the simulation model, the user has to select the most appropriate block for the simulation to be performed. Second, there are multiple alternatives to map the PFR. Since a straightforward 1:1 mapping is not sufficient, the user maps the PFR into a cascade of two blocks.
3. The simulation is performed in Aspen Plus, resulting in a simulation model which is augmented with simulation results. In parallel, the flow sheet is extended with the chemical process steps that have not been specified so far (flashing and splitting).
4. Finally, the integration tool is used to synchronize the parallel work performed in the previous step. This involves information flow in both directions. First, the simulation results are propagated from the simulation model back to the flow sheet. Second, the extensions are propagated from the flow sheet to the simulation model. After these propagations have been performed, mutual consistency is re-established.

4 Requirements

From the motivating example presented in the previous section, we derive the following requirements:

Functionality. An integration tool must manage links between objects of inter-dependent documents. In general, links may be m:n relationships, i.e., a link connects m source objects with n target objects. They may be used for multiple purposes: *browsing*, *consistency analysis*, and *transformation*.

Mode of operation. An integration tool must operate incrementally rather than batch-wise. It is used to propagate changes between inter-dependent documents. This is done in such a way that only actually affected parts are modified. As a consequence, manual work does not get lost, as it happens in the case of batch converters.

Direction. In general, an integration tool may have to work in both directions. That is, if d_1 is changed, the changes are propagated into d_2 and vice versa.

Mode of interaction. While an integration tool may operate automatically in simple scenarios, it is very likely that user interactions are required to resolve non-deterministic choices.

Time of activation. In single-user applications, it may be desirable to have changes propagate eagerly. This way, the user is informed promptly about the consequences of the changes performed in the respective documents. In multi user scenarios, however, *deferred propagation* is usually required. In this case, each user keeps control of the export and import of changes from/to his local workspace.

Integration rules. An integration tool is driven by rules defining which *object patterns* may be related to each other. It must provide support for defining and applying these rules.

Traceability. An integration tool must record a trace of the rules which have been applied. This way, the user may reconstruct later on which decisions have been performed during the integration process.

Adaptability. An integration tool must be adaptable to a specific application domain. Adaptability is achieved by defining suitable integration rules and controlling their application (e.g., through priorities). It must be possible to modify the rule base on the fly.

A posteriori integration. An integration tool must work with heterogeneous tools supplied by different vendors. To this end, it has to access these tools via corresponding wrappers which provide abstract and unified interfaces.

5 Framework for Building Integration Tools

Figure 2 provides an overview of the *framework* for *tool integration* which we have developed with our industrial partner. At the heart of this framework, the integrator core offers basic functionality. In particular, it includes the basic control logic, i.e., the algorithms for document integration. The integrator core accesses the integration document which stores fine-grained links and records the application of integration rules. Furthermore, it is connected to the tools and documents to be integrated via respective *wrappers*, which are used to abstract from tool-specific details (a posteriori integration). The integrator user interface is used to control the integrator interactively. The rules which drive the integrator are specified in a *rule definition tool*. Rules are interpreted by the integrator core; alternatively, they may be hard-coded and compiled for more efficient execution[1].

[1] Currently, the latter requires manual programming.

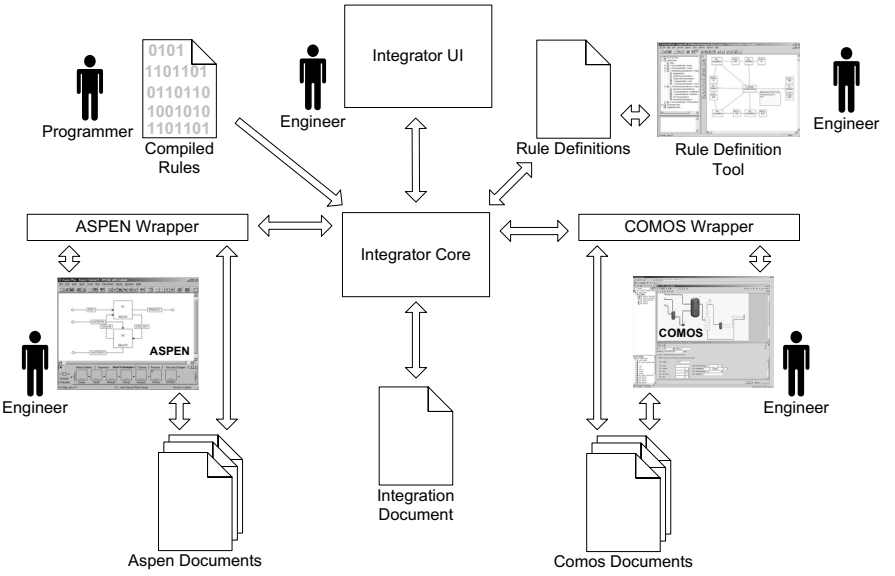


Fig. 2. Framework for tool integration

Let us illustrate the operation of this framework by the example of Section 3:

1. The flow sheet designer creates an initial flow sheet in COMOS PT. Here, COMOS PT is used as it stands.
2. The simulation expert uses the integrator to create a simulation model. The integrator accesses the flow sheet through the COMOS PT wrapper which provides a graph-based view on the source graph. Similarly, the Aspen Plus wrapper offers an updatable view on the target graph. The simulation expert activates source-to-target productions through the interactive interface of the integrator. Source-to-target relationships are stored in the integration document, which plays the role of the correspondence graph.
3. The flow sheet designer and the simulation expert operate in parallel locally on their respective documents.
4. The changes are synchronized with the help of the integrator. To synchronize the changes, both source-to-target and target-to-source productions are applied.

6 Definition of Rules

For the definition of rules, we decided to rely on the *Unified Modeling Language* [9] primarily for pragmatic reasons. The UML is a wide-spread modeling language which is supported by CASE tools such as Rational Rose, TogetherJ, etc. Although the UML is based on an object-oriented rather than on a graph-based data model, there are strong relationships to graphs and graph rewriting systems. For example, an *object diagram*

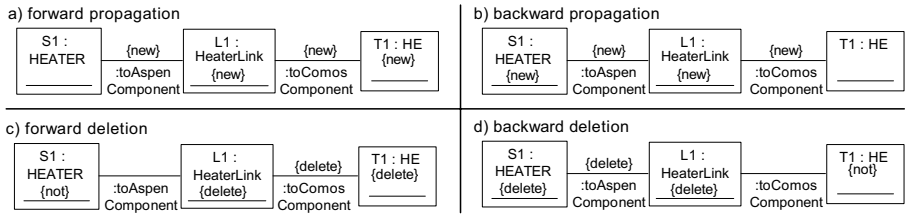


Fig. 3. Encoding graph rewrite rules with collaboration diagrams

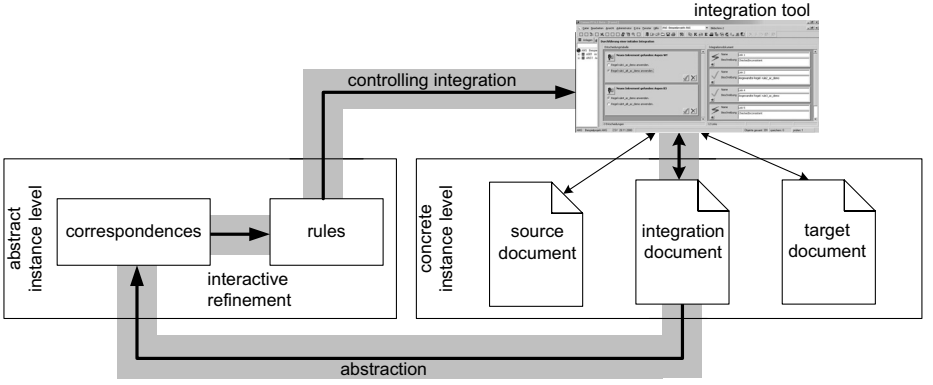


Fig. 4. Modeling process

showing a set of objects connected by links may be viewed as a graph. Likewise, a *collaboration diagram* extending a static object diagram with operations for creating/deleting objects or links corresponds to a graph rewrite rule.

Figure 3 illustrates how graph rewrite rules are expressed as collaboration diagrams. All of these rules deal with simple 1:1 correspondences between heater elements in flow sheets and heater blocks in simulation models. Left- and right-hand side of a graph rewrite rule are merged into a single diagram. Creation and deletion of objects and links are indicated by annotations new and delete, respectively. Rules a) and b) are *constructive* since they insert objects and links into the target (source) document after the source (target) document has been extended. In contrast, the *destructive* rules c) and d) are applied to propagate deletions: If the source (target) object is not present any more, the target (source) object as well as the link object have to be deleted. Please note that in general users may perform not only insertions, but also changes and deletions to source and target documents. Thus, we have to deal with general *graph rewrite rules* rather than only with generating productions.

So far, we have tacitly assumed that the rule base is given when the integrator is applied. In fact, it is fairly difficult to define an appropriate and comprehensive rule set beforehand. Rather, the rules have to be learned through experience. This is achieved through a *round-trip modeling process* which is illustrated in Figure 4. Let us assume an initial rule base to start with. The user may apply these rules to establish correspondences

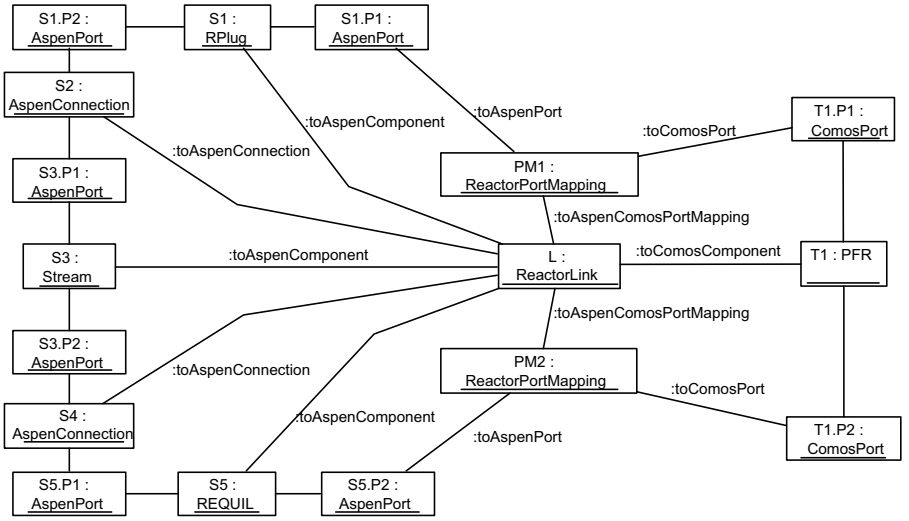


Fig. 5. Complex correspondence

between source and target document. If the user wishes to establish a certain correspondence even though a specific rule is not available, he may resort to built-in ad-hoc rules by means of which "untyped" correspondences may be created. Here, the user has to specify correspondences manually. Concrete correspondences stored in actual integration documents may be transformed into abstract correspondences defining mutually related graph patterns. Subsequently, these static correspondences may be transformed into dynamic rules. After that, the integrator may be used with the improved rule set.

Figure 5 gives an example of a complex correspondence which is represented by an object diagram. This correspondence is abstracted from the link between the plug flow reactor in COMOS PT and the cascade of reactor blocks in Aspen Plus, as illustrated in Figure 1[2]. From an object diagram, we may derive collaboration diagrams by introducing new and delete annotations. This may be performed in two steps. In the first step, a synchronous rule is defined. In the second step, source-to-target and target-to-source rules may be derived from the synchronous rule.

So far, only the structural aspects of correspondences and rules have been addressed. In addition, *attributes* have to be considered. In practice, elements of flow sheets and simulation models may carry a large number of attributes which have to be kept consistent with each other. Therefore, rules for attribute assignments have to be provided as well. In the UML, the relationships between attribute values may be defined in the Object Constraint Language (OCL). For further details on attribute assignments, the reader is referred to [5].

[2] Please note that a simplified notation was employed in Figure 1, while Figure 5 shows the actual internal graph representation. In particular, connections are represented as objects, as well as the end points (ports) of both connections and devices.

7 Implementation

The framework introduced in Section 5 was implemented in cooperation with our industrial partner innotec, the provider of COMOS PT. The implementation was performed such that the integrator would interact smoothly with COMOS PT (and Aspen Plus). Furthermore, it was required to keep the implementation as slim as possible and to avoid rucksacks of infrastructure software. For these reasons, we did not use the *PROGRES* environment [10], which is still heavily used in other projects carried out in our group. Rather, a *light-weight implementation* was realized which is tailored towards the specific requirements of our application domain and does not provide a general and powerful graph rewriting machinery.

For the points to be made in the next section, it is not important to go into the details of the implementation. However, we do have to convey an overall understanding of how the integrator works. The integrator is provided with the source document, the target document, and the integration document. Both the source document and the target document may have been modified after the last run of the integrator (see e.g. the last step of our example in Section 3). To re-establish consistency, the integrator searches source and target documents for elements which do not participate in correspondences. These elements are scheduled for source-to-target or target-to-source transformations. In the next step, the set of candidate rules is identified for each scheduled element. If there is no such rule, the user may apply a built-in ad hoc rule. If there is more than one rule, the user has to select the appropriate one. If there is exactly one rule, the rule is applied automatically. In addition, the integrator performs a run through the integration document to check the consistency of the correspondences already stored in the integration document. Each correspondence stores a reference to the respective rule. All correspondences whose source or target patterns were modified as marked as inconsistent. If essential elements of those patterns were deleted, the correspondence is deleted as well. If possible, repair actions are initiated to re-establish consistency. In addition to structural rules, the integrator also handles attribute rules (through the execution of script code).

8 Discussion

After having recalled the theoretical foundations of triple graph grammars in Section 2 and having presented a practical application in the following sections, we now reflect on the experiences we have made in the described application.

By and large, triple graph grammars constitute a powerful conceptual framework for addressing integration problems for the following reasons:

- For complex m:n relationships, it pays off to introduce a correspondence graph in between the source and the target graph.
- Triple productions declaratively specify coupling of graph structures and abstract from the different possible modes of use: synchronous coupling, source-to-target (and target-to-source) translation, and incremental change propagation.

On the other hand, the actual definitions as given in [2] bear some restrictions which prevented their use in our context:

- Graphs are not typed, and nodes do not carry attributes. Both types and attributes are very important in our application domain.
- Inter-graph relationships are represented by graph morphisms. Usually, morphisms are defined between graphs of the same type. Furthermore, they have to preserve not only source and target nodes of edges, but also types. This is not the case for the relationships between the correspondence graph and source or target graph. In addition, a correspondence node may be related to only one source and target node, respectively. Thus, complex correspondences cannot be modeled in the way we have done it (see Figure 5); rather, they have to spread over multiple correspondence nodes which are grouped only implicitly[3]. Altogether, it seems more appropriate to represent inter-graph relationships by *inter-graph edges* instead.
- The definitions deal only with graph *grammars*. However, we are concerned with graph *rewriting systems*: The user may also apply deleting or modifying transformations. Grammars are adequate for batch translations: Given a source graph SG, construct a target graph TG. However, we have to deal with general editing rather than merely with constructing operations.

Finally, we faced some practical problems in our application domain:

- The original proposal tacitly assumes that we may start from given grammars for the source and target graphs. In practice, these grammars are not available. Usually, tools provide a procedural interface (e.g., OLE) for reading and writing the data stored in native data structures. There is no definition of the underlying graphical language. At best, the tool builder may provide a documented textual interchange format (typically XML).
- Likewise, it is by no means straightforward to define the triple rules. In fact, the correspondences between flow sheets and simulation models may be defined only through practical experience. Therefore, we introduced our round-trip modeling process illustrated in Figure 4.
- In our application domain, we only have a fairly weak notion of *consistency*. The rules describing correspondences between flow sheets and simulation models are of heuristic nature. Similar observations apply e.g. to the relationships between requirements definitions and software architectures in software engineering [6].
- In [2], it is assumed that the decoupling of transformations is achieved with the help of graph parsers: Only when we know the production sequence on the source graph can we apply the corresponding productions on the target graph. Building of graph parsers is complicated anyway, but it completely breaks down when the parsing problem is undecidable[4]. For these reasons, we have never considered building graph parsers. Rather, the changes performed on source and target graph are determined in a completely different way by traversing source, target, and correspondence graph, as described in Section 7.

[3] In the example given in [2], this grouping is introduced informally by composite node identifiers.
[4] The original proposal assumes monotonic productions to guarantee decidability.

9 Related Work

Triple graph grammars have their roots in the IPSEN project [11] which dealt with tightly integrated software development environments. Originally, only a priori integration was considered, i.e., tools were designed for integration from the very beginning. Lefering [6] used triple graph grammars to develop incremental integration tools for the coupling of requirements definitions and software architectures. Later on, triple graph grammars were applied in several software engineering projects outside the scope of the IPSEN project. In particular, they were used for the re-engineering of software systems. In the Varlet project [12], incremental integration tools were built for mapping relational to object-oriented database schemas. In ReforDi [13], tools were developed for migrating mainframe applications to a client-server architecture. Here, the structure graph of the original Cobol application was mapped to an object-based architecture with the help of triple rules. Both projects relied on the PROGRES environment [10] as the underlying specification and implementation machinery. Finally, [14] reports on an application of triple graph grammars in chemical engineering. To some extent, this work served as a starting point for the project described in this paper.

Graph transformations have been used for the specification of integration tools in a couple of other projects as well. Some of this work is devoted to *model transformations*, where a given model is transformed into another notation [15,16]. Model transformation tools usually operate in batch mode without user interaction, i.e., they work like a compiler. In contrast, the applications we study demand for incremental, interactive integration tools. Closely related problems are studied in the ViewPoints project [17], which investigates methods and tools for maintaining consistency between related view points (documents in our terminology) in software engineering. Enders et al. [18] describe how consistency analysis and repair actions in the ViewPoints approach are specified with the help of distributed graph transformations. Here, the more restricted pair graph grammar approach coined by Pratt [1] is applied.

10 Conclusion

We have presented an industrial application of triple graph grammars. Incremental integration tools are used to maintain consistency between inter-dependent design documents in chemical engineering. We have also discussed the strengths and limitations of the triple graph grammar approach. In our future work, we will evaluate the tools which we built in cooperation with our industrial partner. Furthermore, we will generalize the framework such that it can be applied outside the chemical engineering domain.

References

1. Pratt, T.W.: Pair grammars, graph languages and string-to-graph translations. Journal of Computer and System Sciences **5** (1971) 560–595
2. Schürr, A.: Specification of graph translators with triple graph grammars. In Mayr, E.W., Schmidt, G., Tinhofer, G., eds.: Proceedings 20th Workshop on Graph-Theoretic Concepts in Computer Science WG 1994. LNCS 903, Herrsching, Germany, Springer (1994) 151–163

3. Nagl, M., Marquardt, W.: SFB-476 IMPROVE: Informatische Unterstützung übergreifender Entwicklungsprozesse in der Verfahrenstechnik. In Jarke, M., Pasedach, K., Pohl, K., eds.: Informatik '97: Informatik als Innovationsmotor. Informatik aktuell, Aachen, Germany, Springer (1997) 143–154
4. Becker, S., Haase, T., Westfechtel, B., Wilhelms, J.: Integration tools supporting cooperative development processes in chemical engineering. In: Proceedings International Conference on Integrated Design and Process Technology (IDPT-2002), Pasadena, California (2002) 10 pp. CD ROM proceedings.
5. Becker, S., Westfechtel, B.: UML-based definition of integration models for incremental development processes in chemical engineering. In: Proceedings International Conference on Integrated Design and Process Technology (IDPT-2003), Austin, Texas (2003) 10 pp. To appear.
6. Lefering, M.: Tools to support life cycle integration. In: Proceedings of the 6th Software Engineering Environments Conference, Reading, UK, IEEE Computer Society Press (1993) 2–16
7. innotec GmbH: COMOS PT Documentation, http://www.innotec.de. (2003)
8. Aspen-Technology: Aspen Plus Documentation, http://www.aspentech.com. (2003)
9. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison Wesley, Reading, Massachusetts (1999)
10. Schürr, A., Winter, A.J., Zündorf, A.: The PROGRES approach: Language and environment. [19] 487–550
11. Nagl, M., ed.: Building Tightly-Integrated Software Development Environments: The IPSEN Approach. LNCS 1170. Springer, Berlin, Germany (1996)
12. Jahnke, J., Zündorf, A.: Applying graph transformations to database re-engineering. [19] 267–286
13. Cremer, K., Marburger, A., Westfechtel, B.: Graph-based tools for re-engineering. Journal of Software Maintenance and Evolution: Research and Practice **14** (2002) 257–292
14. Cremer, K., Gruner, S., Nagl, M.: Graph transformation based integration tools: Applications to chemical process engineering. [19] 369–394
15. Baresi, L., Mauri, M., Pezzè, M.: PLCTools: Graph transformation meets PLC design. Electronic Notes in Theoretical Computer Science **72** (2002) 10 pp.
16. de Lara, J., Vangheluwe, H.: Computer aided multi-paradigm modeling to process petri-nets and statecharts. In Corradini, A., Ehrig, H., Kreowski, H.J., Rozenberg, G., eds.: Proceedings of the 1st International Conference on Graph Transformations. LNCS 2505, Barcelona, Spain, Springer (2002) 239–253
17. Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., Goedicke, M.: Viewpoints: A framework for integrating multiple perspectives in system development. International Journal of Software Engineering and Knowledge Engineering **2** (1992) 31–58
18. Enders, B., Heverhagen, T., Goedicke, M., Tröpfner, P., Tracht, R.: Towards an integration of different specification methods by using the Viewpoint framework. Transactions of the SDPS **6** (2002) 1–23
19. Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook on Graph Grammars and Computing by Graph Transformation: Applications, Languages, and Tools. Volume 2. World Scientific, Singapore (1999)

The Minimum Degree Heuristic and the Minimal Triangulation Process

Anne Berry[1], Pinar Heggernes[2], and Geneviève Simonet[3]

[1] LIMOS UMR CNRS 6158, Isima, 63173 Aubière, France.
berry@isima.fr
[2] Informatics, University of Bergen, N-5020 Bergen, Norway.
pinar@ii.uib.no
[3] LIRMM, 161 Rue Ada, 34392 Montpellier, France.
simonet@lirmm.fr

Abstract. The Minimum Degree Algorithm, one of the classical algorithms of sparse matrix computations, is a heuristic for computing a minimum triangulation of a graph. It is widely used as a component in every sparse matrix package, and it is known to produce triangulations with few fill edges in practice, although no theoretical bound or guarantee has been shown on its quality. Another interesting behavior of Minimum Degree observed in practice is that it often results in a minimal triangulation. Our goal in this paper is to examine the theoretical reasons behind this good performance. We give new invariants which partially explain the mechanisms underlying this heuristic. We show that Minimum Degree is in fact resilient to error, as even when an undesirable triangulating edge with respect to minimal triangulation is added at some step of the algorithm, at later steps the chances of adding only desirable edges remain intact. We also use our new insight to propose an improvement of this heuristic, which introduces at most as many fill edges as Minimum Degree but is guaranteed to yield a minimal triangulation.

1 Introduction

For the past forty years, problems arising from applications have given rise to challenges for graph theorists, and thus also to a wealth of graph-theoretic results. One of these is computing a minimum triangulation. Although the problem originally comes from the field of sparse matrix computations [20], it has applications in various areas of computer science.

Large sparse symmetric systems of equations arise in many areas of engineering, like the structural analysis of a car body, or the modeling of air flow around an airplane wing. The physical structure can often be thought of as covered by a mesh where each point is connected to a few other points, and the related sparse matrix can simply be regarded as an adjacency matrix of this mesh. Such systems are solved through standard methods of linear algebra, like Gaussian elimination, and during this process non-zero entries are inserted into cells of the matrix that originally held zeros, which increases both the storage

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 58–70, 2003.

requirement and the time needed to solve the system. It was observed early that finding a good pivotal ordering of the matrix can reduce the amount of *fill* thus introduced: in 1957, Markowitz [14] introduced the idea behind the algorithm known today as Minimum Degree, choosing a pivot row and column at each step of the Gaussian elimination to locally minimize the product of the number of corresponding off-diagonal non-zeros. Tinney and Walker [22] later applied this idea to symmetric matrices, and Rose [20] developed a graph theoretical model of it.

As early as 1961, Parter [18] presented an algorithm, known as Elimination Game (EG), which simulates Gaussian elimination on graphs by repeatedly choosing a vertex and adding edges to make its neighborhood into a clique before removing it, thus introducing the connection between sparse matrices and graphs. In view of the results of [8], the class of graphs produced by EG is exactly the class of chordal graphs. Thus when the given graph is not chordal, Gaussian elimination and EG correspond to embedding it into a chordal graph by adding edges, a process called *triangulation*. As can be observed on the example in Figure 1, the number of fill edges in the resulting triangulation is heavily dependent on the order in which EG processes the vertices. This ordering of the graph corresponds to the pivotal ordering of the rows and columns in Gaussian elimination.

As mentioned above, it is of primary importance to add as few edges as possible when running EG. The corresponding problem is that of computing a *minimum triangulation*, which is NP-hard [23]. It is possible to compute in polynomial time a triangulation which is *minimal*, meaning that an inclusion-minimal set of edges is added [16], [21]. However, such a triangulation can be far from minimum, as can be seen from the example of Figure 1(b). As a result, researchers have resorted to heuristics, of which one of the most universally used and studied is *Minimum Degree (MD)*: this runs EG by choosing at each step a vertex of minimum degree in the transitory elimination graph, as illustrated by Figure 1(d). This algorithm is widely used in practice, and it is known to produce low fill triangulations. In addition, MD is also observed [5] to produce triangulations which are often minimal or close to minimal.

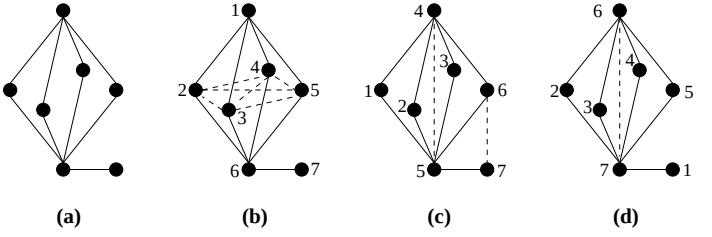


Fig. 1. (a) A graph G, and various triangulations of G by EG through the given orderings: (b) A minimal triangulation with $O(n^2)$ fill edges. (c) A non-minimal triangulation of G with less fill. (d) A minimum triangulation of G.

MD has given rise to a large amount of research with respect to improving the running time of its practical implementations, and the number of papers

written on this subject is in the hundreds [1], [9]. However, very little is proved about its quality. It has in fact been analyzed theoretically only to a limited extent, which makes it difficult to gain control over this heuristic in order to improve it yet further, although recent research has been done on algorithms for low fill minimal triangulations [5], [6], [19].

In this paper, we use recent graph theoretical results on minimal triangulation and minimal separation to explain, at least in part, why MD yields such good results. In fact, it turns out that one of the reasons why MD works so well is that the EG algorithm is remarkably *robust*, in the sense that it is resilient to error: if at some step of the process an undesirable edge with respect to minimal triangulation is added, at later steps the chances of adding only desirable edges remain intact. One of our contributions here is that we use the insight we have gained on the mechanisms which govern EG and MD to propose an algorithmic process which improves the results obtained by MD, giving triangulations that are both minimal and have low fill.

The remainder of this extended abstract is organized as follows: Section 2 gives the graph theoretic background, introduces EG formally, and gives previous results on minimal separation and minimal triangulation. In Section 3, we give some new invariants for EG and MD, and explain why these algorithms are resilient to error, and why in many cases MD computes a minimal triangulation. Section 4 proposes a new algorithmic process in view of using the results of the paper to improve MD.

2 Preliminaries

Given a graph $G = (V, E)$, we denote $n = |V|$ and $m = |E|$. For any subset S of V, $G(S)$ denotes the subgraph of G induced by S, and $\mathrm{C}_G(S)$ denotes the set of connected components of $G(V - S)$. For the sake of simplicity, we will use informal notations such as $H = G + \{e\} + \{x\}$ when H is obtained from G by adding edge e and vertex x. For any vertex v of G, $N_G(v)$ denotes the neighborhood of v in G, and $N_G[v]$ denotes the set $N_G(v) \cup \{v\}$. For a given set of vertices $X \subset V$, $N_G(X) = \cup_{v \in X} N_G(v) - X$, and $N_G[X] = \cup_{v \in X} N_G(v) \cup X$. We will omit the subscripts when there is no ambiguity.

A vertex is *simplicial* if its neighborhood is a clique. We will say that we *saturate* a set of vertices X when we add to the graph all the edges necessary to make X into a clique. A graph is *chordal*, or *triangulated*, if it contains no chordless cycle of length ≥ 4. The set F of edges added to an arbitrary graph $G = (V, E)$ to obtain a triangulation $H = (V, E + F)$ of G is called a *fill*.

A function $\alpha : V \rightarrow \{1, 2, ..., n\}$ is called an *ordering* of the vertices of $G = (V, E)$, and (G, α) will denote a graph G the vertices of which are ordered according to α. We will use $\alpha = (v_1, v_2, ..., v_n)$, where $\alpha(v_i) = i$.

The algorithmic description of Elimination Game (EG) given below defines the notations we will use in the rest of this paper:

Algorithm Elimination Game
Input: A graph $G = (V, E)$, and an ordering α of the vertices in G.
Output: A triangulation G^+_α of G.
$G^1_\alpha \leftarrow G$; $G^+_\alpha \leftarrow G$;
for $k = 1$ **to** n **do**
Let F be the set of edges necessary to saturate $N_{G^k_\alpha}(v_k)$ in G^k_α;
$G^{k+1}_\alpha \leftarrow G^k_\alpha + F - \{v_k\}$; $G^+_\alpha \leftarrow G^+_\alpha + F$;

According to the definition used in [16], we will call $G^k_\alpha(\{v_k, ..., v_n\} - N_{G^k_\alpha}[v_k])$ the *section graph at step k*. It is immediate from the description of EG that for any distinct $i, j \in [k, n]$, edge v_iv_j is present in G^k_α iff there is a path in G between v_i and v_j (the path may have only one edge) all intermediate vertices of which are numbered $< k$ (i.e. do not belong to G^k_α). Similarly, the edges added during an execution of EG are well defined as v_iv_j is an edge of G^+_α iff v_iv_j is an edge of G or there is a path in G between v_i and v_j, all intermediate vertices of which have a number which is strictly smaller than $\min\{i, j\}$ [21].

The Minimum Degree (MD) heuristic is based on EG: it takes as input an unordered graph G, and computes an ordering α along with the corresponding triangulation G^+_α, by choosing at each step a vertex of minimum degree in G^k_α and numbering it as v_k.

Before proceeding to the next section, we will need some results on minimal separation. The notion of a minimal separator was introduced by Dirac [7] to characterize chordal graphs. Given a graph $G = (V, E)$, a vertex set $S \subset V$ is a *minimal separator* if $G(V - S)$ has at least two connected components C_1 and C_2 such that $N_G(C_1) = N_G(C_2) = S$ (C_1 and C_2 are called *full components*).

Characterization 1. ([7]) *A graph is chordal iff all its minimal separators are cliques.*

Recent research [11], [17], [2] has shown that minimal separators are central to minimal triangulations. The idea behind this is that forcing a graph into respecting Dirac's characterization will result into a minimal triangulation, by repeatedly choosing a not yet processed minimal separator and saturating it. We will need the definition of crossing separators, which characterize the separators that disappear when a saturation step of this process is executed:

Definition 1. ([11]) *Let S and S' be two minimal separators of G. S and S' are said to be* crossing *if there exist two connected components C_1, C_2 of $G(V - S)$, such that $S' \cap C_1 \neq \emptyset$ and $S' \cap C_2 \neq \emptyset$ (the crossing relation is symmetric).*

The saturation process described above can be generalized by choosing and simultaneously saturating a set of pairwise non-crossing minimal separators instead of a single minimal separator at each step, until a chordal graph is obtained. We will refer to this generalized process as the *Saturation Algorithm*. Given a set S of minimal separators of G, we will denote G_S the graph obtained from G by saturating all the separators belonging to S.

The following results from the works of Kloks, Kratsch and Spinrad [11] and Parra and Scheffler [17] provide a proof of this algorithm and will be used in Sections 3 and 4.

Theorem 1. ([11], [17]) *A graph $H = (V, E + F)$ is a minimal triangulation of $G = (V, E)$ iff there is a maximal set* S *of pairwise non-crossing minimal separators of G such that $H = G_{\mathrm{S}}$.*

Corollary 1. *A graph $H = (V, E+F)$ is a minimal triangulation of $G = (V, E)$ iff H is chordal and there is a set* S *of pairwise non-crossing minimal separators of G such that $H = G_{\mathrm{S}}$.*

Lemma 1. ([17]) *Let $G = (V, E)$ be a graph, let* S *and* S′ *be sets of pairwise non-crossing minimal separators of G and G_{S}, respectively. Then $\mathrm{S} \cup \mathrm{S}'$ is a set of pairwise non-crossing minimal separators of G.*

Lemma 2. ([17]) *Let $G = (V, E)$, be a graph and* S *a set of pairwise non-crossing minimal separators of G. Then any minimal triangulation of G_{S} is a minimal triangulation of G.*

We will also use the notion of *substar*, which was introduced by Lekkerkerker and Boland [12] in connection with their characterization of chordal graphs.

Definition 2. ([12]) *Given a graph $G = (V, E)$ and a vertex x of G, the* substars *of x in G are the neighborhoods in G of the connected components of $G(V - N[x])$.*

In fact, although Lekkerkerker and Boland seemed not to be aware of this, the set of substars of some vertex x is exactly the set of minimal separators included in the neighborhood of x. LB-simpliciality of a vertex was defined in [3] in the following way for more convenient terminology.

Definition 3. *A vertex x is* LB-simplicial *if every substar of x is a clique.*

This was implicitly used by [12] to characterize chordal graphs as graphs such that every vertex is LB-simplicial, but the notion of substar is also very useful in the context of minimal triangulation, because it provides a fast and easy way of repeatedly finding sets of pairwise non-crossing minimal separators when running the Saturation Algorithm. This is fully described in [4], with in particular the following lemma:

Lemma 3. ([4]) *The substars of a vertex x in a graph G are pairwise non-crossing in G.*

3 Properties of EG Related to Minimal Triangulation

We will now examine how EG behaves with respect to the minimal separators of the graph which is to be triangulated.

3.1 EG and Partial Minimal Triangulation

We will first extend the definition of substar given in Section 2 to that of *substars* of (G, α).

Definition 4. *Given $(G = (V, E), \alpha)$, we will say that a set $S \subset V$ of vertices is a* substar of (G, α) *if there is some step k of EG such that S is a substar of v_k in G^k_α, which will be referred to as a* substar defined at step k *of EG.*

Clearly, during the execution of the EG, at each step k, making the currently processed vertex v_k simplicial will saturate these substars, and may also add some extraneous edges which do not have both endpoints in some common substar, so that two kinds of edges can be added:

- Edges which have both endpoints in some common substar defined at step k. We will refer to these edges as *substar fill edges*.
- Edges which do not have both endpoints in some common substar defined at step k. We will refer to these as *extraneous edges*.

In Section 2, we mentioned that in G and for a given vertex v, the substars of v are the minimal separators included in the neighborhood of v. One of our most interesting discoveries is that, in fact, *all* the substars defined by EG are minimal separators of the input graph, whether or not extraneous edges have been added at earlier steps. This fact is stated in Theorem 2, and its proof is based on the following lemma, which is interesting in its own right, as it describes a strong correspondence between the structures of G and G^k_α.

Lemma 4. *Given $(G = (V, E), \alpha)$ and an integer $k \in [1, n]$, let $G^k_\alpha = (V^k_\alpha, E^k_\alpha)$ and $S \subseteq V^k_\alpha$. The connected components of $G^k_\alpha(V^k_\alpha - S)$ are the sets $C \cap V^k_\alpha$ where C is a connected component of $G(V - S)$ such that $C \cap V^k_\alpha \neq \emptyset$, with the same neighborhoods, i.e. $N_{G^k_\alpha}(C \cap V^k_\alpha) = N_G(C)$.*

Proof. Let $S \subseteq V^k_\alpha$. We have to prove that $\mathrm{C}_{G^k_\alpha}(S) = \{C \cap V^k_\alpha,\ C \in \mathrm{C}_G(S) \mid C \cap V^k_\alpha \neq \emptyset\}$ and $\forall C \in \mathrm{C}_G(S)$ such that $C \cap V^k_\alpha \neq \emptyset$, $N_{G^k_\alpha}(C \cap V^k_\alpha) = N_G(C)$. Let $C \in \mathrm{C}_G(S)$ such that $C \cap V^k_\alpha \neq \emptyset$ and let $C' = C \cap V^k_\alpha$. Let us show that $C' \in \mathrm{C}_{G^k_\alpha}(S)$ and $N_{G^k_\alpha}(C') = N_G(C)$. $G^k_\alpha(C')$ is connected because for any vertices x and y in C', there is a path P in $G(C)$ between x and y, and by the properties of EG, the sub-sequence of P containing only the vertices belonging to V^k_α is a path in $G^k_\alpha(C')$ between x and y. Let us show that $N_{G^k_\alpha}(C') \subseteq N_G(C)$. Let $x \in N_{G^k_\alpha}(C')$ and $y \in C'$ such that $xy \in E^k_\alpha$. By the description of EG, there is a path in G between x and y all intermediate vertices of which belong to $V - V^k_\alpha$, and therefore belong to $V - S$ and consequently belong to C, so $x \in N_G(C)$. Let us show that $N_G(C) \subseteq N_{G^k_\alpha}(C')$. Let $x \in N_G(C)$ and $y \in C$ such that $xy \in E$. As $C' \neq \emptyset$, we may choose $z \in C'$. Let P be a path in $G(C)$ between y and z and let z' be the first vertex of P from y belonging to V^k_α. Vertex $z' \in C'$, and due to EG, $xz' \in E^k_\alpha$, so $x \in N_{G^k_\alpha}(C')$. Thus $N_{G^k_\alpha}(C') = N_G(C)$. As $C' \neq \emptyset$, $C' \subseteq V^k_\alpha - S$, $G^k_\alpha(C')$ is connected and $N_{G^k_\alpha}(C') = N_G(C) \subseteq S$, it follows that $C' \in \mathrm{C}_{G^k_\alpha}(S)$. Therefore, $\{C \cap V^k_\alpha,\ C \in \mathrm{C}_G(S) \mid C \cap V^k_\alpha \neq \emptyset\} \subseteq \mathrm{C}_{G^k_\alpha}(S)$. As $\cup_{C \in \mathrm{C}_G(S)}(C \cap V^k_\alpha) = V^k_\alpha - S$, the reverse inclusion holds too.

Theorem 2. *Every substar of* (G, α) *is a minimal separator of* G.

Proof. Let S be a substar defined at step k. S is a minimal separator of G^k_α, and by Lemma 4, there are at least as many full components of $G(V - S)$ as of $G^k_\alpha(V^k_\alpha - S)$. So S is also a minimal separator of G.

We are now ready to state our Main Theorem:

Theorem 3. *The set of substars of* (G, α) *forms a set of pairwise non-crossing minimal separators of* G.

Proof. Let S and S' be two substars of (G, α) defined at steps k and k' respectively, with $k \leq k'$. By Theorem 2, they are both minimal separators of G. Let us show that they are non-crossing in G. If $k = k'$ then they are non-crossing in G^k_α by Lemma 3, so they are non-crossing in G by Lemma 4. We suppose now that $k < k'$. S is a clique of G^{k+1}_α and $S' \subseteq V^{k+1}_\alpha$, so there is a connected component C of $G^{k+1}_\alpha(V^{k+1}_\alpha - S')$ such that $S \subseteq S' \cup C$. By Lemma 4, there is a connected component C' of $G(V - S')$ containing C, so $S \subseteq S' \cup C'$. Hence S and S' are non-crossing in G.

Note that this theorem does not guarantee that the set of substars defines a set of pairwise non-crossing minimal separators which is maximal. For instance, for any non complete graph G, if v_1 is a universal vertex of G then there is no substar of (G, α) whereas G has at least one minimal separator. A less trivial counterexample is given in Figure 2(b) of Example 1.

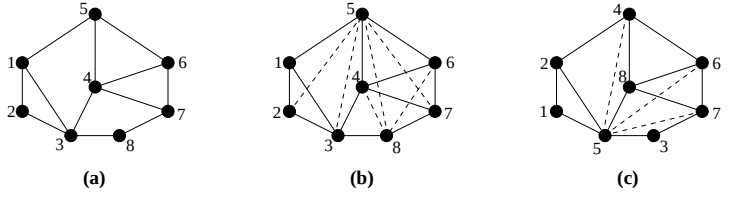


Fig. 2. Two executions of EG on the same graph with (b) an arbitrary ordering, and (c) an MD ordering.

Example 1. Figure 3.1 shows two executions of EG on graph G. A graph G and an ordering α are given in (a). The minimal separators of G are: $\{1, 3\}$, $\{3, 5\}$, $\{3, 7\}$, $\{1, 4, 6\}$, $\{1, 4, 7\}$, $\{1, 4, 8\}$, $\{3, 5, 7\}$, $\{4, 5, 7\}$, $\{4, 5, 8\}$, $\{4, 6, 8\}$, $\{3, 4, 6\}$.

We now demonstrate the execution of EG on (G, α) resulting in the graph shown in (b). **Step 1**: $N(1) = \{2, 3, 5\}$, $C_1 = \{4, 6, 7, 8\}$, $N(C_1) = \{3, 5\}$; substar fill edge 35 and extraneous edge 25 are added. **Step 2**: $N(2) = \{3, 5\}$; $C_2 = \{4, 6, 7, 8\}$, $N(C_2) = \{3, 5\}$; 2 is simplicial, so no edge is added. **Step 3**: $N(3) = \{4, 5, 8\}$, $C_3 = \{6, 7\}$, $N(C_3) = \{4, 5, 8\}$; substar fill edges 48 and 58 are added. **Step 4**: $N(4) = \{5, 6, 7, 8\}$; 4 is universal, so no component is defined; extraneous edges 57 and 68 are added; the remaining graph becomes a clique; no further edge is added. The set of substars of (G, α) is thus $\{\{3, 5\}, \{4, 5, 8\}\}$, which is a set of pairwise non-crossing minimal separators of G, but not a maximal one as $\{\{3, 5\}, \{4, 5, 8\}, \{4, 5, 7\}\}$ and $\{\{3, 5\}, \{4, 5, 8\}, \{4, 6, 8\}\}$ are also sets of pairwise

non-crossing minimal separators of G. If only substar fill edges are preserved, a chordless cycle 5678 remains in the graph thus obtained. In order to saturate a maximal set of pairwise non-crossing minimal separators of G, $\{4, 5, 7\}$ or $\{4, 6, 8\}$ should also be saturated.

On the same graph, MD yields a minimal (and even minimum) triangulation, as shown in (c).

We would like to end this subsection with a discussion on the robustness of EG and MD regarding the process of defining non-crossing minimal separators of G. If, during the EG process, no extraneous edge is added, then the triangulation which is computed is minimal. However, due to Theorem 3, even when extraneous edges have been added, all substar fill edges added later belong to a set of pairwise non-crossing minimal separators of G, and therefore to a minimal triangulation of G. Thus if only a few extraneous edges are added during EG process, they will not destroy the property that all the substar fill edges are "useful" edges and that these few extraneous edges are the only "unnecessary" edges introduced. This makes EG a fault-tolerant procedure.

3.2 Conditions for EG and MD to Produce Minimal Triangulations

We have seen that EG does not necessarily compute a minimal triangulation. However, as mentioned earlier, MD is observed in practice to often produce orderings that are minimal or close to minimal, in addition to low fill. We will now give an explanation of this good behavior of MD through Lemma 5.

Lemma 5. *Let v_k be a vertex of minimum degree in G^k_α such that the union of all substars defined at step k is equal to a substar S defined at step k. Then only substar fill edges are added at step k.*

Proof. Assume by contradiction that there is an extraneous edge yz added at step k. Then y or z, say y, is not in S. $N_{G^k_\alpha}(y) \subseteq N_{G^k_\alpha}[v_k] - \{y, z\}$, so that $|N_{G^k_\alpha}(y)| < |N_{G^k_\alpha}(v_k)|$, which contradicts the fact that v_k is a vertex of minimum degree in G^k_α.

This result shows in particular that when the section graph is connected at some step of MD, then fill edges are added at that step only within the single substar defined. As is clear from the proof of this lemma, this is not the case in general for EG. Moreover, in most practical applications [15], the input graph is sparse; although no statistical result has been established on this, intuitively, a vertex x of minimum degree quite often defines only one substar, which usually corresponds to a connected section graph. MD run on sparse graphs thus stands a significantly higher chance of generating minimal triangulations than EG.

4 Improving the Results of Minimum Degree

Based on the results of the previous sections, we now present a variant of MD that produces a minimal triangulation and yields at most the same amount of

fill, because it computes a minimal triangulation which is a subgraph of the graph computed by MD.

We have seen in Section 3 that the set S of non-crossing minimal separators of G defined by the set of all substars of (G, α) is not necessarily maximal. Thus if we remove all extraneous edges from G_α^+ (or equivalently, add to G only the substar fill edges), the graph G_S thus obtained might fail to be triangulated. However, by Lemma 2, we know that any minimal triangulation H of G_S is a minimal triangulation of G. We thus propose a process which computes a minimal triangulation of the graph G_S obtained by running MD on G and removing the extraneous edges. In order to give the MD approach its chances, we will repeat the process of running MD on the partially triangulated graph obtained and removing the extraneous edges, until a chordal graph is obtained. However, at the beginning of each iteration, we will remove all LB-simplicial vertices, according to the following result:

Lemma 6. *Let $G = (V, E)$ be a graph, X the set of LB-simplicial vertices of G, and $G' = G(V-X) = (V', E')$. For any minimal triangulation $H' = (V', E'+F')$ of G', the graph $H = (V, E + F')$ is a a minimal triangulation of G.*

Proof. H is chordal because for any cycle C in H of length ≥ 4, either C is in H' and then C has at least one chord, or C contains a vertex x of X and then the neighbors of x in C induce a chord of C, as they belong to the substar of x in H defined by the component containing the other vertices of C, which is also substar of x in G (otherwise there would be an edge yz in F' such that y belongs to a connected component D of $G(V - N_G[x])$ and z does not belong to $N_G[D]$; by Theorem 1, y and z would be in a minimal separator of G', so they would be vertices of a chordless cycle C' in G', C' would have to contain two distinct vertices in $N_G(D)$ inducing a chord of C', a contradiction). So H is a a triangulation of G. It is a minimal one because for any chordal graph $H_1 = (V, E + F_1')$ with $F_1' \subseteq F'$, the graph $H_1' = (V', E' + F_1')$ is chordal too, so that $F_1' = F'$.

Thus the LB-simplicial vertices can only cause MD to add extraneous edges, as well as unnecessarily increasing some vertex degrees, which justifies our systematically eliminating them from the graph at each step. Note that any step of the MD process tends to create LB-simplicial vertices, so removing these can make a significant difference regarding the quality of the fill obtained. We now present the new algorithm.

Algorithm Minimal Minimum Degree (MMD)
Input: A graph G.
Output: A minimal triangulation H of G.
Run MD on G, which defines an ordering α and a set of substars S of (G, α);
$G' \leftarrow G_S$; $H \leftarrow G_S$;
while G' is not chordal **do**
  Remove all LB-simplicial vertices from G';
  Run MD on G', which defines an ordering α and a set of substars S of (G', α);
  $G' \leftarrow G'_S$; $H \leftarrow H_S$;

It should be noted that graphs can be constructed such that no execution of MD can produce a minimal triangulation. Such an example is a graph consisting of two large cliques, connected by a single path of length ≥ 2. The graph is chordal, but vertices on the path will be chosen by MD at first steps, introducing unnecessary fill. MMD will not encounter any problem with that kind of graph, since the only minimal triangulation of a chordal graph is the graph itself.

We now prove that MMD gives a fill which is as least as good a that of MD, by showing that MMD yields a subgraph of the graph computed by MD.

Theorem 4. *Let* S *be the set of substars of* (G, α)*. Then any minimal triangulation* H *of* G_{S} *is a minimal triangulation of* G *which is a subgraph of* G^+_α*.*

Proof. In [4] the following property (P) is proved: Let S be a set of pairwise non-crossing minimal separators of G. Then $\forall S \in \mathrm{S}$, $\mathrm{C}_G(S) = \mathrm{C}_{G_{\mathrm{S}}}(S)$ and $\forall C \in \mathrm{C}_G(S)$, $N_G(C) = N_{G_{\mathrm{S}}}(C)$. We will use (P) in our proof. Let H be a minimal triangulation of G_{S}. By Theorem 3 and Lemma 2, H is a minimal triangulation of G. Let us show that H is a subgraph of G^+_α. By Theorem 1 and the fact that G_{S} is a subgraph of G^+_α, it is sufficient to show that any minimal separator of G_{S} is a clique of G^+_α. Let T be a minimal separator of G_{S}, let u and v be two vertices of T with $\alpha(u) < \alpha(v)$ and let $k = \alpha(u)$ (i.e. $u = v_k$). Let us show that uv is an edge of G^+_α, i.e. $v \in N_{G^k_\alpha}(v_k)$. We assume by contradiction that $v \notin N_{G^k_\alpha}(v_k)$. Let S be the substar of (G, α) defined at step k by the component containing v. S is a minimal separator of G^k_α separating the vertices v_k and v of T. By Lemma 4 S is also a minimal separator of G separating v_k and v, and by Theorem 3 and (P), it is a minimal separator of G_{S} separating v_k and v. So S and T are crossing in G_{S}. Hence S intersects at least two components of $\mathrm{C}_{G_{\mathrm{S}}}(T)$. But as $S \in \mathrm{S}$, S is a clique of G_{S} so that S cannot intersect two components of $\mathrm{C}_{G_{\mathrm{S}}}(T)$, a contradiction.

Theorem 5. *MMD computes a minimal triangulation of* G *which is a subgraph of* G^+_α*, where* α *is the MD ordering computed at the beginning of the algorithm.*

Proof. MMD terminates, because at the beginning of each step, the LB-simplicial vertices are all removed and this process cannot make a non LB-simplicial vertex become LB-simplicial. This is because a vertex is not LB-simplicial iff it belongs to a chordless cycle of length ≥ 4, and such a cycle remains after removing LB-simplicial vertices. Thus the vertex of minimum degree which is chosen first is not LB-simplicial; as a result, making it simplicial cannot fail to add at least one substar fill edge. Let us now prove MMD correctness. Let H be the output graph, let S_0 be the set of substars computed at the beginning of the algorithm and S' be the union of those computed in the **while**-loop. Thus $H = (G_{\mathrm{S}_0})_{\mathrm{S}'}$, G_{S_0} being the input graph of the **while**-loop. By Theorem 4 it is sufficient to show that H is a minimal triangulation of G_{S_0}, or more generally that for any input graph G' of the **while**-loop, the graph $G'_{\mathrm{S}'}$, where S' is the union of the sets of substars computed in the **while**-loop, is a minimal triangulation of G'. Let us prove this property by induction on the number p of iterations of the

while-loop before G' gets chordal. It trivially holds for $p = 0$, as in that case G' is chordal and S' is the empty set. We suppose that it holds when the number of iterations of the **while**-loop before G' gets chordal is p. Let us show that it holds when this number is $p + 1$. Let G'_1 be the graph obtained from G' by removing all its LB-simplicial vertices, let S'_1 be the set of substars computed at the first iteration of the **while**-loop and S'' be the union of those computed at the following iterations, and let G'' be the graph obtained at the end of the first iteration. Thus $G'' = (G'_1)_{S'_1}$ and $S' = S'_1 \cup S''$. By induction hypothesis, $G''_{S''}$ is a minimal triangulation of G'', so by Theorem 3 and Lemma 2, it is also a minimal triangulation of G'_1. $G''_{S''} = ((G'_1)_{S'_1})_{S''} = (G'_1)_{S'_1 \cup S''} = (G'_1)_{S'}$. Thus the graphs $G'_{S'}$ and $G''_{S''}$ are obtained from G' and G'_1 respectively by adding the same set F' of edges, so by Lemma 6, $G'_{S'}$ is a minimal triangulation of G', which completes the proof by induction and therefore the proof of MMD correctness.

With practical tests, we have compared MMD against MD with respect to the number of edges in the resulting triangulation. We have done a simple and straightforward implementation of MMD in Matlab, and we have run the tests both on randomly generated graphs of varying density, and on graphs from Matrix Market [15]. On each graph G, we first generated an MD ordering α. Then we compared the number of fill edges in G^+_α to the number of fill edges produced by MMD. As expected by Theorem 5, the number of fill edges resulting from MMD was always less than or equal to the number of fill edges resulting from MD. An interesting point is that on most graphs, MMD required only two iterations of the **while**-loop. The reduction in the number of fill edges achieved by MMD was not very large, because of MD's remarkably good performances. However, this improved algorithm may both give significant results on very large graphs and help researchers gain a better evaluation of how close MD gets to an optimal solution.

Finally, we would like to mention that existing MD codes in use are very fast although the theoretical running time of these implementations is not good [10]. In addition, other iterative procedures for computing minimal triangulations have been implemented to run fast in practice [19]. Thus we believe that, with some effort, MMD can be implemented to run efficiently in practice.

5 Conclusion

Our contributions in this paper are threefold. We have found new invariants for the Elimination Game process, proving that it defines and saturates a set of minimal separators of the input graph, with a remarkably fault-tolerant behavior. We have shown that Minimum Degree has additional properties that gives it a high chance of actually producing minimal triangulations, thereby explaining this practical behavior of MD. Finally, we have given a new algorithm that produces minimal triangulations with low fill, based on our findings about EG and MD.

Acknowledgment. We thank Elias Dahlhaus for interesting discussions on minimal triangulations.

References

1. P. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.*, 17:886–905, 1996.
2. A. Berry. Désarticulation d'un graphe. *PhD Dissertation, LIRMM, Montpellier, December 1998.*
3. A. Berry, J. R. S. Blair, and P. Heggernes. Maximum Cardinality Search for Computing Minimal Triangulations. In L. Kucera, editor, *Graph Theoretical Concepts in Computer Science - WG 2002*, Springer Verlag, 2002. Lecture Notes in Computer Science.
4. A. Berry, J.-P. Bordat, P. Heggernes, G. Simonet, and Y. Villanger. A wide-range algorithm for minimal triangulation from an arbitrary ordering. Reports in Informatics 243, University of Bergen, Norway, 2003, and Research Report 02-200, LIRRM, Montpellier, France. *Submitted to Journal of Algorithms, November 2002.*
5. J. R. S. Blair, P. Heggernes, and J. A. Telle. A practical algorithm for making filled graphs minimal. *Theoretical Computer Science*, 250:124–141,2001.
6. E. Dahlhaus. Minimal elimination ordering inside a given chordal graph. In R. H. Möhring, editor, *Graph Theoretical Concepts in Computer Science*, pages 132–143. Springer Verlag, 1997. Lecture Notes in Computer Science 1335.
7. G.A. Dirac. On rigid circuit graphs. *Anh. Math. Sem. Univ. Hamburg*, 25:71–76, 1961.
8. D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Math.*, 15:835–855, 1965.
9. J. A. George and J. W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31:1–19, 1989.
10. P. Heggernes, S. Eisenstat, G. Kumfert, and A. Pothen. The computational complexity of the Minimum Degree algorithm. *Proceedings of 14th Norwegian Computer Science Conference, NIK 2001, University of Tromsø, Norway.* Also available as ICASE Report 2001-42, NASA/CR-2001-211421, NASA Langley Research Center, USA.
11. T. Kloks, D. Kratsch, and J. Spinrad. On treewidth and minimum fill-in of asteroidal triple-free graphs. *Theoretical Computer Science*, 175:309–335, 1997.
12. C. G. Lekkerkerker and J. Ch. Boland. Representation of a finite graph by a set of intervals on the real line. *Fund. Math.*, 51:45–64, 1962.
13. J. W. H. Liu. Equivalent sparse matrix reorderings by elimination tree rotations. *SIAM J. Sci. Stat. Comput.*, 9:424–444, 1988.
14. H. M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, 3:255–269, 1957.
15. Matrix Market *Web site:* `http://math.nist.gov/MatrixMarket/`
16. T. Ohtsuki, L. K. Cheung, and T. Fujisawa. Minimal triangulation of a graph and optimal pivoting order in a sparse matrix. *J. Math. Anal. Appl.*, 54:622–633, 1976.
17. A. Parra and P. Scheffler. How to use the minimal separators of a graph for its chordal triangulation. *Proceedings of the 22nd International Colloquium on Automata, Languages and Programming (ICALP '95), Lecture Notes in Computer Science*, 944:123–134, 1995.

18. S. Parter. The use of linear graphs in Gauss elimination. *SIAM Review,* 3:119–130, 1961.
19. B. Peyton. Minimal orderings revisited. *SIAM J. Matrix Anal. Appl.*, 23:271–294, 2001.
20. D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In R. C. Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, 1972.
21. D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5:266–283, 1976.
22. W. F. Tinney and J. W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization In Proceedings of the IEEE, 55:1801–1809, 1967.
23. M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2:77–79, 1981.

Generalized Parametric Multi-terminal Flows Problem

Pascal Berthomé[1], Madiagne Diallo[2], and Afonso Ferreira[3⋆]

[1] Laboratoire de Recherche en Informatique (LRI), UMR 8623, CNRS, Université Paris-Sud, 91405, Orsay-Cedex, France
Pascal.Berthome@lri.fr

[2] Laboratoire PRiSM, Université de Versailles, 45 Av. des Etats-Unis, 78035 Versailles-Cedex, France
Madiagne.Diallo@prism.uvsq.fr

[3] CNRS, I3S, INRIA-Sophia Antipolis, Projet Mascotte, 2004 Rt. des Lucioles, BP93, F-06902 Sophia Antipolis, France
Afonso.Ferreira@inria.fr

Abstract. Given an undirected edge-weighted n-nodes network in which a single edge-capacity is allowed to vary, Elmaghraby studied the sensitivity analysis of the multi-terminal network flows. The procedure he proposed requires the computation of as many Gomory-Hu cut trees as the number of critical capacities of the edge, leading to a pseudo-polynomial algorithm.
In this paper, we propose a fully polynomial algorithm using only two Gomory-Hu cut trees to solve the Elmaghraby problem and propose an efficient generalization to the case where k edge capacities can vary. We show that obtaining the all-pairs maximum flows, for the case where k edge capacities vary in the network, is polynomial whenever $k = O(\text{polylog } n)$, since we show that it can be solved with the computation of 2^k Gomory-Hu trees.

Keywords: Multi-Terminal Flows, Gomory-Hu Cut Tree, Parametric Flows, Dynamic Networks, Max-Flow, Min-Cut, Sensitivity Analysis, Network Management.

1 Introduction

In the late 1950's, the single source–single terminal maximum flow problem was popularized by the resolution of Ford and Fulkerson [9]. They specially showed the connection between the maximum flow and the min cut problems in extension of Menger's theorem.

In the setting of a connected, undirected graph with constant edge-weights, the multi-terminal network flows problem consists in finding the all pairs maximum flows in the network. Clearly, this problem is solvable with $n(n-1)/2$

⋆ Partially supported by the Europeans projects RTN ARACNE and FET CRESCCO, and by the French AS *Dynamo*.

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 71–80, 2003.

single source–single terminal maximum flow computations. In 1961, Gomory and Hu [10] delivered an ingenious method to solve this maximum flow analysis problem using only $n-1$ maximum flow computations. They summarized their results in a tree referred to as GH cut tree (in the literature) reflecting the all pairs maximum flows in the network. Later in 1990, Gusfield [11] provided a simpler procedure to obtain the GH cut tree but using also $n-1$ maximum flow computations.

In 1964, Elmaghraby [8] was the first to extend the former problem to the case where a single edge is allowed to vary in capacity. He did the sensitivity analysis of the multi-terminal network flows, *i.e.*, the analysis of the capacity variation effects on the all pairs maximum flows in the network. As far as the variation is concerned, some maximum flows will be affected by the variation and others not. This give rise to the concept of *critical capacities*, *i.e.*, values for which a current flow changes of behavior by either beginning to vary with the parametric capacity or stopping to vary and remaining constant for the rest of the parameterization process. Thus the former sensitivity analysis problem turns to be the one of finding all the *critical capacities*, since the flows remain unchanged when the parametric capacity vary between two *critical capacities*. To solve this problem, Elmaghraby proposed an algorithm that computes as many GH cut trees as the number of possible critical capacities, leading to a pseudo-polynomial algorithm.

Recently, Diallo and Hamacher [6], showed that the analysis in [8] fails to determine all affected flows and provided an algorithmic improvement on the Elmaghraby analysis without improving the pseudo-polynomiality of the algorithm. In [5], it is proposed some pseudo-polynomial heuristics for the generalization of the Elmaghraby method.

In this paper we improve all methods and propose an efficient generalization of the sensitivity analysis of multi-terminal network flows. To do so, we start first by showing a fully polynomial algorithm that requires the computation of only two Gomory-Hu trees in order to solve the Elmaghraby sensitivity analysis problem. Second, since our algorithm is very simple compared to the existing pseudo-polynomial solution, it can be generalized to many varying capacities. Our main result is then that obtaining the all-pairs maximum flows for any value of the parameters, in the case where k edge capacities vary in the network, is polynomial whenever $k = O(\text{polylog } n)$, since we show that it can be solved with the computation of 2^k Gomory-Hu trees. In particular, the critical capacities become not a central key but a simple consequence of our results, since for more than one parametric edge-capacity, this notion of critical capacity may become more complex than obtaining directly the values of the maximum flows.

The multi-terminal network flows problem with constant capacity has many known applications in the fields of transports, energy and telecommunications (see for example [4, 5, 8] and references therein). The parametric multi-terminal network flows problem also reflects in these fields problems including link breakdown, capacity improvement, bandwidth reservation, network expansion.

In the remainder, we present in Section 2 some basic definitions and briefly describe the main ideas of Elmaghraby's method, and its improvements by Diallo and Hamacher. Section 3 is devoted to the details of our method in the case of a single edge-capacity variation. In Section 4, we provide the generalization to several parametric edge-capacities. We close the paper with concluding remarks and perspectives. Interested readers are referred to the technical report [2] where some examples illustrate our algorithms.

2 Definitions and Preliminaries

In this section we provide a precise definition of a GH cut tree and some preliminary results. Throughout this paper, we assume that the reader is familiar with general concepts of graph theory and network flows. For example, we refer to [1, 9, 12].

2.1 Definitions

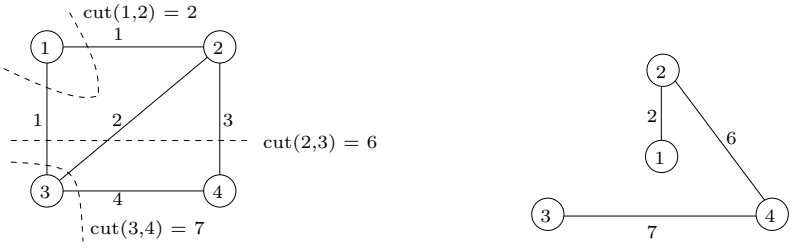
Let G be a connected undirected graph. We denote by $V(G)$ the vertex-set of G and by $E(G)$ its edge-set. A *network* is intended to be G associated to an edge-weight function $c : E(G) \to R^+$ called *edge-capacity* function. A flow from a source vertex s to a terminal vertex t in G is given by a function $f : E(G) \to R^+$. The flow has to be conserved in each vertex, excepted in vertices s and t, *i.e*, $\forall u \in V \setminus \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$. We notice that the graphs we deal with are symmetric thus each undirected edge $[u, v]$ can be replaced by two directed arcs (u, v) and (v, u), both with the same arc capacity, *i.e.*, $c(u, v) = c(v, u)$. For each $(u, v) \in E(G)$, $f(u, v) \leq c(u, v)$. For the sake of simplicity, we denote the maximum flow from a source s to a terminal t as f_{st}.

Definition 1. (GH cut tree) *Given a network $G = (V, E)$, a GH cut tree $T = (V, F)$ obtained from G is a weighted tree with the same set of vertices V with the two following properties:*

Equivalent flow tree: *the value of the maximum flow between any $s, t \in G$ is equal to the value of the maximum flow in T between s and t, i.e., the smallest of the capacities of the edges on the unique path from s to t in T; thus the maximum flows between all pairs of vertices in G are represented in T;*

Cut property: *the removal of any edge of capacity c from T separates its vertices into two classes, where the cut in G given by this partition has capacity c as well.*

In Figure 1(b), we illustrate a GH cut tree T obtained from the given network G with the illustrated minimum cuts in Figure 1(a). As shown in [10], $n-1$ min-cut computations are sufficient to obtain the global structure of the GH cut tree. We notice that GH cut trees are not unique. Algorithms to compute a GH cut tree are provided in [10] and in [11]. An experimental study of minimum cut algorithms and a comparison of algorithms producing GH cut trees are provided in [3].



(a) G (b) GH tree

Fig. 1. A graph and one of its Gomory-Hu cut trees

2.2 Sensitivity Analysis and All Critical Capacities

A natural extension of the multi-terminal network flows problem with constant edge-capacities is the analysis of the effects of a single edge-capacity parameterization on the all pairs maximum flows. This was effectively the aim of Elmaghraby in [8]. He mainly described a procedure to analyze the case where a single capacity decreases linearly and gave a sketch of the procedure to solve the capacity increasing case.

The sensitivity analysis problem solved in [8] can be stated as follows. Given the former network description, and an edge $e = (i, j)$ of $E(G)$ with a capacity given by $c(e) = \bar{c} - \epsilon$, $0 \leq \epsilon \leq \bar{c}$, where $\bar{c}$ is the initial capacity, one has to determine the set of all pairs maximum flows for all values of ϵ.

The Elmaghraby procedure can be briefly described as follows. With the decrease on the capacity $c(e)$, it is clear that some maximum flow values will be modified and others not. If the edge e is present in a minimum cut, this implies a reduction in the respective maximum flow value. As far as the variation is concerned, some maximum flow values that were constant may begin to decrease linearly with respect to ϵ. The value of $c(e)$ for which some flow changes of behavior is called *critical capacity*. Thus, in the interval between two critical capacities prevails the *status quo*, *i.e.* , no maximum flow behavior change occurs. With this remark, Elmaghraby transfers the sensitivity analysis of multi-terminal network flows to the problem of determining all the critical capacities, since in each such interval a GH cut tree computation provides the desired maximum flows.

In order to obtain the first critical capacity $\hat{\lambda}^0$, a GH cut tree is computed with the parameter set to zero. From this cut tree, an incident-like $E(G) \times (n-1)$ matrix is constructed [7]. This matrix defines the minimum cuts in the cut tree in terms of the original edges in G. Thus, based on this matrix, one identifies the set $\mathcal{C}$ of the minimum cuts that contain the parameterized edge and its complement set $\bar{\mathcal{C}}$ in the cut tree. The critical capacity $\hat{\lambda}^0$ is then the smallest value of the parameter λ for which a cut would move from $\bar{\mathcal{C}}$ to $\mathcal{C}$. With $\hat{\lambda}^0$ at hand, in order to compute the next critical value, we decrease the parametric

capacity in G by $\hat{\lambda}^0$ and repeat the same procedure with this updated capacity. The analysis ends when the parametric capacity is null, *i.e.*, $\epsilon = \bar{c}$.

In [8], it is described with detail how to obtain the critical capacities. We just emphasize that *to obtain each critical capacity, a GH cut tree computation is needed, thus leading to a pseudo-polynomial algorithm.*

Furthermore, when the parameter is negative, *i.e.*, when the capacity of the edge increases, Elmaghraby proposed to set the capacity of the tested edge to a big enough value and do the backward process using the decreasing case procedure to determine the critical capacities. What again leads to another pseudo-polynomial algorithm with some additional operations.

In [6], Diallo and Hamacher showed that the $E(G)\times(n-1)$ matrix used by Elmaghraby [7] to determine the minimum cuts that contain the investigated edge may fail to provide all such minimum cuts. They provided a counter-example and deliver a very simple algorithm to generally test whether or not a minimum cut between a given pair of vertices contains a given edge.

In the sequel, we show that if a single capacity is varying, then two GH cut trees are enough to compute all the critical capacities. Furthermore, the method we provide delivers simultaneously the critical capacities for both the capacity decreasing and capacity increasing problems. Nevertheless, the advantage of Elmaghraby's method is that one can obtain a minimum cut for any value of the parameter.

3 Computing Critical Capacities with Two GH Cut Trees

In this section, we show that if a single capacity is parameterized, then only two GH cut trees are needed to obtain all the critical capacities. To do so, we first provide a way to obtain the critical capacity with respect to a single maximum flow, and then, in Section 3.2, we explore this result to obtain the critical capacities for the all pairs maximum flows.

3.1 Critical Capacity with Respect to a Single Maximum Flow

For s and t two vertices of G, we define $f_{s,t}(\lambda)$ as the value of the maximum flow between s and t when the capacity of the edge e is λ. We denote by $f^0_{s,t}$ (or simply f^0) the maximum flow $f_{s,t}(0)$, *i.e.* , when the edge e is removed from the network, and by $f^\infty_{s,t}$ (or simply f^∞) the $\lim_{\lambda\to\infty} f_{s,t}(\lambda)$, *i.e.* , the maximum flow when there is no constraint on the edge e. This latter value is finite for all pairs, except when $(s,t) = e$. It can be simply computed by setting the capacity of the tested edge e to the sum of the capacities of its adjacent edges.

One interesting point is to observe the global behavior of the function $f_{s,t}(\lambda)$, for a given pair (s,t), for which the maximum value changes of behavior during the parameterization. As shown in Figure 2, it is composed by two distinct parts:

- as far as the capacity λ of the edge e increases, the maximum flow increases in the same way. During this stage, the parameterized edge is present in any s,t-minimum cut;

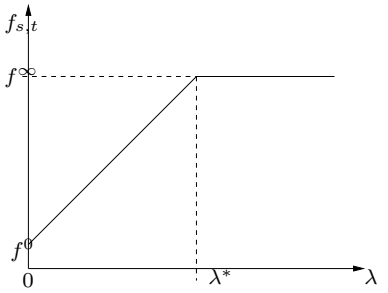


Fig. 2. behavior of a sensitive maximum flow function $f_{s,t}(\lambda)$

– at some value λ^* of λ, namely the critical capacity, the maximum flow becomes saturated (except if $\{s,t\} = e$). During this stage, the parameterized edge is out of all s,t-minimum cuts.

However, a maximum flow f_{st} may never depend on the parameter, in this case, $f^0_{s,t} = f^{\infty}_{s,t}$, thus we admit that its critical capacity is $\lambda^*_{s,t} = 0$.

Notice that $f_{ij} \to \infty$ as $\lambda \to \infty$, thus by convention we also admit that $f^{\infty}_{i,j} = \infty$.

Lemma 1. *Let $G = (V, E)$ be a network and $e = (i, j)$ one edge of $E(G)$ with parametric capacity $\lambda \geq 0$. Let p and q be two vertices of G. The critical capacity $\lambda^*_{p,q}$ exists if $\{p, q\} \neq \{i, j\}$ and satisfies:*

$$\lambda^*_{p,q} = f^{\infty}_{p,q} - f^0_{p,q}. \tag{1}$$

Proof. This is a direct consequence of the behavior of the maximum flow function: it grows linearly from $f^0_{p,q}$ up to $f^{\infty}_{p,q}$, thus the breakpoint is when the capacity equals to $f^{\infty}_{p,q} - f^0_{p,q}$. □

Corollary 1. *The critical capacity λ^* of λ for an arbitrary maximum flow can be computed using only two maximum flow computations.*

Proof. Using Lemma 1, we deduce that only f^0 and f^{∞} are necessary to compute λ^*. Furthermore, in order to compute f^{∞}, the result of f^0 can be used as initial value. □

Corollary 2. *Let $G = (V, E)$ be a network and $e = (i, j)$ one edge of $E(G)$ with capacity $\lambda \geq 0$. Let s and t be two vertices of G. The maximum flow $f_{s,t}(\lambda)$ verifies:*

$$f_{s,t}(\lambda) = \begin{cases} f^0_{s,t} + \lambda & \text{if } \lambda < f^{\infty}_{s,t} - f^0_{s,t} \\ f^{\infty}_{s,t} & \text{otherwise} \end{cases} \tag{2}$$

or more simply:

$$f_{s,t}(\lambda) = \min(f^0_{s,t} + \lambda, f^{\infty}_{s,t}). \tag{3}$$

Note that, in the case of a network for which the investigated edge is a cut-edge, then, the previous formula is also valid, since when the investigated edge is removed, the maximum flow between two vertices, one in each side of the cut-edge, is null due to disconnectivity, and the critical capacity is simply f^{∞}.

3.2 Critical Capacities for the All Pairs Maximum Flows

Theorem 1. *Let $G = (V, E)$ be a network with n nodes. If only one edge capacity is allowed to vary, then the set of all critical capacities can be computed using two GH cut trees followed by $O(n^2)$ operations.*

Proof. Two remarks shall be made. First, for a given single pair of vertices s and t of the network, Lemma 1 provides a simple way to compute the unique critical capacity if we know the values of the maximum flows $f^0_{s,t}$ and $f^{\infty}_{s,t}$. Second, the GH procedure provides an efficient way to compute the all pairs maximum flows.

Thus, the desired result can be obtained by the computation of a GH cut tree in the absence of the investigated edge e in order to obtain all the $f^0_{s,t}$, $\forall s, t \in V(G)$, and the computation of a second GH cut tree where the capacity of the edge e is set to ∞. This latter computation provides all the $f^{\infty}_{s,t}$ $\forall s, t \in V(G)$. With these two values of maximum flows at hand, using Lemma 1, one get all critical capacities by computing and sorting increasingly the differences

$$f^{\infty}_{s,t} - f^0_{s,t}, \quad \forall s, t \in V(G).$$

The final step considers $n(n-1)$ pairs of vertices, thus it takes $O(n^2)$ operations to be performed. □

The algorithm can be directly obtained from this proof. Notice that it does not matter the case (increasing or decreasing) we deal with. Once the algorithm is performed, the critical values for both type of cases can be obtained.

4 Analyzing the Effects of Several Parametric Edge-Capacities

In this section, we study the case where more than one capacity vary either or not independently.

4.1 Analyzing the Effects of Two Parametric Capacities

We examine in detail the case where the capacities of two edges vary independently. The main result of this section is that the algorithm given by the former Theorem 1 can also be applied in the current case, and only four (actually $\mathbf{2^2}$) maximum flow computations are needed to compute any maximum flow value, whatever the value of the capacities of the selected edges are.

The general problem in this section can be formally stated as follows. Given a network $G = (V, E)$ and two distinct edges e_1 and e_2, we want to determine the maximum flow between all pairs of vertices when the capacity of e_1 is λ and the capacity of e_2 is μ, where $\lambda, \mu \geq 0$ are varying.

On the effects of two parametric capacities on a single maximum flow. In this paragraph, we consider two selected vertices s and t in the network and provide a way to compute $f_{s,t}(\lambda, \mu)$. Before stating our results, one can remark that all the partial functions $\lambda \mapsto f_{s,t}(\lambda, \mu_0)$, with μ_0 fixed, have the same profile as illustrated in Figure 2: the maximum flow first increases up to a saturation step (critical capacity), and then stagnates. The partial functions $\mu \mapsto f_{s,t}(\lambda_0, \mu)$, with λ_0 fixed, behave analogously.

As previously, we denote $f_{s,t}^{0,0}$ the maximum flow between s and t when both edges e_1 and e_2 are removed (respective capacities set to 0) from the network. The flows $f_{s,t}^{0,\infty}$, $f_{s,t}^{\infty,0}$ and $f_{s,t}^{\infty,\infty}$ can be defined in a similar way considering the capacities $c(e_1)$ and $c(e_2)$ set to 0 or to ∞. In other words we denote the maximum flows $f_{s,t}^{0,0}$, $f_{s,t}^{0,\infty}$, $f_{s,t}^{\infty,0}$ and $f_{s,t}^{\infty,\infty}$ as *extreme flows*.

Theorem 2. *Let $G = (V, E)$ be a network, e_1 and e_2 two different edges of E, and s and t two distinct vertices of V. Then, the maximum flow $(f_{s,t}(\lambda, \mu))$ between s and t with the capacity of e_1 set to λ and the capacity of e_2 set to μ can be directly obtained from the four maximum flows $f_{s,t}^{0,0}$, $f_{s,t}^{0,\infty}$, $f_{s,t}^{\infty,0}$ and $f_{s,t}^{\infty,\infty}$. The maximum flow $(f_{s,t}(\lambda, \mu))$ can be computed as follows:*

$$f_{s,t}(\lambda, \mu) = \min(f_{s,t}^{0,0} + \mu + \lambda, f_{s,t}^{0,\infty} + \lambda, f_{s,t}^{\infty,0} + \mu, f_{s,t}^{\infty,\infty}). \tag{4}$$

Proof. The main point of this proof is to decompose the computation of the general maximum flow into several computations of simple maximum flows and use Corollary 2 to obtain the desired values.

Thus, let us consider that μ is fixed. As noted previously, the partial function $\lambda \mapsto f_{s,t}(\lambda, \mu)$ can be obtained if both maximum flows $f_{s,t}(0, \mu)$ and $f_{s,t}(\infty, \mu)$ are known by using Corollary 2 or its closed form given in Equation 3:

$$f_{s,t}(\lambda, \mu) = \min(f_{s,t}(0, \mu) + \lambda, f_{s,t}(\infty, \mu)) \tag{5}$$

At this step, it remains to compute $f_{s,t}(0, \mu)$ and $f_{s,t}(\infty, \mu)$. For the former, we consider the partial function $\mu \mapsto f_{s,t}(0, \mu)$. Again, this function can be described using Corollary 2:

$$f_{s,t}(0, \mu) = \min(f_{s,t}(0, 0) + \mu, f_{s,t}(0, \infty)) \tag{6}$$

Using their definitions, Equation 6 can be rewritten as:

$$f_{s,t}(0, \mu) = \min(f_{s,t}^{0,0} + \mu, f_{s,t}^{0,\infty}) \tag{7}$$

Similarly, $f_{s,t}(\infty, \mu_0)$ can be obtained by the following equation:

$$f_{s,t}(\infty, \mu) = \min(f_{s,t}^{\infty,0} + \mu, f_{s,t}^{\infty,\infty}) \tag{8}$$

Consequently, we have:

$$f_{s,t}(\lambda, \mu) = \min(\min(f_{s,t}^{0,0} + \mu, f_{s,t}^{0,\infty}) + \lambda, \min(f_{s,t}^{\infty,0} + \mu, f_{s,t}^{\infty,\infty})) \tag{9}$$

Using simple properties of the min function, we have:

$$f_{s,t}(\lambda, \mu) = \min(f_{s,t}^{0,0} + \mu + \lambda, f_{s,t}^{0,\infty} + \lambda, f_{s,t}^{\infty,0} + \mu, f_{s,t}^{\infty,\infty}). \tag{10}$$

□

As previously, the proof yields an algorithm to compute the resulting maximum flows.

Effects on the All Pairs Maximum Flows. From the property given by Theorem 2, we can deduce the all pairs maximum flows theorem for two parametric capacities, given below.

Theorem 3. *Let $G = (V, E)$ be a network, and e_1 and e_2 be two different edges of E. Then, the all pairs parametric maximum flow problem can be solved using the computation of four GH cut trees if the capacity of both edges e_1 and e_2 vary.*

Proof. This is a direct consequence of Theorem 2. We need to compute all the $f_{s,t}^{0,0}$, $f_{s,t}^{0,\infty}$, $f_{s,t}^{\infty,0}$ and $f_{s,t}^{\infty,\infty}$ for all the pairs (s, t) of vertices. The set of $f_{s,t}^{0,0}$, for all the vertices s and t can be obtained by the computation of a GH cut tree considering the network G in which have been removed both edges e_1 and e_2. The three other GH cut trees can be obtained similarly considering the presence or the removal of each tested edge with the infinite capacity.

Once the four GH cut-trees are computed, all values of maximum flows can be obtained using Theorem 2. The four GH cut trees provide the four extremal maximum flows and the desired maximum flow can be obtained using Equation 4. □

4.2 On the Generalization to Several Parametric Capacities

In this section, we consider the case where the capacities of more than two edges vary. Let $e_1, e_2, \ldots, e_k$ be the selected edge capacities that will be parametrized by $\lambda_1, \lambda_2, \ldots$ and λ_k, $k \leq m$, where m is the number of edges in the network.

Theorem 4. *Let $G = (V, E)$ be a network, k be an integer, and $e_1, e_2, \ldots, e_k$ be k different edges. All pairs maximum flows for all values of the parameters can be computed by using 2^k GH cut tree computations if the capacities of the edges vary independently.*

Proof. This result can be obtained by a simple recurrence. The basic case $k = 1$ is solved in Section 3.2. The sketch of the general case strictly follows the proof of Theorem 3. The main idea is to consider as fixed one of the parameters and use the recursion hypothesis for this case, leading to 2^{k-1} maximum flow computations. Then, it remains to develop each maximum flow computation in terms of the final dimension. Thus, for each computation, two maximum flows are necessary by using Corollary 2, leading to 2^k maximum flow computations.

The graphs on which the GH cut trees have to be computed are the variations of the initial graphs where the considered edges are either removed or their capacity set to infinity. □

5 Conclusion

In this paper, we have shown how to obtain efficiently allways the all pairs maximum flows when the capacities of some edges are parameterized. Previously, to solve the single parametric capacity case, as many GH cut tree computations were needed as the number of critical capacities, whereas in our approach, only two GH cut trees are required. Our work provides on the one hand a major improvement for the single varying capacity case and on the other hand, provides an efficient algorithm to solve the generalized varying capacities case.

An algorithmic improvement on our paper would be to provide a more efficient way to reuse a GH cut tree to compute a next one. We know that $(n-1)$ maximum flow computations are required to compute one GH cut tree. However, since there exists an important relationship between all the graphs for which we compute the GH cut trees, there should be a way to compute less than $2^k(n-1)$ maximum flows overall. We can expect a significant improvement on computation time in this way.

References

1. C. Berge. *Graphs and Hypergraphs*. North Holland, Amsterdam, 1973.
2. P. Berthomé, M. Diallo, and A. Ferreira. Generalized parametric multi-terminal flow problem. Technical Report LRI-1355, LRI, March 2003. available at http://www.lri.fr/~ berthome/.
3. C.S. Chekuri, A.V. Goldberg, D.R. Karger, M.S. Levine, and C. Stein. Experimental study of minimum cut algorithms. In *ACM Symposium On Discrete Algorithms*, pages 324–333, New Orleans, Louisiana, 5–7 January 1997.
4. J. Cohen and E.P. Duarte Jr. Fault-tolerant routing of TCP/IP PDU's on general topology backbones. In *Design of Reliable Communication Networks*, Budapest, Hungary, October 2001.
5. M. Diallo. Parametric multi-terminal networks flow analysis. Master's thesis, Dpt of Mathematics, University of Kaiserslautern, Germany, October 2000.
6. M. Diallo and H.W. Hamacher. Parametric capacity and multi-terminal network flows. Technical Report 60, AG-Hamacher, University of Kaiserslautern, Germany, 2002.
7. S.E. Elmaghraby. On the relationship between the cut-tree and the fundamental cut-set of multi-terminal flow networks. *Journal of Franklin Institute*, October 1964.
8. S.E. Elmaghraby. Sensitivity analysis of multi-terminal network flows. *J. ORSA*, 12:680–688, 1964.
9. L.R. Ford and D.R. Fulkerson. *Flows in Networks*. Princeton Univ. Press, Princeton, NJ, 1973.
10. R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, December 1961.
11. D. Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal of Computing*, 19:143–155, 1990.
12. T. C. Hu. *Combinatorial Algorithms*. Addison-Wesley Publ. Comp., Reading, Mass., 1982.

Canonical Decomposition of Outerplanar Maps and Application to Enumeration, Coding, and Generation

(Extended Abstract)

Nicolas Bonichon, Cyril Gavoille, and Nicolas Hanusse

LaBRI*, Université Bordeaux I, France.
{bonichon,gavoille,hanusse}@labri.fr

Abstract. In this article we define a canonical decomposition of rooted outerplanar maps into a spanning tree and a list of edges. This decomposition, constructible in linear time, implies the existence of bijection between rooted outerplanar maps with n nodes and bicolored rooted ordered trees with n nodes where all the nodes of the last branch are colored white. As a consequence, for rooted outerplanar maps of n nodes, we derive:

- an enumeration formula, and an asymptotic of $2^{3n-\Theta(\log n)}$;
- an optimal data structure of asymptotically $3n$ bits, built in $O(n)$ time, supporting adjacency and degree queries in worst-case constant time;
- an $O(n)$ expected time uniform random generating algorithm.

1 Introduction

A graph is *outerplanar* if it can be drawn on the plane with non-intersecting edges such that all the nodes lie on the boundary of the infinite face, also called *outerface*. Characterization of outerplanar graphs has been given by Chartrand and Harary [CH67]: a graph is outerplanar if and only if it has neither $K_{2,3}$ nor K_4 as a minor. A linear time recognition algorithm has been given by Mitchell [Mit79]. Labeled and unlabeled outerplanar graphs can be randomly generated in $O(n^4 \log n)$ space and $O(n^2)$ time [BK03] after a preprocessing of $O(n^5)$ time. Among graph properties, outerplanar graphs contain trees, have tree-width at most two, and are exactly the graphs of *pagenumber* one [Bil92]. Recall that a graph G has pagenumber k if k is the smaller integer for which G has a *k-page* embedding, also called *book* embedding. In such an embedding the nodes are drawn on a straight line (the spine of a book), and the edges are partitioned into k *pages*, each page consisting of non-intersecting edges.

* Laboratoire Bordelais de Recherche en Informatique

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 81–92, 2003.

A *planar map* is a connected graph drawn on the sphere with non-intersecting edges (see [CM92] for a survey). A planar map is *outerplanar* if all the nodes lie on one face, called the outerface. For convenience, outerplanar maps are drawn on the plane such that the outerface corresponds to the infinite face. A map is *rooted* if one of its edges, the root, is distinguished and oriented. In this case, the map is drawn on the plane in such a way that whenever traveling clockwise around the boundary of the outerface, the tail of the root edge is traversed before its head. A *planted tree* is the rooted planar map of a rooted tree such that the tail of the root of the map coincides with the root of the tree. In the literature, planted trees are also named ordered rooted trees. All the maps considered in this paper are planar, rooted, and are simple (have no loops and multi-edges).

Some sub-classes of outerplanar maps are well-known. Planted trees and maximal outerplanar maps, i.e., the map of an outerplanar graph with the maximum number of edges, are counted by the Catalan numbers. Finally, biconnected outerplanar maps can be seen as dissections of a convex polygon, and their number can be counted by Schröder numbers. For these three sub-classes there exist linear time random generation algorithms [ARS97,BdLP99,ES94].

Besides the combinatorial aspect and random generation, a lot of attention is given in Computer Science to *efficiently* represent discrete objects. By "efficiently", we mean that the representation should be succinct, i.e., the storage of these objects requires few bits, and that the time to compute such representation should be polynomial in its size. Fast manipulation of the so encoded objects and easy access to a part of the code are also desirable properties. Typically, adjacency query, i.e., check if two nodes are neighbors or not, and degree query, i.e., how many neighbors a node has, should be given very fast.

For instance, a folklore encoding of n-edge planted trees, based on a clockwise depth-first traversal, yields a representation with $2n - O(1)$ bits. This coding length is asymptotically optimal since the number of possible n-edge planted trees is the nth Catalan number $\frac{1}{n+1}\binom{2n}{n} \sim 2^{2n-O(\log n)}$. Completing this coding by an efficient data structures of length $o(n)$ bits, it has been shown in [MR01, CLL01] that adjacency and degree queries can then be answered in constant time, assuming that: 1) nodes of the tree are labeled according to the depth-first traversal (i.e., the node i must be the ith node encountered in the clockwise prefix order of the tree); and 2) standard arithmetic operations on integers of $\Omega(\log n)$ bits can be performed in constant time.

Outerplanar graphs are an interesting class of graphs because they are isomorphic to graphs of pagenumber one. Our contribution is an optimal $3n$-bit encoding for outerplanar maps. We point out that there exist many 1-page embeddings for a graph of pagenumber one. From the asymptotic formula of Flajolet and Noy [FN99], any encoding of 1-page embeddings requires $3.37n$ bits[1].

Let us sketch our technique. First we show that an outerplanar map admits a canonical decomposition into a particular rooted spanning tree (called *well-*

[1] In their article, 1-page embeddings are called non-crossing graphs.

orderly tree and defined in Section 2), and a set of additional edges (u, v) such that v is the first node after u (in a clockwise preorder of the tree) that is not a descendant of v. This decomposition can be computed in linear time. Then, we give three applications to this decomposition: enumeration formula (Section 3), efficient encoding (Section 4), and random generation algorithm with experiments (Section 5).

Hereafter we denote by $T_{n,d}$ the number of n-node planted trees where the depth of the clockwise last leaf is d (formulas for these numbers are given in [Fel68,Kre70]). Our canonical decomposition gives a bijection between outerplanar maps and bicolored trees (more precisely planted trees in which the nodes are colored either white or black), and where all the nodes (including the root) of the clockwise last branch are colored white. Clearly these last objects are counted by the following numbers: $\sum_{d=1}^{n-1} 2^{n-d-1} T_{n,d}$. From this bijection, we show that the number of n-node outerplanar maps is asymptotically $\frac{2^{3n}}{36n\sqrt{\pi n}}$.

An information-theoretic optimal encoding algorithm is deduced from the previous decomposition. It takes a $O(n)$ time and uses at most $3n - 6$ bits, for $n \geqslant 2$, as follows: $2n-4$ bits are used to encode the spanning tree, and $n-2$ bits (at most) are used to encode the additional edges. Adding a standard $o(n)$-bit data structure to this coding [MR01,CLL01], adjacency and degree queries can be then answered in worst-case constant time.

Using a grammar to produce bicolored rooted ordered trees with n nodes where all the nodes of the last branch are colored white, and using Goldwurm's algorithm [Gol95], a random outerplanar map can be generated uniformly with $O(n)$ space and $O(n^2)$ average time. Using Floating-Point Arithmetic [DZ99], this average time complexity can be reduced to $O(n^{1+\epsilon})$. In Section 5, we propose a $O(n)$ expected time and $O(n)$ space complexity generating algorithm. It can generate outerplanar maps with a given number of nodes, or with a given number of nodes and of edges.

Due to space limitations, some proofs are not given.

2 The Well-Orderly Tree of an Outerplanar Map

In [BGH03] the authors introduced the *well-orderly trees*, a special case of the *orderly spanning trees* [CLL01]. Let T be a rooted spanning tree of a planar map H. Two nodes are *unrelated* if neither of them is an ancestor of the other in T. An edge of H is unrelated if its endpoints are unrelated. Let $v_1, v_2 \ldots, v_n$ be the clockwise preordering of the nodes in T. A node v_i is *well-orderly* in H w.r.t. T if the incident edges of v_i in H form the following blocks (possibly empty) in clockwise order around v_i:

- $B_P(v_i)$: the edge incident to the parent of v_i;
- $B_<(v_i)$: unrelated edges incident to nodes v_j with $j < i$;
- $B_C(v_i)$: edges incident to the children of v_i;

– $B_>(v_i)$: unrelated edges incident to nodes v_j with $j > i$; and
– the clockwise first edge $(v_i, v_j) \in B_>(v_i)$, if it exists, verifies that v_i is a descendant of the parent of v_j;

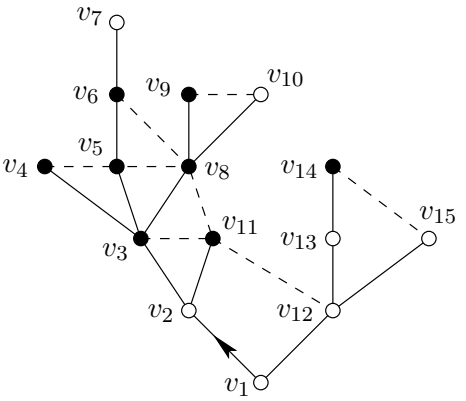


Fig. 1. A rooted outerplanar map, and its well-orderly tree depicted with solid edges.

T is a *well-orderly tree* of H if all the nodes of T are well-orderly in H, and if the root of T belongs to the boundary of the outerface of H (see Fig. 1 for an example). Note that a well-orderly tree is necessarily a spanning tree. Observe also that for every edge e of H, with $e = (v_i, v_j)$ and $i < j$, we have either $e \in B_C(v_i)$ (i.e., $e \in T$), or $e \in B_>(v_i)$. For convenience, the clockwise first edge of $B_>(v_i)$, if it exists, is called the *front edge* of v_i.

All planar maps do not admit a well-orderly tree. Actually, we have:

Lemma 1 ([BGH03]). *Every rooted planar map admits at most one well-orderly tree rooted at the tail of the root edge of the map.*

We will show that in fact every outerplanar map admits a well-orderly tree. It can be computed by the following recursive algorithm Traversal. H is the rooted outerplanar map, and r is the tail of its root edge. Traversal$(H, \varnothing, r)$ returns the well-orderly tree T of H rooted at r, the second parameter is the current set of edges of the tree.

Theorem 1. *Every rooted outerplanar map H has a well-orderly tree T, computable in linear time, rooted at the tail of the root edge of H. Moreover, for every node u, if $(u, v) \in B_>(u)$, then v is the next unrelated node with u in the clockwise preordering of T. In particular, $|B_>(u)| \leqslant 1$ for every u.*

Proof. Let T be the set of edges returned by Traversal$(H, \varnothing, r)$ (cf. Algorithm 1), where r is the tail of the root edge of H. Let us denote by T_i and by v_i respectively

Algorithm 1 Traversal(H, T, u).

$C \leftarrow \{(u, v) \in H \mid v \notin T\}$
$T \leftarrow T \cup C$
for all edges $(u, v) \in C$ taken in the clockwise order around u **do**
  $T \leftarrow$ Traversal(H, T, v)
end for
return T

the second and the third parameters of the ith call of Traversal. We have $T_1 = \varnothing$ and $v_1 = r$. Note that there are exactly n calls to Traversal, where n is the number of nodes of H. By induction, T_i is a tree with i nodes, for every $i \in \{1, \ldots, n\}$. Therefore $T_n = T$ is a spanning tree of H. An important observation is that the clockwise preordering of the nodes of T is precisely $v_1, \ldots, v_n$, and that T_i is a subtree of T_{i+1}, for $i \in \{1, \ldots, n-1\}$.

This algorithm can be easily implemented to run in $O(n)$ time. Let us show that the tree T satisfies the properties of Theorem 1. This is done thanks to the following properties.

Unrelated Edges. Every edge of H is either in T, or an unrelated edge. Indeed, let (v_i, v_j) be any edge with $i < j$. Clearly, v_j is not an ancestor of v_i. When v_i is being treated: 1) if $v_j \in T_i$, then v_j is not a descendant of v_i, and thus (v_i, v_j) is an unrelated edge; and 2) if $v_j \notin T_i$, then v_j becomes a child of v_i in T_{i+1}, thus an edge of T_n since T_i is a subtree of T_{i+1}.

Blocks. The incident edges of v_i form clockwise around v_i the four blocks $B_P(v_i)$, $B_<(v_i)$, $B_C(v_i)$, and $B_>(v_i)$. Since T is a spanning tree of a planar map, all the edges of $B_<(v_i)$ are (clockwise) after $B_P(v_i)$ and before the edges of $B_>(v_i)$. Let us show that the edges of $B_C(v_i)$ are after the edges of $B_<(v_i)$ and before the edges of $B_>(v_i)$.

Let us consider the ith call of Traversal. Let (v_i, v_t) be the last edge (in clockwise order around v_i) toward a node that is before v_i in T_i. Let v_k be the nearest common ancestor of v_i and v_t. The path from v_k to v_i, the edge (v_i, v_t) and the path from v_t to v_k defines a region of the plane, R, distinct from the outerface. Since H is outerplanar there is no node inside the region R. So all the neighbors of v_i that are not in T_i follow the edge (v_i, v_t) in the clockwise order around v_i. Hence, the edges of $B_C(v_i)$ are after the edges of $B_<(v_i)$.

The same reasoning can be done to show that the edges of $B_C(v_i)$ are before the edges of $B_>(v_i)$.

Descendant of the Parent. For every $(v_i, v_j) \in B_>(v_i)$, v_i is a descendant of the parent of v_j. Assume $(v_i, v_j) \in B_>(v_i)$. In particular $(v_i, v_j) \notin T$. When v_i is being treated, the node v_j must be in T_i, otherwise v_j becomes a child of v_i in T_{i+1}, and thus in T. By construction, the only nodes v_j's of T_i that are after the node v_i in the prefix clockwise preordering of T_i are such that their parent is an ancestor of v_i.

At this point, we have proved that every node in T is well-orderly, and thus T is a well-orderly tree of H.

Next Unrelated Node. Let $(v_i, v_j) \in B_>(v_i)$. Let v_k be the first unrelated node with v_i clockwise after v_i in T. Assume that $v_j \neq v_k$, so $i < k < j$. Let v_t be the parent of v_j in T. The cycle composed of the path in T from v_t to v_i, and of the edges (v_i, v_j) and (v_j, v_t) defines a region of the plane, R, distinct from the outerface. Because $i < k < j$, v_k must belong to R or to its boundary. As v_j is the only node of the boundary of R that is unrelated with v_i, v_k must belong to R: a contradiction with the fact that H is an outerplanar map. It follows that $v_j = v_k$. Therefore, we have showed that $(v_i, v_j) \in B_>(v_i)$ implies that v_j is the next unrelated node with v_i in T.

This completes the proof of Theorem 1. □

Combining Lemma 1 and Theorem 1, it is clear that an outerplanar map H has an unique decomposition into a well-orderly tree T, and a set of edges of the kind $(u, v) \in B_>(u)$, where v is the next unrelated node after u in T. (Recall that an edge of H belongs either to T or to a $B_>$ block). Conversely, given T, and a piece of information on whether $B_>(u)$ is empty or not for each node u, one can uniquely determine the corresponding rooted outerplanar map. The coding of the cardinality of each $B_>$ block can be done by coloring the nodes of T as follows: if $|B_>(u)| = 0$, u is colored white, and if $|B_>(u)| = 1$, u is colored black. Observing that for every node u of the clockwise last branch of T, $|B_>(u)| = 0$, we obtain:

Corollary 1. *There is a bijection, computable in linear time, between the n-node bicolored rooted trees where all the nodes of the last branch (including the root) are colored white, and the n-node rooted outerplanar maps.*

Recall that a graph (or a map) is k-connected if G has more than k nodes and if, for every subset X of fewer than k nodes, $G \setminus X$ is connected [Die00]. *biconnected* is a synonym for 2-connected.

Theorem 2. *There is a bijection, computable in linear time, between the $(n-1)$-node bicolored rooted trees with a white root, all leaves colored in black, and the set of n-node rooted biconnected outerplanar maps.*

Observe that if the well-orderly tree of an outerplanar map H has its clockwise last leaf of depth d, then from Corollary 1 H has no more than $(n-1)+n-(d+1) = 2n-2-d$ edges. In particular, the depth of the last leaf of the well-orderly tree of any maximal outerplanar map (i.e., having $2n-3$ edges) must be $d = 1$. As the well-orderly tree is unique, this yields another bijective proof of the well known following result:

Corollary 2. *There is a bijection, computable in linear time, between maximal n-node outerplanar maps and planted trees with $n-1$ nodes.*

3 Enumeration of Outerplanar Maps

Let $T_{n,d}$ be the number of rooted n-node plane trees whose clockwise last leaf is of depth d. These numbers are called the ballot numbers (or Delannoy numbers), and we have [Fel68,Kre70]:

$$T_{n,d} \;=\; \frac{d}{2n-2-d}\binom{2n-2-d}{n-1-d}, \quad \text{for all } n > d > 0.$$

So, there are exactly $2^{n-1-d}T_{n,d}$ bicolored trees whose all the nodes of the last branch (including the root of the tree) are colored white. Let M_n be the number of n-node outerplanar maps.

Theorem 3. *For all $n \geqslant 2$, $M_n = \sum_{d=1}^{n-1} 2^{n-d-1}T_{n,d}$, and $M_n \sim \frac{2^{3n}}{36n\sqrt{\pi n}}$.*

Theorem 4. *The number $M_{n,m}$ of rooted outerplanar maps with $n \geqslant 3$ nodes and $m \geqslant n-1$ edges is:*

$$M_{n,m} \;=\; \sum_{d=1}^{2n-2-m} \binom{n-d-1}{m-n+1} T_{n,d}\ .$$

4 Coding Supporting Adjacency and Degree Queries

From Corollary 1, every outerplanar map with n nodes can be represented by a bicolored tree with n nodes. A standard coding for n-node planted trees uses $2n-4$ bits if $n \geqslant 2$, and the colors can be stored using $n-2$ bits at most, observing that the color associated to the last branch (containing the last leaf and the root) is known (white). For the tree encoding, we use a clockwise depth-first traversal of the tree, each edge being traversed twice starting from the root. During the traversal, we output "1" if the edge is traversed for the first time, and "0" otherwise. This leads to a $2(n-1)$-bit encoding. Actually, if the tree has at least one edge ($n \geqslant 2$) two bits can be saved observing that the previous coding always starts we "1" and ends with "0".

This leads to a $3n-6$ bits encoding. By Lemma 3, the length of this coding is asymptotically optimal, up to an adding factor of $O(\log n)$ bits. It follows:

Theorem 5. *Every n-node rooted outerplanar map or every outerplanar graph, $n \geqslant 2$, can be coded (and decoded) in $O(n)$ time and using a representation on at most $3n-6$ bits.*

In the following, we show how to extend this coding with an extra $o(n)$ bits (still constructible in linear time) so that the data structure supports adjacency and degree queries in worst-case constant time. For that we present below efficient well-known data structures for binary strings and balanced string of parentheses.

4.1 Constant Time Queries in Strings of Parentheses

Let S be a string of symbols defined over a given alphabet. We denote by $S[i]$ the ith symbol of S, $i \geqslant 1$. Let $\mathsf{select}(S, i, \star)$ be the position of the ith $\star$ in S. Let $\mathsf{rank}(S, j, \star)$ be the number of symbols $\star$ before or at the jth position of S. That is if $\mathsf{select}(S, i, \star) = j$ then $\mathsf{rank}(S, j, \star) = i$.

Now, let S be a string of open and closed parentheses, i.e., $\star \in \{(,)\}$. Two parentheses $S[i] = ($ and $S[j] =)$ match if $i < j$ and the difference of the number of open and closed parentheses between them is null. The string S is *balanced* if for each parenthesis, there is a matching parenthesis. Let $\mathsf{match}(S, i)$ be the position in S of the matching parenthesis of $S[i]$. $S[k]$ is *enclosed* by $S[i]$ and $S[j]$ if $i < k < j$. Let $\mathsf{enclose}(S, i_1, i_2)$ be the closest matching parenthesis pair (j_1, j_2) that encloses $S[i_1]$ and $S[i_2]$. Finally, let $\mathsf{wrapped}(S, j)$ denote twice the number of pairs of matching parentheses (i_1, i_2) such that $\mathsf{enclose}(S, i_1, i_2) = (j, \mathsf{match}(S, j))$.

The following results are valid in the word-RAM model, in which standard arithmetic operations on binary words of length $\Omega(\log n)$ can be done in constant time.

Lemma 2 ([MR01,CLL01]). *For every balanced string S of parentheses (or a binary string) of length $O(n)$, the operations* select, rank, match, enclose, *and* wrapped *can be done in worst-case constant time using an auxiliary table of $o(|S|)$ bits and $O(|S|)$ preprocessing time.*

Actually, Lemma 2 holds for the operations select and rank, even if S is not balanced.

4.2 Coding of Outerplanar Maps

We can associate to any planted tree T a balanced string of parentheses (or Dyck word) as follows: during a clockwise depth-first traversal of T starting at the root, whenever an edge is traversed for the first time, output an open parenthesis and otherwise output an closed parenthesis. For convenience, an open parenthesis (resp. a closed parenthesis) is added at the beginning (resp. at the end) of the output string. One can think this latter transformation as an extra edge entering in the root of T. If T has n nodes, the final string of parentheses contains $2n$ symbols as each of T (plus the extra edge) is traversed twice. The final string is called the *clockwise prefix coding* of T.

Consider T an n-node planted tree, and its clockwise prefix coding S_T. Let $v_1, \ldots, v_n$ be the clockwise preordering of the nodes of T. To perform efficiently queries on T, we label the node v_i by its index i, so that an adjacency query between the nodes v_i and v_j is simply the pair $\{i, j\}$. By construction of the clockwise prefix coding, the ith open parenthesis corresponds to the edge of T entering in v_i. And, the matching parenthesis of the ith open parenthesis

corresponds to the edge of T leaving v_i. In other words, each node v_i can be seen as a pair (i, j) of matching parentheses, i.e., such that $S_T[i]$ matches with $S_T[j]$ in S_T. See Example 1.

Lemma 2 and the clockwise prefix coding leads to two useful operations for the tree that can be done in constant time: adjacency and degree.

Two nodes are adjacent in T if one of them is the parent of the other one. One can determine the parent of a node v_i finding the position p of the ith open parenthesis using $p = \mathsf{select}(S_T, i, ($), and the position of the closest enclosing pair (j_1, j_2) of $(p, \mathsf{match}(S_T, p))$ with $(j_1, j_2) = \mathsf{enclose}(p, \mathsf{match}(S_T, p))$. Then, the parent of v_i is v_j where $j = \mathsf{rank}(S_T, j_1, ($.

As for the degree, one can check that the number of children of v_i is $\frac{1}{2}\mathsf{wrapped}(S_T, \mathsf{select}(S_T, i, ()))$. We just have to add 1 if v_i is not the root, i.e., if $i \neq 1$.

From the previous explanations, one can claim the following lemma:

Lemma 3 ([CLL01]). *Let T be an n-node planted tree and its clockwise prefix coding S_T. The parent, degree and adjacency of a node can be done in constant time using an auxiliary table of $o(|S_T|)$ bits and a $O(n)$ preprocessing time.*

We encode an outerplanar map H with two binary strings:

1. S_T of length $2n$: the clockwise prefix coding of the well-orderly tree T of H;
2. S_B of length n: a binary string such that $S_B[j] = |B_>(v_i)|$, where v_i is the node corresponding to the jth closed parenthesis of S_T.

Given j, i can be obtained by $i = \mathsf{rank}(S_T, \mathsf{match}(S_T, \mathsf{select}(S_T, j,))), ($. Observe that if $j > n - 2$, then v_i belongs to the last branch of T, and thus $|B_>(v_i)| = 0$. Thus, S_B can reduce to the first $n-2$ entries. To perform $|B_>(v_i)|$ given i, one can compute $j = \mathsf{rank}(S_T, \mathsf{match}(S_T, \mathsf{select}(S_T, i, ()),))$, and return $S_B[j]$.

The sequence of nodes $v_{i_1}, v_{i_2}, \ldots, v_{i_j}$ is a *branch* of T if v_{i_j} is a leaf, and if for every $t \in \{1, \ldots, j-1\}$, $v_{i_{t+1}}$ is the clockwise last child of v_{i_t}. The last branch consists therefore of all the nodes of the path between the root and the clockwise last leaf of T. The branches partition the nodes of T and there is exactly one branch per leaf. One can check that a branch of T corresponds to a maximal block of closed parentheses in S_T.

Example 1. The outerplanar map of Fig. 1 can be encoded by the following two strings: $S_T = ((((()(((()))(()()))()))(((()))()))$ and $S_B = 101110111010000$. The sequence of nodes v_3, v_8, v_{10} is a branch. To know if node v_3 has a front edge, do the following sequence of operations: $p = \mathsf{select}(S_T, 3, () = 3$, $\mathsf{match}(S_T, p) = 18$, $\mathsf{rank}(S_T, 18,)) = 8$. Since $S_B[8] = 1$, node v_3 has a front edge.

Lemma 4. *Let $v_{i_1}, \ldots, v_{i_j}$ be a branch of T. Then, $|B_<(v_{i_j+1})| = \sum_{t=1}^{j} |B_>(v_{i_t})|$. Moreover, $|B_<(v_{i_j+1})|$ can be computed in constant time.*

Theorem 6. *Every rooted outerplanar map with n nodes admits a $3n + o(n)$-bit encoding, built in linear time, such that adjacency and degree queries can be computed in worst-case constant time.*

Proof. We proposed a $3n$-bit encoding for a rooted outerplanar map. Adding auxiliary tables of size $o(n)$, Lemmas 2 and 3 provide us constant time for computing the parent and the degree of v_i in T.

Adjacency: it remains to check the adjacency for the edges of H that does not belong to T. Nodes v_i and v_j, with $i < j$, are adjacent in H if v_j is the next unrelated node after v_i (that is $j = r - 1$ where v_r is the leaf of the branch of v_i) and if $|B_>(v_i)| = 1$.

Degree: the degree of node v_i is the sum of the degree in the tree T (see Lemma 3), the cardinality of $B_<(v_i)$ (see Lemma 4) and of $B_>(v_i)$. □

Starting from a connected outerplanar graph, we can compute a rooted outerplanar map using the algorithm presented in [CNAO85]. So the previous results on outerplanar maps can also be applied to outerplanar graphs.

5 Uniform Random Generation

To randomly generate an outerplanar map, one can randomly generate a bicolored tree where the nodes of the last branch are colored white. Thanks to Corollary 1, one can then construct in linear time the corresponding random outerplanar map.

Theorem 7. *A rooted outerplanar map with n nodes or with n nodes and m edges can be generated uniformly at random in $O(n)$ expected time.*

Proof. Let $\mathcal{B}_{n,b}$ be the set of all bicolored rooted trees with n nodes such that:

1. the root is colored white;
2. the clockwise last leaf is colored white; and
3. there are exactly b nodes colored black.

Let $\mathcal{B}_n = \bigcup_{b=0}^{n-2} \mathcal{B}_{n,b}$. From the bijection between outerplanar maps and some bicolored trees (cf. Corollary 1), the outerplanar maps with n nodes are in bijection with a subset, say $\tilde{\mathcal{M}}_n$, of bicolored trees. Clearly, $\tilde{\mathcal{M}}_n \subset \mathcal{B}_n$. Similarly, the outerplanar maps with n nodes and m edges are in bijection with a subset, say $\tilde{\mathcal{M}}_{n,m}$, of bicolored trees. The trees of $\tilde{\mathcal{M}}_{n,m}$ have exactly $m - (n-1)$ black nodes, so $\tilde{\mathcal{M}}_{n,m} \subset \mathcal{B}_{n,m-n+1}$.

The algorithm we proposed is an accept-reject algorithm: it consists in repeating a uniformly generation of an element of $\mathcal{B}_n$ (or of $\mathcal{B}_{n,m-n+1}$) until we get

an element of $\tilde{\mathcal{M}}_n$ (or of $\tilde{\mathcal{M}}_{n,m}$). This clearly provides a uniform random generation on the set $\tilde{\mathcal{M}}_n$ (or on $\tilde{\mathcal{M}}_{n,m}$), and thus on the corresponding outerplanar maps.

A rooted tree can be generated in linear time using for example the Arnold and Sleep algorithm [AS80]. The colors of the $n-2$ nodes are colored black with probability $1/2$ (recall that the root and the last leaf are forced to be colored white by definition of $\mathcal{B}_n$). This provides an $O(n)$ time algorithm to generate an element of $\mathcal{B}_n$.

To generate an element of $\mathcal{B}_{n,m-n+1}$, we have to select exactly $b = m-n+1$ black nodes among $n-2$ (the root and the last leaf are forced to be colored white). Selecting b elements among $n-2$ can be done in $O(n)$ time [Wil77] (the procedure uses $O(n)$ arithmetic operations on $O(\log n)$ bit integers). This provides an $O(n)$ time algorithm to generate an element of $\mathcal{B}_{n,m-n+1}$.

Testing whether $T \in \mathcal{B}_n$ belongs to $\tilde{\mathcal{M}}_n$ (or whether $T \in \mathcal{B}_{n,m-n+1}$ belongs to $\tilde{\mathcal{M}}_{n,m}$) clearly takes $O(n)$ time: it suffices to test whether all the inner nodes of the last branch are white or not.

As the bijection between bicolored trees and maps is linear, it remains to show that the average number of rejects in the above procedure is bounded by a constant.

Observe that every tree $T \in \mathcal{B}_n$ whose the last leaf is of depth 1 belongs to $\tilde{\mathcal{M}}_n$ and to $\tilde{\mathcal{M}}_{n,m-n+1}$ as well. Thus the probability to accept a tree $T \in \mathcal{B}_n$ in $\tilde{\mathcal{M}}_n$ (or in $\tilde{\mathcal{M}}_{n,m-n+1}$) is at least (we use the fact that $\sum_{d=1}^{n-1} T_{n,d} = c_{n-1}$ and $T_{n,1} = c_{n-2}$, where $c_n = \frac{1}{n+1}\binom{2n}{n}$ is the nth Catalan number counting the number of n-edge rooted trees):

$$\frac{T_{n,1}2^{n-2}}{|\mathcal{B}_n|} = \frac{T_{n,1}2^{n-2}}{\sum_{d=1}^{n-1} T_{n,d}2^{n-2}} = \frac{c_{n-2}}{c_{n-1}} = \frac{n}{4n-6} > \frac{1}{4}\,.$$

It follows that the average number of rejects is at most 4, that completes the proof. □

The algorithm presented in the proof of Theorem 7 can be enhanced into an anticipated-reject version. In this version, the bicolored tree is constructed from the end (so from the last branch). As soon an inner black node appears in the last branch, we reject. The advantage of this version is that fewer random bits are needed to generate an outerplanar map since the expected length of the last branch is $O(1)$, so only $O(1)$ bits would be wasted before acceptation of the whole bicolored tree. An implementation of the enhanced algorithm is available in the PIGALE Library (*http://pigale.sourceforge.net/*).

References

[ARS97] L. Alonso, J. L. Rémy, and R. Schott. A linear-time algorithm for the generation of trees. *Algorithmica*, 17(2):162–182, 1997.

[AS80] D. B. Arnold and M. R. Sleep. Uniform random generation of balanced parenthesis strings. *ACM Trans. Programming Languages and Systems*, 2(1):122–128, 1980.

[BdLP99] E. Barcucci, A. del Lungo, and E. Pergola. Random generation of trees and other combinatorial objects. *Theoretical Computer Science*, 218(2):219–232, 1999.

[BGH03] Nicolas Bonichon, Cyril Gavoille, and Nicolas Hanusse. An information-theoretic upper bound of planar graphs using triangulation. In 20^{th} *Annual Symposium on Theoretical Aspects of Computer Scie nce (STACS)*, volume 2607 of Lecture Notes in Computer Science, pages 499–510. Springer, February 2003.

[Bil92] T. Bilski. Embedding graphs in books: A survey. *IEE Proceedings-E*, 139(2):134–138, March 1992.

[BK03] Manuel Bodirsky and Mihyun Kang. Generating random outerplanar graphs. In 1^{st} *Workshop on Algorithms for Listing, Counting, and Enumeration (ALICE)*, January 2003.

[CH67] G. Chartrand and F. Harary. Planar permutation graphs. *Ann. Inst. Henry Poincare, Sec. B3*, pages 433–438, 1967.

[CLL01] Yi-Ting Chiang, Ching-Chi Lin, and Hsueh-I Lu. Orderly spanning trees with applications to graph encoding and graph drawing. In 12^{th} *Symposium on Discrete Algorithms (SODA)*, pages 506–515. ACM-SIAM, January 2001.

[CM92] R. Cori and A. Machi. Maps, hypermaps and their automorphisms: a survey i, ii, iii. *Expo. Math.*, 10:403–467, 1992.

[CNAO85] Norishige Chiba, Takao Nishizeki, Shigenobu Abe, and Takao Ozawa. A linear algorithm for embedding planar graphs using pq-trees. *Journal of Computer and System Sciences*, 30(1):54–76, 1985.

[Die00] Reinhard Diestel. *Graph Theory (second edition)*, volume 173 of Graduate Texts in Mathematics. Springer, February 2000.

[DZ99] Alain Denise and Paul Zimmermann. Uniform random generation of decomposable structures using floating-point arithmetic. *Theoretical Computer Science*, 218:233–248, 1999.

[ES94] P. Epstein and J.-R. Sack. Generating triangulations at random. *ACM Trans. Model. and Comput. Simul.*, 4:267–278, 1994.

[Fel68] W. Feller. *An Introduction to Probability Theory and its Applications*, volume 1. John Wiley & Sons, 1968.

[FN99] P. Flajolet and M. Noy. Analytic combinatorics of non-crossing configurations. *Discrete Mathematics*, 204:203–229, 1999.

[Gol95] M. Goldwurm. Random generation of words in an algebraic language in linear binary space. *Information Processing Letters*, 54:229–233, 1995.

[Kre70] G. Kreweras. Sur les éventails de segments. *Cahiers du Bureau Universitaire de Recherche Opérationnelle*, 15:1–41, 1970.

[Mit79] S. L. Mitchell. Linear algorithms to recognize outerplanar and maximal outerplanar graphs. *Inform. Proc. Letters*, pages 229–232, 1979.

[MR01] J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses, static trees and planar graphs. *SIAM Journal on Computing*, 31(3):762–776, 2001.

[Wil77] H.S. Wilf. A unified setting for sequencing, ranking, and selection algorithms for combinatorial objects. *Advances in Mathematics*, 24:281–291, 1977.

The Complexity of the Matching-Cut Problem for Planar Graphs and Other Graph Classes
(Extended Abstract)

Paul Bonsma

Department of Applied Mathematics, University of Twente, The Netherlands
bonsma@math.utwente.nl

Abstract. The Matching-Cut problem is the problem to decide whether a graph has an edge cut that is also a matching. Chvátal studied this problem under the name of the Decomposable Graph Recognition problem, and proved the problem to be $\mathcal{NP}$-complete for graphs with maximum degree 4, and gave a polynomial algorithm for graphs with maximum degree 3. In this paper it is shown that the problem is $\mathcal{NP}$-complete when restricted to planar graphs with girth 5 and planar graphs with maximum degree 4. In addition, for claw-free graphs and planar graphs with girth at least 7 polynomial algorithms to find matching-cuts are described.

1 Introduction

Let $G = (V, E)$ be a graph. A matching is a subset of E such that no two edges in the set share a common end vertex. A subset M of E is an edge cut if there is a subset S of the vertices ($S \neq \emptyset$, $S \neq V$) such that M is the set of edges with exactly one end vertex in S. So an edge cut can be written as $[S, \overline{S}]$ (with $\overline{S} = V \backslash S$). The *Matching-Cut problem* is the problem to decide whether a given graph has an edge cut that is also a matching (a matching-cut):

Matching-Cut:
Instance: A graph $G = (V, E)$.
Question: Does G have an edge cut $[S, \overline{S}]$ that is also a matching?

Throughout this paper only connected graphs will be considered. A graph with a matching-cut is called *decomposable*, and other graphs *indecomposable*.

Previous Results. Chvátal [3] studied this problem under the name of the *Decomposable Graph Recognition problem*, and showed that the problem is $\mathcal{NP}$-complete for graphs with maximum degree 4 (using the 3-uniform Hypergraph Bicolorability problem), and gave a polynomial algorithm to solve the problem for graphs with maximum degree 3. Moshi [8] gave polynomial algorithms for this problem for line graphs and quadrangulated graphs. Le and Randerath [7]

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 93–105, 2003.

adapted Chvátal's construction to prove the $\mathcal{NP}$-completeness of the problem for bipartite graphs in which all vertices in one class of the bipartition have degree 3 and all vertices in the other class have degree 4. Recently, unaware of Chvátal's result, Patrignani and Pizzonia [9] also proved the $\mathcal{NP}$-completeness of the problem for graphs with maximum degree 4 using a different reduction, though from nearly the same problem (Not-All-Equal-3-Satisfiability). In addition they presented a linear time algorithm for series-parallel graphs. They also posed the question whether the problem is $\mathcal{NP}$-complete for the class of planar graphs.

Other results on matching-cuts appear in [4] and [5]: in [4] matching-cuts are studied in the context of network applications, and [5] contains an extremal result regarding matching cuts (see Section 4.1 for more details.)

New Results. In this paper the Matching-Cut problem is shown to be $\mathcal{NP}$-complete for planar graphs, using a reduction from Planar Graph 3-colorability. This is done in Section 2. By changing the components used in the transformation, in Section 3 the $\mathcal{NP}$-completeness of the problem is proved for the more restricted classes of planar graphs with girth 5 and planar graphs with maximum degree 4. Next, in Section 4 for some graph classes polynomial algorithms for Matching-Cut are described. These classes are claw-free graphs (this generalizes the result on line graphs) and planar graphs with girth at least 7. We also observe that for any fixed k, a linear time algorithm can be constructed to solve Matching-Cut for graphs with treewidth bounded by k. Such an algorithm generalizes the algorithm for series-parallel graphs.

2 The $\mathcal{NP}$-Completeness of Planar Matching-Cut

In this section, the $\mathcal{NP}$-completeness of the Matching-Cut problem is proved when instances are restricted to the class of planar graphs. This $\mathcal{NP}$-completeness proof is by a polynomial transformation from the following graph coloring problem:

Planar Graph 3-Colorability
Instance: A planar graph $G = (V, E)$
Question: Does G have a vertex coloring using at most 3 colors?

In [6], this problem is shown to be $\mathcal{NP}$-complete. In order to prove the $\mathcal{NP}$-completeness of Planar Matching-Cut, two intermediate problems will be used that are also $\mathcal{NP}$-complete. First we need the fact that Graph 3-Colorability is still $\mathcal{NP}$-complete when restricted to a smaller set of instances:

Planar Hamiltonian Graph 3-Colorability:
Instance: A planar graph $G = (V, E)$ with a Hamilton cycle $H \subseteq E$.
Question: Does G have a vertex coloring using at most 3 colors?

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

Before we can use Graph 3-Colorability to prove the $\mathcal{NP}$-completeness of Planar Matching-Cut, we transform the problem into an artificial, but more suitable vertex coloring problem in which every vertex is incident with only one vertex that has to be colored with a different color, and at most two that have to be colored with the same color:

Segment 3-Colorability
Instance: A set of vertices V and three edge sets A, B and C such that $G = (V, A \cup B \cup C)$ is a 3-regular planar multigraph with Hamilton cycle $A \cup B$.
Question: Can we find a color function $f : V \to \{1, 2, 3\}$ such that if $uv \in A$ then $f(u) = f(v)$ and if $uv \in C$ then $f(u) \neq f(v)$?

An instance of Segment 3-Colorability will be denoted by the multigraph $G = (V, A \cup B \cup C)$. Note that this instance gives a multigraph, so if for instance B and C both contain an edge between vertices u and v, in G these are considered to be different edges. Edges in the sets A, B and C will be called respectively A-edges, B-edges and C-edges. If $G = (V, A \cup B \cup C)$ is an instance of Segment 3-Colorability, the components of the graph (V, A) are called *segments*. All vertices of one segment receive the same color in a solution of this problem, and therefore the problem is called segment coloring. Note that C is a perfect matching in G.

Using lemma 1, we can prove the $\mathcal{NP}$-completeness of Segment 3-Colorability:

Lemma 2. *Segment 3-Colorability is $\mathcal{NP}$-complete.*

Proof: We can construct an instance G' of Segment 3-Colorability from an instance G of Planar Hamiltonian Graph 3-Colorability such that:

- the vertices of G correspond to the segments of G',
- the edges of G correspond to the edge set C of G': G' has a C-edge between vertices of segments x and y if and only if there is an edge between the vertices of G corresponding to x and y, and
- the vertex ordering given by the Hamilton cycle H in G is the same as the segment ordering given by the Hamilton cycle $A \cup B$ in G'.

Using the first two properties above, it is easy to see that Segment 3-Colorability on G' is equivalent with Planar Hamiltonian Graph 3-Colorability on G. An example of this transformation is shown in Figure 1. □

Now, using Lemma 2 we can prove the $\mathcal{NP}$-completeness of Planar Matching-Cut.

Theorem 1. *Planar Matching-Cut is $\mathcal{NP}$-complete*

Proof: For this proof, we transform an instance $G = (V, A \cup B \cup C)$ of Segment 3-Colorability into an instance G' of Planar Matching-Cut. For every vertex in V we will introduce a *vertex component* in G'. (Throughout this proof, we do

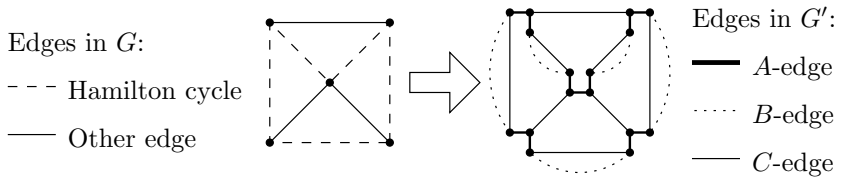


Fig. 1. An example of the transformation in the proof of Lemma 2

not use the graph theoretical meaning of the word component.) If two vertices in V are joined by an A-edge, the corresponding vertex components will be connected using a *vertex connection component*. If two vertices in V are joined by a B-edge, the corresponding vertex components will be connected using a *segment connection component*. In order to properly label the components that will be introduced, first label the *vertices* of G with labels $1, \ldots, k_1$. Label the *edges* in A with labels $k_1 + 1, \ldots, k_2$. Label the *edges* in B with $k_2 + 1, \ldots, k_3$. Note that for C-edges no components will be introduced.

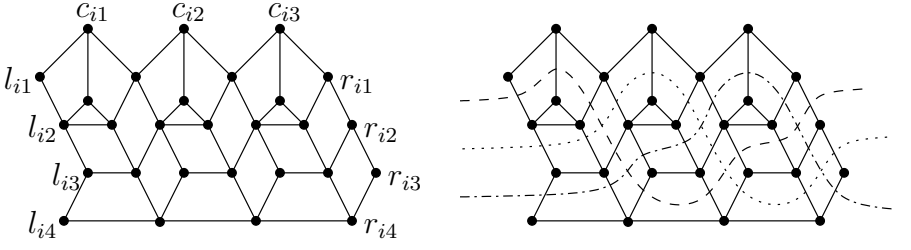


Fig. 2. A vertex component with three possible matching-cuts

For every vertex $i \in V(G)$, introduce a vertex component as shown in Figure 2. There are three possible matching-cuts through these components, also shown in Figure 2. It can be checked that these are the only matching-cuts. If edge $l_{ij}l_{i(j+1)}$ is in the matching-cut, we will say that *this vertex component is cut by cut* j *(*$j = 1, 2, 3$*)*. Below, this will correspond to a coloring of vertex i in G with color j. Observe that in this case, $r_{ij}r_{i(j+1)}$ is also in the matching-cut, and vertex c_{ij} is incident with one of the edges in the matching-cut, whereas c_{ik} for $k \neq j$ is not.

For each edge i in A ($i = k_1 + 1, \ldots, k_2$), we introduce a vertex connection component as shown in Figure 3. It is easy to see that there are exactly three possible matching-cuts through these components. Observe that if edge $l_{ij}l_{i(j+1)}$ is in the matching-cut, then edge $r_{ij}r_{i(j+1)}$ also is.

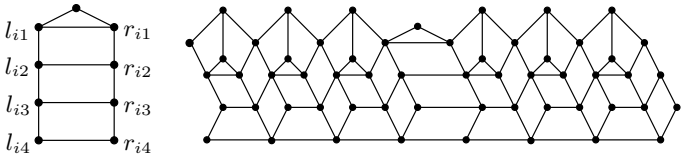


Fig. 3. A vertex connection component and a vertex connection

If in G, there is an A-edge k from i to j, the two vertex components i and j will be connected by vertex connection component k as follows: identify r_{il} with l_{kl} for $l = 1, 2, 3, 4$. This new vertex is called r_{il}. Now, for $l = 1, 2, 3$ there are two edges from r_{il} to $r_{i(l+1)}$. Delete one of these. Next, identify r_{kl} with l_{jl} for $l = 1, 2, 3, 4$ and call the new vertices l_{jl}. Also delete one of all the double edges introduced here. An example of such a *vertex connection* is also shown in Figure 3. This connection has the following property:

Property 1. If two vertex components i and j are connected by a vertex connection component, then for any matching-cut: i is cut by cut l if and only if j is cut by cut l ($l = 1, 2, 3$).

Clearly, to achieve this property, vertex components can also be connected without using a vertex connection component, but for the alternative constructions in the next section, vertex connection components are useful.

For each edge i in B ($i = k_2 + 1, \ldots, k_3$), we introduce a segment connection component as shown in Figure 4.

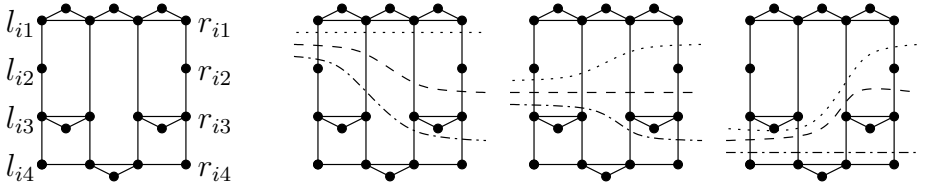


Fig. 4. A segment connection component and the nine possible matching-cuts

Because the triangle edges cannot be in a matching-cut, there are nine possible matching-cuts through these components, all of which cut exactly one edge $l_{ij}l_{i(j+1)}$ and one edge $r_{ik}r_{i(k+1)}$. There is one cut for every combination of j and k ($j = 1, 2, 3$, $k = 1, 2, 3$).

If in G there is a B-edge from i to j, we can connect the vertex components i and j using a segment connection component in the same way as described above for vertex connection components. This *segment connection* has the following property:

Property 2. If two vertex components i and j are connected by a segment connection component, then for any matching-cut: i is cut by this matching-cut if and only if j is cut by this matching-cut. Any combination of cuts through i and j is possible.

The C-edges of G determine the last type of connection between vertex components: if $ij \in C$, identify c_{il} with c_{jl} $(l = 1, 2, 3)$. This connection is called an *edge connection*. We know that if vertex component i is cut by cut l, then c_{il} is incident with an edge of this matching-cut. So because a matching-cut is a matching, this connection has the following property:

Property 3. If two vertex components i and j are connected by an edge connection, then for any matching-cut and $l = 1, 2, 3$: it is not possible that i is cut by cut l and j is cut by cut l.

Let G' be the graph that is constructed by introducing vertex components for every vertex in G and connecting them with vertex connection components, segment connection components and edge connections for every edge in respectively A, B and C as described above. Using the observed properties of these connections, G' has the following properties:

- Because $A \cup B$ gives a Hamilton cycle in G, and the fact that the different components have no matching-cuts other than the indicated cuts, we use Property 1 and Property 2 to conclude that if G' has a matching-cut, this matching-cut cuts every vertex component.
- Property 1 shows that if G' has a matching-cut, all vertex components that correspond to vertices in the same segment of G are cut by the same cut l.
- By Property 3, in any matching-cut, two vertex components that correspond to vertices in segments that are joined by a C-edge must be cut by different cuts.
- Property 2 shows that segment connection components do not impose additional constraints on the possible combinations of cuts through vertex components.

Now it is easy to see how any matching-cut in G' corresponds to a proper segment 3-coloring of G and vice versa.

The only question left is whether G' is indeed an instance for Planar Matching-Cut, so whether G' is planar. Actually, with vertices of the vertex components labeled as in Figure 2, the edge connections, vertex connections and segment connections can not be made without destroying planarity. But it is possible to find labelings for the different vertex components such that G' will be planar.

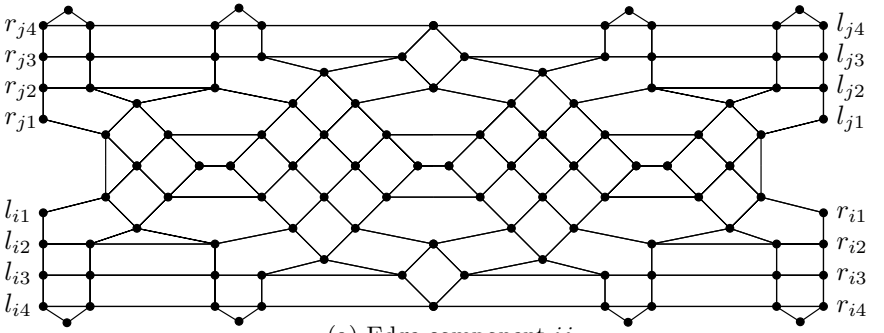
This concludes the $\mathcal{NP}$-completeness proof of Planar Matching-Cut. □

3 The $\mathcal{NP}$-Completeness of Matching-Cut for Several Other Graph Classes

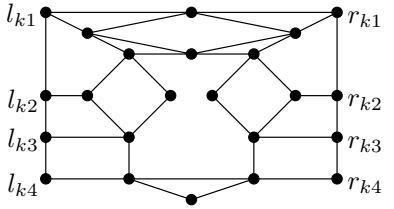
3.1 Planar Graphs with Maximum Degree 4

As Chvátal proved the Matching-Cut problem to be $\mathcal{NP}$-complete for graphs with maximum degree 4, the question arises whether the problem is also $\mathcal{NP}$-complete for planar graphs with maximum degree 4. To prove that this is indeed the case, we outline how the construction used in the proof of Theorem 1 should be altered such that the resulting graph G' is a planar graph with maximum degree 4, and still has the properties needed in the proof.

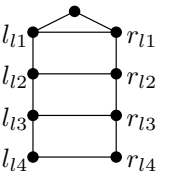
For this alteration, in the graph G' from the proof of Theorem 1, we will replace all segment connection components by new segment connection components. The vertex components will be replaced pairwise: in G', every vertex component occurs in a pair of vertex components connected with an edge connection. Call this subgraph, composed of vertex components i and j, *edge component* ij.



(a) Edge component ij



(b) Segment connection component k



(c) Vertex connection component l

Fig. 5. Components with maximum degree 4

Replace each edge component ij in G' by the new edge component ij shown in Figure 5(a). This component is obviously planar and has maximum degree 4. Furthermore, the vertices l_{ik}, r_{ik}, l_{jk} and r_{jk} $(k = 1, 2, 3, 4)$ have smaller degree

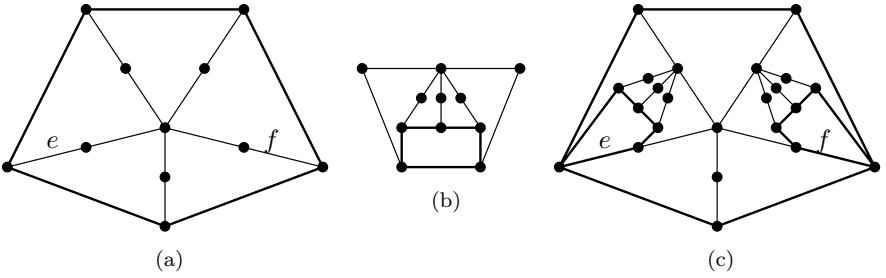


Fig. 6. Making an indecomposable graph with girth 5

such that further connections are possible. With a little effort it can be checked that the possible combinations of matching-cuts through this new edge component are similar to the possible combinations of matching-cuts through the original edge component. There is one difference: the matching-cuts are 'reversed', so for instance the matching-cut that contains the edge $l_{i1}l_{i2}$ also contains the edge $r_{i3}r_{i4}$ and the matching-cut that contains the edge $l_{i3}l_{i4}$ also contains the edge $r_{i1}r_{i2}$. This is easily remedied by replacing each edge component in G' by *two* of the edge components in Figure 5(a), and connecting them with two of the vertex connection components shown in Figure 5(c).

Replace each segment connection component (Figure 4) by the component shown in Figure 5(b). This component is again planar, has maximum degree 4 and has a set of matching-cuts equivalent to the nine matching-cuts of the original segment connection component. For the vertex connection components we again use the component shown in Figure 5(c).

If we replace all subgraphs of G' exactly as described, then next to some edge components, vertices of degree 5 will occur. This problem can easily be solved by inserting extra vertex connection components (with a double edge between l_{l4} and r_{l4} instead of between l_{l1} and r_{l1}) and/or deleting vertex connection components.

This shows how the construction can be altered such that the resulting graph has maximum degree 4 and still has all the necessary properties. Therefore, the Matching-Cut problem is $\mathcal{NP}$-complete for the class of planar graphs with maximum degree 4.

3.2 Planar Graphs with Large Girth

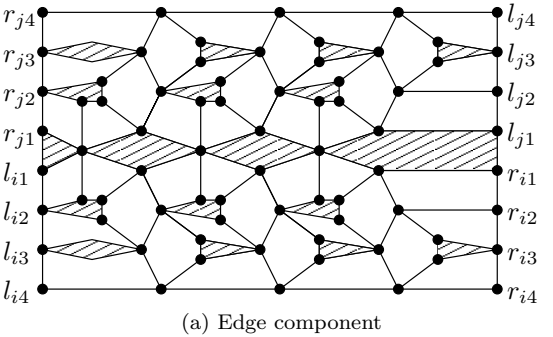
After we observe that the indecomposable graphs that we have seen all contain small cycles (triangles), the question arises whether the Matching-Cut problem is still $\mathcal{NP}$-complete for graphs with large girth (the girth of a graph is the length of a shortest cycle). First consider simple graphs without triangles or, for a stronger result, bipartite simple graphs. Moshi [8] observes that if in a

Matching-Cut instance G every edge is replaced by two parallel edges and both of those edges are subdivided with one vertex, this gives a bipartite Matching-Cut instance G' that is equivalent with G. This operation does not destroy planarity, so Matching-Cut is $\mathcal{NP}$-complete for planar bipartite graphs.

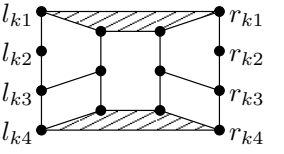
Before we can prove that the Matching-Cut problem is $\mathcal{NP}$-complete for planar graphs with girth 5, we must find such a graph that is indecomposable. In Figure 6(a), a planar graph with girth 5 is shown. It can be checked that none of the thick edges can be part of a matching-cut. A different embedding of this graph is shown in Figure 6(b). If we draw this second embedding into two of the faces of the first embedding, we obtain the planar graph with girth 5 in Figure 6(c). Again, the thick edges can not be part of a matching-cut. Therefore it can be checked that this graph has no matching-cuts.

In a similar way, for any $d \geq 5$ embeddings of indecomposable planar graphs with girth 5 can be constructed such that the outer face has degree d. Call such a graph an *indecomposable perimeter-d graph*.

Now we can outline how to change the proof of Theorem 1 such that the resulting graph is a planar graph with girth 5, which proves the $\mathcal{NP}$-completeness of the Matching-Cut problem for planar graphs with girth 5. Consider the components in Figure 7. Replace edge components and segment connection components in the graph G' from the proof of Theorem 1 by these new components. Recall that vertex connection components do not have to be used. Now, edges on the boundary of the shaded faces can be subdivided until all 2-cycles, 3-cycles and 4-cycles are removed (observe that in the graph obtained from G', all these small cycles contain such an edge). To ensure that none of the edges around the shaded faces can be part of a matching-cut, insert in every shaded face of degree d an indecomposable perimeter-d graph. By this 'insert' operation we mean the operation we also used to obtain the graph in Figure 6(c). It can be checked that the resulting graph again has girth 5. So this yields a planar graph of girth 5 that has the desired matching-cut properties.



(a) Edge component



(b) Segment connection component

Fig. 7. Components for the construction with girth 5

4 Graph Classes for Which Matching-Cut Is Easy

For some graph classes the matching-cut problem can be solved efficiently: Chvátal [3] described an algorithm for graphs with maximum degree 3, and Patrignani and Pizzonia [9] described an algorithm for series-parallel graphs. Moshi [8] describes algorithms for line graphs and quadrangulated graphs. Quadrangulated graphs are graphs without chordless cycles of length 5 or more. We first observe that the property of having a matching-cut can be described in monadic second order logic, and therefore results in [1] and [2] imply that for any graph class with fixed bounded treewidth the problem can be solved in polynomial time. This generalizes the result on series-parallel graphs. Below a few other positive results on the matching-cut problem are described. We first continue the study of the influence of the girth on the complexity of the problem for planar graphs.

4.1 Planar Graphs with Girth at Least Seven

In this section, it is shown that the Matching-Cut problem is easy for the class of planar graphs with girth at least 7. In fact, it is shown that every such graph has a matching-cut.

Theorem 2. *Every planar graph with girth at least 7 has a matching-cut*

Proof: Suppose this is not true. Let G be a planar graph with girth 7 without a matching cut. We have the following observations:

1. $\delta(G) \geq 2$.
2. G can not contain two vertices with degree 2 that are neighbors. Using the fact that G has no triangles, this structure gives a matching-cut.
3. G can not contain vertices v with at least $d(v) - 1$ neighbors with degree 2. Using the fact that G has no cycles of length 4 or less, this would give a matching-cut $[S, \overline{S}]$. (Let S be v together with its degree 2 neighbors.)

Now define for every $v \in V(G)$: $e(v) := |\{u \in N(v) : d(u) = 2\}|$, the number of neighbors of v with degree 2. So by Observation 3 above: $e(v) \leq d(v) - 2$ for all $v \in V(G)$. Now, if $d(v) = 3$ and $e(v) = 1$, call v a *critical 3-vertex*. If $d(v) = 3$ and $e(v) = 0$, call v a *simple 3-vertex*. Then in addition, we can observe the following:

4. G can not contain two critical 3-vertices that are neighbors. Using the fact that G has no cycles of length 5 or less, this would give a matching-cut $[S, \overline{S}]$. (Let S be these two vertices together with their degree 2 neighbors.)
5. G can not contain a simple 3-vertex with more than one critical 3-vertex as neighbor. Using the fact that G has no cycles of length 6 or less, this would give a matching-cut $[S, \overline{S}]$. (Let S be these degree 3 vertices together with their degree 2 neighbors.)

Now assign weights to the vertices in the following way:

$$w(v) := d(v) + e(v) \text{ if } d(v) \geq 3$$
$$w(v) := 0 \qquad\quad \text{if } d(v) = 2$$

Observation 2 above shows that every vertex of degree 2 contributes 1 to the weights of exactly two vertices of degree at least 3, and thus

$$\sum_v d(v) = \sum_v w(v).$$

Now for every critical 3-vertex v, with x and y the two neighbors with degree at least 3, apply the following redistribution of weights once:

$$w(v) := w(v) - \tfrac{1}{2}$$
$$w(x) := w(x) + \tfrac{1}{4}$$
$$w(y) := w(y) + \tfrac{1}{4}$$

Clearly, after this is done for every critical 3-vertex, the sum of the weights over all vertices remains the same. Now we will show that after this weight redistribution step: $w(v) \leq \frac{7}{2}(d(v) - 2)$ for every vertex v.

- If $d(v) = 2$, then $w(v)$ is not changed by the redistribution, so $w(v) = 0 = \frac{7}{2}(d(v) - 2)$.
- If v is a critical 3-vertex, then because of Observation 4, the redistribution will not add weight to $w(v)$, only substract once. So $w(v) = d(v)+e(v)-\frac{1}{2} = \frac{7}{2} = \frac{7}{2}(d(v) - 2)$.
- If v is a simple 3-vertex, then because of Observation 5, v will gain at most $\frac{1}{4}$ in the redistribution step. So $w(v) \leq d(v) + e(v) + \frac{1}{4} = \frac{13}{4} < \frac{7}{2}(d(v) - 2)$.
- If v has $d(v) \geq 4$, then v can gain at most $\frac{1}{4}$ for every neighbor with degree at least 3, so $w(v) \leq d(v) + e(v) + \frac{1}{4}(d(v) - e(v)) = \frac{5}{4}d(v) + \frac{3}{4}e(v) \leq \frac{5}{4}d(v) + \frac{3}{4}(d(v) - 2) = 2d(v) - \frac{3}{2} < \frac{7}{2}(d(v) - 2)$. For the last step we use $d(v) \geq 4$.

Now let g be the girth of G, and let n be the number of vertices, m the number of edges and k the number of faces in an embedding of G. Using Euler's formula we have $m - n = k - 2$. Let F be the set of faces in this embedding. Then for $f \in F$, $d(f)$ denotes the face degree. For graphs without bridges such as G, $d(f)$ is equal to the number of edges on the boundary of f. We get:

$$gk \leq \sum_{f \in F} d(f) = 2m = \sum_{v \in V} d(v) = \sum_{v \in V} w(v) \leq \sum_{v \in V} \frac{7}{2}(d(v) - 2) = \frac{7}{2}(2m - 2n) =$$

$$\frac{7}{2}(2k - 4) = 7k - 14,$$

so $g < 7$, a contradiction. □

Note that the proof above describes an efficient algorithm that will always find a matching-cut in planar graphs with girth at least 7: every such graph must have one of the structures mentioned in Observations 1 to 5 in the above proof.

Remark. At WG2003 the open question was presented whether the above result is also true for planar graphs with girth 6. Andrzej Proskurowski pointed out that [5] might be useful for answering this question. Indeed, the main result of [5] is that for every indecomposable graph on n vertices with m edges, $m \geq \lceil 3(n-1)/2 \rceil$, and that this bound is best possible. For planar graphs with girth 6 we have $m \geq 3k$. Rewriting Euler's formula then gives $m \leq 3(n-2)/2$, which proves that all planar graphs with girth at least 6 are decomposable. We believe that the alternative proof in this section, even though the result is weaker, is still useful for the understanding of the problem.

4.2 Claw-Free Graphs

A claw is a $K_{1,3}$. A claw-free graph is a graph without induced claws. For any connected claw-free graph $G = (V, E)$, let $M \subseteq E$ be the set of edges that are not part of a triangle. Since G is claw-free, $\Delta(V, M) \leq 2$, so the graph (V, M) is a set of paths and cycles. Let $U \subseteq V$ be the set of vertices with degree 2 in (V, M). If $v \in U$, then $d_G(v) = 2$. So if (V, M) contains a cycle, then because G is connected, G must be this cycle. The cycle has at least 4 vertices, so G has a matching-cut. Now assume (V, M) is a set of paths. Then

- if (V, M) contains a path on at least 4 vertices, then G contains two neighbors of degree 2 that are not part of a triangle, so G has a matching-cut.
- if $(V \backslash U, E \backslash M)$ is not connected, then G has a matching-cut M': for each path in (V, M), put one edge in M'. Now M' is a matching and an edge cut.

We will show that these are the only cases in which G has a matching-cut, if G is not a cycle. Suppose G has a matching-cut $[S, \overline{S}]$. $[S, \overline{S}]$ can only contain edges in M. If it contains two edges of the same path in (V, M), then this path must contain at least 4 vertices. Otherwise, there is a path in (V, M) with end vertices $u \in S$ and $v \in \overline{S}$. Also $u, v \in V \backslash U$. Because $[S, \overline{S}] \subseteq M$, u and v are also in different components of $(V \backslash U, E \backslash M)$, and therefore $(V \backslash U, E \backslash M)$ is not connected.

This gives a polynomial decision algorithm for Matching-Cut for claw-free graphs. We remark that in the $\mathcal{NP}$-completeness proof of Chvátal [3] a $K_{1,4}$-free graph is constructed, so for $K_{1,4}$-free graphs the Matching-Cut problem is $\mathcal{NP}$-complete. Line graphs are claw-free, so this generalizes the result in [8].

References

1. S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.
2. H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.
3. V. Chvátal. Recognizing decomposable graphs. *Journal of Graph Theory*, 8:51–53, 1984.
4. A. M. Farley and A. Proskurowski. Networks immune to isolated line failures. *Networks*, 12(4):393–403, 1982.

5. A. M. Farley and A. Proskurowski. Extremal graphs with no disconnecting matching. In *Proceedings of the 14th South-Eastern Conference on combinatorics, graph theory, and computing*, volume 41 of *Congressus Numerantium*, pages 153–165, 1984.
6. M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified $\mathcal{NP}$-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.
7. V. B. Le and B. Randerath. On stable cutsets in line graphs. *Theoretical Computer Science*, 301:463–475, 2003.
8. A. M. Moshi. Matching cutsets in graphs. *Journal of Graph Theory*, 13(5):527–536, 1989.
9. M. Patrignani and M. Pizzonia. The complexity of the matching-cut problem. In *Graph-theoretic concepts in computer science (Boltenhagen, 2001)*, volume 2204 of *Lecture Notes in Computer Science*, pages 284–295, 2001.

Tree Spanners for Bipartite Graphs and Probe Interval Graphs

Andreas Brandstädt[1], Feodor F. Dragan[2], Hoang-Oanh Le[1], Van Bang Le[1], and Ryuhei Uehara[3]

[1] Institut für Theoretische Informatik, Fachbereich Informatik, Universität Rostock, 18051 Rostock, Germany.
{ab,hoang-oanh.le,le}@informatik.uni-rostock.de
[2] Dept. of Computer Science, Kent State University, Ohio, USA.
dragan@cs.kent.edu
[3] Natural Science Faculty, Komazawa University, Tokyo, Japan.
uehara@komazawa-u.ac.jp

Abstract. A tree t-spanner T in a graph G is a spanning tree of G such that the distance between every pair of vertices in T is at most t times their distance in G. The tree t-spanner problem asks whether a graph admits a tree t-spanner, given t. We first substantially strengthen the known results for bipartite graphs. We prove that the tree t-spanner problem is NP-complete even for chordal bipartite graphs for $t \geq 5$, and every bipartite ATE–free graph has a tree 3-spanner, which can be found in linear time. The best known before results were NP-completeness for general bipartite graphs, and that every convex graph has a tree 3-spanner. We next focus on the tree t-spanner problem for probe interval graphs and related graph classes. The graph classes were introduced to deal with the physical mapping of DNA. From a graph theoretical point of view, the classes are natural generalizations of interval graphs. We show that these classes are tree 7-spanner admissible, and a tree 7-spanner can be constructed in $O(m \log n)$ time.

Keywords: Chordal bipartite graph, Interval bigraph, NP-completeness, Probe interval graph, Tree spanner

1 Introduction

A *tree t-spanner* T in a graph G is a spanning tree of G such that the distance between every pair of vertices in T is at most t times their distance in G. The *tree t-spanner problem* asks whether a graph admits a tree t-spanner, given t. The notion is introduced by Cai and Corneil [8,9], which finds numerous applications in distributed systems and communication networks; for example, it was shown that tree spanners can be used as models for broadcast operations [1] (see also [23]). Moreover, tree spanners were used in the area of biology [2], and approximating the bandwidth of graphs [27]. We refer to [24,26,6] for more background information on tree spanners.

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 106–118, 2003.

The tree t-spanner problem is NP-complete in general [9] for any $t \geq 4$. However, it can be solved efficiently for some particular graph classes. Especially, the complexity of the tree t-spanner problem is well investigated for chordal graphs and its subclasses. For $t \geq 4$ the problem is NP-complete for chordal graphs [6], strongly chordal graphs are tree 4-spanner admissible [3] (i.e., every strongly chordal graph has a tree 4-spanner), and the following graph classes are tree 3-spanner admissible: interval graphs [18], directed path graphs [17], split graphs [27] (see also [6]).

We first focus on the tree t-spanner problem for bipartite graphs and its subclasses. The class of bipartite graphs is wide and important from both practical and theoretical points of view. However, the known results for the complexity of the tree t-spanner problem for bipartite graph classes are few comparing to chordal graph classes. The NP-completeness is only known for general bipartite graphs (this result can be deduced from the construction in [9]), and the problem can be solved for regular bipartite graphs, and convex graphs as follows; a regular bipartite graph is tree 3-spanner admissible if and only if it is complete [18]; and any convex graph is tree 3-spanner admissible [27].

We substantially strengthen the known results for bipartite graph classes, and reduce the gap. We show that the tree t-spanner problem is NP-complete even for chordal bipartite graphs for $t \geq 5$. The class of chordal bipartite graphs is a bipartite analog of chordal graphs, introduced by Golumbic and Goss [13], and has applications to nonsymmetric matrices [12]. We also show that every bipartite asteroidal-triple-edge–free (ATE–free) graph has a tree 3-spanner, and such a tree spanner can be found in linear time. The class of ATE–free graphs was introduced by Müller [22] to characterize interval bigraphs. The class of interval bigraphs is a bipartite analog of interval graphs and was introduced by Harary, Kabell, and McMorris [14].

Our results reduce the gap between the upper and lower bounds of the complexity of the tree t-spanner problem for bipartite graph classes since the following proper inclusions are known [22,7]; convex graphs $\subset$ interval bigraphs $\subset$ bip. ATE–free graphs $\subset$ chordal bipartite graphs $\subset$ bipartite graphs.

We next focus on the tree t-spanner problem on probe interval graphs and related graph classes. The class of probe interval graphs was introduced by Zhang to deal with the physical mapping of DNA, which is a problem arising in the sequencing of DNA (see [28,21,20,29] for background). A probe interval graph is obtained from an interval graph by designating a subset P of vertices as *probes*, and removing the edges between pairs of vertices in the remaining set N of *nonprobes*. In the original papers [28,29], Zhang introduced two variations of probe interval graphs. An enhanced probe interval graph is the graph obtained from a probe interval graph by adding the edges joining two nonprobes if they are adjacent to two independent probes. The class of STS-probe interval graphs is a subset of the probe interval graphs; in those graphs all probes are independent.

From the graph theoretical point of view, it has been shown that all probe interval graphs are weakly chordal [21], and enhanced probe interval graphs are chordal [28,29]. In full version, we show that (1) the class of STS-probe interval

graphs is equivalent to the class of convex graphs (hence the class is tree 3-spanner admissible), and (2) the class of the (enhanced) probe interval graphs is incomparable with the classes of strongly chordal graphs and rooted directed path graphs.

Hence, from both viewpoints of graph theory and biology, the tree t-spanner problem for (enhanced) probe interval graphs is worth investigating. Especially, it is natural to ask that if those graph classes are tree t-spanner admissible for fixed integer t. We give the positive answer to that question: The classes of probe interval graphs and enhanced probe interval graphs are tree 7-spanner admissible. A tree 7-spanner of a (enhanced) probe interval graph can be constructed in $O(m + n \log n)$ time if it is given with an interval model. Therefore, using the recognition algorithms in [15,19], we can construct a tree 7-spanner for a given (enhanced) probe interval graph $G = (P, N, E)$ in $O(m \log n)$ time.

Due to space limitation, proofs are omitted, and can be found in full version[1].

2 Preliminaries

Given a graph $G = (V, E)$ and a subset $U \subseteq V$, the *subgraph of G induced by U* is the graph (U, F), where $F = \{\{u, v\} | \{u, v\} \in E \text{ for } u, v \in U\}$, and denoted by $G[U]$. For a subset F of E, we sometimes unify the edge set F and its *edge induced subgraph* (U, F) with $U = \{v | \{u, v\} \in F \text{ for some } u \in V\}$. For two vertices u and v on G, the *distance* of the vertices is the minimum length of the paths joining u and v, and denoted by $d_G(u, v)$. The *disk* of radius k centered at v is the set of all vertices with distance at most k to v, $D_k(v) = \{w \in V : d_G(v, w) \leq k\}$, and the *$k$th neighborhood* $N_k(v)$ of v is defined as the set of all vertices at distance k to v, that is $N_k(v) = \{w \in V : d_G(v, w) = k\}$. By $N(v)$ we denote the *neighborhood* of v, i.e., $N(v) := N_1(v)$. More generally, for a subset $S \subseteq V$ let $N(S) = \cup_{v \in S} N(v)$ denote the *neighborhood* of S.

A *tree t-spanner* T in a graph G is a spanning tree of G such that for each pair u and v in G, $d_T(u, v) \leq t \cdot d_G(u, v)$. We say that G is *tree t-spanner admissible* if it contains a tree t-spanner. The *tree t-spanner problem* is to determine, for given graph and positive integer t, if the graph admits a tree t-spanner. A class C of graphs is *tree t-spanner admissible* if every graph in C is tree t-spanner admissible. On the tree t-spanner problem, the following result plays an important role:

Lemma 1. [9] *A spanning tree T of G is a tree t-spanner if and only if for every edge $\{u, v\}$ of G, $d_T(u, v) \leq t$.*

It is well known that a graph G is bipartite if and only if G contains no cycle of odd length [16]. Thus, for each positive integer k, a tree $2k$-spanner of a bipartite graph G is also a tree $(2k - 1)$-spanner. Hence we will consider a tree t-spanner for each odd number t for bipartite graphs.

A graph (V, E) with $V = \{v_1, v_2, \cdots, v_n\}$ is an *interval graph* if there is a set of intervals $\mathcal{I} = \{I_1, I_2, \cdots, I_n\}$ such that $\{v_i, v_j\} \in E$ if and only if $I_i \cap I_j \neq \emptyset$

[1] Full version is available at `http://www.komazawa-u.ac.jp/~uehara/ps/t-span.pdf`

for each $1 \leq i, j \leq n$. We call the set $\mathcal{I}$ *interval representation* of the graph. For each interval I, we denote by $R(I)$ and $L(I)$ the right and left endpoints of the interval, respectively. A bipartite graph (X, Y, E) with $X = \{x_1, x_2, \cdots, x_{n_1}\}$ and $Y = \{y_1, y_2, \cdots, y_{n_2}\}$ is an *interval bigraph* if there are families of intervals $\mathcal{I}_X = \{I_1, I_2, \cdots, I_{n_1}\}$ and $\mathcal{I}_Y = \{J_1, J_2, \cdots, J_{n_2}\}$ such that $\{x_i, y_j\} \in E$ if and only if $I_i \cap J_j \neq \emptyset$ for each $1 \leq i \leq n_1$ and $1 \leq j \leq n_2$. We also call the families of intervals $(\mathcal{I}_X, \mathcal{I}_Y)$ *interval representation* of the graph. We sometimes unify a vertex v_i and its corresponding interval I_i; I_v denotes the interval corresponding to the vertex v, and $R(v)$ and $L(v)$ denote $R(I_v)$ and $L(I_v)$, respectively.

An edge which joins two vertices of a cycle but is not itself an edge of the cycle is a *chord* of that cycle. A graph is *chordal* if each cycle of length ≥ 4 has a chord. A graph G is *weakly chordal* if G and $\bar{G}$ contain no induced cycle C_k with $k \geq 5$. A bipartite graph G is *chordal bipartite* if each cycle of length ≥ 6 has a chord. Let the neighborhood $N(e)$ of an edge $e = \{v, w\}$ be the union $N(v) \cup N(w)$ of the neighborhoods of the end-vertices of e. Three edges of a graph G form an *asteroidal triple of edges* (ATE) if for any two of them there is a path from the vertex set from one to the vertex set of the other that avoids the neighborhood of the third edge. *ATE–free* graphs are those graphs which do not contain any ATE. This class of graphs was introduced in [22], where it was also shown that any interval bigraph is an ATE–free graph, and any bipartite ATE–free graph is chordal bipartite.

A graph $G = (V, E)$ is a *probe interval graph* if V can be partitioned into subsets P and N (corresponding to the *probes* and *nonprobes*) and each $v \in V$ can be assigned to an interval I_v such that $\{u, v\} \in E$ if and only if both $I_u \cap I_v \neq \emptyset$ and at least one of u and v is in P. In this paper, we assume that P and N are given, and we denote the considered probe interval graph by $G = (P, N, E)$. Let $G = (P, N, E)$ be a probe interval graph. Let E^+ be a set of edges $\{u_1, u_2\}$ with $u_1, u_2 \in N$ such that there are two probes v_1 and v_2 in P such that $\{v_1, u_1\}$, $\{v_1, u_2\}$, $\{v_2, u_1\}$, $\{v_2, u_2\} \in E$, and $\{v_1, v_2\} \notin E$. Each edge in E^+ is called an *enhanced edge*, and the resulting graph $G^+ = (P, N, E \cup E^+)$ is said to be an *enhanced probe interval graph*. See [28,21,20,29] for further details.

3 NP-Completeness for Chordal Bipartite Graphs

In this section we show that, for any $t \geq 5$, the tree t-spanner problem is NP-complete for chordal bipartite graphs. The proof is a reduction from Monotone 3SAT which consists of instances of 3SAT such that each clause contains either only negated variables or only non-negated variables (see [11, LO2]). For which the following family of chordal bipartite graphs will play an important role.

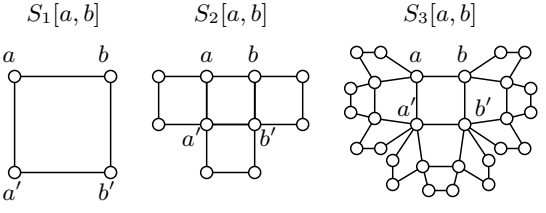


Fig. 1. The graph $S_\ell[a, b]$

First, $S_0[a,b]$ is an edge $\{a,b\}$, and $S_1[a,b]$ is the 4-cycle (a,a',b',b,a). Next, for a fixed integer $\ell > 1$, $S_{\ell+1}[a,b]$ is obtained from one cycle (a,b,b',a',a), $S_\ell[a,a']$, $S_\ell[b,b']$, and $S_\ell[a',b']$ by identifying the corresponding vertices (see Fig. 1). We will connect the vertices a and b to other graphs, and use $S_\ell[a,b]$ as a subgraph of bigger graphs. Sometimes, when the context is clear, we simply write S_ℓ for $S_\ell[a,b]$. In case $\ell > 0$ we write (a,a',b',b,a) for the 4-cycle in $S_\ell[a,b]$ containing the edge $\{a,b\}$. Each of the edges $\{a,a'\},\{a',b'\},\{b,b'\}$ belongs to a unique $S_{\ell-1}$, the *corresponding* $S_{\ell-1}$ in $S_\ell[a,b]$ to $\{a,a'\},\{a',b'\},\{b,b'\}$, respectively. The following observations collect basic facts on S_ℓ used in the reduction later.

Observation 1. *For every integer $\ell \geq 0$, $S_\ell[a,b]$ has a tree $(2\ell+1)$-spanner containing the edge $\{a,b\}$.*

Observation 2. *Let H be an arbitrary graph and let e be an arbitrary edge of H. Let K be an $S_\ell[a,b]$ disjoint from H. Let G be the graph obtained from H and K by identifying the edges e and $\{a,b\}$; see Fig. 2. Suppose that T is a tree t-spanner in G, $t > 2\ell$, such that the (a,b)-path in T belongs to H. Then $d_T(a,b) \leq t - 2\ell$.*

Observation 2 indicates a way to force an edge $\{x,y\}$ to be a tree edge: Choosing $\ell = \lfloor \frac{t-1}{2} \rfloor$ shows that $\{a,b\}$ must be an edge of T.

We now describe the reduction. Let $k \geq 2$ be an integer, and let F be a 3SAT formula with m clauses C_j for $1 \leq j \leq m$, over n variables x_i for $1 \leq i \leq n$.

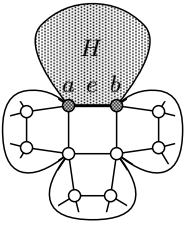


Fig. 2. The graph obtained from H and $S_\ell[a,b]$ by identifying the edge $e = \{a,b\}$

Definition 1. *In a graph G, an edge $\{a,b\}$ is said to be* forced by an S_ℓ *if $\{a,b\}$ appears in some $S_\ell[a,b]$ (as induced subgraph in G) such that $\{a,b\}$ disconnects $S_\ell[a,b]$ from the rest. We require that each two $S_\ell[a,b]$ and $S_{\ell'}[c,d]$ have at most 2 vertices in $\{a,b,c,d\}$ in common. An edge $\{a,b\}$ is said to be* strongly forced *if it is forced by two $S_k[a,b]$.*

By Observation 2, if G has a tree $(2k+1)$-spanner T every strongly forced edge must belong to T.

For each variable x_i create the gadget $G(x_i)$ as follows: (1) Take $2m+4$ vertices $x_i^1,\ldots,x_i^m$, $\overline{x_i}^1,\ldots,\overline{x_i}^m$, p_i,q_i,r_i,s_i, and (2) for $1 \leq j,j' \leq m$, add the edges $\{x_i^j,\overline{x_i}^{j'}\}$, $\{q_i,x_i^j\}$, $\{r_i,x_i^j\}$, $\{p_i,\overline{x_i}^j\}$, $\{s_i,\overline{x_i}^j\}$, and $\{p_i,r_i\},\{r_i,s_i\},\{s_i,q_i\}$. Furthermore, (3) each of the edges $\{p_i,r_i\},\{r_i,s_i\},\{s_i,q_i\}$, and $\{x_i^j,\overline{x_i}^j\}$ with $1 \leq j \leq m$, is a strongly forced edge, (4) force each edge $\{a,b\} \in \{\{q_i,x_i^j\} : 1 \leq j \leq m\} \cup \{\{r_i,x_i^j\} : 1 \leq j \leq m\} \cup \{\{p_i,\overline{x_i}^j\} : 1 \leq j \leq m\} \cup \{\{s_i,\overline{x_i}^j\} : 1 \leq j \leq m\} \cup \{\{x_i^j,\overline{x_i}^{j'}\} : 1 \leq j,j' \leq m, j \neq j'\}$ by an $S_{k-1}[a,b]$. (See Fig.3; in the figure, the S_k and S_{k-1} are omitted, and thick edges are strongly forced).

The vertex x_i^j ($\overline{x_i}^j$, respectively) will be connected to the clause gadget of clause C_j if x_i ($\overline{x_i}$, respectively) is a literal in C_j. All edges $\{r_i,x_i^j\}$ $(1 \leq j \leq m)$

or else all edges $\{s_i, \overline{x_i}^j\}$ $(1 \leq j \leq m)$ will belong to any tree $(2k+1)$-spanner (if any) of the graph G which we are going to describe.

Definition 2. *A clause is* positive *(*negative*, respectively) if it contains only variables (negation of variables).*

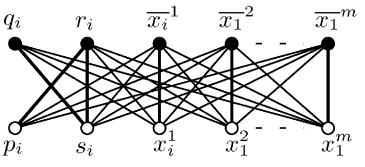


Fig. 3. $G(x_i)$

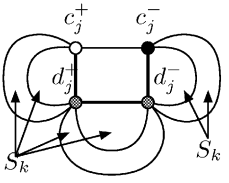


Fig. 4. $G(C_j)$

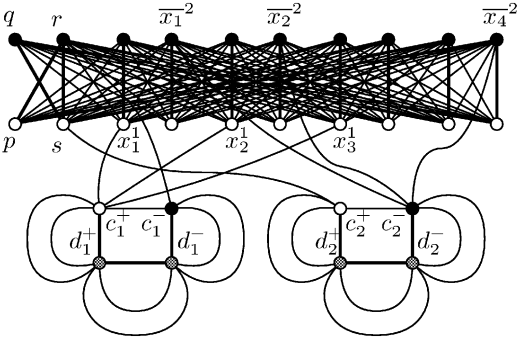


Fig. 5. The reduction given $C_1 = (x_1, x_2, x_3)$ and $C_2 = (\overline{x_1}, \overline{x_2}, \overline{x_4})$

We note that each clause is either positive or negative since given formula is an instance of Monotone 3SAT. For each clause C_j, $G(C_j)$ is the 4-cycle $(c_j^+, d_j^+, d_j^-, c_j^-, c_j^+)$ where $\{c_j^+, d_j^+\}$, $\{d_j^+, d_j^-\}$, and $\{d_j^-, c_j^-\}$ are strongly forced edges (see Fig.4).

Finally, the graph $G = G(F)$ is obtained from all $G(v_i)$ and $G(C_j)$ by identifying all vertices p_i, q_i, r_i and s_i to a single vertex p, q, r, and s, respectively (thus, $\{p, r\}$, $\{r, s\}$ and $\{s, q\}$ are edges in G), and adding the following edges: (1) Connect every x_i^j with every $\overline{x_{i'}}^{j'}$ $(i \neq i')$. (2) For every positive clause C_j: If x_i is in C_j then connect x_i^j with c_j^+ and force the edge $\{x_i^j, c_j^+\}$ by an $S_{k-2}[x_i^j, c_j^+]$. Connect c_j^- with r and force the edge $\{c_j^-, r\}$ by an $S_{k-2}[c_j^-, r]$. (3) For every negative clause C_j: If $\overline{x_i}$ is in C_j then connect $\overline{x_i}^j$ with c_j^- and force the edge $\{\overline{x_i}^j, c_j^-\}$ by an $S_{k-2}[\overline{x_i}^j, c_j^-]$. Connect c_j^+ with s and force the edge $\{c_j^+, s\}$ by an $S_{k-2}[c_j^+, s]$.

The description of the graph $G = G(F)$ is complete. Clearly, G can be constructed in polynomial time. See Fig. 5 for an example.

Lemma 2. *G is chordal bipartite.*

Lemma 3. *Suppose G admits a tree $(2k+1)$-spanner. Then F is satisfiable.*

Lemma 4. *Suppose F is satisfiable. Then G admits a tree $(2k+1)$-spanner.*

Theorem 3. *For every fixed $k \geq 2$, the Tree $(2k+1)$-Spanner problem is* NP*-complete for chordal bipartite graphs.*

4 Tree 3-Spanners for Bipartite ATE-Free Graphs

In this section we show that any bipartite ATE–free graph admits a tree 3-spanner.

We say that a vertex u of a graph G has a *maximum neighbor* if there is a vertex w in G such that $N(N(u)) = N(w)$. We will need the following result from [5].

Lemma 5. [5] *Any chordal bipartite graph G has a vertex with a maximum neighbor.*

It is easy to deduce from results [4, Lemma 4.4], [5, Corollary 5] and [10, Corollary 1] that a vertex with a maximum neighbor of a chordal bipartite graph can be found in linear time by the following procedure.

PROCEDURE NICE-VERTEX. Find a vertex with a maximum neighbor
Input: A chordal bipartite graph $G = (X \cup Y, E)$.
Output: A vertex with a maximum neighbor.
Method:
initially all vertices $v \in X \cup Y$ are unmarked;
repeat
among unmarked vertices of X select a vertex x such that $N(x)$ contains the maximum number of marked vertices;
mark x and all its unmarked neighbors;
until all vertices in Y are marked;
output the vertex of Y marked last.

Now let $G = (V, E)$ be a connected bipartite ATE–free graph and u be a vertex of G which has a maximum neighbor (recall that G is chordal bipartite and therefore such a vertex u exists).

Lemma 6. *Let S be a connected component of a subgraph of G induced by set $V \setminus D_{k-1}(u)$ ($k \geq 1$). Then, there is a vertex $w \in N_{k-1}(u)$ such that $N(w) \supset S \cap N_k(u)$.*

Lemma 6 suggests the following algorithm for constructing a spanning tree of G.

PROCEDURE SPAN-ATEG. Tree 3-spanners for bip. ATE–free graphs
Input: A bipartite ATE–free graph $G = (V, E)$ and a vertex u of G with a maximum neighbor.
Output: A spanning tree $T = (V, E')$ of G (rooted at u).
Method:
set $E' := \emptyset$;
set $q := \max\{d_G(u, v) : v \in V\}$;
let s_i^q, $i \in \{1, \ldots, p_q\}$ be the vertices of $N_q(u)$;
for every $i \in \{1, \ldots, p_q\}$ **do**
pick a neighbor w of s_i^q in $N_{q-1}(u)$;
add edge $\{s_i^q, w\}$ to E';
for $k := q - 1$ **downto** 1 **do**

compute the connected components $S_1^k, \ldots, S_{p_k}^k$ of
$G[N_k(u) \cup \{s_i^{k+1}, i \in \{1, \ldots, p_{k+1}\}\}]$;
for every $i \in \{1, \ldots, p_k\}$ **do**
set $S := S_i^k \cap N_k(u)$;
pick a vertex w in $N_{k-1}(u)$ such that $N(w) \supset S$;
for each $v \in S$ add the edge $\{v, w\}$ to E';
shrink component S_i^k to a vertex s_i^k and make s_i^k adjacent in G to
all vertices from $N(S_i^k) \cap N_{k-1}(u)$.

It is easy to see that the graph $T = (V, E')$ constructed by this procedure is a spanning tree of G and its construction takes only linear time. Moreover, T is a shortest path tree of G rooted at u since for any vertex $x \in V$, $d_G(x, u) = d_T(x, u)$ holds.

Theorem 4. *Let $T = (V, E')$ be a spanning tree of a bipartite ATE–free graph $G = (V, E)$ output by PROCEDURE SPAN-ATEG. Then, for any $x, y \in V$, we have $d_T(x, y) \leq 3 \cdot d_G(x, y)$ and $d_T(x, y) \leq d_G(x, y) + 2$.*

Since any interval bigraph is a bipartite ATE–free graph, and any convex graph is an interval bigraph, we have the following corollaries.

Corollary 1. *Any interval bigraph $G = (V, E)$ admits a spanning tree T such that $d_T(x, y) \leq 3 \cdot d_G(x, y)$ and $d_T(x, y) \leq d_G(x, y) + 2$ hold for any $x, y \in V$. Moreover, such a tree T can be constructed in linear time.*

Corollary 2. [27] *Any convex graph $G = (V, E)$ admits a spanning tree T such that $d_T(x, y) \leq 3 \cdot d_G(x, y)$ and $d_T(x, y) \leq d_G(x, y) + 2$ hold for any $x, y \in V$. Moreover, such a tree T can be constructed in linear time.*

5 Tree 7-Spanners for (Enhanced) Probe Interval Graphs

In this section we show that any (enhanced) probe interval graph admits a tree 7-spanner.

Let $G = (P, N, E)$ be a connected probe interval graph. We assume that an interval representation of G is given (if not, an interval model for G can be constructed by a method described in [19] in $O(m \log n)$ time, where $n = |P| + |N|$ and $m = |E|$). Let $\mathcal{I} = \{I_x : x \in P\}$ be the intervals in the interval model representing the probes and $\mathcal{J} = \{J_y : y \in N\}$ be the intervals representing the nonprobes.

First we discuss two simple special cases. If $N = \emptyset$ then clearly $G = (P, E)$ is an interval graph. It is known (see [25]) that for any interval graph G and any vertex u of G there is a shortest path spanning tree T of G rooted at u such that $d_T(x, y) \leq d_G(x, y) + 2$ holds for any x, y. In fact, a procedure similar to PROCEDURE SPAN-ATEG produces such a spanner in linear time for any interval graph G and any start vertex u. Evidently, T is a tree 3-spanner of G.

To describe other special case, we will need the following notion. A connected probe interval graph $G = (P, N, E)$ is *superconnected* if for any two intersecting

intervals $I_v, I_w \in \mathcal{I}$ there is always an interval $J_y \in \mathcal{J}$ such that $I_v \cap I_w \cap J_y \neq \emptyset$. For a superconnected probe interval graph G, a tree 4-spanner can be constructed easily. First we ignore all edges in $G[P]$ to get an interval bigraph $G' = (X = P, Y = N, E')$ and then run PROCEDURE SPAN-ATEG on G'. We claim that a spanning tree T of G', produced by that procedure, is a tree 4-spanner of G. Indeed, for any edge $\{x, y\}$ of G such that $x \in P$ and $y \in N$, $d_T(x, y) \leq 3$ holds by Corollary 1; it is an edge of G', too. Now consider an edge $\{v, w\}$ of G with $v, w \in P$. Since G is superconnected, there is a vertex $y \in N$ such that $I_v \cap I_w \cap J_y \neq \emptyset$, i.e., $d_{G'}(v, w) = 2$. Then, by Corollary 1, we have $d_T(v, w) \leq d_{G'}(v, w) + 2 = 2 + 2 = 4$. Consequently, T is a tree 4-spanner of G.

To get a tree 7-spanner for an arbitrary connected probe interval graph $G = (P, N, E)$, we will use the following strategy. First we decompose the graph G into subgraphs $G_0, G_1, \ldots, G_k$ such that G_i and G_j $(i \neq j)$ share at most one common vertex and each G_i is either an interval graph or a superconnected probe interval graph. Then iteratively, given a tree 7-spanner T^i for $G_0 \cup G_1 \cup \ldots \cup G_i$ $(i < k)$ and a tree t-spanner T_{i+1} $(t \leq 4)$ of G_{i+1}, we will extend T^i to a tree 7-spanner T^{i+1} for $G_0 \cup G_1 \cup \ldots \cup G_i \cup G_{i+1}$ by either making all vertices of G_{i+1} adjacent in T^{i+1} to a common neighbor in $G_0 \cup G_1 \cup \ldots \cup G_i$ (if it exists) or by gluing trees T^i and T_{i+1} at a common vertex.

Now we give a formal description of the decomposition algorithm. Let $S_0, S_1, \ldots, S_q$ be segments of the union $\cup_{y \in N} J_y$. (see Fig. 3 for an illustration).

Clearly, all probe interval graphs G_{2i+1} $(i = 1, \ldots, q)$ are superconnected and a decomposition of G into G_0,G_1,..., G_{2q+2} can be done in linear time if endpoints of the intervals $\mathcal{I} \cup \mathcal{J}$ are sorted.

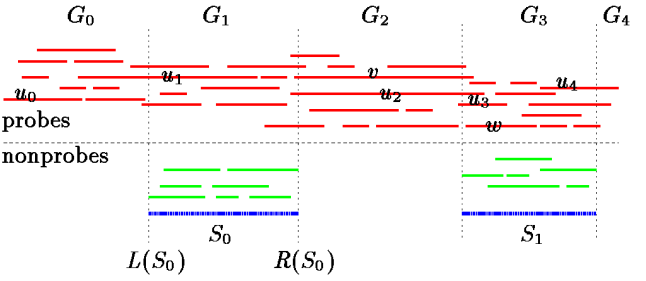


Fig. 3. Segments and a decomposition of a probe interval graph

PROCEDURE DECOMP. A decomposition of a probe interval graph

Input: A probe interval graph G and its interval representation $(\mathcal{I}, \mathcal{J})$.

Output: Subgraphs $G_0, G_1, \ldots, G_{2q+2}$ of G, where G_{2i} $(i \in \{0, \ldots, q+1\})$ is an interval graph and G_{2i+1} $(i \in \{0, \ldots, q\})$ is a superconnected probe interval graph, and special vertices u_j $(j = 1, \ldots, 2q+2)$, where u_j belongs to G_{j-1} and G_j.

Method:

for $i = 0$ **to** q **do**
/* define an interval graph */
set $\mathcal{X} := \{I_x \in \mathcal{I} : L(x) \leq L(S_i)\}$;
on intervals $\mathcal{X}$ define an interval graph G_{2i};
let I^* be an interval from $\mathcal{X}$ with maximum $R(\cdot)$ value;
set u_{2i+1} := a vertex of G corresponding to I^*;
set $\mathcal{I} := \mathcal{I} \setminus (\mathcal{X} \setminus \{I^*\})$;

/* define a superconnected probe interval graph */
set $\mathcal{Y} := \{I_y \in \mathcal{J} : I_y \subseteq S_i\}$;
set $\mathcal{X} := \{I_x \in \mathcal{I} : L(x) \leq R(S_i)\}$;
define a probe interval graph G_{2i+1} with probes $\mathcal{X}$ and nonprobes $\mathcal{Y}$;
let I^* be an interval from $\mathcal{X}$ with maximum $R(\cdot)$ value;
set u_{2i+2} := a vertex of G corresponding to I^*;
set $\mathcal{I} := \mathcal{I} \setminus (\mathcal{X} \setminus \{I^*\})$;
define on $\mathcal{I}$ an interval graph G_{2q+2}.

Lemma 7. *For any $i = 2, \ldots, 2q+2$, $R(u_i) \geq R(u_{i-1})$ holds.*

Now, for an interval graph G_0 (if it is not empty), we can construct a tree 3-spanner $T_0 = T_0(u_0)$ rooted at any vertex u_0 of G_0. For an interval graph G_{2i} $(i = 1, \ldots, q+1)$, we can construct a tree 3-spanner $T_{2i} = T_{2i}(u_{2i})$ rooted at vertex u_{2i} (see PROCEDURE DECOMP). Since all those trees are shortest path trees, the neighborhoods of vertex u_{2i} in G_{2i} and T_{2i} coincide.

Let G^-_{2i+1} be an interval bigraph obtained from a superconnected probe interval graph G_{2i+1} by ignoring all edges between probes and deleting all probes I_v such that $I_v \subset I_{u_{2i+1}}$.

Lemma 8. *For any $i = 0, \ldots, q$, vertex u_{2i+1} has a maximum neighbor in G^-_{2i+1}.*

Let $T^-_{2i+1} = T^-_{2i+1}(u_{2i+1})$ be a tree 3-spanner of an interval bigraph G^-_{2i+1} constructed starting at vertex u_{2i+1}, $i \in \{0, \ldots, q\}$ (see PROCEDURE SPANATEG). Clearly, the neighborhoods of vertex u_{2i+1} in G^-_{2i+1} and T^-_{2i+1} coincide. We can extend tree T^-_{2i+1} to a spanning tree $T_{2i+1} = T_{2i+1}(u_{2i+1})$ of G_{2i+1} by adding, for each probe I_v of G_{2i+1} such that $I_v \subset I_{u_{2i+1}}$, a pendant vertex v adjacent to u_{2i+1}.

Lemma 9. *$T_{2i+1}(u_{2i+1})$ is a tree 4-spanner for G_{2i+1}, $i \in \{0, \ldots, q\}$. Moreover, for any edge $\{w, u_{2i+1}\}$ of G_{2i+1}, $d_{T_{2i+1}}(w, u_{2i+1}) \leq 2$ holds.*

Now we are ready to construct a spanning tree T for the original probe interval graph $G = (P, N, E)$. We say that a vertex v of G *dominates* a subgraph G_k of G if every vertex of G_k, different from v, is adjacent to v in G.

PROCEDURE SPAN-PIG. Tree 7-spanner for probe interval graphs
Input: A probe interval graph $G = (P, N, E)$, its interval representation $(\mathcal{I}, \mathcal{J})$ and a decomposition of G into graphs $G_0, G_1, \ldots, G_{2q+2}$.
Output: A spanning tree $T = (P \cup N, E')$ of G.
Method:
set $E' = \emptyset$ and $k := 0$;
while $k \leq 2q+2$ **do**
  if there is an index j such that $k \leq j$ and u_k dominates G_j **then do**
    find the largest index j with that property;
    for each v in $G_k \cup \ldots \cup G_j$ $(v \neq u_k)$ add edge $\{v, u_k\}$ to E';
    set $k := j + 1$;
  else do

if k is even **then do**
find a tree 3-spanner $T_k(u_k)$ of an interval graph G_k;
add all edges of $T_k(u_k)$ to E';
if k is odd **then do**
find a tree 4-spanner $T_k(u_k)$ of a superconnected probe interval graph G_k;
add all edges of $T_k(u_k)$ to E';
set $k := k + 1$.

It is easy to see that the tree T constructed by PROCEDURE SPAN-PIG is a spanning tree of G and its construction takes only linear time.

Lemma 10. *If for graph G_k ($k \in \{0, \ldots, 2q+2\}$) there exists a vertex $u_i \in \{u_0, \ldots, u_k\}$ which dominates G_k, then there is a vertex $u_s \in \{u_0, \ldots, u_k\}$ such that $d_T(x, u_s) \leq 1$ holds for any x in G_k. Otherwise, if such vertex u_i does not exist, then for any vertices x, y of G_k, $d_T(x, y) = d_{T_k}(x, y)$ holds.*

Corollary 3. *For any vertices x, y of G_k ($k \in \{0, \ldots, 2q+2\}$), $d_T(x, y) \leq \max\{2, d_{T_k}(x, y)\}$ holds.*

Lemma 11. *T is a tree 7-spanner for G.*

Theorem 5. *Any probe interval graph G admits a tree 7-spanner. Moreover, such a tree 7-spanner can be constructed in $O(m \log n)$ time, or in $O(m+n \log n)$ time if the intersection model of G is given in advance.*

Now let $G = (P, N, E)$ be an enhanced probe interval graph with probes P and nonprobes N.

Corollary 4. *Any enhanced probe interval graph $G = (P, N, E)$ admits a tree 7-spanner. Moreover, such a tree spanner can be constructed in $O(m \log n)$ time.*

6 Concluding Remarks

In the paper, we have shown that the tree t-spanner problem is NP-complete even for chordal bipartite graphs for $t \geq 5$. The complexity of the tree 3-spanner problem is still open. We have also shown that every (enhanced) probe interval graph has a tree 7-spanner. However, it is also open whether the graph classes are tree t-spanner admissible for smaller t.

Acknowledgements. The authors are greatful to an anonymous referee for simplifying the reduction in Section 3.

References

1. B. Awerbuch, A. Baratz, and D. Peleg. Efficient broadcast and light-weighted spanners. manuscript, 1992.
2. H.-J. Bandelt and A. Dress. Reconstructing the Shape of a Tree from Observed Dissimilarity Data. *Advances in Applied Mathematics*, 7:309–343, 1986.
3. A. Brandstädt, V. Chepoi, and F. Dragan. Distance Approximating Trees for Chordal and Dually Chordal Graphs. *J. of Algorithms*, 30(1):166–184, 1999.
4. A. Brandstädt, V.D. Chepoi, and F.F. Dragan. The Algorithmic Use of Hypertree Structure and Maximum Neighbourhood Orderings. *Disc. Appl. Math.*, 82:43–77, 1998.
5. A. Brandstädt, F. Dragan, V. Chepoi, and V. Voloshin. Dually Chordal Graphs. *SIAM J. Disc. Math.*, 11(3):437–455, 1998.
6. A. Brandstädt, F.F. Dragan, H.-O. Le, and V.B. Le. Tree Spanners on Chordal Graphs: Complexity, Algorithms, Open Problems. In *ISAAC 2002*, pages 163–174. LNCS Vol. 2518, Springer-Verlag, 2002.
7. A. Brandstädt, V.B. Le, and J.P. Spinrad. *Graph Classes: A Survey*. SIAM, 1999.
8. L. Cai and D.G. Corneil. Tree Spanners: an Overview. *Congressus Numerantium*, 88:65–76, 1992.
9. L. Cai and D.G. Corneil. Tree Spanners. *SIAM J. Disc. Math.*, 8(3):359–387, 1995.
10. F.F. Dragan and V.I. Voloshin. Incidence Graphs of Biacyclic Hypergraphs. *Disc. Appl. Math.*, 68:259–266, 1996.
11. M.R. Garey and D.S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. Freeman, 1979.
12. M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.
13. M.C. Golumbic and C.F. Goss. Perfect Elimination and Chordal Bipartite Graphs. *J. of Graph Theory*, 2:155–163, 1978.
14. F. Harary, J.A. Kabell, and F.R. McMorris. Bipartite intersection graphs. *Comment. Math. Univ. Carolin.*, 23:739–745, 1982.
15. J.L. Johnson and J.P. Spinrad. A Polynomial Time Recognition Algorithm for Probe Interval Graphs. In *Proc. 12th SODA*, pages 477–486. ACM, 2001.
16. D. König. *Theorie der endlichen und unendlichen Graphen (in German)*. Akademische Verlagsgesellschaft, 1936.
17. H.-O. Le and V.B Le. Optimal Tree 3-Spanners in Directed Path Graphs. *Networks*, 34:81–87, 1999.
18. M.S. Madanlal, G. Venkatesan, and C. P. Rangan. Tree 3-Spanners on Interval, Permutation and Regular Bipartite Graphs. *IPL*, 59:97–102, 1996.
19. R.M. McConnell and J.P. Spinrad. Construction of Probe Interval Models. In *Proc. 13th SODA*, pages 866–875. ACM, 2002.
20. T.A. McKee and F.R. McMorris. *Topics in Intersection Graph Theory*. SIAM, 1999.
21. F.R. McMorris, C. Wang, and P. Zhang. On Probe Interval Graphs. *Disc. Appl. Math.*, 88:315–324, 1998.
22. H. Müller. Recognizing Interval Digraphs and Interval Bigraphs in Polynomial Time. *Disc. Appl. Math.*, 78:189–205, 1997. Erratum is available at `http://www.comp.leeds.ac.uk/hm/pub/node1.html`.
23. D. Peleg. *Distributed Computing: A Locally-Sensitive Approach*. Monographs on Discrete Mathematics and Applications. SIAM, 2000.

24. D. Peleg and A.A. Schäffer. Graph Spanners. *J. of Graph Theory*, 13(1):99–116, 1989.
25. E. Prisner. Distance approximating spanning trees. In *Proc. of STACS'97*, pages 499–510. LNCS Vol. 1200, Springer-Verlag, 1997.
26. J. Soares. Graph Spanners: a Survey. *Congress Numerantium*, 89:225–238, 1992.
27. G. Venkatesan, U. Rotics, M.S. Madanlal, J.A. Makowsy, and C.P. Rangan. Restrictions of Minimum Spanner Problems. *Inf. and Comp.*, 136:143–164, 1997.
28. P. Zhang. Probe Interval Graphs and Its Applications to Physical Mapping of DNA. manuscript, 1994.
29. P. Zhang. United States Patent. Method of Mapping DNA Fragments. [Online] Available `http://www.cc.columbia.edu/cu/cie/techlists/patents/5667970.htm`, July 3 2000.

A Simple Linear Time LexBFS Cograph Recognition Algorithm

Anna Bretscher[1], Derek Corneil[1], Michel Habib[2], and Christophe Paul[2]

[1] Dept. of Computer Science, University of Toronto
[2] LIRMM, Université Montpellier II

Abstract. This paper introduces a new simple linear time algorithm to recognize cographs (graphs without an induced P_4). Unlike other cograph recognition algorithms, the new algorithm uses a multisweep Lexicographic Breadth First Search (LexBFS) approach, and introduces a new variant of LexBFS, called LexBFS$^-$, operating on the complement of the given graph G and breaking ties with respect to an initial LexBFS. The algorithm either produces the cotree of G or identifies an induced P_4).

1 Introduction

Cographs or *complement reducible* graphs have the property of being defined both recursively and by a simple forbidden induced subgraph characterization. Cographs are the family of graphs constructed from a single vertex under the closure of the operations of union and complementation or equivalently, union and join (the join of $G_1, G_2, \ldots, G_k$ is the graph $G_1 \cup G_2 \cup \ldots \cup G_k$ together with all edges between vertices of different G_i). As shown by Lerchs (see [4]) these operations uniquely define a tree representation of the cograph referred to as a *cotree*. See Fig. 1 for a cograph $G = (V, E)$ and the corresponding cotree T_G. Leaves represent the vertex set V and each internal node represents the union (0) or join (1) operations on the children. The significance of the 0(1) nodes is captured by the fact that $xy \in E$ iff the smallest subtree containing x and y is rooted at a 1 node. Note that the labels on any path from a leaf to the root of the cotree alternate. Alternatively, Lerchs (see [4]) also proved that cographs are exactly those graphs that do not contain an induced P_4 (i.e. a path on 4 vertices).

The cotree is algorithmically significant as a tool for reducing NP-hard problems such as colouring, clique detection, Hamiltonicity, etc., on arbitrary graphs to fast linear time problems on cographs [4]. Cographs arise in applications such as examination scheduling problems [14] and automatic clustering of index terms. Surprisingly, despite the structural simplicity of cographs, constructing linear time recognition algorithms has been challenging. The first such algorithm to recognize cographs and construct the cotree was achieved by Corneil, Perl and Stewart [5] in 1985. The algorithm examines each vertex in turn, inserting it into the cotree if the graph is a cograph and returns "false" otherwise. Recently,

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 119–130, 2003.

Habib and Paul [9] have used vertex splitting to produce a new linear time recognition algorithm for cographs. Note that any linear time modular decomposition algorithm, such as [12,8,6], immediately yields a somewhat complicated, linear time cograph recognition algorithm.

One of the most elegant recognition algorithms for a restricted family of graphs was developed in 1976 by Rose, Tarjan and Lueker [13] for chordal graphs. This algorithm introduced a restricted form of Breadth First Search called Lexicographic Breadth First Search (LexBFS). They showed that a graph is chordal iff the ordering of any LexBFS is a perfect elimination ordering. Surprisingly, LexBFS has only recently resurfaced appearing in recognition algorithms for interval graphs [3,10], unit interval graphs [2] and bipartite permutation graphs [1] as well as structural related algorithms on a host of other graph families. Given the success of LexBFS on families of graphs related to cographs, it is natural to wonder whether there exists a simple, linear time, LexBFS based recognition algorithm for cographs.

The remainder of the paper reveals such an algorithm. We will define a variant of LexBFS denoted LexBFS$^-$ and a property of cographs called the *Neighbourhood Subset Property* which form the basis of the algorithm. In particular, Algorithm Recognize_Cograph determines whether the input graph G is a cograph using the Neighbourhood Subset Property, returning the cotree if G is a cograph and an induced P_4 if G is not a cograph.

Algorithm Recognize_Cograph(G)
Input: Graph G.
Output: Cotree T if G is a cograph, an induced P_4 otherwise.
  Compute LexBFS(G) resulting in τ
  Compute LexBFS$^-(\overline{G}, \tau)$ resulting in $\overline{\sigma}_1$
  Compute LexBFS$^-(G, \overline{\sigma}_1)$ resulting in σ_2
  if $\overline{\sigma}_1, \sigma_2$ satisfy the *Neighbourhood Subset Property*
      return (Construct_Cotree(V))
  else
      return (Construct_P_4)

The general graph theory notation of this paper follows that of West [15]. The remainder of this section provides an overview of LexBFS and cotree related terminology. The following section provides the intuition of the new algorithm and introduces LexBFS$^-$, a new variant of LexBFS. The paper concludes with some details behind the algorithm and its correctness.

1.1 LexBFS

We define a LexBFS as in Korte and Möhring [11]. We say that vertices x and y *disagree* on z if one of x, y is adjacent to z. Given a graph $G = (V, E)$, LexBFS produces an ordering $\sigma : (v_1, v_2, \ldots, v_n)$ of V such that if there exists

$1 \leq i < j < k \leq n$ in which v_j, v_k disagree on v_i, then the leftmost vertex on which they disagree is adjacent to v_j. The algorithm details are given below in ALGORITHM LEXBFS(G, P). We use $\sigma(y) = i$ to indicate that y is the i^{th} vertex of σ. Note that the numbering convention in this paper follows that of [3] where the ordering σ is the order in which vertices are visited during the LexBFS.

There are several different paradigms for implementing LexBFS. The term *lexicographic* stems from the original *labelling* paradigm in which vertices pass a label to each unnumbered neighbour. The earlier a vertex is visited, the higher the label passed. The next vertex is chosen such that it has the lexicographically largest label. Another conceptualization of LexBFS involves pivots and partitions [10]. The *partitioning* paradigm will prove the most useful for illustrating the new variant, LexBFS$^-$.

The *partitioning* paradigm considers a pivot (the current vertex) and all unnumbered vertices. A vertex is considered to be visited if it has been a pivot, i.e. numbered. The algorithm begins with a partition list L initialized to a single ordered set P of the vertices in V. The first pivot is the first vertex of P and is first in the ordering σ. Let $N(x)$ refer to the *neighbourhood* of x. For each pivot x, remove x from its partition and define $N' = \{v | v \in N(x)$ and v unnumbered$\}$. For each set P in the partition L, all members of N' belonging to P are removed from P and inserted into a new set P' positioned immediately to the left of P.

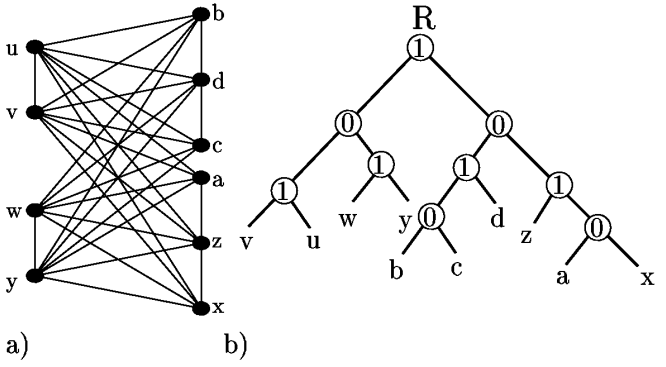


Fig. 1. a) A cograph G. b) An embedding of the cotree T_G of G.

For example, for the graph in Fig. 1, if x is the first vertex selected, then the remaining vertex set is split into two partitions $\boxed{\text{y u v w z}}$ and $\boxed{\text{d c a b}}$ where $\boxed{\text{y u v w z}}$ is moved ahead of $\boxed{\text{d c a b}}$ since each of $\{yuvwz\}$ is adjacent to x while $\{dcab\}$ are not adjacent to x.

The LexBFS terminates when all vertices have been visited. Algorithm LEXBFS is a variation of the version appearing in [10]. Table 1 walks through a LexBFS of the cograph of Fig. 1.

ALGORITHM LEXBFS(G, P)
Input: a connected graph $G = (V, E)$ and an initial ordering P of V
Output: an ordering σ of the vertices of G
1 $L \leftarrow (P)$; *L, the list of partitions, is initialized to the partition P*
2 $i \leftarrow 1$; *The counter for assigning vertex positions*
3 **while** there exists a set in $L = (P_1, \ldots, P_k)$ of unnumbered vertices **do**
4 Let P_l be the leftmost partition of unnumbered vertices;
5 Remove the first vertex x from P_l; *the pivot*
6 $\sigma(x) \leftarrow i$;
7 $i \leftarrow i + 1$;
8 **for** each partition $P_j, j \geq l$ **do**
9 Let $P' = \{v | v \in N(x) \cap P_j\}$; *$P'$ are the vertices of P_j adjacent to x*
10 **if** P' is non-empty and $P' \neq P_j$ **then**
11 Remove P' from P_j;
12 Insert P' to the left of P_j in L; *P' forms a new partition*
13 **end for**
14 **end while**
15 **return** (σ);
end LEXBFS

Table 1. Step by step LexBFS of G from Fig. 1.

$\sigma(v)$	v	$N'(v)$	Partitions
			x d y u v w c a z b
1	x	{u v w y z}	y u v w z | d c a b
2	y	{a b c d w z}	w z | u v | d c a b
3	w	{a b c d z}	z | u v | d c a b
4	z	{a u v}	u v | a | d c b
5	u	{a b c d v}	v | a | d c b
6	v	{a b c d}	a | d c b
7	a	{ }	d c b
8	d	{b c}	c b
9	c	{b}	b
10	b	{ }	

Definition 1 *The vertices in the leftmost partition of line 5 of* ALGORITHM LEXBFS(G, V) *are tied, in that their neighbourhoods of numbered vertices are identical. We refer to such a set of tied vertices as a* slice, S.

An example of a slice is the partition $\boxed{\text{y u v w z}}$ in Table 1 defined by the neighbourhood of x. Notice that the entire vertex set is initially a slice.

Given a slice S of tied vertices, ALGORITHM LEXBFS breaks the tie in an arbitrary manner by selecting the first vertex of the leftmost partition. Some variants of LexBFS use a more sophisticated tie breaking mechanism. For example, LexBFS$^+$ [3] breaks ties by considering the ordering τ, produced by a previous LexBFS sweep, and selects the vertex x in S numbered latest in τ. The partitioning paradigm easily facilitates this tie breaking mechanism in the following way. The initial partition P is the vertex set in *reverse* τ order. Since ALGORITHM LEXBFS takes the first vertex of the leftmost partition P_l as the next pivot, this implementation automatically satisfies the tie breaking rule. The main insight of this paper involves the introduction of a new tie breaking mechanism discussed in Section 2.2 on LexBFS$^-$.

1.2 Cotrees

Some notation with respect to cotrees will be necessary in subsequent sections. Let T be an embedding of the cotree of a cograph $G = (V, E)$ rooted at R. For each leaf x, let r_x be the root of the largest subtree of the embedding such that x is the rightmost leaf. Let P_x be the directed path in T from x to r_x. The internal nodes on P_x alternate between 0 and 1 nodes. Let $(0_1^x, 0_2^x, \ldots, 0_k^x)$ be the sequence of 0 nodes on P_x. Similarly let the sequence $(1_1^x, 1_2^x, \ldots, 1_{k'}^x)$ denote the 1 nodes. Note that $|k - k'| \leq 1$.

Now consider the subtree t' rooted at 0_i^x, the i^{th} 0 node on P_x. Let z be the child of 0_i^x that lies on P_x. Define the subtree $\mathbf{T_{0i}^x}$ to be t' without the subtree rooted at z. An analogous definition holds for T_{1i}^x. Figure 2 highlights a subtree T_{02}^x for the cotree of Fig. 1.

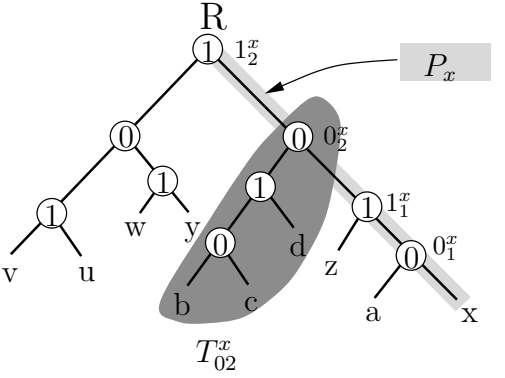


Fig. 2. Let $t = T$ and therefore $r = R$. Dark grey indicates the subtree T_{02}^x and light grey the path P_x.

Observation 1 *Cographs are closed under complementation. The cotree* $\overline{T}$ *of* $\overline{G}$*, the complement of* G*, is exactly* T *with 0 and 1 nodes interchanged.*

Observation 2 *Cographs are hereditary in the sense that any induced subgraph is also a cograph. Furthermore, given cograph* G *with cotree* T_G*, an induced, rooted subtree* t *of* T_G *is the cotree of an induced subgraph of* G*.*

2 Unfolding the Cograph Recognition Algorithm

2.1 Relating Slices to Subtrees

An intuitive sense for the new cograph recognition algorithm is best developed by observing how LexBFS reveals the structure of a given cograph. Consider the LexBFS σ depicted in Table 1, that started with the arbitrarily chosen vertex x,

$$\sigma : \mathbf{xywzuvadcb}.$$

In an attempt to reveal underlying relationships between the ordering σ and the cograph structure of G we investigate properties of the slices generated during the course of the search. First, notice that selecting x defines the following slice containing vertices adjacent to x (ordered as they appear in σ),

$$[\mathbf{ywzuv}].$$

After numbering each of $ywzu$ and v, only vertices not adjacent to x remain (see Table 1, row 6), and are partitioned according to their adjacencies with $ywuzv$. In particular, one vertex, a, is adjacent to each of $ywzuv$ and therefore defines its own slice [**a**]. The remaining vertices form the slice

$$[\mathbf{dcb}].$$

The slices define an ordered partitioning of σ into the following sequence:

$$\sigma : \mathbf{x}[\mathbf{ywzuv}][\mathbf{a}][\mathbf{dcb}].$$

Recall that in Algorithm LexBFS(G, P), each pivot is selected from a slice, or the leftmost partition P_l. The set of *all* slices constructed during the sweep defines nested ordered partitionings of σ. We often use the term *subslice* to refer to a slice nested inside another slice. We now present notation to refer to the first level of nested subslices of a slice.

Definition 2 *Let* $S = [x, S^A(x), \langle S^N(x)\rangle]$ *be an arbitrary slice constructed during a LexBFS sweep where* x *is the first vertex of* S*. We consider only the first level of nested subslices of* S*. Let* $S^A(x) = [$*the subslice of* S *of vertices adjacent to* $x]$*. Each subsequent subslice of* S *contains vertices not adjacent to* x*. Let* $\langle S^N(x)\rangle$ *where* $\langle S^N(x)\rangle = S_1^N(x), S_2^N(x), \ldots, S_k^N(x)$ *denote the subslices not adjacent to* x *in* S*.*

Recalling that the entire vertex set is initially a slice, if we let $S = \sigma$, then $S^A(x) = [ywzuv]$, $S_1^N(x) = [a]$ and $S_2^N(x) = [dcb]$. For all $i \geq 3$, $S_i^N(x)$ is empty and is therefore omitted . If t is the first vertex of S and $S^A(t)$ is empty but $S_1^N(t)$ is non-empty, then $S^A(t) = [\ ]$ and is preserved as a place holder. The following surprising observation forms the basis of the new algorithm.

Observation 3 *Let σ denote a LexBFS of the cograph G of Fig. 1 starting at vertex x and let T denote an* embedding *of the cotree of G such that x is the rightmost leaf of T. Let S be the initial slice, then S is partitioned into $x, S^A(x), \langle S^N(x)\rangle$ and* ***each slice*** *$S_i^N(x)$* ***contains exactly the leaves of the subtree*** *T_{0i}^x* ***of*** *T.*

Figure 3 illustrates Observation 3. Notice that $\mathbf{S_1^N(x) = [a]}$ contains exactly the leaf of $\mathbf{T_{01}^x}$ and similarly that $\mathbf{S_2^N(x) = [dcb]}$ contains exactly the leaves of $\mathbf{T_{02}^x}$. In fact, Observation 3 holds for *any* LexBFS of *any* cograph. Unfortunately, as seen in Fig. 3, the vertices in $S^A(x)$ are *not* partitioned into their respective subtrees. Naturally, we would like to have a similar relationship for the T_{1i}^x

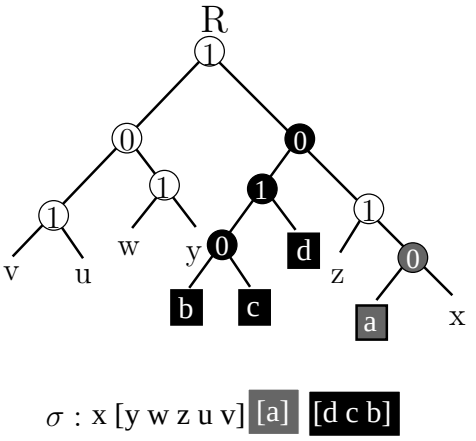


Fig. 3. The pairing of slices with subtrees T_{0i}^x.

subtrees. Recall Observation 1 that the cotree $\overline{T}$ of $\overline{G}$ is exactly T with the 0 and 1 nodes interchanged. Thus a LexBFS of the *complement* graph $\overline{G}$, starting at x, should partition vertices into slices of leaves of the subtrees T_{1i}^x.

Akin to the notation for slices of σ, we denote an arbitrary slice in $\overline{\sigma}$ by $\overline{S}$. Similarly, the sequence of subslices of a slice $\overline{S}$ for which x is the first vertex is as follows:

- vertices adjacent to x in $\overline{G}$ are denoted by $\overline{S^A}(x)$ and
- subslices of vertices not adjacent to x are denoted by the sequence $\langle \overline{S_i^N}(x)\rangle = \overline{S_1^N}(x), \overline{S_2^N}(x), \ldots, \overline{S_k^N}(x)$.

As expected, a LexBFS of $\overline{G}$ starting at x, does indeed define a partitioning of the initial slice $\overline{S} = V$ into $x, \overline{S^A}(x), \langle \overline{S^N}(x) \rangle$ where each $\overline{S_i^N}(x)$ ***contains exactly the leaves of*** T_{1i}^x in the embedding T.

Referring to Fig. 3 again, a LexBFS of $\overline{G}$ starting at x supports our claim, for example,

$$\overline{\sigma} : x[dacb][z][uwyv], \tag{1}$$

reveals that $\overline{\mathbf{S_1^N}}(\mathbf{x}) = [\mathbf{z}]$ contains the leaf of $\mathbf{T_{11}^x}$ and $\overline{\mathbf{S^N}}_\mathbf{2}(\mathbf{x}) = [\mathbf{uwyv}]$ contains the leaves of $\mathbf{T_{12}^x}$. As with Observation 3, this property generalizes to *any* LexBFS of the complement of *any* cograph.

As mentioned, it is necessary that the LexBFS $\overline{\sigma}$ start at x in order to determine the correct T_{1i}^x leaves on the path P_x and in general, for any cograph, $\overline{\sigma}$ must start at the same vertex as σ. In addition, the hope is that the above observations about slices and subtrees applies recursively within each subtree. Specifically, the subslices of $\overline{S_i^N}(x)$ defined by $\overline{\sigma}$ should determine the leaves of the subtrees of T_{1i}^x. Similarly, the recursion should hold for the subslices of $S_i^N(x)$ of σ and the subtrees of T_{0i}^x.

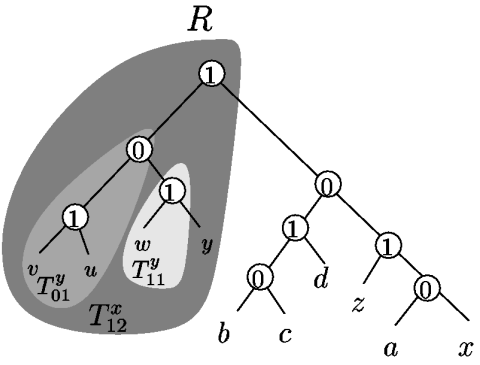
Recall Observation 2 which says that the subgraph H of G induced by the leaves of T_{1i}^x is a cograph. Additionally we know that $\overline{S_i^N}(x)$ contains the leaves of T_{1i}^x. Therefore, if y is the first vertex of $\overline{S_i^N}(x)$, define P_y to be the path from y to the root of T_{1i}^x. We can now define the subtrees T_{0j}^y and T_{1j}^y rooted on P_y.

Since $\overline{S_i^N}(x)$ contains the leaves of T_{1i}^x, the vertices of $\overline{S_i^N}(x)$ induce H. Therefore the LexBFS $\overline{\sigma}$ restricted to $\overline{S_i^N}(x)$ is a LexBFS of $\overline{H}$ and assuming that y is the first vertex of $\overline{S_i^N}(x)$, $\overline{\sigma}$ partitions $\overline{S_i^N}(x)$ into y, $\overline{S^A}(y)$, $\langle \overline{S^N}(y) \rangle$. Therefore, in H or equivalently T_{1i}^x, each $\overline{S_j^N}(x)$ contains exactly the leaves of T_{1j}^y.

However, we still need to resolve which leaves of T_{1i}^x belong to which T_{0j}^y. This requires that σ contain a sequence of slices $\langle S^N(y) \rangle$ such that each $S_j^N(y)$ contains the leaves of each T_{0j}^y. Notice that this implies that at the very least, σ number y **before** numbering any vertices of the sequence $\langle S^N(y) \rangle$, i.e. before the leaves of each subtree T_{0j}^y. In fact, it turns out that σ must number y before all other leaves in T_{1i}^x.

Returning to the slice $\overline{S_i^N}(x)$, recall that the selection process for choosing the first vertex to be numbered is arbitrary in that it is simply the first vertex in the leftmost partition. Consider selecting vertex y from $\overline{S_i^N}(x)$ such that y appears **earliest** in the ordering σ. This ensures that in σ, y is numbered before any other vertex in $\overline{S_i^N}(x)$, i.e. before any other leaf in T_{1i}^x. This new tie breaking mechanism defines a new variant of LexBFS referred to as **LexBFS$^-$** or **LexBFS *minus***.

Figure 4 indicates the recursion on the subtree T_{12}^x using a LexBFS$^-$ on $\overline{G}$. Notice that the LexBFS$^-$ version of $\overline{\sigma}$ in Fig. 4 is different than the arbitrary LexBFS of $\overline{G}$ chosen in (1) above.



$$\sigma = x[y[wz][uv]][a][dcb] \Rightarrow S_1(y) = [uv]$$
$$\overline{\sigma} = x[a[d[][cb]]][z][y[uv][w]] \Rightarrow \overline{S}_1(y) = [w]$$

Fig. 4. LexBFS$^-$ of $\overline{G}$.

2.2 LexBFS$^-$

Definition 3 *A LexBFS$^-(\overline{G}, \sigma)$ is a LexBFS of the* ***complement*** *graph $\overline{G}$ that uses a previous LexBFS σ of G to break ties. That is, a vertex x of a slice S is selected if its index in σ is smallest.*

Observe that given a graph G, a LexBFS of $\overline{G}$ is found by implementing Algorithm LexBFS with P' inserted to the *right* of P_j at line 12. In addition, the implementation of the LexBFS$^-$ tie breaking mechanism mirrors the implementation of LexBFS$^+$. In particular, if the initial partition P used by Algorithm LexBFS is assigned the *exact* ordering of the previous sweep, then the minus tie breaking mechanism is implemented automatically.

Lemma 1 *LexBFS$^-(\overline{G}, \sigma)$ can be implemented in linear time with respect to G.*

As shown in Algorithm Recognize_Cograph, the new cograph recognition algorithm implements three sweeps leading to the following notation and theorem. Let τ denote the initial LexBFS ordering, $\overline{\sigma}_1$ denote the second ordering resulting from LexBFS$^-(\overline{G}, \tau)$ and σ_2 denote the third ordering resulting from LexBFS$(G, \overline{\sigma}_1)$.

Theorem 1 *If $G = (V, E)$ is a cograph with $\overline{\sigma}_1$ and σ_2 as defined above, then the cotree T of G can be built in linear time from the slice sequences $\langle S^N(v)\rangle$ (as determined by σ_2) and $\langle \overline{S^N}(v)\rangle$ (as determined by $\overline{\sigma}_1$) for each $v \in V$.*

In the next section we describe various aspects of Algorithm Recognize_Cograph, including details of how to construct the cotree if the input graph is a cograph.

3 The Cograph Recognition Algorithm

Having computed $\overline{\sigma}_1$ and σ_2, Algorithm Recognize_Cograph determines whether $\overline{\sigma}_1$ and σ_2 satisfy the *Neighbourhood Subset Property* and if so, calls Construct_Cotree described in the following section.

3.1 Constructing the Cotree

Algorithm Construct_Cotree is a high level description of the algorithm to build the cotree. An indication of theorems needed for the correctness of the algorithm follows with complete implementation and proof details available in the journal version of the paper.

The algorithm to build the cotree walks through the sequences $\langle S^N(v)\rangle$ and $\langle \overline{S^N}(v)\rangle$ for each vertex $v \in V$, and builds a path P_{vr} from v to the root r of the subtree containing all vertices in $\langle S^N(v)\rangle \bigcup \langle \overline{S^N}(v)\rangle$. For each 0_i^v node on P_{vr}, the algorithm is recursively called to build the subtree T_{0i}^v and likewise for each 1_i^v node and the subtree T_{1i}^v.

Algorithm Construct_Cotree(S)
Global Var: Slice sequences $\langle S^N(v)\rangle$, $\langle \overline{S^N}(v)\rangle$, $\forall v \in V$,
defined by $\overline{\sigma}_1$ and σ_2.
Input: S: the current slice
Output: The cotree of $G(S)$
$x \leftarrow S[1]$; *Assign x the 1^{st} vertex of the slice S*
Starting at x construct a path T of alternating 0 and 1 nodes where each 0 node is represented by the slice $S_i^N(x)$ and each 1 node by $\overline{S_i^N}(x)$
for each i
Assign to the i^{th} 0 node the subtree Construct_Cotree($S_i^N(x)$);
Assign to the i^{th} 1 node the subtree Construct_Cotree($\overline{S_i^N}(x)$);
return(T);
end Construct_Cotree

Correctness. For the following two Lemmas, we assume that a graph $G = (V, E)$, and LexBFS$^-$ $\overline{\sigma}_1$ and LexBFS$^-$ σ_2 are given. Let S be a slice of σ_2 such that S contains the vertices of a subtree t of T (possibly T itself). Let r represent the root of t. If v is the first vertex of S, then the path from v to r is P_{vr}.

Lemma 2 *Each slice $S_i^N(v)$ in the sequence $\langle S^N(v)\rangle$ contains the vertices of the subtree T_{0i}^v rooted on P_{vr}. Similarly, each slice $\overline{S_i^N}(v)$ in the sequence $\langle \overline{S^N}_{(v)}\rangle$ contains the vertices of the subtree T_{1j}^v.*

Lemma 3 *Let x be the first vertex in the ordering σ_2 (and therefore the first vertex in the ordering $\overline{\sigma}_1$). The path P_xR from x to the root R of the cotree T_G of G can be constructed from the slice sequences $\langle S^N(x)\rangle$ and $\langle \overline{S^N}(x)\rangle$.*

Lemmas 2 and 3 are the driving force behind Theorem 2.

Theorem 2 *Given $\overline{\sigma}_1$ and σ_2 of a cograph G, the cotree T_G can be constructed from the slices of $\overline{\sigma}_1$ and σ_2.*

3.2 The Neighbourhood Subset Property

Recall that during a LexBFS, a slice is a set of vertices all with equal neighbourhood of numbered vertices. Therefore it is natural to talk about the numbered neighbourhood of a slice at the time that it is defined.

Definition 4 *Given a slice $S_i^N(v)$ we define the* numbered neighbourhood *of $S_i^N(v)$ to be*

$$N_i(v) = \{y \mid y \text{ numbered and } y \in N(z),\ \forall z \in S_i^N(v)\}\ .$$

Similarly we define the neighbourhood of a subtree T_{0i}^v to be

$$N_{0i}^v = \{y \mid y \notin T_{0i}^v \text{ and } y \in N(z), \forall z \in T_{0i}^v\}\ .$$

Lemma 4 *Let T_{0i}^v and T_{0i+1}^v be subtrees on the path P_{vR}. The following hold:*

- *The leaves of T_{0i}^v are not adjacent to any leaves of T_{0i+1}^v.*
- $N_{0i+1}^v \subset N_{0i}^v$

A similar lemma appears in [7].

Definition 5 *A LexBFS satisfies the* Neighbourhood Subset Property *iff*

$$N_{i+1}(v) \subset N_i(v), \forall v \in V, \forall i \geq 1\ .$$

Lemma 4 coupled with the fact that for cographs, the vertices of $S_i^N(v)$ and $\overline{S_i^N}(v)$ are exactly the leaves of T_{0i}^v and T_{1i}^v respectively, forms the basis of the proof of Theorem 3.

Theorem 3 *Given graph $G = (V, E)$ and $\overline{\sigma}_1$ and σ_2 of* Algorithm Recognize_Cograph(G). *G is a cograph iff $\overline{\sigma}_1$ and σ_2 satisfy the* Neighbourhood Subset Property.

If a pair of slices of a LexBFS fail the Neighbourhood Subset Property then by Theorem 3, a P_4 exists. In addition, the point where the condition fails provides a starting point to construct a P_4 from the numbered neighbourhoods of the slices. This is accomplished by algorithm Construct_P_4.

Lemma 5 *Given a pair of slices $S_i^N(v)$ and $S_{i+1}^N(v)$ failing the Neighbourhood Subset Property, a* P_4 *can be constructed in $\mathcal{O}(|N(v)|)$ time.*

In conclusion, the complexity of Algorithm Recognize_Cographs(G) is linear in time and space.

Acknowledgements. The authors would like to thank the anonymous referees for many helpful suggestions regarding the presentation of this paper. Additionally, the first and second authors would like to thank the Natural Sciences and Engineering Research Council of Canada for financial assistance.

References

[1] Jou-Ming Chang, Chin-Wen Ho, and Ming-Tat Ko.: LexBFS-ordering in Asteroidal Triple-free Graphs.ISAAC: 10th International Symposium on Algorithms and Computation, **LNCS 1741**, A. Aggarwal and C. Pandu Rangan (Eds.), Springer-Verlag, Berlin, (1999) 163–172.

[2] Derek G. Corneil.: A Simple 3-sweep LBFS Algorithm for the Recognition of Unit Interval Graphs. Discrete Applied Mathematics.(to appear)

[3] Derek G. Corneil, Steven Olariu and Lorna K. Stewart.: The Ultimate Interval Graph Recognition Algorithm? (Extended Abstract). Symposium on Discrete Algorithms. (1998), 175–180.

[4] D. G. Corneil, H. Lerchs and L. Stewart Burlingham.: Complement Reducible Graphs. Discrete Applied Math.**3** 1(1981) 63–174.

[5] D. G. Corneil, Y. Pearl and L. Stewart.: A linear recognition algorithm for cographs. SIAM Journal of Computing. **14(4)** (1985) 926–934.

[6] A. Cournier, M. Habib.: A new linear algorithm for modular decomposition.**LNCS 787** (1994) 68–84.

[7] E. Dahlhaus.: Efficient parallel recognition algorithms for cographs and distance hereditary graphs. Discrete Applied Mathematics. **57** (1995) 29–45.

[8] E. Dahlhaus, J. Gustedt, and R. M. McConnell.: Efficient and practical modular decomposition. 8th Annual ACM-SIAM Symposium On Discrete Algorithms (SODA). (1997) 26–35.

[9] M. Habib and C. Paul.: A new vertex splitting algorithm for cograph recognition. Technical Report, April 2000. **URL** http://citeseer.nj.nec.com/habib00new.html.

[10] M. Habib, R. M. McConnell, C. Paul, L. Viennot.: Lex-BFS and Partition Refinement, with Applications to Transitive Orientation, Interval Graph Recognition and Consecutive Ones Testing. Theoretical Computer Science. **234** (2000) 59–84.

[11] N. Korte and H. Möhring.: An incremental linear-time algorithm for recognizing interval graphs. SIAM J. Comput. **18** (1989) 68–81.

[12] R. M. McConnell and J. Spinrad.: Linear-Time Modular Decomposition and Efficient Transitive Orientation of Comparability Graphs.Proceedings of the ACM-SIAM Symposium on Discrete Algorithms. (1994) 536–545.

[13] D. J. Rose, R. E. Tarjan, and G. S. Lueker.: Algorithmic aspects of vertex elimination on graphs. SIAM J. Comput. **5** (1976) 266–283.

[14] S. Shew.: A Cograph Approach to Examination Scheduling. M.Sc Thesis, Dept. of Computer Science, University of Toronto. (1986).

[15] D. B. West.: Introduction to Graph Theory, 2^{nd} Edition. Prentice Hall, Upper Saddler River, (2001).

Backbone Colorings for Networks

Hajo Broersma[1], Fedor V. Fomin[2], Petr A. Golovach[3], and Gerhard J. Woeginger[4]

[1] University of Twente, The Netherlands
broersma@math.utwente.nl
[2] University of Bergen, Norway
fomin@ii.uib.no
[3] Syktyvkar State University, Russia
golovach@ssu.komi.com
[4] University of Twente, The Netherlands
g.j.woeginger@math.utwente.nl

Abstract. We study backbone colorings, a variation on classical vertex colorings: Given a graph $G = (V, E)$ and a spanning subgraph H (the backbone) of G, a backbone coloring for G and H is a proper vertex coloring $V \to \{1, 2, \ldots\}$ in which the colors assigned to adjacent vertices in H differ by at least two. We concentrate on the cases where the backbone is either a spanning tree or a spanning path.
For tree backbones of G, the number of colors needed for a backbone coloring of G can roughly differ by a multiplicative factor of at most 2 from the chromatic number $\chi(G)$; for path backbones this factor is roughly $\frac{3}{2}$. In the special case of split graphs G, the difference from $\chi(G)$ is at most an additive constant 2 or 1, for tree backbones and path backbones, respectively. The computational complexity of the problem 'Given a graph G, a spanning tree T of G, and an integer ℓ, is there a backbone coloring for G and T with at most ℓ colors?' jumps from polynomial to NP-complete between $\ell = 4$ (easy for all spanning trees) and $\ell = 5$ (difficult even for spanning paths).

1 Introduction and Related Research

Our work is motivated by the general framework for coloring problems related to frequency assignment. In this application area graphs are used to model the topology and mutual interference between transmitters (receivers, base stations): the vertices of the graph represent the transmitters; two vertices are adjacent in the graph if the corresponding transmitters are so close (or so strong) that they are likely to interfere if they broadcast on the same or 'similar' frequency channels. The problem in practice is to assign the frequency channels to the transmitters in such a way that interference is kept at an 'acceptable level'. This has led to various different types of coloring problems in graphs, depending on different ways to model the level of interference, the notion of similar frequency channels, and the definition of acceptable level of interference (See e.g. [13],[18]). One way of putting these problems into a more general framework is the following.

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 131–142, 2003.

> For a given graph G_2 and for a given spanning subgraph G_1 of G_2, determine a vertex coloring that satisfies certain restrictions of type 1 in G_1, and certain restrictions of type 2 in G_2.

Many known coloring problems related to frequency assignment fit into this general framework: First of all suppose that $G_2 = G_1^2$, i.e. G_2 is obtained from G_1 by adding edges between all pairs of vertices that are at distance 2 in G_1. If one just asks for a proper vertex coloring of G_2 (and G_1), this is known as the distance-2 coloring problem. Research has concentrated on the case that G_1 is a planar graph. We refer to [1], [3], [4], [16], [19], and [20] for more details. In some versions of this problem one puts the additional restriction on G_1 that the colors should be sufficiently separated, in order to model practical frequency assignment problems in which interference should be kept at an acceptable level. One way to model this is to use positive integers for the colors (modeling certain frequency channels) and to ask for a coloring of G_1 and G_2 such that the colors on adjacent vertices in G_2 are different, whereas they differ by at least 2 on adjacent vertices in G_1. This problem is known as the radio coloring problem and has been studied in [2], [5], [6], [7], [8], [9], and [17].

The so-called radio labeling problem models a practical setting in which all assigned frequency channels should be distinct, with the additional restriction that adjacent transmitters should use sufficiently separated frequency channels. Within the above framework this can be modeled by considering the graph G_1 that models the adjacencies of n transmitters, and taking $G_2 = K_n$, the complete graph on n vertices. The restrictions are clear: one asks for a proper vertex coloring of G_2 such that adjacent vertices in G_1 receive colors that differ by at least 2. We refer to [12] and [15] for more particulars.

In this paper, we model the situation that the transmitters form a network in which a certain substructure of adjacent transmitters (called the backbone) is more crucial for the communication than the rest of the network. This means we should put more restrictions on the assignment of frequency channels along the backbone than on the assignment of frequency channels to other adjacent transmitters. The backbone could e.g. model so-called hot spots in the network where a very busy pattern of communications takes place, whereas the other adjacent transmitters supply a more moderate service. We consider the problem of coloring the graph G_2 (that models the whole network) with a proper vertex coloring such that the colors on adjacent vertices in G_1 (that model the backbone) differ by at least 2. Throughout the paper we consider two types of backbones: spanning trees and a special type of spanning trees that are better known as Hamiltonian paths.

1.1 Terminology and Notation

All graphs in this paper are connected, finite, simple, and undirected. Consider a graph $G = (V, E)$ with a spanning tree $T = (V, E_T)$. A vertex coloring f of V is *proper*, if $|f(u) - f(v)| \geq 1$ holds for all edges $uv \in E$. A vertex coloring is a *backbone* coloring for (G, T), if it is proper and if additionally $|f(u) - f(v)| \geq 2$

holds for all edges $uv \in E_T$ in the spanning tree T. The chromatic number $\chi(G)$ is the smallest integer k for which there exists a proper coloring $f : V \to \{1, \ldots, k\}$. The backbone coloring number $\text{BBC}(G, T)$ of (G, T) is the smallest integer ℓ for which there exists a backbone coloring $f : V \to \{1, \ldots, \ell\}$. When dealing with colorings, we say that two colors z_1 and z_2 are *adjacent* if and only if $|z_1 - z_2| = 1$.

A *Hamiltonian path* of the graph $G = (V, E)$ is a simple path that contains all the vertices of G. A *split graph* is a graph whose vertex set can be partitioned into a *clique* (i.e. a set of mutually adjacent vertices) and an *independent set* (i.e. a set of mutually nonadjacent vertices), with possibly edges in between. The size of a largest clique in G is denoted by $\omega(G)$. Split graphs are perfect graphs, and hence satisfy $\chi(G) = \omega(G)$.

1.2 Results

We start our investigations of the backbone coloring number by analyzing its relation to the classical chromatic number. How far away from $\chi(G)$ can $\text{BBC}(G, T)$ be in the worst case? To answer this question, we introduce for integers $k \geq 1$ the values

$$\mathcal{T}(k) \;:=\; \max\{\, \text{BBC}(G, T) :\; T \text{ is a spanning tree of } G, \text{ and } \chi(G) = k\}$$

It turns out that this function $\mathcal{T}(k)$ behaves quite primitively:

Theorem 1 *$\mathcal{T}(k) = 2k - 1$ for all $k \geq 1$.*

The upper bound $\mathcal{T}(k) \leq 2k - 1$ in this theorem in fact is straightforward to see. Indeed, consider a proper coloring of G with colors $1, \ldots, \chi(G)$, and replace every color i by a new color $2i - 1$. The resulting coloring uses only odd colors, and hence constitutes a 'universal' backbone coloring for any spanning tree T of G. The proof of the matching lower bound $\mathcal{T}(k) \geq 2k - 1$ is more involved and will be presented in Section 2.

Next, let us discuss the situation where the backbone tree is a Hamiltonian path. Similarly as with spanning trees, we introduce for integers $k \geq 1$ the values

$$\mathcal{P}(k) \;:=\; \max\{\, \text{BBC}(G, P) :\; P \text{ is a Hamiltonian path of } G, \text{ and } \chi(G) = k\}$$

In Section 3 we will exactly determine all these values $\mathcal{P}(k)$ and observe that they roughly grow like $3k/2$. Their precise behavior is summarized in the following theorem.

Theorem 2 *For $k \geq 1$ the function $\mathcal{P}(k)$ takes the following values:*

(a) For $1 \leq k \leq 4$: $\mathcal{P}(k) = 2k - 1$;
(b) $\mathcal{P}(5) = 8$ and $\mathcal{P}(6) = 10$;
(c) For $k \geq 7$ and $k = 4t$: $\mathcal{P}(4t) = 6t$;
(d) For $k \geq 7$ and $k = 4t + 1$: $\mathcal{P}(4t + 1) = 6t + 1$;
(e) For $k \geq 7$ and $k = 4t + 2$: $\mathcal{P}(4t + 2) = 6t + 3$;
(f) For $k \geq 7$ and $k = 4t + 3$: $\mathcal{P}(4t + 3) = 6t + 5$;

Next, let us turn to the special case of backbone colorings on split graphs. Split graphs were introduced by Hammer & Földes [14]; see also the book [11] by Golumbic. They form an interesting subclass of the class of perfect graphs. The combinatorics of most graph problems becomes easier when the problem is restricted to split graphs. The following theorem is a strengthening of Theorems 1 and 2 for the special case of split graphs. Because of space restrictions, its proof has been moved to the full version of this extended abstract.

Theorem 3 *Let $G = (V, E)$ be a split graph.*

(a) For every spanning tree T in G, $\text{BBC}(G, T) \leq \chi(G) + 2$.
(b) If $\omega(G) \neq 3$, then for every Hamiltonian path P in G, $\text{BBC}(G, P) \leq \chi(G)+1$.

Both bounds are best possible.

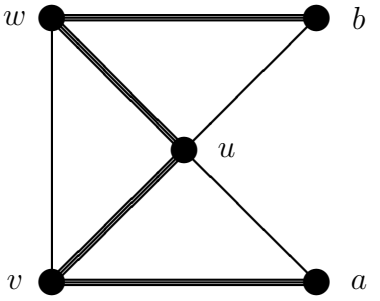


Fig. 1. A split graph G with a Hamiltonian path P (bold edges), such that $\chi(G) = 3$ and $\text{BBC}(G, P) = 5$.

Let us show here why for split graphs with clique number 3 the statement in Theorem 3.(b) does not work. Consider the split graph G on five vertices from Figure 1. Vertices v, u, w form a clique, vertex a is adjacent to v, u and vertex b is adjacent to u, w. The clique number (and hence the chromatic number) of this graph is equal to 3. Let $P = (a, v, u, w, b)$ be the Hamiltonian path. We claim that $\text{BBC}(G, P) > \chi(G) + 1$. To the contrary, assume that G, P has a backbone coloring with colors 1,2,3,4. It is easy to see that u can not be colored with color 2 or 3; otherwise we are forced to use the same color for v and w, a clear contradiction. Now suppose that u is colored with color 1 (the case when u is colored with color 4 is similar). Then one of its neighbors in P must have color 3 and the other one color 4. Without loss of generality assume that v has color 3. Vertex a is adjacent in P to v, so the colors 2,3,4 are forbidden for a and the only valid color for a is 1. But a is adjacent in G to u which has color 1 as well. This contradiction completes the proof of the claim.

Finally, we discuss the computational complexity of computing the backbone coloring number: *"Given a graph G, a spanning tree T, and an integer ℓ, is* $\text{BBC}(G,T) \leq \ell$*?"* Of course, this general problem is NP-complete. It turns out that for this problem the complexity jump occurs between $\ell = 4$ (easy for all spanning trees) and $\ell = 5$ (difficult even for Hamiltonian paths).

Theorem 4

(a) The following problem is polynomially solvable for any $\ell \leq 4$: Given a graph G and a spanning tree T, decide whether $\text{BBC}(G,T) \leq \ell$.

(b) The following problem is NP-complete for all $\ell \geq 5$: Given a graph G and a Hamiltonian path P, decide whether $\text{BBC}(G,P) \leq \ell$.

2 Tree Backbones and the Chromatic Number

This section is devoted to a proof of the lower bound statement $\mathcal{T}(k) \geq 2k-1$ in Theorem 1. Consider some arbitrary $k \geq 1$. We will construct a graph G with chromatic number $\chi(G) = k$, and a spanning tree T of G, such that $\text{BBC}(G,T) = 2k-1$.

The graph G is a complete k-partite graph that consists of k independent sets $V_1, \ldots, V_k$ that are all of cardinality k^k. Clearly, $\chi(G) = k$. The spanning tree T is defined as the final tree in the following inductive construction: The tree T_0 is a star with root in V_1 and $k-1$ leaves in the $k-1$ sets $V_2, \ldots, V_k$, one in each set. For $j = 1, \ldots, k$ the tree T_j is constructed from the tree T_{j-1}, by creating $k-1$ new vertices for every vertex v in T_{j-1} and by attaching them to v. If v is in the set V_q, then every independent set V_i with $i \neq q$ contains exactly one of these new vertices. Note that all newly created vertices are leaves in the tree T_j. It is easy to see that the tree T_j consists of k^{j+1} vertices that are equally distributed among the sets $V_1, \ldots, V_k$. We denote the vertex set of T_j by $V(T_j)$. Note that $V(T_j) \subset V(T_{j+1})$.

Consider a backbone coloring of (G,T) with ℓ colors where $T = T_k$ is the final tree in the above sequence of trees. Since G is complete k-partite, any color that is used in some set V_i cannot be used in any V_j with $j \neq i$. We denote by C_i the set of colors that are used on vertices in V_i. We now go through a number of steps; in every step, the colors in one of the color sets C_i are labeled with the labels A and B.

> (Step s). If there exists some (yet unlabeled) color set C_i such that $|C_i|-1$ of the colors in C_i are adjacent to a color with label A, then: Label these $|C_i|-1$ colors by label B. Label the remaining color in C_i by label A.

Eventually, there will be no more color class that satisfies the condition in the if-part: Either, all colors have been labeled, or each of the remaining unlabeled color classes contains at least two colors that are not adjacent to any color with label A. If this is the case at the start, then $|C_i| \geq 2$ for all i, and we obtain $\ell \geq 2k$. We denote by $a \leq k$ the number of steps performed; we assume $a \geq 1$. We denote by $\pi(s)$ $(s = 1, \ldots, a)$ the index of the color set that is labeled in step s. Moreover, we denote by $c_{\pi(s)}$ the unique color in $C_{\pi(s)}$ that is labeled A.

Lemma 5 *For any integer s with $1 \leq s \leq a$ the following statements hold.*
(L1). The backbone coloring colors all vertices in $V(T_{k-s}) \cap V_{\pi(s)}$ with $c_{\pi(s)}$.
(L2). The color $c_{\pi(s)}$ is not adjacent to any color $c_{\pi(q)}$ with $q < s$.

Proof. The proofs of (L1) and (L2) are done simultaneously by induction on s. In step $s = 1$, only a color class $C_{\pi(1)}$ with $|C_{\pi(i)}| = 1$ can be labeled. Then the (unique) color in $C_{\pi(1)}$ is labeled by A, and thus becomes color $c_{\pi(1)}$. But by the definition of $C_{\pi(1)}$, in this case *all* vertices in $V_{\pi(1)}$ are colored by $c_{\pi(1)}$. Statement (b) is trivial for $s = 1$.

Now assume that we have proved the statements up to step $s - 1 < a$, and consider step s. Every color in $C_{\pi(s)} - \{c_{\pi(s)}\}$ (if any) is labeled by B, and is adjacent to some color that has been labeled by A in an earlier step. Let D be the set of these adjacent colors. By the inductive assumption, the colors in D are the only possible colors (from their corresponding color sets) that can be used on the vertices in $V(T_{k-s+1})$. Every vertex v in $V(T_{k-s}) \cap V_{\pi(s)}$ is adjacent to $k - 1$ leaves in T_{k-s+1}, and therefore all the colors in D show up on these leaves. Consequently, they block all colors from $C_{\pi(s)}$ for vertex v except color $c_{\pi(s)}$. This proves statement (L1). In case color $c_{\pi(s)}$ was adjacent to some color x labeled by A in an earlier step, the above argument with $D \cup \{x\}$ instead of D yields that there is no possible color for vertex v. This proves statement (L2). □

Let L^A denote the set of colors that are labeled by A. Since every step labels exactly one color by A, $|L^A| = a$. Let L^+ denote the set of colors z for which $z-1$ is in L^A; clearly, $|L^+| \geq |L^A| - 1 = a - 1$. By statement (L2) in Lemma 5, the sets L^+ and L^A are disjoint. Moreover, there are $k - a$ color sets with unlabeled colors. Since they do not meet the condition in the if-part of the labeling step, each of them contains at least two colors that are not adjacent to any color with label A. These $2(k - a)$ colors are not contained in $L^A \cup L^+$. To summarize, we have found $|L^A| + |L^+| + 2(k - a)$ pairwise distinct colors in the range $1, \ldots, \ell$. Therefore,

$$\ell \;\geq\; |L^A| + |L^+| + 2(k-a) \;\geq\; a + (a-1) + 2(k-a) \;=\; 2k - 1.$$

Note that these arguments also go through in the extremal case $a = k$. This completes the proof of the lower bound statement in Theorem 1.

3 Path Backbones and the Chromatic Number

This section is devoted to a proof of Theorem 2. The upper bound is proved in Section 3.1 by case distinctions. The lower bound is proved in Section 3.2; this proof uses a similar idea as the proof in Section 2, but the actual arguments are quite different.

3.1 Proof of the Upper Bounds

We start with statement (c) in Theorem 2. Hence, consider a graph $G = (V, E)$ with $\chi(G) = 4t$ for some $t \geq 2$, and let $V_1, \ldots, V_{4t}$ denote the corresponding independent sets in the $4t$-coloring. Furthermore, let $P = (V, E_P)$ be a Hamiltonian path in G. Consider the following color sets:

– For $i = 1, \ldots, 3t$, we define the color set $C_i = \{2i - 1\}$.
– For $i = 1, \ldots, t$, we define the color set $C'_i = \{2i, 2t + 2i, 4t + 2i\}$.

Note that these $4t$ color sets are pairwise disjoint, and that all the used colors are from the range $1, \ldots, 6t$.

We construct a backbone coloring for (G, P) that for $i = 1, \ldots, 3t$ colors the vertices in the independent set V_i with the color in color set C_i, and that for $i = 1, \ldots, t$ colors the vertices in the independent set V_{3t+i} with one of the three colors in color set C'_i. The vertices in $V_{3t+1}, \ldots, V_{4t}$ are colored greedily and in arbitrary order: Consider some vertex v in V_{3t+i} that is to be colored with one of the colors $2i, 2t + 2i, 4t + 2i$. In the worst case, the neighbors of v along the Hamiltonian path P have already been colored by colors x and y, and thus forbid the four colors $x - 1, x + 1, y - 1, y + 1$ for vertex v. Since $t \geq 2$, the three colors in $C'_i = \{2i, 2t + 2i, 4t + 2i\}$ are pairwise at distance at least four, whereas $x - 1$, $x + 1$ and $y - 1$, $y + 1$ are at distance two. Therefore, the intersection $C'_i \cap \{x - 1, x + 1, y - 1, y + 1\}$ contains at most two elements, and C'_i contains at least one feasible color for vertex v. This completes the proof of $\mathcal{P}(4t) \leq 6t$ for all $t \geq 2$.

The cases $k = 4t + 1$, $k = 4t + 2$, $k = 4t + 3$ with $t \geq 2$ follow by simple modifications of the above argument: For $k = 4t+1$, we add the color set $C_{3t+1} = \{6t + 1\}$. For $k = 4t + 2$, we furthermore add the color set $C_{3t+2} = \{6t + 3\}$. And for $k = 4t + 3$, we furthermore add the color set $C_{3t+3} = \{6t + 5\}$. This proves $\mathcal{P}(4t + 1) \leq 6t + 1$, $\mathcal{P}(4t + 2) \leq 6t + 3$, and $\mathcal{P}(4t + 3) \leq 6t + 5$ for all $t \geq 2$, and settles the upper bounds in Theorem 2 for all $k \geq 8$.

The upper bounds in Theorem 2 for all $k \leq 4$ follow trivially from Theorem 1. For $k = 5$, we use the above argument with five color sets

$$D_1 = \{1\}, \quad D_2 = \{3\}, \quad D_3 = \{5\}, \quad D_4 = \{8\}, \quad D_5 = \{2, 6, 7\}.$$

For $k = 6$, we add a sixth color set $D_6 = \{10\}$. Finally, for $k = 7$ we use the seven color sets $D'_1 = \{1\}$, $D'_2 = \{3\}$, $D'_3 = \{5\}$, $D'_4 = \{7\}$, $D'_5 = \{9\}$, $D'_6 = \{11\}$, and $D'_7 = \{2, 6, 10\}$. These three constructions prove $\mathcal{P}(5) \leq 8$, $\mathcal{P}(6) \leq 10$, and $\mathcal{P}(7) \leq 11$. The proof of the upper bounds in Theorem 2 is complete.

3.2 Proof of the Lower Bounds

We consider a complete k-partite graph G with $k \geq 2$ that consists of k independent sets $V_1, \ldots, V_k$ that are all of cardinality $2\Pi_k$. Here Π_k denotes the number of different permutations of $1, 1, 2, 2, 3, 3, \ldots, k, k$ in which no two subsequent symbols are the same. It is routine to deduce by inclusion-exclusion that $\Pi_k = \sum_{j=0}^{k} (-1)^j \binom{k}{j} \frac{(2k-j)!}{2^{k-j}}$. The Hamiltonian path P consists of Π_k segments with $2k$ vertices each. Every such segment visits every independent set exactly twice, since we let each segment correspond to one permutation π of the $2k$ indices $1, 1, 2, 2, 3, 3, \ldots, k, k$ that contributes to the total number of Π_k defined before, and we let the segment visit the independent sets exactly in the order

$V_{\pi(1)}, V_{\pi(2)}, \ldots, V_{\pi(2k)}$. Since G is complete k-partite it is clear that these segments can be combined (in many ways) to form a Hamiltonian path in G. It is also obvious that $\chi(G) = k$.

Consider some fixed backbone coloring of (G, P) with ℓ colors. Since G is complete k-partite, any color that shows up in some set V_i cannot show up in any V_j with $j \neq i$. We denote by C_i the set of colors that are used on vertices in V_i. If $|C_i| = 1$, then V_i is called *mono-chromatic*; if $|C_i| = 2$, then V_i is *bi-chromatic*; if $|C_i| \geq 3$, then V_i is *poly-chromatic*. We denote by s_1, s_2, and s_3 the number of mono-chromatic, bi-chromatic, and poly-chromatic sets, respectively. Then clearly

$$s_1 + s_2 + s_3 \;=\; k \tag{1}$$

and

$$s_1 + 2s_2 + 3s_3 \;\leq\; \ell. \tag{2}$$

Colors that are used on mono-chromatic, bi-chromatic, poly-chromatic sets, are called mono-chromatic, bi-chromatic, poly-chromatic colors, respectively. We say that two bi-chromatic colors x, y with $1 \leq x < y \leq \ell$ are *partner* colors, if $C_i = \{x, y\}$ holds for some bi-chromatic set V_i.

Clearly, we may assume there are mono-chromatic colors. Now consider the following process that labels some of the colors in $\{1, 2, \ldots, \ell\}$ with the labels A and B, and that creates a number of arcs among the labeled colors.

> (Phase 1). All mono-chromatic colors are labeled by label A.
> (Phase 2). Repeat the following step over and over again, as long as the condition in the if-part is met:
> If there exists an unlabeled bi-chromatic color y that is adjacent to another color z that has already been labeled A at an earlier point in time, then y is labeled B and its partner color x is labeled A. Moreover, we create an arc going from z to y, and another arc going from y to x.

This process eventually terminates, since the step in the second phase can be performed at most s_2 times. We denote by a and b the number of A-labels and B-labels in the final situation after termination.

Lemma 6 *After termination, the following properties are satisfied.*

- *(T1)* $a = b + s_1$.
- *(T2) For every labeled color z, there is a unique directed path from some mono-chromatic color to z.*
- *(T3) Out of two adjacent colors z and $z+1$, at least one is not labeled A.*

Proof. Proof of (T1). After the first phase, there are exactly s_1 colors with A-labels and no vertices with B-labels. Every time the step in the second phase is performed, exactly one new label A and one new label B are created.

Proof of (T2). This is straightforward from the definition of the second phase.

Proof of (T3). Suppose for the sake of contradiction that the adjacent colors z and $z+1$ are both labeled A. By (T2), there exists a directed path from some

mono-chromatic color $x_{\phi(0)}$ to z (note that $x_{\phi(0)} = z$ might hold). This path goes through colors $x_{\phi(0)}, y_{\phi(1)}, x_{\phi(1)}, y_{\phi(2)}, x_{\phi(2)}, \ldots, y_{\phi(f)}, x_{\phi(f)}$, with $x_{\phi(f)} = z$. Every color $x_{\phi(i)}$ has an A-label, and every color $y_{\phi(i)}$ has a B-label. Every color $y_{\phi(i)}$ is adjacent to color $x_{\phi(i-1)}$. Moreover, the colors $x_{\phi(i)}$ and $y_{\phi(i)}$ are used on the independent set $V_{\phi(i)}$. By similar considerations, we find a directed path from some mono-chromatic color $x_{\psi(0)}$ to $z+1$ that goes through colors $x_{\psi(0)}, y_{\psi(1)}, x_{\psi(1)}, \ldots, y_{\psi(g)}, x_{\psi(g)}$, with $x_{\psi(g)} = z+1$. Every color $x_{\psi(i)}$ has an A-label, and every color $y_{\psi(i)}$ has a B-label. Colors $x_{\psi(i)}$ and $y_{\psi(i)}$ are used on the independent set $V_{\psi(i)}$.

Note that the colors in the directed path from $x_{\phi(0)}$ to z are pairwise distinct, and that the colors in the directed path from $x_{\psi(0)}$ to $z+1$ are pairwise distinct. By the construction of the complete k-partite graph G, there exists a subpath Q of the Hamiltonian path P that visits the independent sets in the ordering

$$V_{\phi(0)}, V_{\phi(1)}, V_{\phi(2)}, \ldots, V_{\phi(f)}, V_{\psi(g)}, V_{\psi(g-1)}, V_{\psi(g-2)}, \ldots, V_{\psi(1)}, V_{\psi(0)}.$$

Let $v_{\phi(i)}$ and $v'_{\psi(j)}$ be the corresponding vertices on Q. What are the possible colors for these vertices in the backbone coloring under investigation? Vertex $v_{\phi(0)}$ is in a mono-chromatic set, and so it must get color $x_{\phi(0)}$. Vertex $v_{\phi(1)}$ is in a bi-chromatic set, and can be colored by color $x_{\phi(1)}$ or by color $y_{\phi(1)}$. However, $v_{\phi(0)}$ is adjacent to $v_{\phi(1)}$, and its color $x_{\phi(0)}$ is adjacent to $y_{\phi(1)}$. Therefore, $v_{\phi(1)}$ must be colored by $x_{\phi(1)}$. Analogous arguments show that every vertex $v_{\phi(i)}$ is colored by color $x_{\phi(i)}$, and that every vertex $v'_{\psi(i)}$ is colored by color $x_{\psi(i)}$.

Now we arrive at the desired contradiction: Vertex $v_{\phi(f)}$ is colored by color $x_{\phi(f)} = z$, vertex $v'_{\psi(g)}$ is colored by color $x_{\psi(g)} = z+1$, and hence two adjacent vertices on the backbone are colored by adjacent colors. □

Let L denote the set of colors z for which $z+1$ is labeled A after termination. If color 1 is labeled A, then $|L| = a-1$, and otherwise $|L| = a$. In any case, $|L| \geq a-1$. No color in L can be labeled A, since this would contradict property (T3) in Lemma 6. At most b of the colors in L can be labeled B. Hence, L contains at least $a-1-b = s_1 - 1$ unlabeled colors, where the equation follows from (T1). None of these $s_1 - 1$ unlabeled colors can be bi-chromatic; otherwise, there would be another possible step in the second phase. Hence, these $s_1 - 1$ unlabeled colors in L must all be poly-chromatic. Among the ℓ colors used by the backbone coloring, there are s_1 mono-chromatic ones, $2s_2$ bi-chromatic ones, and at least $s_1 - 1$ poly-chromatic ones. Therefore,

$$2s_1 + 2s_2 - 1 \;\leq\; \ell. \tag{3}$$

Adding inequality (2) to inequality (3), and subtracting three times the equation in (1) yields

$$3k + s_2 - 1 \;\leq\; 2\ell. \tag{4}$$

Since s_2 is non-negative, (4) implies that $\ell \geq \lceil (3k-1)/2 \rceil$. For the three cases (c) $k = 4t$, (d) $k = 4t+1$, (e) $k = 4t+2$ in Theorem 2 this already implies the claimed lower bounds (c) $\ell \geq 6t$, (d) $\ell \geq 6t+1$, and (e) $\ell \geq 6t+3$, respectively. The case (f) $k = 4t+3$ can be handled as follows: If $s_1 + s_2 \geq 3t+3$, then (3)

implies $\ell \geq 6t+5$. If $s_1 + s_2 \leq 3t+2$, then subtracting three times (1) from (2) yields

$$\ell - 3k \;\geq\; -2s_1 - s_2 \;\geq\; -2(s_1+s_2) \;\geq\; -6t-4,$$

and hence $\ell \geq 6t+5$ as desired in statement (f).

3.3 The Lower Bounds for the Small Cases

It remains to settle the 'small' cases $k \leq 6$ in statements (a) and (b) of Theorem 2. The cases $k=1$ and $k=2$ are trivial.

Proof of the Case k=3. Suppose that for the case $k=3$ there is a backbone coloring of (G,T) with $\ell \leq 4$ colors. Then the equations and inequalities (1)–(4) do not have any solution s_1, s_2, s_3 over the non-negative integers. This settles the case $k=3$.

Proof of the Case k=4. Suppose that for the case $k=4$ there is a backbone coloring of (G,T) with $\ell \leq 6$ colors. Then the equations and inequalities (1)–(4) have $s_1=3$, $s_2=0$, $s_3=1$ as unique solution over the non-negative integers. Up to symmetric cases Lemma 6.(T3) only allows $C_1=\{1\}$, $C_2=\{3\}$, $C_3=\{5\}$ and $C_1=\{1\}$, $C_2=\{3\}$, $C_3=\{6\}$ as mono-chromatic color sets. In the first case $C_4=\{2,4,6\}$ and in the second case $C_4=\{2,4,5\}$. There exists a vertex $v \in V_4$ that is adjacent to vertices from C_2 and from C_3 on the Hamiltonian path P. In either case, there is no feasible color for this vertex v, and we arrive at the desired contradiction.

Proof of the Case k=5. Suppose for the sake of contradiction that for the case $k=5$ there is a backbone coloring of (G,T) with $\ell \leq 7$ colors. Then the equations and inequalities (1)–(4) have $s_1=4$, $s_2=0$, $s_3=1$ as unique solution over the non-negative integers. By Lemma 6.(T3), the only possible mono-chromatic color sets are $C_1=\{1\}$, $C_2=\{3\}$, $C_3=\{5\}$, $C_4=\{7\}$. Hence, the poly-chromatic color set must be $C_5=\{2,4,6\}$. But there exists a vertex $v \in V_5$ that is adjacent to vertices from C_2 and from C_3 on the Hamiltonian path P. Hence, there is no feasible color for v and we arrive at the desired contradiction.

Proof of the Case k=6. Suppose that for the case $k=6$ there is a backbone coloring of (G,T) with $\ell \leq 9$ colors. Then the equations and inequalities (1)–(4) have only two solutions over the non-negative integers: $s_1=5$, $s_2=0$, $s_3=1$, or $s_1=4$, $s_2=1$, $s_3=1$. Using Lemma 6.(T3), the first solution yields only one possibility for the mono-chromatic color sets, with colors 1,3,5,7,9, respectively. Since there exists a vertex v in the poly-chromatic set that is adjacent to vertices with colors 3 and 7 in P, there is no feasible color for v. We continue with the second solution. Suppose the colors c_1, c_2, c_3 and c_4 for the mono-chromatic color sets C_1, C_2, C_3, C_4 are chosen in increasing order, and let C_5 and C_6 denote the bi-chromatic and poly-chromatic color set, respectively. For a vertex $v_5 \in V_5$ and a vertex $v_6 \in V_6$ that are adjacent to vertices with colors c_2 and c_4 on P, we have

no feasible color within the set $\{c_1, c_2 - 1, c_2, c_2 + 1, c_3, c_4 - 1, c_4\}$ of different colors, and we obtain an extra forbidden color if $c_4 \neq 9$. We conclude that $c_4 = 9$, and by symmetry (using c_3 and c_1) that $c_1 = 1$. If $c_3 \neq c_2+2$, then by considering two vertices from V_5 and V_6 that are adjacent to vertices with colors c_2 and c_3 on P, we obtain the eight forbidden colors $1, c_2 - 1, c_2, c_2 + 1, c_3 - 1, c_3, c_3 + 1$, and 9, so we cannot color both of these vertices. Hence, $c_3 = c_2 + 2$. There remain two possibilities, up to symmetry: $c_2 = 3$ (or 5) or $c_2 = 4$.

If $c_2 = 4$, we have mono-chromatic colors 1,4,6,9; we obtain a contradiction in the following way: considering vertices $v_5 \in V_5$ and $v_6 \in V_6$ adjacent to vertices with colors 1 and 6 in P, we deduce that colors 3 and 8 are not in the same set; similarly with colors 4 and 6, we deduce that colors 2 and 8 are in different sets; finally with colors 6 and 9, we obtain that colors 2 and 3 are in different sets, which is absurd.

We are left with the case that $c_2 = 3$, and with mono-chromatic colors 1,3,5,9. Using colors 3 and 5 as in the previous case, we conclude that colors 7 and 8 cannot be in the same set (V_5 or V_6); using colors 3 and 9, the same holds for colors 6 and 7; using colors 5 and 9, the same holds for colors 2 and 7. The only possibility is a bi-chromatic set $C_5 = \{4, 7\}$ and a poly-chromatic set $C_6 = \{2, 6, 8\}$. Now consider a subpath Q of P on four vertices visiting the sets in the order V_2, V_5, V_6, V_2. Since V_2 has color 3, the only possible color on Q in V_5 is 7, and we cannot find a feasible color on Q in V_6, our final contradiction.

4 The Complexity Results

We only prove the negative results in statement (b) of Theorem 4. The arguments for the positive results in statement (a) can be found in the full version of this extended abstract. The NP-hardness reduction is done from the NP-complete classical ℓ-coloring problem (see Garey & Johnson [10] for more information): Given a graph $H = (V_H, E_H)$, does there exist a proper ℓ-coloring of H?

Let $H = (V_H, E_H)$ be an instance of ℓ-coloring, and let $v_1, v_2, \ldots, v_n$ be an enumeration of the vertices in V_H. We create $3(n-1)$ new vertices a_i, b_i, c_i with $1 \leq i \leq n-1$. For every $i = 1, \ldots, n-1$ we introduce the new edges $v_i a_i$, $a_i b_i$, $b_i c_i$, and $c_i v_{i+1}$. The graph that results from adding these $3(n-1)$ new vertices and these $4(n-1)$ new edges to H is denoted by G. The vertices $v_1, a_1, b_1, c_1, v_2, \ldots, c_{n-1}, v_n$ form a Hamiltonian path P in G. We claim that $\chi(H) \leq \ell$ if and only if $\text{BBC}(G, P) \leq \ell$.

Indeed, assume that $\text{BBC}(G, P) \leq \ell$ and consider such a backbone ℓ-coloring. Then the restriction to the vertices in V_H yields a proper ℓ-coloring of H. Next assume that $\chi(H) \leq \ell$, and consider a proper ℓ-coloring $f : V_H \to \{1, , \ldots, \ell\}$. We extend f to a backbone ℓ-coloring of (G, P): Every vertex b_i receives color 3. If vertex $f(v_i) \leq 3$ then a_i is colored ℓ, and otherwise it is colored 1. If vertex $f(v_{i+1}) \leq 3$, then c_i is colored ℓ, and otherwise it is colored 1. This completes the proof of Theorem 4.

Acknowledgement. This research has been sponsored by NWO-grant 047.008.006, and by EC contract IST-1999-14186: Project ALCOM-FT.

References

1. G. Agnarsson and M. M. Halldórsson, *Coloring powers of planar graphs*. Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (San Francisco) (2000) 654–662.
2. H.L. Bodlaender, T. Kloks, R.B. Tan, and J. van Leeuwen, *λ-coloring of graphs*, in Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS'2000), Springer LNCS 1770, (2000) 395–406.
3. O.V. Borodin, H.J. Broersma, A. Glebov, and J. van den Heuvel, *Stars and bunches in planar graphs. Part I: Triangulations*. Preprint (2001).
4. O.V. Borodin, H.J. Broersma, A. Glebov, and J. van den Heuvel, *Stars and bunches in planar graphs. Part II: General planar graphs and colourings*. Preprint (2001).
5. G.J. Chang and D. Kuo, *The $L(2,1)$-labeling problem on graphs*, SIAM J. Discrete Math. 9 (1996) 309–316.
6. J. Fiala, A.V. Fishkin, and F.V. Fomin, *Off-line and on-line distance constrained labeling of graphs*, in Proceedings of the 9th European Symposium on Algorithms (ESA'2001), Springer LNCS 2161 (2001) 464–475.
7. J. Fiala, T. Kloks, and J. Kratochvíl, *Fixed-parameter complexity of λ-labelings*, Discrete Appl. Math. 113 (2001) 59–72.
8. J. Fiala, J. Kratochvíl, and A. Proskurowski, *Distance constrained labelings of precolored trees*, in Proceedings of the 7th Italian Conference on Theoretical Computer Science (ICTCS'2001), Springer LNCS 2202 (2001) 285–292.
9. D.A. Fotakis, S.E. Nikoletseas, V.G. Papadopoulou and P.G. Spirakis, *Hardness results and efficient approximations for frequency assignment problems and the radio coloring problem*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS 75 (2001) 152–180.
10. M.R. Garey and D.S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York (1979).
11. M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York (1980).
12. J.R. Griggs and R.K. Yeh, *Labelling graphs with a condition at distance* 2, SIAM J. Discrete Math. 5 (1992) 586–595.
13. W.K. Hale, *Frequency assignment: Theory and applications*, Proceedings of the IEEE 68 (1980) 1497–1514.
14. P.L. Hammer and S. Földes, *Split graphs*, Congressus Numerantium 19 (1977) 311–315.
15. J. van den Heuvel, R.A. Leese, and M.A. Shepherd, *Graph labeling and radio channel assignment*, J. Graph Theory 29 (1998) 263–283.
16. J. van den Heuvel and S. McGuinness, *Colouring the square of a planar graph*. Preprint (1999).
17. T.K. Jonas, *Graph coloring analogues with a condition at distance two: $L(2,1)$-labellings and list λ-labellings*. Ph.D. Thesis, University of South Carolina (1993).
18. R.A. Leese, *Radio spectrum: a raw material for the telecommunications industry*, in Progress in Industrial Mathematics at ECMI 98, Teubner, Stuttgart (1999) 382–396.
19. M. Molloy and M.R. Salavatipour, *A bound on the chromatic number of the square of a planar graph*. Preprint (2001).
20. G. Wegner, *Graphs with given diameter and a colouring problem*. Preprint, University of Dortmund (1977).

Greedy Edge-Disjoint Paths in Complete Graphs*

Paz Carmi[1], Thomas Erlebach[2], and Yoshio Okamoto[3]

[1] Department of Computer Science, Ben-Gurion University of the Negev, Israel.
carmip@cs.bgu.ac.il

[2] Computer Engineering and Networks Laboratory, Department of Information Technology and Electrical Engineering, ETH Zürich, Switzerland.
erlebach@tik.ee.ethz.ch

[3] Institute of Theoretical Computer Science, Department of Computer Science, ETH Zürich, Switzerland.
okamotoy@inf.ethz.ch

Abstract. The maximum edge-disjoint paths problem (MEDP) is one of the most classical NP-hard problems. We study the approximation ratio of a simple and practical approximation algorithm, the shortest-path-first greedy algorithm (SGA), for MEDP in complete graphs. Previously, it was known that this ratio is at most 54. Adapting results by Kolman and Scheideler [Proceedings of SODA, 2002, pp. 184–193], we show that SGA achieves approximation ratio $8F + 1$ for MEDP in undirected graphs with flow number F and, therefore, has approximation ratio at most 9 in complete graphs. Furthermore, we construct a family of instances that shows that SGA cannot be better than a 3-approximation algorithm. Our upper and lower bounds hold also for the bounded-length greedy algorithm, a simple on-line algorithm for MEDP.

Keywords: Approximation algorithm, Greedy algorithm, Shortening lemma.

1 Introduction

The maximum edge-disjoint paths problem (MEDP) is one of the most classical NP-hard problems [5]. Several real-world problems concerning the efficient operation of communication networks lead to problems that can be modeled as MEDP.

An instance of MEDP is given by a pair $(G, \mathcal{R})$ of an undirected connected graph $G = (V, E)$ and a multiset $\mathcal{R}$ of unordered pairs of vertices of V. Each

* This work was supported by the Berlin-Zürich Joint Graduate Program "Combinatorics, Geometry, and Computation" (CGC), financed by ETH Zürich and the German Science Foundation (DFG), by the Swiss National Science Foundation under Contract No. 2100-63563.00 (AAPCN), and by EU Thematic Network APPOL II, IST-2001-32007, with funding by the Swiss Federal Office for Education and Science (BBW). A longer version of this paper is available as technical report [1].

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 143–155, 2003.

Table 1. Results about the approximation ratio for MEDP in complete graphs.

approximation ratio	algorithm	contributors [reference]
27	Tripartition	Erlebach and Vukadinović [4]
54	SGA	Erlebach and Vukadinović [4]
17	BGA ($L = 4$), SGA	Kolman and Scheideler [10]
9	BGA ($L = 4$), SGA	this paper

element of $\mathcal{R}$ is called a *request*. Two paths are called *edge-disjoint* if they have no common edge. We consider the problem of connecting as many requests as possible along edge-disjoint paths. A feasible solution to the instance $(G, \mathcal{R})$ of MEDP is given by a pair $(\mathcal{A}, \mathcal{P})$ of a subset $\mathcal{A}$ of $\mathcal{R}$ and a set $\mathcal{P}$ of edge-disjoint paths in G such that each request in $\mathcal{A}$ is connected by a unique path in $\mathcal{P}$. The task is to maximize the size of $\mathcal{A}$, which is denoted by $|\mathcal{A}|$. Note that $|\mathcal{A}| = |\mathcal{P}|$ in a feasible solution $(\mathcal{A}, \mathcal{P})$. A request is called *accepted* if it belongs to $\mathcal{A}$.

Since it is known that MEDP is NP-hard even if input graphs are restricted to complete graphs [4], we are interested in finding approximation algorithms for MEDP. An algorithm for MEDP is called a *ρ-approximation algorithm* if it runs in polynomial time and always outputs a feasible solution $(\mathcal{A}, \mathcal{P})$ for any instance $(G, \mathcal{R})$ which satisfies $\mathrm{Opt} \leq \rho|\mathcal{A}|$, where Opt is the number of the accepted requests in an optimal solution to $(G, \mathcal{R})$. The value ρ is also called *approximation ratio*. We refer to [8] for an introduction and more background information about edge-disjoint paths problems and to [2] and the references given there for recent results about the approximability of MEDP in general undirected and directed graphs.

In this paper, we are mainly interested in MEDP in complete graphs. Table 1 shows the known and new results about the approximation ratio for MEDP in complete graphs. The first approximation algorithm with a constant approximation ratio was given by Erlebach and Vukadinović [4]. In this algorithm, we first divide the vertex set V into three parts V_1, V_2, V_3, and also divide the requests $\mathcal{R}$ into three parts $\mathcal{R}_{12}, \mathcal{R}_{23}, \mathcal{R}_{31}$ such that $\mathcal{R}_{ij}$ consists of those requests with both ends in V_i or one end in V_i and the other in V_j. Then, we solve the problem in which the requests are restricted to $\mathcal{R}_{ij}$ by a certain procedure independently and take the best solution. We call this algorithm the *tripartition algorithm*, but we do not precisely describe this algorithm here. It is important to say that they showed that this algorithm is a 27-approximation algorithm.

Greedy algorithms appear to be the simplest algorithms for MEDP. The first greedy algorithm that we consider is called the *bounded-length greedy algorithm* (BGA). It takes a parameter L and accepts a request only if it can be routed along a path of length at most L. (The *length* of a path is defined as the number of edges on the path.) More formally, the algorithm processes the requests in given order and greedily accepts each request if it can be routed along a path of length at most L that is edge-disjoint from the paths of all previously accepted requests. Otherwise, it rejects the request. BGA was first introduced by Kleinberg [8]. Note

that BGA can process the requests in arbitrary order and is therefore an on-line algorithm, i.e., it can process each request without knowledge about future requests.

Kolman and Scheideler [10] invented a new network parameter F, called the flow number, and they showed that BGA with parameter $L = 4F$ is a $(16F+1)$-approximation algorithm for the unsplittable flow problem (a generalization of MEDP that will be explained in Section 2.1). Indeed, they discovered a "shortening lemma" and used it nicely to obtain this bound. We will see that complete graphs have flow number one. Thus their result implies a 17-approximation algorithm.

In this paper, we adapt their analysis to MEDP and prove, using the shortening lemma, that BGA with parameter $L = 4F$ is actually an $(8F + 1)$-approximation algorithm for undirected graphs with flow number F. This shows that BGA with $L = 4$ is a 9-approximation algorithm for complete graphs.

Another variant of greedy algorithms is the *shortest-path-first greedy algorithm* (SGA). This algorithm starts with $\mathcal{A} = \emptyset$ and repeats the following step until there is no request left in $\mathcal{R}$ that can be connected along a path in G: Let $\{s_i, t_i\}$ be a request in $\mathcal{R}$ such that a shortest path π_i from s_i to t_i in G has minimum length among all the requests in $\mathcal{R}$ (ties can be broken arbitrarily); accept $\{s_i, t_i\}$ (i.e., assign $\mathcal{A} \leftarrow \mathcal{A} \cup \{\{s_i, t_i\}\}$), assign it the path π_i, remove $\{s_i, t_i\}$ from $\mathcal{R}$, and delete all edges of π_i from G. In other words, the algorithm greedily accepts the request that can be connected along a shortest possible path that is disjoint to the paths of previously accepted requests. This algorithm was first given by Kolliopoulos and Stein [9]. Note that SGA is not an on-line algorithm, as it considers all remaining requests in each step.

It is clear that the approximation ratio of SGA is at least as good as that of BGA. To see this, consider an arbitrary instance of MEDP and let BGA process the given requests in the following order: first all requests accepted by SGA (in the same order as SGA accepts them), and then all requests rejected by SGA (in arbitrary order). BGA will accept all requests that were accepted by SGA along paths of length at most L, and no other paths. (We can assume that BGA routes the accepted requests along the same paths as SGA.) Thus, the solution produced by BGA cannot have a larger value than that of SGA if the requests are processed in this order.

Therefore, all upper bounds on the approximation ratio of BGA hold also for SGA. In particular, we obtain that SGA is a 9-approximation algorithm for complete graphs. It is worthwhile to say here that the best previous bound on the approximation ratio of SGA for complete graphs was 54, given by Erlebach and Vukadinović [4].

We obtain another bound in terms of the maximum multiplicity (the maximum number of copies of the same request in the given instance), denoted by $\mu_{\max}$: we show that $\mu_{\max}$ is also an upper bound on the approximation ratio of SGA. Therefore, we get the following theorem.

Theorem 1.1. *The bounded-length greedy algorithm with $L = 4$ is a 9-approximation algorithm for the maximum edge-disjoint paths problem in complete*

graphs. The shortest-path-first greedy algorithm is a $\min\{9, \mu_{\max}\}$*-approximation algorithm.*

Finally, we consider lower bounds on the approximation ratio of BGA and SGA. We will construct a family of instances of MEDP in complete graphs that implies that SGA cannot be better than a 3-approximation algorithm. Of course, this lower bound applies to BGA as well.

The organization of this paper is as follows. We will prove the upper bounds (Theorem 1.1) in Section 2; we will construct some instances which give lower bounds in Section 3; some additional remarks will be stated in the last section.

Notation. The set of reals is denoted by $\mathbb{R}$, and the set of nonnegative reals is denoted by $\mathbb{R}_+$. A path p is sometimes denoted by $(v_1 — v_2 — \cdots — v_l)$, where $v_1, v_2, \ldots, v_l$ are the consecutive vertices along the path p. The length of the path p, i.e. the number of edges in p, is denoted by $\text{length}(p)$. For two paths p and q, $p \cap q$ represents the set of the edges which p and q have in common. So $p \cap q = \emptyset$ means that p and q are edge-disjoint.

2 Upper Bounds

In this section, we will give a better upper bound on the approximation ratio of BGA and SGA for MEDP in complete graphs. In the first subsection, we will review the framework of Kolman and Scheideler [10], in particular the flow number and the shortening lemma, and will see how their result implies a better bound. This yields that BGA with parameter $L = 4$ is a 17-approximation algorithm for MEDP in complete graphs. In the second subsection, we will consider a further improvement. We will show that BGA with $L = 4F$ is an $(8F+1)$-approximation algorithm for MEDP in graphs with flow number F and thus a 9-approximation algorithm for MEDP in complete graphs. While all bounds for BGA hold also for SGA, we give another bound for SGA depending on the maximum multiplicity of the requests. This gives the proof of the main theorem (Theorem 1.1).

2.1 Flow Number of Complete Graphs

In a recent work by Kolman and Scheideler [10], they studied the unsplittable flow problem (UFP) and gave an improved approximation ratio using the flow number of a network and the shortening lemma. First, we are going to review their results.

UFP is a generalization of MEDP. We are given an undirected graph $G = (V, E)$ and a function $u : E \to \mathbb{R}_+$. The number $u(e)$ associated with an edge $e \in E$ is called the *capacity* of e. We are also given a multiset $\mathcal{R}$ of requests as in MEDP. Moreover, we are given two functions $d : \mathcal{R} \to \mathbb{R}_+$ and $r : \mathcal{R} \to \mathbb{R}_+$, where $d(i)$ and $r(i)$ associated with a request $i \in \mathcal{R}$ are called the *demand* and the *profit* of i, respectively. To summarize, an instance of UFP is given by a 5-tuple $(G, \mathcal{R}, u, d, r)$ consisting of an undirected graph G, a multiset $\mathcal{R}$ of requests, the capacity u, the demand d and the profit r. Then a feasible solution

to the instance $(G, \mathcal{R}, u, d, r)$ of UFP is given by a pair $(\mathcal{A}, \mathcal{P})$ of a subset $\mathcal{A}$ of $\mathcal{R}$ and a set $\mathcal{P}$ of paths (not necessarily edge-disjoint) in G such that each request in $\mathcal{A}$ is connected by a unique path in $\mathcal{P}$ and for each edge $e \in E$ the sum of the demands of the requests connected by the paths going through e does not exceed the capacity $u(e)$ of the edge e. A request is called *accepted* if it belongs to $\mathcal{A}$. The task is to maximize the total profit of the accepted requests. MEDP is a special case of UFP. To appreciate this, we only have to set $u(e) = 1$ for all $e \in E$ and $d(i) = r(i) = 1$ for all $i \in \mathcal{R}$.

UFP is also an NP-hard problem, since it contains MEDP as a special case. Let us define what an approximation algorithm for UFP is. (In the previous section, we just defined approximation algorithms for MEDP.) An algorithm for UFP is called a *ρ-approximation algorithm* if it runs in polynomial time and always outputs a feasible solution $(\mathcal{A}, \mathcal{P})$ for any instance $(G, \mathcal{R}, u, d, r)$ which satisfies $\mathrm{Opt} \le \rho \sum_{i \in \mathcal{A}} r(i)$, where Opt is the total profit of the accepted requests in an optimal solution. We say that ρ is an *approximation ratio*, similarly to the case of MEDP. Note that these definitions are consistent with those for MEDP.

We can formulate UFP as a $\{0,1\}$-integer programming problem, as usual in combinatorial optimization. To do that, we introduce two kinds of variables. One is $x \in \{0,1\}^{\mathcal{R}}$, which means that $x_i = 1$ if the request i is accepted and $x_i = 0$ if i is not accepted. The other one is $y^{(i)} \in \{0,1\}^{\mathcal{P}_i}$, where $i \in \mathcal{R}$ and $\mathcal{P}_i$ is the set of all paths in G that connect the request i. The variable $y^{(i)}$ represents that $y_\pi^{(i)} = 1$ if the request i is accepted through the path $\pi \in \mathcal{P}_i$ and $y_\pi^{(i)} = 0$ otherwise.

Here is a formulation of UFP via integer programming:

$$
\begin{aligned}
\text{(IP-UFP):}\quad \text{maximize} \quad & \sum_{i \in \mathcal{R}} r(i) x_i & & \\
\text{subject to} \quad & \sum_{\substack{i \in \mathcal{R}, \pi \in \mathcal{P}_i \\ \text{s.t. } e \in \pi}} d(i) y_\pi^{(i)} \le u(e) & \text{for all } e \in E, & \quad (1)\\
& \sum_{\pi \in \mathcal{P}_i} y_\pi^{(i)} = x_i & \text{for all } i \in \mathcal{R}, & \quad (2)\\
& x_i \in \{0,1\} & \text{for all } i \in \mathcal{R}, & \quad (3)\\
& y_\pi^{(i)} \in \{0,1\} & \text{for all } i \in \mathcal{R} \text{ and } \pi \in \mathcal{P}_i. & \quad (4)
\end{aligned}
$$

The formulation (IP-UFP) is easy to understand but has exponentially many variables. However, we can indeed obtain a formulation of polynomial size as in the network flow problem (using edge variables instead of path variables). See [7], for example.

Now we relax the $\{0,1\}$-constraints (3) and (4), that is, we replace them with $x_i \in [0,1]$ and $y_\pi^{(i)} \in [0,1]$. Then, we obtain a linear programming problem, called the *multicommodity flow problem* and denoted by (LP-UFP).

There is a variant of the multicommodity flow problem, called the concurrent multicommodity flow problem. The *concurrent multicommodity flow problem* is defined as follows:

$$
\begin{array}{lll}
\text{(ConMFP):} & \text{maximize} & x_1 \\
& \text{subject to} & \displaystyle\sum_{\substack{i\in\mathcal{R},\pi\in\mathcal{P}_i \\ \text{s.t. } e\in\pi}} d(i)y_\pi^{(i)} \le u(e) \quad \text{for all } e \in E, \\
& & \displaystyle\sum_{\pi\in\mathcal{P}_i} y_\pi^{(i)} = x_i \quad \text{for all } i \in \mathcal{R}, \\
& & x_i = x_j \quad \text{for all } i, j \in \mathcal{R}, \qquad (5) \\
& & 0 \le x_i \le 1 \quad \text{for all } i \in \mathcal{R}, \\
& & 0 \le y_\pi^{(i)} \le 1 \quad \text{for all } i \in \mathcal{R} \text{ and } \pi \in \mathcal{P}_i.
\end{array}
$$

In a feasible solution to the concurrent multicommodity flow problem (ConMFP), all x_i's are forced to be the same by the constraint (5). Then, this problem represents the situation that if we are forced to accept the same fraction of all the requests, we want to know how large the fraction can be. Note that any feasible solution to the concurrent multicommodity flow problem (ConMFP) is also a feasible solution to the multicommodity flow problem (LP-UFP), but the converse is not always true.

Now we define the flow number of a network as in [10]. (Here, a *network* is a pair (G, u) of an undirected graph $G = (V, E)$ and a capacity function $u : E \to \mathbb{R}_+$.) We are given a network (G, u). Consider an instance $I(G, u)$ of the concurrent multicommodity flow problem that is constructed as follows. The set $\mathcal{R}$ of requests consists of all the unordered pairs of the vertices, i.e., $\mathcal{R} = \{\{s, t\} : s, t \in V,\ s \ne t\}$. The demand $d : \mathcal{R} \to \mathbb{R}_+$ is defined as $d(\{s, t\}) := (\sum_{\{s,v\}\in E} u(\{s, v\}) \sum_{\{t,v\}\in E} u(\{t, v\}))/(2\sum_{e\in E} u(e))$. So we have an instance $I(G, u) := (G, \mathcal{R}, u, d)$ of the concurrent multicommodity flow problem.

For a feasible solution $\xi = (x, (y^{(i)} : i \in \mathcal{R}))$ to an instance $(G, \mathcal{R}, u, d)$ of the concurrent multicommodity flow problem, we define two parameters: the dilation and the congestion. The *dilation* $D(\xi)$ of ξ is the length of a longest flow path in ξ, i.e., $D(\xi) := \max\left\{\text{length}(\pi) : y_\pi^{(i)} > 0\right\}$. The *congestion* $C(\xi)$ of ξ is the inverse of the objective value of ξ, i.e., $C(\xi) := \frac{1}{x_1}$. Finally, the *flow number* $F(G, u)$ of the network (G, u) is defined as

$$F(G, u) := \min\{\max\{D(\xi), C(\xi)\} : \xi \text{ is a feasible solution to } I(G, u)\}.$$

Now it is easy to determine the flow number of a complete graph with unit capacity.

Lemma 2.1. *Let $G = (V, E)$ be a complete graph and $u : E \to \mathbb{R}_+$ be given as $u(e) = 1$ for every $e \in E$. Then $F(G, u) = 1$.*

Proof. First, it is clear that $F(G, u) \ge 1$. To see that $F(G, u) \le 1$, note that $I(G, u)$ consists of a request $\{s, t\}$ with demand $d(\{s, t\}) = (n - 1)/n$ for every

edge $\{s,t\} \in E$. This means that we can route the whole demand of every request along its direct edge, resulting in a solution with dilation 1 and congestion 1. □

Invoking the flow number, Kolman and Scheideler [10] derived several interesting statements. One is the so-called "shortening lemma."

Lemma 2.2 (shortening lemma [10]). *Let an instance I of the concurrent multicommodity flow problem in a graph with flow number F be given. Then for any $\epsilon \in (0,1]$ and any feasible solution to I with objective value f, there exists a feasible solution ξ to I such that the objective value is $f/(1+\epsilon)$ and $D(\xi) \leq 2(1+1/\epsilon)F$.*

From Lemma 2.2, we can conclude the following corollary.

Corollary 2.1. *If we are given a feasible solution to an instance of UFP in which the profit is equal to the demand, then we can transform this solution into a feasible solution to the corresponding instance of the multicommodity flow problem such that it uses only paths of length at most $4F$ but the objective value is half as much as the original one.*

Proof. Given an instance $I = (G, \mathcal{R}, u, d, d)$ of UFP, let $(x, (y^{(i)} : i \in \mathcal{R}))$ be a feasible solution to I. Then we set $\overline{\mathcal{R}} := \{i \in \mathcal{R} : x_i = 1\}$, and consider the instance $\overline{I} = (G, \overline{\mathcal{R}}, u, d)$ of the concurrent multicommodity flow problem. Here if we set $\overline{x}_i = x_i$ for all $i \in \overline{\mathcal{R}}$ and $\overline{y}_\pi^{(i)} = y_\pi^{(i)}$ for all $i \in \overline{\mathcal{R}}$ and $\pi \in \mathcal{P}_i$, then $(\overline{x}, (\overline{y}^{(i)} : i \in \overline{\mathcal{R}}))$ is a feasible solution to $\overline{I}$ and the objective value is 1. Now we can apply the shortening lemma with $\epsilon = 1$. Then we get a feasible solution to $\overline{I}$ such that the objective value is $1/2$ and the dilation is at most $4F$. This also forms a feasible solution to the instance of the multicommodity flow problem corresponding to the original instance I of UFP, and it shows that the objective value is half as large as the original solution and the dilation is at most $4F$. □

Using this corollary, Kolman and Scheideler [10] showed that BGA with $L = 4F$ is a $(16F+1)$-approximation algorithm for UFP in networks with flow number F, provided that the profit of each request is equal to its demand, the maximum demand is at most the minimum capacity, and the algorithm processes the requests in order of non-increasing profits. Since these conditions are satisfied for MEDP and the flow number of complete graphs with unit capacity is one by Lemma 2.1, we immediately obtain that BGA with $L = 4$ (and thus also SGA) is a 17-approximation algorithm for MEDP in complete graphs. In the next subsection, we derive a better bound.

2.2 A 9-Approximation Algorithm

We will show that SGA is a 9-approximation algorithm for MEDP in complete graphs. Notice again that MEDP is a special case of UFP, setting $u(e) = 1$ for all $e \in E$ and $d(i) = r(i) = 1$ for all $i \in \mathcal{R}$. First, we will refine the analysis by Kolman and Scheideler for UFP [10] to obtain a better bound for MEDP.

Theorem 2.1. *The bounded-length greedy algorithm with parameter $L = 4F$ (and thus also the shortest-path-first greedy algorithm) is an $(8F+1)$-approximation algorithm for the maximum edge-disjoint paths problem in undirected graphs with flow number F.*

Proof. Let $\mathcal{A}$ and $\mathcal{P}(\mathcal{A})$ be the set of accepted requests and the set of paths obtained by BGA, respectively. Let further $\mathcal{O}$ be the set of accepted requests in an optimal solution. By Corollary 2.1, there exists a feasible solution $\xi = (x, (y^{(i)} : i \in \mathcal{R}))$ of the corresponding multicommodity flow problem such that $D(\xi) \leq 4F$ and the objective value for ξ is $|\mathcal{O}|/2$. Notice that the proof of Corollary 2.1 shows that $\sum_{\pi \in \mathcal{P}_i} y_\pi^{(i)} \leq 1/2$ holds for all $i \in \mathcal{R}$.

Let $\mathcal{P}(\xi)$ be the set of the paths π satisfying $y_\pi^{(i)} > 0$. Take any path $p \in \mathcal{P}(\mathcal{A})$. Note that $\text{length}(p) \leq 4F$ by definition of BGA. Then we have

$$\sum_{\substack{\pi \in \mathcal{P}(\xi) \\ \text{s.t. } \pi \cap p \neq \emptyset}} y_\pi^{(i)} \leq \sum_{\substack{\pi \in \mathcal{P}(\xi) \\ \text{s.t. } \exists e \in \pi \cap p}} y_\pi^{(i)} \leq \sum_{e \in p} \sum_{\substack{\pi \in \mathcal{P}(\xi) \\ \text{s.t. } e \in \pi}} y_\pi^{(i)} \leq \sum_{e \in p} 1 \leq \text{length}(p) \leq 4F, \quad (6)$$

using constraint (1) of (LP-UFP).

Now we divide $\mathcal{P}(\xi)$ into two parts $\mathcal{P}_1(\xi)$ and $\mathcal{P}_2(\xi)$ as follows: $\mathcal{P}_1(\xi) = \{\pi \in \mathcal{P}(\xi) : \text{ the request connected by } \pi \text{ does not belong to } \mathcal{A}\}$ and $\mathcal{P}_2(\xi) = \mathcal{P}(\xi) \setminus \mathcal{P}_1(\xi)$.

Claim. For each path $\pi \in \mathcal{P}_1(\xi)$ there exists some path $p \in \mathcal{P}(\mathcal{A})$ such that $\pi \cap p \neq \emptyset$, i.e., π and p have a common edge.

Proof (of Claim). Consider the contrary. Then, we must have some request $r \in \mathcal{R} \setminus \mathcal{A}$ which is connected by a path of length at most $4F$ consisting of edges that no path in $\mathcal{P}(\mathcal{A})$ uses. So BGA with $L = 4F$ should have accepted this request. A contradiction. □

Now we analyze the approximation ratio. Using the claim above and (6), we get

$$\sum_{\pi \in \mathcal{P}_1(\xi)} y_\pi^{(i)} \leq \sum_{p \in \mathcal{P}(\mathcal{A})} \sum_{\substack{\pi \in \mathcal{P}(\xi) \\ \text{s.t. } \pi \cap p \neq \emptyset}} y_\pi^{(i)} \leq 4F \ |\mathcal{P}(\mathcal{A})| = 4F \ |\mathcal{A}|.$$

On the other hand, we have $\sum_{\pi \in \mathcal{P}_2(\xi)} y_\pi^{(i)} \leq \sum_{i \in \mathcal{A}} \sum_{\pi \in \mathcal{P}_i} y_\pi^{(i)} \leq \sum_{i \in \mathcal{A}} 1/2 = |\mathcal{A}|/2$. This gives

$$\text{Opt} = |\mathcal{O}| = 2 \times (\text{the objective value for } \xi) = 2 \sum_{i \in \mathcal{R}} \sum_{\pi \in \mathcal{P}_i} y_\pi^{(i)} = 2 \sum_{\pi \in \mathcal{P}(\xi)} y_\pi^{(i)}$$

$$= 2 \left(\sum_{\pi \in \mathcal{P}_1(\xi)} y_\pi^{(i)} + \sum_{\pi \in \mathcal{P}_2(\xi)} y_\pi^{(i)} \right) \leq 2(4F|\mathcal{A}| + |\mathcal{A}|/2) = (8F+1)|\mathcal{A}|.$$

Thus, we have shown that the approximation ratio is at most $8F + 1$. □

From Lemma 2.1 and Theorem 2.1, we immediately obtain the following corollary.

Corollary 2.2. *The bounded-length greedy algorithm with parameter $L = 4$ (and thus also the shortest-path-first greedy algorithm) is a 9-approximation algorithm for the maximum edge-disjoint paths problem in complete graphs.*

Now we consider the multiplicities of requests. For a request $\{s, t\} \in \mathcal{R}$, if $\mathcal{R}$ has μ copies of $\{s, t\}$, then we say the *multiplicity* of $\{s, t\}$ in $\mathcal{R}$ is μ. Define $\mu_{\max}(\mathcal{R})$ as the maximum of the multiplicities of the requests in $\mathcal{R}$. If there is no risk of confusion, we denote it simply by $\mu_{\max}$. Let $\mathcal{R}_1$ be the set of requests neglecting the multiplicities (i.e., with only one copy of each unique request in $\mathcal{R}$). The proof of the following lemma is straightforward.

Lemma 2.3. *For the maximum edge-disjoint paths problem in complete graphs with the multiset $\mathcal{R}$ of requests, the shortest-path-first greedy algorithm gives a solution $\mathcal{A}$ such that each request in $\mathcal{R}_1$ is accepted and connected by a path of length 1 (i.e. just one edge). Moreover, there exists an optimal solution with the same property.*

Then we have $|\mathcal{R}_1| \leq |\mathcal{A}|$. Moreover, $\mu_{\max}|\mathcal{R}_1|$ is an obvious upper bound on $|\mathcal{R}|$. Also, $|\mathcal{R}|$ bounds Opt. This shows $\mathrm{Opt} \leq |\mathcal{R}| \leq \mu_{\max}|\mathcal{R}_1| \leq \mu_{\max}|\mathcal{A}|$. Combining this analysis and Corollary 2.2, we obtain Theorem 1.1.

3 Lower Bounds

In the previous section, we have shown that SGA is a $\min\{9, \mu_{\max}\}$-approximation algorithm. The next question is: how tight is this analysis? If we want to construct a tight example for the bound 9, then it must have maximum multiplicity at least 9. Unfortunately, we do not know an instance that achieves the bound above. However, we can construct two instances of MEDP in complete graphs to show non-trivial lower bounds on the approximation ratio of SGA and BGA. The first one has $\mu_{\max} = 2$ and achieves the ratio 4/3. The second one has an arbitrarily large $\mu_{\max}$ and achieves the ratio $3 - \epsilon$ for arbitrarily small $\epsilon > 0$. Both lower bounds apply to SGA and therefore also to BGA.

3.1 Case of $\mu_{\max} = 2$

In this subsection, we construct an instance of MEDP in complete graphs with $\mu_{\max} = 2$ for which SGA has approximation ratio 4/3. Our example has 8 vertices and 16 requests. The optimal solution accepts all the requests, while SGA may return a set of 12 requests. Then the ratio is 4/3. Let the vertex set be $V = \{1, 2, 3, 4, 5, 6, 7, 8\}$. Then a set $\mathcal{R}$ consists of the following 16 requests: $r_1 = \{1, 3\}$, $r_2 = \{1, 3\}$, $r_3 = \{5, 3\}$, $r_4 = \{5, 3\}$, $r_5 = \{1, 7\}$, $r_6 = \{1, 7\}$, $r_7 = \{5, 7\}$, $r_8 = \{5, 7\}$, $r_9 = \{2, 4\}$, $r_{10} = \{2, 4\}$, $r_{11} = \{8, 6\}$, $r_{12} = \{8, 6\}$, $r_{13} = \{3, 8\}$, $r_{14} = \{3, 6\}$, $r_{15} = \{7, 2\}$, $r_{16} = \{7, 4\}$. Note that $\mu_{\max} = 2$.

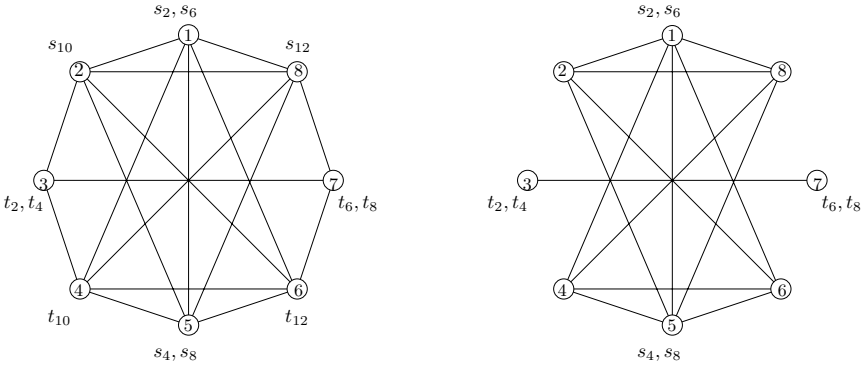


Fig. 1. Situation after removing the requests in $\mathcal{R}_1$ from $\mathcal{R}$ and their paths from K_8.

By Lemma 2.3, the output of SGA and the optimal solution accept each request in $\mathcal{R}_1$ through the corresponding edge. We have $\mathcal{R}_1 = \{r_1, r_3, r_5, r_7, r_9, r_{11}, r_{13}, r_{14}, r_{15}, r_{16}\}$. So we can remove these requests from $\mathcal{R}$ and also remove the corresponding edges from K_8. This procedure results in the situation shown in Figure 1 (left). Now we can see that an optimal solution accepts all the remaining requests. Next, consider SGA. In the situation of Figure 1 (left), all remaining requests can be connected along paths of length 2. Assume that SGA accepts r_{10} along (2 – 3 – 4) and r_{12} along (6 – 7 – 8). The requests and the edges which are left are shown in Figure 1 (right). We can see that there is no path for r_2, r_4, r_6 or r_8. So SGA accepts only 12 of the 16 requests and its approximation ratio is 4/3 on this instance.

3.2 Not Better than 3-Approximation

In this subsection, we construct a family of instances of MEDP in complete graphs for which SGA has approximation ratio $3 - \epsilon$ for arbitrarily small $\epsilon > 0$.

Consider the complete graph with $2n$ vertices for large n. Let $k \in \mathbb{N}$ be a parameter that is divisible by 3 and satisfies $3n/5 \leq k < n$. Let $V_v \cup V_a \cup V_b \cup V_c$ be the vertex set, where $V_v = \{v_1, \ldots, v_{2(n-k)}\}$, $V_a = \{a_1, \ldots, a_{2k/3}\}$, $V_b = \{b_1, \ldots, b_{2k/3}\}$ and $V_c = \{c_1, \ldots, c_{2k/3}\}$. The set of requests $\mathcal{R} = \mathcal{R}_v \cup \mathcal{R}_a \cup \mathcal{R}_b \cup \mathcal{R}_c$ is as follows: $\mathcal{R}_v$ contains $2n - i$ requests between v_i and v_{i+1} for all $i \in \{1, 3, 5, \ldots, 2(n-k)-1\}$, a total of $n^2 - k^2$ requests. $\mathcal{R}_a$ contains $n - k + 1$ requests between a_i and a_{i+1} for all $i \in \{1, 3, 5, \ldots, 2k/3 - 1\}$, a total of $k(n-k+1)/3$ requests. $\mathcal{R}_b$ and $\mathcal{R}_c$ consist of $k(n-k+1)/3$ requests each and are constructed in the same way as $\mathcal{R}_a$.

First, observe that the optimal solution can accept all requests: For every $i \in \{1, 3, 5, \ldots, 2(n-k)-1\}$, the $2n-i$ requests $\{v_i, v_{i+1}\}$ are accepted by routing one of them along the direct edge $\{v_i, v_{i+1}\}$ and routing the other $2n-i-1$ along paths of length two via each of the $2n-2k-i-1$ vertices $v_{i+2}, v_{i+3}, \ldots, v_{2(n-k)}$ and each of the $2k$ vertices in $V_a \cup V_b \cup V_c$.

One copy of each of the requests in $\mathcal{R}_a \cup \mathcal{R}_b \cup \mathcal{R}_c$ is accepted along the direct edge. The remaining copies of requests of $\mathcal{R}_a$ can be accepted along paths of length two via intermediate vertices in V_b, those of $\mathcal{R}_b$ along paths of length two via intermediate vertices in V_c, and those of $\mathcal{R}_c$ along paths of length two via intermediate vertices in V_a. Note that there are $n-k$ remaining copies of each request in $\mathcal{R}_a \cup \mathcal{R}_b \cup \mathcal{R}_c$ and $2k/3$ vertices in each of V_a, V_b, V_c. By our choice of k with $k \geq 3n/5$, we have $n-k \leq 2k/3$, and the number of available candidates for intermediate vertices is indeed sufficient.

Thus the optimal solution accepts all of the $n^2 + kn + k - 2k^2$ requests.

Now consider the output of SGA. The algorithm first accepts the requests with a shortest path of length 1. So we have n accepted requests: one between v_i and v_{i+1} for $i \in \{1, 3, \ldots, 2(n-k)-1\}$, one between a_i and a_{i+1} for $i \in \{1, 3, \ldots, 2k/3-1\}$, one between b_i and b_{i+1} for $i \in \{1, 3, \ldots, 2k/3-1\}$, and one between c_i and c_{i+1} for $i \in \{1, 3, \ldots, 2k/3-1\}$. Then the algorithm accepts requests with a shortest path of length 2. Consider the remaining $n-k$ requests between a_1 and a_2. Assume that they are accepted along paths $(a_1 \text{ — } v_i \text{ — } a_2)$ where $i \in \{1, 3, \ldots, 2(n-k)-1\}$. In the same manner, for $j \in \{3, 5, \ldots, 2k/3-1\}$, assume that the $n-k$ requests between a_j and a_{j+1} are accepted along paths $(a_j \text{ — } v_i \text{ — } a_{j+1})$ where $i \in \{1, 3, \ldots, 2(n-k)-1\}$. Thus, all of the $k(n-k+1)/3$ requests in $\mathcal{R}_a$ are accepted. Similarly, assume that all of the $2k(n-k)/3$ remaining requests in $\mathcal{R}_b \cup \mathcal{R}_c$ are accepted by using $v_1, v_3, v_5, \ldots, v_{2(n-k)-1}$ as internal vertices on paths of length 2.

All requests in $\mathcal{R}_a \cup \mathcal{R}_b \cup \mathcal{R}_c$ have been accepted now, but all edges from $v_1, v_3, v_5, \ldots, v_{2(n-k)-1}$ to vertices in $V_a \cup V_b \cup V_c$ have already been used by accepted paths. Therefore, the vertices in $V_a \cup V_b \cup V_c$ cannot be used as intermediate vertices on paths for requests in $\mathcal{R}_v$.

Consider the requests in $\mathcal{R}_v$. The remaining $2k$ requests between $v_{2(n-k)-1}$ and $v_{2(n-k)}$ can only be accepted via the vertices of $V_v \setminus \{v_{2(n-k)-1}, v_{2(n-k)}\}$. Since $|V_v \setminus \{v_{2(n-k)-1}, v_{2(n-k)}\}| = 2(n-k-1)$, SGA can only accept $2(n-k-1)$ requests out of $2k$. In total, among the $2k+1$ requests between $v_{2(n-k)-1}$ and $v_{2(n-k)}$, SGA has accepted only $2(n-k-1)+1$ requests. Next, consider the $2k+2$ requests between $v_{2(n-k)-3}$ and $v_{2(n-k)-2}$. They can be accepted only via the vertices of $V_v \setminus \{v_{2(n-k)-3}, v_{2(n-k)-2}, v_{2(n-k)-1}, v_{2(n-k)}\}$. By the same reason as before, SGA can accept only $2(n-k-2)$ requests out of $2k+2$. In total, among the $2k+3$ requests between $v_{2(n-k)-3}$ and $v_{2(n-k)-2}$, SGA accepts only $2(n-k-2)+1$ requests. Continuing in this way from high to low indices, SGA accepts only i requests among $2n-i$ requests between v_i and v_{i+1}, for $i \in \{1, 3, \ldots, 2(n-k)-1\}$. Then the algorithm terminates since all edges in the subgraph induced by V_v are exhausted. So SGA accepts only $n^2 - kn + k$ requests in total.

Let $k = \alpha n$ with $3/5 \leq \alpha < 1$. The approximation ratio is

$$\frac{n^2 + kn + k - 2k^2}{n^2 - kn + k} = \frac{n^2 + \alpha n^2 + \alpha n - 2\alpha^2 n^2}{n^2 - \alpha n^2 + \alpha n} \xrightarrow{n \to \infty} \frac{1 + \alpha - 2\alpha^2}{1 - \alpha} = 1 + 2\alpha.$$

With $\alpha = 1 - \epsilon/2$, we get $1 + 2\alpha = 3 - \epsilon$ as desired.

4 Conclusion

We have studied the approximation ratio achieved by the simple and practical algorithms SGA and BGA for MEDP in complete graphs. Previously, only an upper bound of 54 on the ratio of SGA had been shown, and no non-trivial lower bound was known for these algorithms. We have proved that BGA with $L = 4$ and SGA are 9-approximation algorithms for complete graphs and cannot be better than 3-approximation algorithms. This has substantially reduced the gap between upper and lower bounds on the approximation ratio of these algorithms. However, a gap remains and it would be interesting to discover the exact worst-case approximation ratio of these algorithms.

MEDP in complete graphs is NP-hard, but it is not known whether the problem is APX-hard. Therefore, there is still the possibility that a polynomial-time approximation scheme can be obtained. (MEDP in general undirected graphs is known to be APX-hard, even for trees of rings [3,6].) So the inapproximability of the problem should be considered as well as better approximation algorithms.

Acknowledgements. The authors are grateful to anonymous referees for comments on the earlier version.

References

1. P. Carmi, T. Erlebach and Y. Okamoto, Greedy edge-disjoint paths in complete graphs. TIK-Report 155, Computer Engineering and Networks Laboratory (TIK), ETH Zürich, February 2003.
2. C. Chekuri and S. Khanna, Edge disjoint paths revisited. In: *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2003), 2003, 628–637.
3. T. Erlebach, Approximation algorithms and complexity results for path problems in trees of rings. TIK-Report 109, Computer Engineering and Networks Laboratory (TIK), ETH Zürich, June 2001.
4. T. Erlebach and D. Vukadinović, New results for path problems in generalized stars, complete graphs, and brick wall graphs. In: Proceedings of the 13th International Symposium on Fundamentals of Computation Theory (FCT 2001), *Lecture Notes in Computer Science* **2138**, 2001, 483–494.
5. M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York–San Francisco, 1979.
6. N. Garg, V.V. Vazirani and M. Yannakakis, Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica* **18**, 1997, 3–20.
7. V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd and M. Yannakakis, Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. In: *Proceedings of the 31st Annual ACM Symposium on Theory of Computing* (STOC'99), 1999, 19–28.
8. J.M. Kleinberg, *Approximation algorithms for disjoint paths problems*. Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1996.

9. S.G. Kolliopoulos and C. Stein, Approximating disjoint-path problems using greedy algorithms and packing integer programs. In: Proceedings of the 6th Integer Programming and Combinatorial Optimization Conference IPCO VI, *Lecture Notes in Computer Science* **1412**, 1998, 153–168.
10. P. Kolman and C. Scheideler, Improved bounds for the unsplittable flow problem. In: *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2002), 2002, 184–193.

Error-Correcting Graphs for Software Watermarking

Christian Collberg[1⋆], Stephen Kobourov[1⋆⋆], Edward Carter[1⋆], and Clark Thomborson[2⋆⋆⋆]

[1] Department of Computer Science, University of Arizona
{collberg,kobourov,ecarter}@cs.arizona.edu
[2] Department of Computer Science, University of Auckland
cthombor@cs.auckland.ac.nz

Abstract. In this paper, we discuss graph-theoretic approaches to software watermarking and fingerprinting. Software watermarking is used to discourage intellectual property theft and software fingerprinting is used to trace intellectual property copyright violations. We focus on two algorithms that encode information in software through the use of graph structures. We then consider the different attack models intended to disable the watermark while not affecting the correctness or performance of the program. Finally, we present several classes of graphs that can be used for watermarking and fingerprinting and analyze their properties (resiliency, data rate, performance, and stealthiness).

1 Introduction

Watermarking and fingerprinting [2,3] are popular techniques for protecting the intellectual property of digital artifacts such as images, audio and video. A watermark is a copyright notice that is embedded into the artifact that uniquely identifies its owner. A fingerprint is similar but identifies the *customer* who purchased the copy. Watermarking an object discourages intellectual property theft, whereas fingerprinting allows us to trace the copyright violator.

Several algorithms for watermarking *software* have recently been developed [4,7,12,17]. We focus on two of these algorithms: one by Collberg and Thomborson [4] and the other by Venkatesan *et al.* [17]. Both of these algorithms encode the watermark using graphs which are then embedded into the cover program.

In the remainder of this section we formally present the software watermarking and graph encoding problems. In Section 2 we consider the algorithms of Collberg and Thomborson [4] and by Venkatesan *et al.* [17] in detail. In Section 3 we present models of how an adversary can attack a program to destroy

⋆ Partially supported by NSF grant CCR-0073483 and by AFRL contract F33615-02-C-1146.
⋆⋆ Partially supported by NSF grant ACR-0222920.
⋆⋆⋆ Partial support by NERF of New Zealand under contracts UOAX9906, UOAX0214.

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 156–167, 2003.

the watermark (Section 3), and present classes of graphs suitable for encoding watermarks (Section 4). In Section 5 we evaluate the performance of the new encoding scheme that uses reducible permutation graphs.

1.1 Software Watermarking and Fingerprinting

While watermarking and fingerprinting have different uses, they both rely on encoding a number inside a program. For convenience, in the rest of this paper we refer to both processes as watermarking. Similarly, we assume that the watermark is an integer W. A *software watermarking system* can be formally described by the functions $\mathcal{E}(P, W, k) \to P_w$, $\mathcal{R}(P_w, k) \to W$, and $\mathcal{A}(P) \to P'$. The *embedder* $\mathcal{E}$ embeds the watermark W into the (cover) program P using the secret key k, yielding a watermarked program P_w. The *recognizer* $\mathcal{R}$ extracts W from P_w, also using k as the key. The *attack function* $\mathcal{A}$ models the types of attacks that an adversary may launch against a watermarked program in order to foil recognition. For example, a common attack model includes semantics-preserving transformations such as code optimization and binary translation. Attacks can also be non-semantics-preserving, allowing the attacker to alter the meaning of the program. A good watermarking system must have the following properties:

High resiliency: A watermarking system must be *resilient* against a reasonable set of de-watermarking attacks. Resiliency refers to the ability to recognize a watermark even after the watermarked program has been attacked, i.e., $\mathcal{R}(\mathcal{A}(\mathcal{E}(P, W, k)), k) \to W$;

High data rate: The ratio of the number of bits encoded by the watermark W to the total size of the watermark should be high;

High stealth: To prevent an adversary from easily locating the watermark, P and P_w should have similar statistical properties;

High performance: The watermarking should not adversely affect the size and execution time of P_w.

Note that any software watermarking technique will exhibit a trade-off between resilience, data rate, performance, and stealth. For example, the resilience of a watermark can easily be increased by exploiting redundancy (i.e., including the mark several times in the cover program), but this will lead to reduced data rate. Similarly, a watermark such as ⌜`static int watermark=314`⌝ has high performance but low stealth.

1.2 Graph Encodings

In this paper we focus on two software watermarking algorithms that encode watermarks as graph structures. In general, such encodings make use of an encoding function e that converts the watermark W into a graph G, $e(W) \to G$, and also of a decoding function d that converts a graph into an integer, $d(G) \to W$. We call the pair (e, d) a *graph codec*. From a graph-theoretic point of view, we are looking for a class of graphs $\mathbb{G}$ and a corresponding codec $(e, d)_{\mathbb{G}}$ with the following properties:

Appropriate graph types: Graphs in $\mathbb{G}$ should be directed multigraphs with an ordering on the outgoing edges, with low max out-degree, to match real program graphs;

High resiliency: $d(G)$ should be insensitive to small perturbations of G (the result of adversarial attacks against the watermarked program) such as insertions or deletions of a constant number of nodes and/or edges. Formally: $G \in \mathbb{G}, d(G) \to W, G' \approx G \Rightarrow d(G') \to W$;

Small size: The size $|P_w| - |P|$ of the embedded watermark should be small;

Efficient codecs: $(e, d)_{\mathbb{G}}$ should be polynomial time efficient.

2 Graph-Based Watermarking Algorithms

Software watermarking algorithms fall in two broad categories, *static* and *dynamic*. A static algorithm recognizes the watermark by examining the (source or compiled) code of the watermarked program. A dynamic algorithm recognizes the watermark by examining the state of the program after it has been executed with a special finite input sequence $i_1, i_2, \ldots, i_n$. Thus, for a dynamic algorithm the recognizer will have the signature $\mathcal{R}(P_w[\![i_1, i_2, \ldots, i_n]\!]) \to W$, where $P[\![I]\!]$ is the state of program P after input I.

2.1 The GTW Algorithm

Static watermarking algorithms typically embed the watermark by permuting segments of code, statements, basic blocks, or arms of switch-statements [7], or by altering instruction frequencies [14], or by inserting a code segment which has no semantic effect. The Graph-Theoretic Watermark (GTW) algorithm of Venkatesan *et al* [17] falls in the last category. The basic embedding technique is as follows:

1. encode the integer watermark value W as a control-flow graph (CFG) G,
2. combine G and the program CFG P to construct P_w,
3. connect G and P by adding bogus control-flow edges,
4. mark the nodes (the *basic blocks*) of G.

In order for the watermark to be stealthy G and P should be tightly integrated. In [17], integration is achieved by performing a random walk over the nodes of G and P and adding edges until P_w is uniformly dense. The watermark is extracted from P_w as follows:

1. let G be the subgraph induced by the set of marked blocks from P_w,
2. compute the watermark using the decoder $d(G)$.

Once again, in order for the watermark to be stealthy, the watermark graph G should look as much like "real" control-flow graphs as possible. Thus, the following properties must hold:

- G should be *reducible* [1]. Reducible graphs are constructed from structured programs, using only properly nested statements such as if-then-else, while-do, repeat-until, etc. Well-structured CFGs can be approximated by series-parallel graphs [10].
- G should have low maximum out-degree. In real code, basic blocks have out-degree 1 or 2. Only switch-statements construct CFGs with out-degree greater than 2, and these are unusual in most programs.
- G should be "skinny". In typical functions, control structures (loops and conditionals) are nested only a few levels deep. Hence, the width of G should be a small constant.

2.2 The CT Algorithm

The algorithm of Collberg and Thomborson [4] (henceforth, *CT*) embeds the watermark in a dynamically built, linked, graph structure. The rationale for this design is that graph-building code is difficult for an adversary to analyze, due to aliasing effects [8]. (The *aliasing problem* determines whether two pointers may/must refer to the same memory location. In the general case this is known to be undecidable [13].) The algorithm proceeds as follows:

1. The watermark number is encoded into a graph G.
2. G is split into a number of subgraphs.
3. Each subgraph is converted into a code sequence that builds the graph. P_w is constructed by inserting the graph-building statements along a special execution path $\langle p_1, p_2, p_3, p_4 \rangle$ of the program.
4. The recognition sequence is started by executing P_w with a special input sequence $i_1, i_2, \ldots, i_n$. This causes the program to follow the execution path $\langle p_1, p_2, p_3, p_4 \rangle$ and the watermark graph G to be built on the *heap*. (A *heap*, in this context, is the memory in a running program where dynamically allocated objects are allocated.)
5. G is extracted from the heap and decoded into W.

In order for the watermark to be stealthy, the CT algorithm requires that the watermark graph G look like a "real" dynamic data structure. Thus, the following properties must hold:

- G should have a low maximum out-degree, typically 2 or 3. Most linked data structures used in real programs are linked lists, binary trees, and sparse graphs, which all have low constant out-degree.
- G should be have a unique root node from which every other node is reachable. Data-structures in real programs are usually passed around using such root nodes.

3 Models of Attack

A successful attack against P_w prevents the recognizer from extracting the watermark while not seriously harming the performance or correctness of the program. It is generally assumed that an attacker has access to the algorithms used

by the embedder and recognizer. Only the watermark key is kept hidden. We distinguish between *semantics-preserving* and *non-semantics-preserving* attacks. A semantics-preserving attack is typically *automatic* and consists of running a set of semantics-preserving transformations over P_w. The transformations may reorganize the code, optimize the code, insert bogus code, remove or add abstractions, etc. in order to confuse the watermark recognizer [5,6]. The advantage of this type of attack is that the attacker does not need to know where in P_w the watermark is hidden. Rather, he can repeatedly apply semantics-preserving transformations uniformly over P_w until the code is sufficiently convoluted to confuse the recognizer.[1] A non-semantics-preserving attack requires the attacker to "guess" the approximate location of the watermark code. This can either be done through manual examination or through statistical analysis. The attacker will then proceed to carefully make minor "tweaks" to the code in an attempt to disable the watermark while not disrupting the normal execution of the program. In the remainder of this section we discuss attacks against graph-based software watermarking algorithms.

3.1 Edge-Flip Attacks

An *edge-flip attack* reorders the outgoing edges of each node in the graph. In the GTW algorithm this amounts to transforming ⌜`if (`p`)` T `else` E⌝ into ⌜`if (!`p`)` E `else` T⌝. In the CT algorithm this attack amounts to reordering the fields within the graph node structure and making appropriate modifications to any code that references the structure. For both algorithms these are simple semantics-preserving attacks that have little or no performance overhead.

3.2 Edge-Addition/Deletion Attacks

In the GTW algorithm a graph edge corresponds to a possible control-flow in the program. Adding an edge is accomplished using opaque predicates. For example, to add an edge between basic blocks B_1 and B_2 an attacker can add a bogus jump from B_1: ⌜`if (`P^F`) goto` B_2⌝. P^F is an *opaque false predicate*, a boolean expression that always evaluates to false but which is difficult for an adversary to evaluate. This is a simple attack with low performance overhead, assuming the opaque predicate is cheap [6].

In the CT algorithm a graph edge is a link between two heap objects. Adding an extra edge requires the attacker to extend each graph node with a bogus pointer field and adding code to link nodes on this field. This can be a very dangerous operation as it may affect which nodes get garbage collected and hence may introduce a memory leak into the program.

For both algorithms edge-deletions are dangerous operations for the attacker. In the GTW algorithm it would require the attacker to remove a branch from

[1] We assume that it is possible for the attacker to study the recognition algorithm to determine the kinds of semantics-preserving transformations that may be effective against a particular watermarking system.

the program. In the CT algorithm a link would have to be removed from a data-structure.

3.3 Node-Addition/Deletion Attacks

Adding a node in the GTW algorithm can be easily accomplished using opaque predicates. It is simply a matter of inserting the code $\ulcorner$`if (`P^F`) B`$\urcorner$ into the code, where B is a bogus basic block. Node additions are also easy in the CT algorithm, and similar to adding edges. Node deletions are difficult for both algorithms since they require a deep understanding of the semantics of the underlying program.

3.4 The Attack Model

Based on these observations, we can formulate a model for attacks against graph-based software watermarks. From what we have seen, not all attacks will be effective against all watermarking systems. Graphs are multi-graphs represented as a pair $(root, edges)$ where *edges* is list of triples $from \xrightarrow{link} to$, where *link* is the number (label) of the outgoing edge. For every node a we require that there be no outgoing edges with the same link number. For example, a graph G would be represented by the tuple $G = (a, \langle a \xrightarrow{1} b, a \xrightarrow{2} c, b \xrightarrow{1} d, c \xrightarrow{1} d, d \xrightarrow{2} a\rangle)$. An attack against a watermark graph G is modeled by transferring G over a lossy communications channel C. The severity of the attack is modeled by the lossy properties of C. A edge-flip attack only affecting a's outgoing edges would change $(a, \langle a \xrightarrow{1} b, a \xrightarrow{2} c, b \xrightarrow{1} d, c \xrightarrow{1} d, d \xrightarrow{2} a\rangle)$ into $(a, \langle a \xrightarrow{2} b, a \xrightarrow{1} c, b \xrightarrow{1} d, c \xrightarrow{1} d, d \xrightarrow{2} a\rangle)$.

Given a lossy channel C, our goal is to construct a class of graphs $\mathbb{G}$ that can be transfered across C unscathed. We call such graphs *error-correcting graphs*. For example, in the next section we present *reducible permutation graphs* which are resilient to edge-flip attacks.

Other types of error-correcting graph models can be considered. For example, it would seem natural to use an adjacency matrix representation of the graph, encode this as a bitstring, and use error-correcting codes [16] to protect it as it is being transfered across the lossy channel. However, for the purposes of software watermarking this model is not as good as the lossy channel model. For example, both the GTW and CT algorithms use graphs that are inherently linked structures. GTW's control-flow graph is implicit in the control-flow of the program and cannot be represented any other way. The CT algorithm gets its strength from the fact that it is a linked data-structure which is known to be computationally difficult to analyze. An adjacency matrix representation would not have this property.

4 Graph Encodings

Any enumerable class of graphs $\mathbb{G}$ may be used for software watermarking, giving us the freedom to choose a class with particularly desirable properties. To

obtain a watermark with efficient encoding and decoding functions, we require the enumeration to be polynomial-time efficient. Stealthy watermarks are chosen from a class of graphs, all of whose members resemble a naturally-arising population of graphs. Watermarks with high data rate are chosen with a uniform probability distribution from a very large class of graphs, each member of which can be embedded with a relatively small amount of overhead in space.

To lower-bound the data rate of a watermarking scheme, we assume that each graph g in $\mathbb{G}$ is chosen as a watermark with equal likelihood, and that the extra space required to embed any such g as a watermark in our target program is no more than s bytes. The data rate of this watermarking scheme is then lower-bounded by $\lg |\mathbb{G}|/s$, and we call this a "high data rate" if it is at least one bit of watermarking information per word of overhead space.

We have been particularly interested in $\mathbb{G}$ that resemble either the linked-list data structures or the trees that are built dynamically by many "naturally occurring" programs.

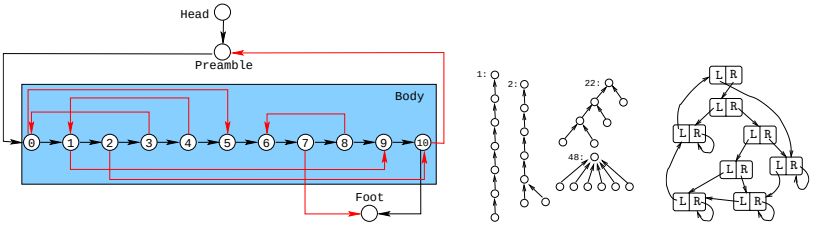
4.1 Parent-Pointer Trees

The parent-pointer data structure is the most space-efficient representation of a tree using nodes with pointer fields. Each node in the data structure has just one pointer field referencing its parent. We may take our class of watermarking graphs to be $\mathbb{G}_{pp}$, the set of all parent-pointer graphs on n nodes, for some suitable constant n; see Fig. 1(b). This class is stealthy, for structures with a single pointer field arise often in programs that would be watermarked. If we take $n = 655$ nodes, then $|\mathbb{G}_{pp}| \approx 2^{1024}$ by the enumeration described in Knuth Vol. 1 [11]. Since one word of storage is required for each pointer in our watermark, the data rate is high: $1024/655 = 1.56$ bits per word. Highly efficient codecs can be obtained by ordering the operations in the enumeration. For example we may assign index 1 to the path of length $n-1$, and we may enumerate all other parent-pointer trees whose roots have indegree one before any tree whose root has indegree larger than one. The highest index may be assigned to the star graph.

Regrettably the parent-pointer data structure has almost no error-correcting properties. An adversary who adds a single node or an edge to a parent-pointer graph g may radically change the watermark value $d(g)$ it represents.

4.2 Planted Planar Cubic Trees

Stronger error-correcting properties can be obtained by restricting the class of trees, for an adversary's modifications may result in a watermark that is outside our class, and our decoder may have a reasonably-efficient algorithm for correcting the "error" introduced by the adversary. For example, we may choose our watermark from the class $\mathbb{G}_{PPCT}$ of planted planar cubic trees; see Fig. 1(c). The enumeration of these trees is especially straightforward and efficient, for they follow a Catalan recurrence [9]. This class has moderate data rate, approximately 0.5 bits per word. The class $\mathbb{G}_{PPCT}$ is very stealthy, for it may be represented



(a) Reducible permutation graph. (b) Parent-pointer. (c) PPCT.

Fig. 1. Graph encodings.

by the commonly-encountered binary tree data structure, although it has two quirks. Each non-leaf node has exactly two children, except for a root node that has outdegree one. The underlying undirected graph is thus cubic (uniform degree 3) except at its leaves and its root. Such data structures are used in any program involving binary search trees, and superficially similar structures arise in any program that has a datatype with exactly two pointer fields.

When we represent a PPCT in a data structure, we might distinguish leaf nodes from non-leaf nodes by using null pointers in their "lchild" and "rchild" fields. However, with this representation, node-addition attacks will be potent (so long as the adversary knows enough not to create nodes with out-degree 1). Edge-addition attacks are also potent, for the decoder will be faced with a combinatorial explosion when trying to decide "which one of the three outgoing edges from a node is bogus?".

The error-correcting properties of PPCTs can be greatly improved, at a modest cost in stealthiness, by using an alternative distinction between leaf and non-leaf nodes. Leaf nodes should have a self-reference in their "rchild" field; non-leaf nodes never refer to themselves. The "lchild" fields of leaf nodes should form a singly-linked list of all the leaf nodes in reverse depth-first search order. The root node of a PPCT should be lchild-linked from the first leaf of the tree, and it should lchild-point to the last leaf of the tree; see Fig. 1(c). We have thus constructed a biconnected digraph as a supergraph of our (child-directed) binary tree, by drawing a directed outercycle linking the root and all the leaves on a planar map of the tree. Including an outercycle in the PPCT representation increases their error-correcting properties. In particular, PPCTs can detect and correct one occurrence of node deletion or insertion. It is also possible to repair edge-flips but we leave the details out of this extended abstract. Multiple edge- and node-deletions are more problematic, for these may disconnect the PPCT. The PPCT may be disconnected into k fragments by $O(k)$ deletions, and we suspect that time exponential in k may be required for the decoder to examine all "legal" reassemblies into PPCTs.

4.3 List-Type Graphs

Radix-list watermark graphs $\mathbb{G}_r$ of order n have the following structure: they are a singly-linked circular list of n nodes, in which each node has an additional pointer field that may point either to NULL or to any other node in the list. Thus $|\mathbb{G}_r| = n^{n+1}$, and the data rate is very close to $(\lg n)/2$. When $n = 255$, these graphs have a data rate of 4 hidden bits per word of overhead [4]. They are reasonably stealthy, for some programs employ circular lists of nodes that contain a single additional pointer, and many programs use nodes that have two pointers. However these graphs have poor error-correcting properties, for they have very little redundancy other than the n-cycle. Node- and edge-addition attacks on such graphs will force our decoder to enumerate all Hamiltonian subgraphs, in order to construct all possible watermark graphs that could have existed prior to the attacks.

We can add redundancy to a radix-list watermark graphs by restricting these to indegree two and outdegree two. The first outgoing arc on each node defines a Hamiltonian cycle. The second arc defines a permutation on the nodes. We have $|\mathbb{G}_p| = n!$ for this class of "permutation watermark graphs", and its data rate is $(\lg n)/2 - O(1)$. Efficient codecs may be built from textbook enumerations of permutations, e.g. [15]. Permutation watermark graphs also have modest error-correcting properties, for the decoder will gain clues about bogus arcs and nodes by noting where in- and out-degrees differ from the required value 2.

4.4 Reducible Permutations

We now introduce a class of list-like watermark graphs with excellent error-correcting properties. Our reducible permutation graphs (RPG) are reducible flow graphs with Hamiltonian paths consisting of four pieces; see Fig. 1(a):

- **header node:** This is the only node in the graph with in-degree zero. It has out-degree one, and every other node in the graph is reachable from it. Any flow graph must have such a node.
- **preamble:** These nodes immediately follow the header node and have no forward edges, except those edges used in the graph's Hamiltonian path. Because of that, edges can be inserted from anywhere in the body to anywhere in this section without making the flow graph irreducible.
- **body:** Edges between these nodes encode a self-inverting permutation. The body starts with the first node along the Hamiltonian path with a forward edge that is not part of that Hamiltonian path. In many cases, the fact that the permutation is self-inverting eliminates the need for a preamble.
- **footer node:** This node has out-degree zero, and it is reachable from every other node in the graph. Any flow graph must have a node like this as well.

Much of the information needed to extract the encoded permutation is contained in the body. The Hamiltonian path, of which there can be at most one in any reducible flow graph, puts nodes in the order by which they have been numbered.

Other edges in the body put the nodes in the order $\sigma(0), \sigma(1), \ldots, \sigma(k-1)$, where k is the number of nodes in the body and σ is the permutation being encoded.

The graph encodes a permutation σ on the set $\{0, 1, \ldots, k-1\}$ such that σ^2 is the identity permutation and $\sigma(0) = 0$. It does so with a cycle among the nodes in the body, except some edges from that cycle cannot be included in the graph because they would make the graph irreducible. Edges from nodes in the body to nodes in the preamble serve to remove the ambiguity created by these missing edges. In Fig. 1(a), nodes 5, 6, 9, and 10 were the tail nodes of such removed edges, since their out-degree to other nodes in the body is only 1. The edges from these nodes to the preamble, or lack thereof, indicates how they are to be sorted in order of the head nodes of their removed edges. No edge to the preamble means the node is to be inserted at the end of the list. An edge to the first node in the preamble, such as node 10 has, indicates that the node is to be inserted before the last node in the list. This puts the nodes in the order 5, 6, 10, 9. If nodes with removed out-edges had to be placed earlier in the list than the second to last position, we would need more nodes in the preamble. Nodes that are the head nodes of removed edges are nodes 2, 3, 4, and 7. Thus, in order to complete the cycle, we add edges from 5 to 2, from 6 to 3, from 10 to 4, and from 9 to 7. Then the nodes in permuted order are 0, 5, 2, 10, 4, 1, 9, 7, 8, 6, and 3. In cycle notation, this permutation is $(0)(1,5)(2)(3,10)(4)(6,9)(7)(8)$, which is its own inverse and is permutation number 5487 in our enumeration. We discuss the RPG encoding in the next section.

5 Properties of Reducible Permutation Encodings

For most of the encoding schemes described in the previous section, it is assumed that the order of outgoing edges is preserved. Encodings that rely on such assumptions are vulnerable to the simplest type of attacks such as edge-flips. Without this assumption, a single permutation graph can be viewed as encoding two different permutations.

The reducible permutation graphs (RPG), however, do not rely on any such assumptions. While their data rate is slightly lower than that of other encoding schemes, we believe their superior error correcting properties make up for it. Moreover, RPGs are very stealthy and fit well into the CT algorithm. We study the properties of RPGs in detail below.

Theorem 1. *A reducible permutation graph $G = (V, E)$ can be created in polynomial time.*

Proof: Let σ be a permutation on $\{0, 1, \ldots, k-1\}$ (which will henceforth be referred to as $\mathbb{Z}_k$) such that $\sigma(0) = 0$ and $\sigma = \sigma^{-1}$. We use the following algorithm to create a reducible permutation graph $G = (V, E)$:

1. Initialization: $V := \{h, v_0, v_1, \ldots, v_k\}$ and $E := \{(v_i, v_{i+1}) : 0 \leq i < k\}$, where h is the header, v_k the footer, and the remaining nodes are the body.
2. For each $i \in \mathbb{Z}_k \backslash \{0\}$ such that $\sigma(i-1) < \sigma(i)$, set $E := E \cup \{(v_{\sigma(i-1)}, v_{\sigma(i)})\}$. At this point G is still acyclic and hence reducible.

3. Compute the dominators of $G - \{h\}$, using v_0 as the root.
4. Initialize $B := \{j \in \{0, 1, \ldots, k-2\} : \sigma(j) > \sigma(j+1)\}$. Initialize $L := \emptyset$.
5. For each $j \in B$, if $v_{\sigma(j+1)}$ dominates $v_{\sigma(j)}$, set $E := E \cup \{(v_{\sigma(j)}, v_{\sigma(j+1)})\}$; else set $L := L \cup \{(v_{\sigma(j)}, v_{\sigma(j+1)})\}$. In the cases where an edge is added to E, it is a backedge, and hence has no effect on the dominance hierarchy.
6. Let $\tau : L \to \mathbb{Z}$ be defined as $\tau((v_i, v_j)) = |\{(v_{i'}, v_{j'}) \in L : i' < i, j' > j\}|$. Let $m = \max \tau(L)$.
7. If $m = 0$, set $E := E \cup \{(h, v_0)\}$ and return $G = (V, E)$.
8. Else, set $V := V \cup \{p_1, p_2, \ldots, p_m\}$. For each $(v_i, v_j) \in L$, if $\tau((v_i, v_j)) > 0$, set $E := E \cup \{(v_i, p_{\tau((v_i, v_j))})\}$, where $\{p_1, p_2, \ldots, p_m\}$ are the preamble vertices. Set $E := E \cup \{(h, p_m), (p_m, p_{m-1}), \ldots, (p_2, p_1), (p_1, v_0)\}$ and return $G = (V, E)$.

The output graph is reducible up to and including step 7. If step 8 is reached (the graph needs a preamble), the preamble is inserted so that the only path from h to any of $\{v_0, v_1, \ldots, v_k\}$ passes through v_0. As a result, dominance relations among these vertices are preserved, and the whole preamble dominates the whole body and the footer node. □

The following four theorems show that RPGs can be decoded in polynomial time and place a lower bound on their data rate. We omit the proofs in this extended abstract, except for Theorem 4, which gives the decoding algorithm.

Theorem 2. *Any acyclic digraph has at most one Hamiltonian path.*

Theorem 3. *Any reducible flowgraph has at most one Hamiltonian path. Furthermore, this Hamiltonian path can be found in polynomial time if it exists.*

Theorem 4. *Reducible permutation graphs can be decoded and corrected for edge-flips in polynomial time.*

Proof: The following algorithm decodes an RPG $G = (V, E)$:

1. Compute the dominance tree of G and verify that G is reducible.
2. Find a Hamiltonian path P in G (in polynomial time from above theorem).
3. Let v_0 be the first node with a forward edge not in P. Let $v_1, v_2, \ldots, v_{k-1}$ be the subsequent nodes in P, excluding f. Let $p_m, p_{m-1}, \ldots, p_1$, be the nodes strictly between h and v_0, if any.
4. Let $B := \{v_0, v_1, \ldots, v_{k-1}\}$.
5. Create an empty ordered list of nodes L. For each $i \in \{0, 1, \ldots, k-1\}$, if v_i has one outgoing edge to $B \cup \{f\}$, insert v_i in L. If v_i has no outgoing edges to $\{p_1, p_2, \ldots, p_m\}$, insert it at the end of the list. Otherwise, if it has an edge to p_j, insert it before the last j elements of the list.
6. For each $i \in \{0, 1, \ldots, k-1\}$, if v_i has only one incoming edge from $B \cup \{h, p_1\}$, add an edge from the first element of L to v_i and remove that first element from L.
7. Initialize an integer array $A[]$ of k elements. Set $A[0] := 0$.

8. For each $i \in \{1, 2, \ldots, k-1\}$, there will now be exactly one edge from $v_{A[i-1]}$ to $B \cup \{f\}$ not contained in the path P. If that edge goes to a vertex v_j, set $A[i] := j$. If that edge goes to f, set $A[i] := A[i-1] + 1$.
9. A will now contain the appropriate permutation.

Since the order of the vertices is determined by the Hamiltonian path P, and since any reducible flow graph has at most one Hamiltonian path, this algorithm will extract the same permutation from any two isomorphic graphs. Therefore, the RPG encoding scheme is resilient to edge flipping attacks. □

Theorem 5. *The data rate for RPGs is at least* $(\lg n - \lg e - 1)/4$.

References

1. A. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, 1986.
2. R. J. Anderson and F. A. Peticolas. On the limits of steganography. *IEEE J-SAC*, 16(4), May 1998.
3. W. Bender, D. Gruhl, N. Morimoto, and A. Lu. Techniques for data hiding. *IBM Systems Journal*, 35(3&4):313–336, 1996.
4. C. Collberg and C. Thomborson. Software watermarking: Models and dynamic embeddings. In *POPL'99*, Jan. 1999.
5. C. Collberg, C. Thomborson, and D. Low. Breaking abstractions and unstructuring data structures. In *ICCL'98*, 1998.
6. C. Collberg, C. Thomborson, and D. Low. Manufacturing cheap, resilient, and stealthy opaque constructs. In *POPL'98*, Jan. 1998.
7. R. L. Davidson and N. Myhrvold. Method and system for generating and auditing a signature for a computer program. US Patent 5,559,884, Sept. 1996. Assignee: Microsoft Corporation.
8. R. Ghiya and L. J. Hendren. Is it a tree, a DAG, or a cyclic graph? A shape analysis for heap-directed pointers in C. In *POPL'96*, pages 1–15, 21–24 Jan. 1996.
9. F. Harary and E. Palmer. *Graphical Enumeration.* Academic Press, New York, 1973.
10. S. Kannan and T. A. Proebsting. Register allocation in structured programs. *Journal of Algorithms*, 29(2):223–237, Nov. 1998.
11. D. E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming.* Addison-Wesley, Reading, MA, USA, third edition, 1997.
12. G. Qu and M. Potkonjak. Analysis of watermarking techniques for graph coloring problem. In *IEEE/ACM Intl. Conference on CAD*, pages 190–193, 1998.
13. G. Ramalingam. The undecidability of aliasing. *ACM TOPLAS*, 16(5):1467–1471, Sept. 1994.
14. J. P. Stern, G. Hachez, F. Koeune, and J.-J. Quisquater. Robust object watermarking: Application to code. In *Information Hiding*, pages 368–378, 1999.
15. A. Tucker. *Applied Combinatorics.* Wiley, 3rd edition, 1994.
16. S. A. Vanstone and P. C. V. Oorschot. *An Introduction to Error Correcting Codes with Applications.* Klewer Academic Publishers, 1989.
17. R. Venkatesan, V. Vazirani, and S. Sinha. A graph theoretic approach to software watermarking. In *4th Intl. Information Hiding Workshop*, 2001.

Completely Connected Clustered Graphs*

Sabine Cornelsen[1] and Dorothea Wagner[2]

[1] Università dell'Aquila, Dipartimento di Ingegneria Elettrica,
cornelse@inf.uni-konstanz.de
[2] University of Karlsruhe, Department of Computer Science,
dwagner@ira.uka.de

Abstract. Planar drawings of clustered graphs are considered. We introduce the notion of completely connected clustered graphs, i.e. hierarchically clustered graphs that have the property that not only every cluster but also each complement of a cluster induces a connected subgraph. As a main result, we prove that a completely connected clustered graph is c-planar if and only if the underlying graph is planar. Further, we investigate the influence of the root of the inclusion tree to the choice of the outer face of the underlying graph and vice versa.

1 Introduction

A frequently used method for visualizing a clustered structure of a graph is to draw every cluster as a simple closed region (e.g. an axis-parallel rectangle) bounded by a simple closed curve. Algorithms and data structures for representing clusterings in general graphs according to this approach can, for example, be found in [1,14,19,20,21]. Feng et al. [11] defined planarity – called c-planarity – for clustered graphs. Since then algorithms for drawing c-planar clustered graphs with respect to different aesthetic criteria were developed [6,7,8,9,10,17].

For connected clustered graphs, i.e. for clustered graphs in which every cluster induces a connected subgraph, Feng et al. [11] gave a c-planarity criterion. They used this criterion to test in quadratic time, whether a connected clustered graph is c-planar. A linear time algorithm that solves this problem was given by Dahlhaus [4]. A first attempt for testing whether certain not necessarily connected clustered graphs are c-planar was done by Gutwenger et al. [13]. As far as we know, the complexity status for testing whether an arbitrary clustered graph is c-planar is still open.

Motivated by drawings of minimal cuts [2,3], we consider completely connected clustered graphs, i.e. clustered graphs in which not only every cluster, but also the complement of each cluster is connected. This graph class has also applications in triangulating c-planar clustered graphs [15]. Very surprisingly it turns out that a completely connected clustered graph is c-planar if only the

* This work was partially supported by the DFG under grant BR 2158/1-1 and WA 654/13-1. It was also partially supported by the Human Potential Program of the EU under contract no HPRN-CT-1999-00104 (AMORE Project).

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 168–179, 2003.

underlying graph is planar. In this paper, we consider varying roots of the inclusion tree. Originally, this was also motivated by drawings of cuts but turned out to be a useful proof technique as well.

The contribution of this paper is as follows. In Section 2, we shortly summarize the definitions and results about c-planar graphs that we will use in this paper. Section 3 introduces completely connected clustered graphs. We first show that c-planarity does not depend on the choice of the root of the inclusion tree. Then, we apply this result to show that a completely connected clustered graph with underlying planar graph is c-planar. The dependence between the outer face of the underlying graph on one hand and the root of the inclusion tree on the other hand is examined in Sect. 4. Finally, we investigate in Sect. 5, whether c-planar clustered graphs can be augmented to completely connected c-planar clustered graphs and whether arbitrary completely connected clustered graphs have a completely connected c-planar clustered subgraph.

2 Preliminaries

A hierarchically clustered graph (G, T, r) – or *clustered graph*, for simplicity – as introduced by Feng et al. [11] consists of

- an (undirected) graph $G = (V, E)$,
- a tree T, and
- an inner vertex r of T such that
- the set of leaves of T is exactly V.

G is called the *underlying graph* and T the *inclusion tree* of (G, T, r). To distinguish vertices of the inclusion tree from vertices in the underlying graph, inner vertices of T are called *nodes*. We denote the tree T rooted at r by (T, r). Each node ν of T represents the *cluster* $V_r(\nu)$ of leaves in the subtree of (T, r) rooted at ν. Let S be any subset of V. By $G(S)$, we denote the subgraph of G induced by S and by $G - S$ we denote the subgraph of G induced by $V \setminus S$. An edge e of G is said to be *incident* to S, if e is incident to a vertex in S and a vertex in $V \setminus S$. If $v_1, v_2 \in V$ are two vertices then $G + \{v_1, v_2\}$ denotes the graph $(V, E \cup \{v_1, v_2\})$. A clustered graph (G, T, r) is *connected*, if each cluster induces a connected subgraph of G. A *c-planar drawing* of a clustered graph (G, T, r) consists of

- a planar drawing of the underlying graph G and
- an inclusion representation[1] of the rooted tree (T, r) such that
- each edge crosses the boundary of the drawing of a node of T at most once.

[1] In an inclusion representation of a rooted tree (T, r), each node of T is represented by a simple closed region bounded by a simple closed curve. The drawing of a node or leaf ν of T is contained in the interior of the region representing a node μ of T if and only if μ is contained in the path from ν to r in T. The drawings of two nodes μ and ν are disjoint if neither μ is contained in the path from ν to r nor ν is contained in the path from μ to r in T.

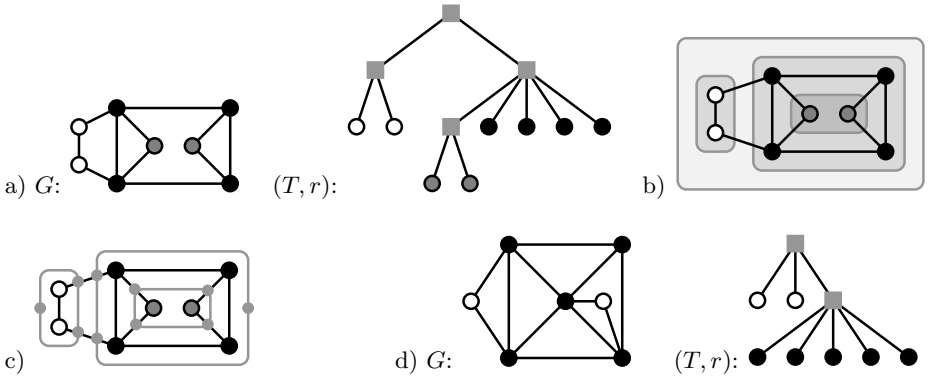


Fig. 1. a) A c-planar clustered graph that is not connected. b) A c-planar drawing of the clustered graph in a. c) The auxiliary graph associated with the c-planar drawing in b. Boundary cycles are grey. d) A connected clustered graph that is not c-planar. The correspondence of vertices of G and leaves of T is indicated by corresponding colors.

Note that the vertices of G are the leaves of T and thus have the same drawing. A clustered graph is *c-planar* if it has a c-planar drawing. A clustered graph with planar underlying graph does not have to be c-planar. An example is given in Fig. 1d. Feng et al. [11] characterized c-planar connected clustered graphs as follows.

Theorem 1 ([11]). *A connected clustered graph (G, T, r) is c-planar if and only if there exists a* c-planar embedding *of G for (G, T, r), i.e. a planar embedding of G together with a fixed outer face such that for each node ν of T all vertices of $V \setminus V_r(\nu)$ are in the outer face of the drawing of $G(V_r(\nu))$.*

With $H \subseteq G$ we denote that a graph H is a spanning subgraph of the graph G. We call a clustered graph (H, T, r) a *subgraph* of the clustered graph (G, T, r), if $H \subseteq G$.

Theorem 2 ([11]). *A clustered graph is c-planar if and only if it is a subgraph of a c-planar connected clustered graph.*

3 Planarity Is Sufficient

A clustered graph (G, T, r) is *completely connected* if and only if for each inner node ν of T both, $G(V_r(\nu))$ and $G - V_r(\nu)$, are connected. An example of a completely connected clustered graph is shown in Fig. 2, while the clustered graph in Fig. 1d is connected but not completely connected.

Remark 1. Let (G, T, r) be a clustered graph. The following statements are equivalent.

1. (G, T, r) is completely connected.
2. (G, T, ν) is connected for every inner node ν of T.
3. (G, T, ν) is completely connected for every inner node ν of T.

In the remainder of this section, we show that a completely connected clustered graph is c-planar if and only if the underlying graph is planar. First, we prove that it does not depend on the choice of the root of the inclusion tree whether a clustered graph is c-planar or not.

For an easier discussion, we associate an *auxiliary graph* $G_{\mathcal{D}}$ with a c-planar drawing $\mathcal{D}$ of the clustered graph (G, T, r). It is the auxiliary graph that was introduced for constructing bend-minimum orthogonal drawings of clustered graphs [2,17]. Let V' be the set of points, in which drawings of edges and boundaries of drawings of clusters intersect. Then the vertex set of $G_{\mathcal{D}}$ is $V \cup V'$. The edge set of $G_{\mathcal{D}}$ contains two types of edges. For an edge $e = \{v, w\}$ of G, let $v_1, \ldots, v_k$ be the points in $\mathcal{D}(e) \cap V'$ in the order they occur in the drawing of e from v to w. Then $G_{\mathcal{D}}$ contains the edges $\{v, v_1\}, \{v_1, v_2\}, \ldots, \{v_k, w\}$. Let $\nu \neq r$ be a node of T. Let $v_1, \ldots, v_k$ be the points in $\partial\mathcal{D}(\nu) \cap V'$ in the order they occur in the boundary $\partial\mathcal{D}(\nu)$ of the drawing of ν. Then $G_{\mathcal{D}}$ contains the edges $\{v_1, v_2\}, \ldots, \{v_{k-1}, v_k\}, \{v_k, v_1\}$. The cycle $v_1, \ldots, v_k$ of $G_{\mathcal{D}}$ is called the *boundary cycle* of ν. (To avoid loops and parallel edges, additional vertices of degree two may be inserted into boundary cycles). We interpret the c-planar drawing $\mathcal{D}$ of (G, T, r) also as a planar drawing of $G_{\mathcal{D}}$. An example of such an auxiliary graph can be found in Fig. 1c.

Lemma 1. *Let (G, T, r) be a c-planar clustered graph and ν a node of T. Then (G, T, ν) is c-planar.*

Proof. Let $\mathcal{D}_r$ be a c-planar drawing of (G, T, r) and let G_r be the auxiliary graph of $\mathcal{D}_r$. By Theorem 2, we may assume that G_r is connected. Let $P : \nu = \nu_1, \nu_2, \ldots, \nu_\ell = r$ be the path in T between ν and r. Then

$$V_\nu(\mu) = \begin{cases} V \setminus V_r(\nu_{i-1}) & \text{if } \mu = \nu_i, i = 2, \ldots, \ell \\ V_r(\mu) & \text{if } \mu \text{ is not in P} \end{cases}$$

for a node $\mu \neq \nu$ of T. Thus for any choice of the outer face, there is a boundary cycle C_μ in G_r that separates $V_\nu(\mu)$ and $V \setminus V_\nu(\mu)$. More precisely, C_μ is the boundary cycle of μ if μ is not in P or C_μ is the boundary cycle of ν_{i-1} if $\mu = \nu_i, i = 2, \ldots, \ell$. We show now that the outer face can be chosen such that $V_\nu(\mu)$ is always contained in the simple region bounded by C_μ.

Let C be the boundary cycle of ν. Let f be a face of $\mathcal{D}_r$ that is contained in the simple region bounded by C, but incident to some edge in C. Let $\mathcal{D}_\nu$ be a drawing of G_r that has the same embedding as $\mathcal{D}_r$, but outer face f. Let $\mu_1, \ldots, \mu_k$ be the adjacent nodes of ν, such that μ_1 is on the path from ν to r in T. Then for $i = 2, \ldots, k$ it holds that $V_\nu(\mu_i) = V_r(\mu_i)$ is still inside the boundary cycle of μ_i and that $V_\nu(\mu_1) = V \setminus V_r(\nu)$ is now inside the boundary cycle of ν. Finally, for all nodes $\mu \neq \nu$ of T, there exists an $i \in \{1, \ldots, k\}$ such that $V_r(\mu) \subseteq V_r(\mu_i)$. Thus, $\mathcal{D}_\nu$ contains the cluster boundaries of all non-root nodes of a c-planar drawing of (G, T, ν). □

If the root r of a clustered graph (G, T, r) is not important – e.g., if we are only interested whether (G, T, r) is c-planar or completely connected – we will omit the root in the notation of the clustered graph. I.e. we refer to (G, T, r) by (G, T). The following theorem gives a surprisingly easy characterization of c-planar completely connected clustered graphs. Note that this result is independently described by Jünger et al. [15].

Theorem 3. *Let (G, T) be a completely connected clustered graph. Then*

$$(G, T) \text{ is c-planar} \iff G \text{ is planar.}$$

Proof. Clearly, G has to be planar if (G, T) is c-planar. For the other direction, we show that for any planar embedding $\mathcal{E}$ and any outer face f_o of G the root r of T can be chosen such that $\mathcal{E}$ together with f_o is a c-planar embedding for (G, T, r). Hence, by Theorem 1, (G, T) is c-planar.

Let $v \in V$ be a vertex that is incident to the outer face f_o. Let r be a node of T that is adjacent to v in T. Let ν be any node of T. Suppose there exists a vertex $w \in V \setminus V_r(\nu)$ that is not drawn in the outer face of $G(V_r(\nu))$. Since (G, T, r) is completely connected, there exists a path from v to w in $G - V_r(\nu)$. But since v and w are contained in different faces of $G(V_r(\nu))$, this is not possible. □

The proofs of Lemma 1 and Theorem 3 even showed that every planar embedding of G is a c-planar embedding for the completely connected clustered graph (G, T). Only the outer face of G has to be chosen according to the root of T or vice versa.

4 The Root and the Outer Face

Let (G, T) be a completely connected clustered graph with planar underlying graph G and let the planar embedding $\mathcal{E}$ of G be fixed. We show in this section how the following two problems can be solved in linear time.

1. Given a fixed outer face f_o of G, which nodes r can be chosen to be the root of T such that $\mathcal{E}$ together with f_o is a c-planar embedding for the clustered graph (G, T, r)?
2. Given a fixed root r of T, which faces can be the outer face of G in a c-planar embedding for the clustered graph (G, T, r)?

For the first problem, let f_o be the outer face of $G = (V, E)$. We call a node r of T *rootable* (with respect to f_o), if the fixed embedding of G together with the outer face f_o is a c-planar embedding of (G, T, r). A c-planar drawing of (G, T, r) is called *compatible* with f_o if the drawing of the underlying graph G corresponds to the given embedding $\mathcal{E}$ of G with the fixed outer face f_o. In the proof of Theorem 3, we made the following observation.

Remark 2. A node of T is rootable, if it is adjacent to a vertex of G that is incident to the outer face f_o.

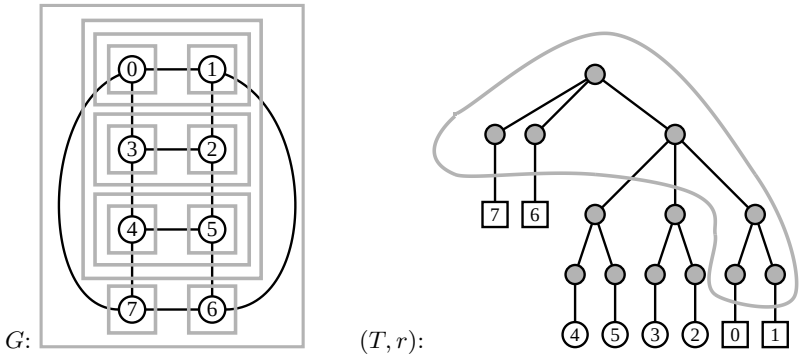


Fig. 2. A completely connected clustered graph. Rootable nodes are encircled. Vertices of G that are incident to the outer face are drawn as quadrangles in the inclusion tree.

Hence, there exists at least one rootable node. The following remark follows immediately from the fact that in a c-planar embedding the complement of a cluster $V_r(\nu)$ is contained in the outer face of the subgraph $G(V_r(\nu))$.

Remark 3. Let r, ν be two nodes of T and let r be rootable. Then $V \setminus V_r(\nu)$ contains a vertex that is incident to f_o.

We use this observation to characterize which nodes among those nodes that are adjacent to a rootable node are rootable themselves. Note that the following lemma is not true, if (G, T) is not completely connected. See Fig. 3 for an example.

Lemma 2. *Let r be a rootable node of T. A node ν of T that is adjacent to r is rootable if and only if $V_r(\nu)$ contains a vertex that is incident to f_o.*

Proof. **"⇒":** If ν is rootable, Remark 3 implies that $V_r(\nu) = V \setminus V_\nu(r)$ contains a vertex that is incident to f_o.

"⇐": Suppose now, that $V_r(\nu)$ contains a vertex that is incident to f_o. Let $\mathcal{D}_r$ be a c-planar drawing of (G, T, r) that is compatible with f_o. Let G_r be the auxiliary graph of $\mathcal{D}_r$.

Let C be the boundary cycle of ν. Since $V_r(\nu)$ contains a vertex that is incident to f_o there is an edge e in C that is contained in f_o. Since (G, T) is completely connected and ν is adjacent to the root r, it follows that e is incident to the outer face of $\mathcal{D}_r$:

Else let f be the face of $\mathcal{D}_r$ that is incident to e, but not contained in the simple region bounded by C. Let $C_f : v_1, \ldots, v_\ell$ be a cycle that is contained in the boundary of f such that f is contained in the simple region bounded by C_f. Suppose C_f contains boundary edges of a node μ. Let $\{v_j, v_{j+1}\}$ be the first and $\{v_{k-1}, v_k\}$ be the last such edge in C_f. Replace $v_j, \ldots, v_k$ in C_f by a path from v_j to v_k in $G(V_r(\mu))$. Since ν is adjacent to the root r, the interior of the simple region bounded by the original C_f does not intersect

the simple region bounded by the boundary cycle of μ. Thus the modified C_f is still a cycle that bounds a simple region containing f. Proceeding like this, we can eliminate every boundary edge in C_f. Thus, f is contained in the simple region bounded by a cycle of G, which contradicts the fact that e is contained in f_o.

Now, let f be the inner face of $\mathcal{D}_r$ that is incident to e. A drawing of G_r with outer face f can be obtained by redrawing e through the outer face of $\mathcal{D}_r$. This yields a c-planar drawing of (G, T, ν) that is compatible with f_o. □

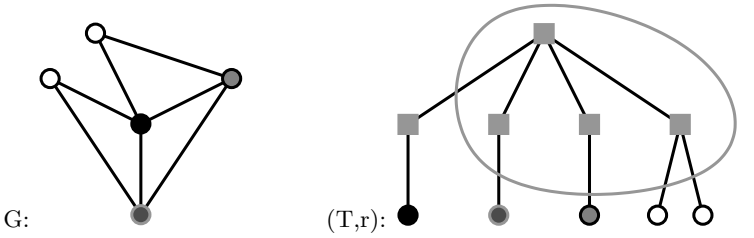


Fig. 3. A clustered graph for which Lemma 2 is not true. Rootable nodes are again encircled.

By Remark 3, the statement in Lemma 2 could also be formulated as follows. An adjacent node ν of a rootable node r is rootable if and only if both, $V_r(\nu)$ and $V \setminus V_r(\nu)$, contain a vertex that is incident to the outer face. This leads to the general characterization of rootable nodes.

Theorem 4. *Let (G, T) be a completely connected clustered graph. A node ν of T is rootable with respect to a fixed outer face f_o if and only if at least two connected components of $T - \nu$ contain vertices of G that are incident to f_o.*

Proof. **"⇒":** Suppose node ν is rootable but has a neighbor μ such that all vertices that are incident to the outer face are contained in $V_\nu(\mu)$. In this case $V_\nu(\mu)$ contains all vertices of G. Since ν is a node, i.e. an inner vertex of T, there would be leaves of T that are not vertices of G. This contradicts the definition of inclusion trees.

"⇐": Suppose now that at least two connected components of $T - \nu$ contain vertices of G that are incident to the outer face, but that ν is not rootable. Let r be a rootable node of T. Let ν_i be the first node on the path $r = \nu_1, \nu_2, \ldots, \nu_\ell = \nu$ that is not rootable. Note that $V \setminus V_r(\nu)$ is the set of leaves in one connected component of $T - \nu$. Thus, by the precondition, $V_r(\nu)$ has to contain at least one vertex that is incident to the outer face. Since $V_r(\nu) \subseteq V_r(\nu_i) = V_{\nu_{i-1}}(\nu_i)$, cluster $V_{\nu_{i-1}}(\nu_i)$ also contains a vertex that is incident to the outer face. Thus, by Lemma 2, ν_i is rootable, contradicting the choice of ν_i. □

The previous lemma and its proof lead to the following corollary.

Corollary 1. *The set of rootable nodes induces a subtree of T.*

Using Remark 2 and Theorem 4, the rootable nodes can be found in linear time by a bottom-up top-down approach starting from an arbitrary node r of T as described in Algorithm 1.

The idea of the algorithm is as follows. The node array PRED represents the predecessor of each node or leaf in the rooted tree (T, r). The outer face f_o of G is represented by the boolean node array OUTER-FACE, i.e. OUTER-FACE(v) is true if and only if v is a vertex of G – and hence a leaf of T – that is incident to the outer face.

In a first step, the algorithm proceeds from the leaves to the root r of T. It sets ONE(ν) to true if $V_r(\nu)$ contains at least one vertex that is incident to f_o. There are two cases in which ROOTABLE(ν) is set to true: if ν is adjacent to a vertex of G that is incident to f_o or if ν has at least two children μ_1, μ_2 in (T, r) such that $V_r(\mu_1)$ as well as $V_r(\mu_2)$ contain vertices that are incident to f_o.

In a second step, the algorithm proceeds from the root r to the leaves of T. It sets ROOTABLE(μ) to true, if both, $V_r(\mu)$ and $V \setminus V_r(\mu)$, contain vertices that are incident to f_o. In the end, the rootable nodes are exactly the nodes for which ROOTABLE is true.

We can also apply Theorem 4 to solve the second problem in linear time. Let r be a root of T and let $T_1, \ldots, T_k$ be connected components of $T - r$. Let v be a vertex of G and let $i \in \{1, \ldots, k\}$ be such that v is contained in T_i. Then v gets the label i. Now, by Theorem 4, each face that is incident to at least two vertices with different labels can be chosen as the outer face.

5 Subgraphs and Supergraphs

Di Battista et al. [5] showed that every connected clustered graph has a c-planar connected clustered subgraph: Proceeding from the leaves to the root of the inclusion tree, construct a subgraph in which every cluster induces a spanning tree. Unfortunately, not every completely connected clustered graph has a completely connected subgraph that is c-planar: See the clustered graph (G, T, r) in Fig. 4 for an example. G is a subdivision of a $K_{3,3}$ and hence is not planar. But the clustered graph (H, T, r) is not completely connected for any proper subgraph $H \subseteq G$. In the rest of this section, we show that at least Theorem 2 can be extended to completely connected clustered graphs. This result was independently obtained by Jünger et al. [15].

Theorem 5. *Every c-planar clustered graph is a subgraph of a c-planar completely connected clustered graph.*

Proof. Let (G, T, r) be a c-planar clustered graph with a fixed c-planar embedding. By Theorem 2, there exists a graph $G_0 \supseteq G$ such that (G_0, T, r) is c-planar

Algorithm 1: Finding the rootable nodes.

Input : tree T, boolean node array OUTER-FACE
Output : boolean node array ROOTABLE initialized to FALSE
Data : node array PRED, boolean node array ONE initialized to false

```
BOTTOM-UP(node ν) begin
    foreach incident edge e = {ν, μ} of ν do
        if μ ≠ PRED(ν) then
            PRED(μ) ← ν;
            BOTTOM-UP(μ);
            if μ is a leaf and OUTER-FACE(μ) then
                ROOTABLE(ν) ← TRUE;
                ONE(ν) ← TRUE;
            else
                ROOTABLE(ν) ← ROOTABLE(ν) ∨ (ONE(ν) ∧ ONE(μ));
                ONE(ν) ← ONE(ν) ∨ ONE(μ);
end
TOP-DOWN(node ν) begin
    foreach incident edge e = {ν, μ} of ν do
        if μ ≠ PRED(ν) then
            ROOTABLE(μ) ← ROOTABLE(μ) ∨ (ONE(μ) ∧ ROOTABLE(ν));
            TOP-DOWN(μ);
end
begin
    choose a node r of T;
    PRED(r) ← r;
    BOTTOM_UP(r);
    TOP_DOWN(r);
end
```

and connected. Let $\nu_1, \ldots, \nu_i$ be the nodes of T in arbitrary order. We construct graphs $G_0 \subseteq \ldots \subseteq G_k$ such that

- (G_i, T, r) is c-planar and
- $G_i - V_r(\nu_i)$ is connected

We prove the existence of G_i by induction on the number k of components of $G_{i-1} - V_r(\nu_i)$. If $k = 0$ then $G_{i-1} - V_r(\nu_i)$ is connected and (G_{i-1}, T, r) is c-planar. Hence $G_i = G_{i-1}$ fulfills the required conditions.

Let $k > 1$. Let e_1, e_2 be two edges that are incident to $V_r(\nu_i)$ and to different connected components of $G_{i-1} - V_r(\nu_i)$. We may assume that e_1 and e_2 are consecutive in the cyclic order around $V_r(\nu_i)$. Let $v_1 = e_1 \cap (V \setminus V_r(\nu_i))$ and $v_2 = e_2 \cap (V \setminus V_r(\nu_i))$.

Then $G_{i-1} + \{v_1, v_2\}$ is c-planar: Let r' be the root of the smallest subtree of T containing v_1 and v_2. Since v_1 and v_2 are not contained in the same connected

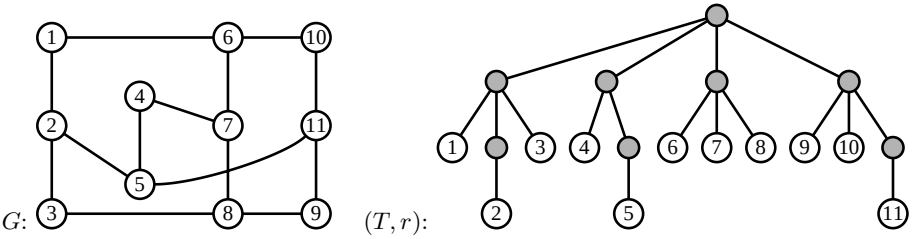


Fig. 4. A completely connected clustered graph that has no c-planar clustered subgraph which is still completely connected.

component of $G_i - V_r(\nu_i)$ but every cluster is already connected, the subtree rooted at r' also contains ν_i. Let $\mu \neq r'$ be the first node on the path from r' to ν_i. No edge crosses the cluster boundary of μ between e_1 and e_2. Else the area between e_1 and e_2 would contain vertices in $V_r(\mu) \setminus V_r(\nu_i)$ – contradicting that $V_r(\mu)$ induces a connected subgraph of G_{i-1}. Hence, we can route the edge $\{v_1, v_2\}$ along the edge e_1 the cluster boundary of μ and the edge e_2. By the choice of μ, this route crosses every cluster boundary at most once. Hence, we obtain a c-planar drawing of $G_{i-1} + \{v_1, v_2\}$.

The number of connected components of $(G_{i-1} + \{v_1, v_2\}) - V_r(\nu_i)$ is $k - 1$. Hence, we can apply the inductive hypothesis to finish the proof. □

The above proof describes a construction for obtaining a c-planar completely connected clustered graph by adding edges to a c-planar clustered graph. Note, however, that the number of additional edges depends on a c-planar drawing of the given graph and on the ordering of the nodes in the inclusion tree. The problem of minimizing the number of additional edges is a generalization of the $\mathcal{NP}$-complete problem planar biconnectivity augmentation [16].

6 Conclusion

We introduced completely connected clustered graphs, i.e. clustered graphs for which not only every cluster but also the complement of each cluster is connected. We made the surprising observation that every planar embedding of the underlying graph of a completely connected clustered graph is already a c-planar embedding. Only the outer face of the underlying graph has to be chosen according to the root of the inclusion tree or vice versa. Fixing the root (or the outer face), we gave a linear time algorithm that decides which faces can be chosen as the outer face (or which nodes can be chosen as the root). Finally, we showed that every c-planar clustered graph is a subgraph of a completely connected c-planar clustered graph.

Acknowledgments. We thank Ulrik Brandes for introducing to us the bottom-up top-down approach and Christian Fieß for implementing the algorithm that determines the rootable nodes.

References

1. F. Bertault and M. Miller. An algorithm for drawing compound graphs. In J. Kratochvíl, editor, *Proceedings of the 7th International Symposium on Graph Drawing (GD '99)*, volume 1731 of *Lecture Notes in Computer Science*, pages 197–204. Springer, 1999.
2. U. Brandes, S. Cornelsen, and D. Wagner. How to draw the minimum cuts of a planar graph. In J. Marks, editor, *Proceedings of the 8th International Symposium on Graph Drawing (GD 2000)*, volume 1984 of *Lecture Notes in Computer Science*, pages 103–114. Springer, 2001.
3. S. Cornelsen. *Drawing Families of Cuts in a Graph.* PhD thesis, Universität Konstanz, 2003. `http://www.ub.uni-konstanz.de/kops/volltexte/2003/975/`.
4. E. Dahlhaus. A linear time algorithm to recognize clustered planar graphs and its parallelization. In C. L. Lucchesi and A. V. Moura, editors, *Proceedings of the 3rd Latin American Symposium on Theoretical Informatics (LATIN '98)*, volume 1380 of *Lecture Notes in Computer Science*, pages 239–248. Springer, 1998.
5. G. Di Battista, W. Didimo, and A. Marcandalli. Planarization of clustered graphs. In M. Jünger and P. Mutzel, editors, *Proceedings of the 9th International Symposium on Graph Drawing (GD 2001)*, volume 2265 of *Lecture Notes in Computer Science*, pages 60–74. Springer, 2002.
6. P. Eades, R. F. Cohen, and Q. Feng. How to draw a planar clustered graph. In D.-Z. Du and M. Li, editors, *Proceedings of the 1st Annual International Conference on Coputing and Combinatorics (COCOON '95)*, volume 959 of *Lecture Notes in Computer Science*, pages 21–30. Springer, 1995.
7. P. Eades and Q. Feng. Multilevel visualization of clustered graphs. In North [18], pages 101–112.
8. P. Eades, Q. Feng, and X. Lin. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. In North [18], pages 146–157.
9. P. Eades, Q. Feng, and H. Nagamochi. Drawing clustered graphs on an orthogonal grid. *Journal on Graph Algorithms and Applications*, 3(4):3–29, 1999.
10. P. Eades, H. Nagamochi, and Q. Feng. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. Technical Report 98–03, Department of Computer Science and Software Engineering, University of Newcastle, Australia, 1998. Available at `ftp://ftp.cs.newcastle.edu.au/pub/techreports/tr98-03.ps.Z`.
11. Q. Feng, R. F. Cohen, and P. Eades. Planarity for clustered graphs. In P. Spirakis, editor, *Proceedings of the 3rd European Symposium on Algorithms (ESA '95)*, volume 979 of *Lecture Notes in Computer Science*, pages 213–226. Springer, 1995.
12. M. T. Goodrich and S. G. Kobourov, editors. *Proceedings of the 10th International Symposium on Graph Drawing (GD 2002)*, volume 2528 of *Lecture Notes in Computer Science*. Springer, 2002.
13. C. Gutwenger, M. Jünger, S. Leipert, P. Mutzel, M. Percan, and R. Weiskircher. Advances in c-planarity testing of clustered graphs. In Goodrich and Kobourov [12], pages 220–235.
14. M. L. Huang and P. Eades. A fully animated interactive system for clustering and navigating huge graphs. In S. H. Whitesides, editor, *Proceedings of the 6th International Symposium on Graph Drawing (GD '98)*, volume 1547 of *Lecture Notes in Computer Science*, pages 374–383. Springer, 1998.

15. M. Jünger, S. Leipert, and M. Percan. Triangulating clustered graphs. Technical Report zaik2002–444, Zentrum für Angewandte Informatik Köln, 2002. Available at `http://www.zaik.uni-koeln.de/%7Epaper/preprints.html?show=zaik2002-444`.
16. G. Kant and H. L. Bodlaender. Planar graph augmentation problems. In F. Dehne, J.-R. Sack, and N. Santoro, editors, *Algorithms and Data Structures, 2nd Workshop (WADS '91)*, volume 519 of *Lecture Notes in Computer Science*, pages 286–298. Springer, 1991.
17. D. Lütke-Hüttmann. Knickminimales Zeichnen 4-planarer Clustergraphen. Master's thesis, Universität des Saarlandes, 1999. (Diplomarbeit).
18. S. C. North, editor. *Proceedings of the 4th International Symposium on Graph Drawing (GD '96)*, volume 1190 of *Lecture Notes in Computer Science*. Springer, 1996.
19. M. Raitner. HGV: A library for hierarchies, graphs, and views. In Goodrich and Kobourov [12], pages 236–243.
20. K. Sugiyama and K. Misue. Visualization of structural information: Automativ drawing of compound digraphs. *IEEE Transactions on Systems, Man and Cybernetics*, 21:876–892, 1991.
21. X. Wang and I. Miyamoto. Generating customized layouts. In F. J. Brandenburg, editor, *Proceedings of the 3rd International Symposium on Graph Drawing (GD '95)*, volume 1027 of *Lecture Notes in Computer Science*, pages 504–515. Springer, 1996.

An FPT Algorithm for Set Splitting

Frank Dehne[1], Michael R. Fellows[2], and Frances A. Rosamond[2]

[1] Carleton University, Ottawa, Canada,
frank@dehne.net
http://www.dehne.net
[2] Univ. of Newcastle, Australia
{mfellows,fran}@cs.newcastle.edu.au

Abstract. An FPT algorithm with a running time of $O(n^4 + 2^{O(k)}n^{2.5})$ is described for the Set Splitting problem, parameterized by the number k of sets to be split. It is also shown that there can be no FPT algorithm for this problem with a running time of the form $2^{o(k)}n^c$ unless the satisfiability of n-variable 3SAT instances can be decided in time $2^{o(n)}$.

1 Introduction

Consider a collection F of subsets of a finite set X. The task is to find a partition of X into two disjoint subsets X_0 and X_1 which maximizes the number of subsets of F that are split by the partition, i.e. not entirely contained in either X_0 or X_1. This problem, called the Max Set Splitting problem, is NP-complete [9] and APX-complete [15].

Andersson and Engebretsen [3] as well as Zhang [16] presented approximation algorithms that provide solutions within a factor of 0.7240 and 0.7499, respectively. A 1/2 approximation algorithm for the special case of the Max Set Splitting problem where the size of X_0 is given was presented in [1]. A variation of the Set Splitting problem, called Max E_m Splitting, in which all sets in F contain the same number of elements m is NP-hard for any fixed $m \geq 2$ [13]. As pointed out in [16], Max E_m Splitting is a special case of Max E_m NAE-SAT. Max E_m splitting is approximable within a factor of 0.8787 for $m \leq 3$ and approximable within a factor of $\frac{1}{1-2^{1-m}}$ for $m \geq 4$ [12,17,18], and it is NP-hard for factor $\frac{1}{\frac{8}{7}-1}$ [10]. In [4] it is shown that a polynomial time approximation scheme exists for $|F| = \Theta(|X|^k)$.

In this paper, we show that the set splitting problem is *fixed parameter tractable* [8] if we consider as parameter, k, the number of sets in F that are split by a given partition of X, for arbitrary size sets in F. We present an FPT algorithm with a running time $O(n^4 + 2^{O(k)}n^{2.5})$, parameterized by the number k of sets to be split. We also show that there can be no FPT algorithm for this problem with a running time of the form $2^{o(k)}n^c$ unless the satisfiability of n-variable 3SAT instances can be decided in time $2^{o(n)}$.

There are a variety of fundamental computational problems that concern families $\mathcal{F} \subseteq 2^X$ of subsets of a base set X, such as Hitting Set (finding a

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 180–191, 2003.

small subset X' of the base set such that every set in $\mathcal{F}$ contains an element of X'), Set Packing (finding a large pairwise disjoint subfamily $\mathcal{F}'$ of $\mathcal{F}$), and Set Splitting. Important applications of these problems on families of sets arise in the analysis of *micro-array* data. Micro-array data can be viewed as a matrix, where the columns represent "features" such as whether a gene is "on" or "off", or whether a sample is cancerous or not, while the rows represent the samples. Each row can be viewed as representing a subset of the column space. An example of how Set Splitting is relevant is given by the following scenario. The set X represents a set of genetic markers (such as SNPs) of an individual. A set $A \in \mathcal{F}$ represents a combination of markers associated with a phenotypic condition that is exhibited by an individual but not by either parent. The partition of X maximizing the number of sets that are split represents a parsimonious hypothesis explaining why the phenotypic signals are not present in the parents (since for a set of markers that is split, the entire set is not present in either parent).

Since the Set Splitting is NP-complete and APX-complete, it is important to find new approaches that can solve problem instances of practical importance. Fixed parameter tractability has provided such solutions for various other NP-complete problems [8], and FPT algorithms have considerably increased the size of solvable problem instances for some of these problems [6]. In this paper, we contribute the first exploration of the parameterized complexity of one of the classic problems about families of sets, Set Splitting, parameterized by the number of sets to be split.

2 A FPT Algorithm for Set k-Splitting

A set k-splitting, $\mathcal{SSP}(X, F, k)$, for a collection F of n subsets of a finite set X is formally defined as a partition $[X_0, X_1]$ of X into two disjoint subsets X_0 and X_1 such that at least k sets in F are split by the partition. See Figure 1(a) for an illustration. For the remainder we assume, w.l.o.g., that $X = \bigcup_{S \in F} S$. Let $|X| = n$. Define a predicate $\Pi_{\mathcal{SSP}}(X, F, k)$ by $\Pi_{\mathcal{SSP}}(X, F, k) =$ TRUE if $\mathcal{SSP}(X, F, k)$ exists and $\Pi_{\mathcal{SSP}}(X, F, k) =$ FALSE if $\mathcal{SSP}(X, F, k)$ does not exist.

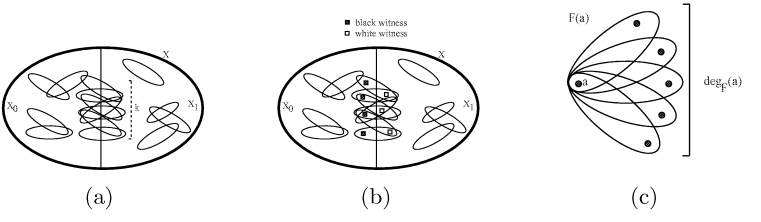


Fig. 1. (a)Illustration Of A Set k-Splitting $\mathcal{SSP}(X, F, k) = [X_0, X_1]$. (b) A k-Witness Structure. (c) Illustration Of F(a) And $\deg_F(a)$.

Definition 1. *Consider a problem instance* (X, F, k). *A sequence* $W = (b_1, \ldots, b_k, w_1, \ldots, w_k) \in X^{2k}$ *is called a* k**-witness structure** *for* F *if and only if* $(b_1, \ldots, b_k) \cap (w_1, \ldots, w_k) = \emptyset$ *and there exist* k *subsets* $S_i \in F$ *such that* $\{b_i, w_i\} \subseteq S_i$ *for all* $i = 1, \ldots, k$.

For a k-witness structure $W = (b_1, \ldots, b_k, w_1, \ldots, w_k)$, we will call b_i the *black* witness element for S_i, and w_i the *white* witness element for S_i. See Figure 1(b) for an illustration. Note that, some black [white] witness elements b_i, b_j $[w_i, w_j]$ may be identical. A witness element in a k-witness structure $W = (b_1, \ldots, b_k, w_1, \ldots, w_k)$ is called *private* if it is unique in W; otherwise it is called *shared*.

For any $W = (b_1, \ldots, b_k, w_1, \ldots, w_k)$, we refer to the process of replacing all occurrences of an element b_i or w_j by another element $b_{i'}$ or $w_{j'}$, respectively, as *deleting* b_i or w_j. A k-witness structure $W = (b_1, \ldots, b_k, w_1, \ldots, w_k)$ is *non-redundant* if any W' obtained from W by deleting any single element b_i or w_j is *not* a k-witness structure.

Observation 1 *Consider a problem instance* (X, F, k). *If there exists a* k*-witness structure then there also exists a non-redundant* k*-witness structure.*

Lemma 1. *(a) Every* k*-splitting* $\mathcal{SSP}(X, F, k) = [X_0, X_1]$ *implies at least one* k*-witness structure* $(b_1, \ldots, b_k, w_1, \ldots, w_k)$. *(b) Every* k*-witness structure* $(b_1, \ldots, b_k, w_1, \ldots, w_k)$ *implies at least one* k*-splitting* $\mathcal{SSP}(X, F, k) = [X_0, X_1]$

Proof. (a) Consider a k-splitting $\mathcal{SSP}(X, F, k) = [X_0, X_1]$ which splits k sets $S_1, \ldots, S_k$. This implies a k-witness structure $(b_1, \ldots, b_k, w_1, \ldots, w_k)$ where each b_i is an arbitrary element in $S_i \cap X_0$ and each w_i is an arbitrary element in $S_i \cap X_1$, $i = 1, \ldots, k$. (b) A k-witness structure $(b_1, \ldots, b_k, w_1, \ldots, w_k)$ implies a k-splitting $[X_0, X_1]$ where $X_0 = \{b_1, \ldots, b_k\}$ and $X_1 = X - X_0$.

For a k-witness structure $W = (b_1, \ldots, b_k, w_1, \ldots, w_k)$ let $F(W) = \{S \in F | \{b_i, w_i\} \subseteq S$ for some $1 \leq i \leq k\}$ be the collection of (at least k) sets $S \in F$ which are split by W. For each $a \in X$ let $\mathrm{F}(a) \subset F$ be the collection of sets $S \in F$, $|S| \geq 2$, which contain a, and let $\deg_F(a) = |\mathrm{F}(a)|$. See Figure 1(c) for an illustration.

Algorithm 1 *Kernelization*: Convert the given problem instance (X, F, k) into an equivalent *reduced* problem instance.

(1) Apply the following rules as often as possible.

Rule 1: IF there exists an element $a \in X$ with $\deg_F(a) > k$ THEN report $\mathcal{SSP}(X, F, k) = [\{a\}, X - \{a\}]$ and STOP.

Rule 2: IF there exists a set $S \in F$ with $|S| \leq 1$ THEN set $F \leftarrow F - \{S\}$.

Rule 3: IF there exists a set $S \in F$ with $|S| \geq 2k$ THEN set $F \leftarrow F - \{S\}$ and $k \leftarrow k - 1$.

Rule 4: IF there exists a set $S \in F$, $|S| \geq 2$, which contains an element $a \in S$ with $\deg_F(a) = 1$ THEN set $F \leftarrow F - \{S\}$ and $k \leftarrow k - 1$.

Rule 5: IF there exist three different elements $a_1, a_2, a_3 \in X$ with $\emptyset \neq \mathrm{F}(a_1) \subset \mathrm{F}(a_2) \subset \mathrm{F}(a_3)$ THEN set $S \leftarrow S - \{a_1\}$ for all $S \in F$. (Note: May need to re-apply Rule 2.)

(2) Set $X \leftarrow \bigcup_{S \in F} S$.

— End of Algorithm —

In the following, we prove the correctness of the above rules. The parts marked "($\Rightarrow$)" show that $\Pi_{\mathcal{SSP}}(X, F, k) = \text{TRUE}$ before the application of a rule implies $\Pi_{\mathcal{SSP}}(X, F', k') = \text{TRUE}$ after the application of that rule. The sections marked "($\Leftarrow$)" show the opposite direction, i.e. $\Pi_{\mathcal{SSP}}(X, F', k') = \text{TRUE}$ after the application of a rule implies $\Pi_{\mathcal{SSP}}(X, F, k) = \text{TRUE}$ before the application of that rule.

Proof Of Correctness For Rule 1. If $\deg_F(a) > k$ then $[\{a\}, X - \{a\}]$ splits k or more sets. □

Proof Of Correctness For Rule 2. Consider a set $S \in F$ with $|S| \leq 1$. ($\Rightarrow$) Since S can not be split by any $\mathcal{SSP}(X, F, k) = [X_0, X_1]$, any such $[X_0, X_1]$ also splits k sets in $F - \{S\}$. ($\Leftarrow$) If there exists a $\mathcal{SSP}(X, F - \{S\}, k) = [X_0, X_1]$ then the same $[X_0, X_1]$ is also a set k-splitting $\mathcal{SSP}(X, F, k)$. □

Proof Of Correctness For Rule 3. Consider a set $S \in F$ with $|S| \geq 2k$. ($\Rightarrow$) If $[X_0, X_1]$ is a k-splitting for F then $[X_0, X_1]$ is either a k-splitting for $F - \{S\}$ (if S is not split by $[X_0, X_1]$) or $[X_0, X_1]$ is a $k-1$-splitting for $F - \{S\}$ (if S is split by $[X_0, X_1]$). ($\Leftarrow$) Consider a $k-1$-splitting $[X_0, X_1]$ of $F - \{S\}$ with witness structure $W = (b_1, \ldots, b_{k-1}, w_1, \ldots, w_{k-1})$ by Lemma 1(a). Since $|S| \geq 2k$, there exist two elements $b_k, w_k \in S - W$. Hence $(b_1, \ldots, b_k, w_1, \ldots, w_k\}$ is a witness structure for a k-splitting of F; see Lemma 1(b). □

Proof Of Correctness For Rule 4. Consider an $S \in F$ with $|S| \geq 2$ which contains an element $a \in S$ with $\deg_F(a) = 1$. ($\Rightarrow$) If $[X_0, X_1]$ is a k-splitting for F then $[X_0, X_1]$ is a $k-1$-splitting for $F - \{S\}$. ($\Leftarrow$) Consider a $k-1$-splitting $[X_0, X_1]$ for $F - \{S\}$ with witness structure $W = (b_1, \ldots, b_{k-1}, w_1, \ldots, w_{k-1})$ by Lemma 1(a). Since $\deg_F(a) = 1$, a is not an element of W. Since $|S| \geq 2$, S contains another element $a' \neq a$. Assume $a' \in W$, w.l.o.g. $a' = b_j$ for some $1 \leq j \leq k-1$, then $W' = (b_1, \ldots, b_{k-1}, w_1, \ldots, w_{k-1}, w_k)$ with $w_k = a$ is a k-witness structure for a k-splitting of F (Lemma 1(b)). Assume $a' \notin W$, then $W'' = (b_1, \ldots, b_{k-1}, b_k, w_1, \ldots, w_{k-1}, w_k)$ with $b_k = a'$ and $w_k = a$ is a k-witness structure and implies a k-splitting of F (Lemma 1(b)). □

Proof Of Correctness For Rule 5. Consider three different elements $a_1, a_2, a_3 \in X$ with $\emptyset \neq \mathrm{F}(a_1) \subset \mathrm{F}(a_2) \subset \mathrm{F}(a_3)$. Let $F' = \{S - \{a_1\} | S \in F\}$. ($\Rightarrow$) Assume that $[X_0, X_1]$ is a k-splitting for F with *non-redundant* k-witness structure $W = (b_1, \ldots, b_k, w_1, \ldots, w_k)$. If $a_1 \notin W$ then W is also a witness structure for a

k-splitting of F'. Assume $a_1 \in W$, w.l.o.g. $a_1 = b_i$ for some $1 \leq i \leq k-1$. If both, a_2 and a_3 were contained in W then they can not both be black [white] witness elements since, otherwise, W would not be minimal (a_2 could be replaced by a_3). However, if a_2 and a_3 were both contained in W and had different colors, then W would not be minimal as well since a_1 could be replaced by either a_2 or a_3. Hence, only either a_2 or a_3 can be in W and must have a color different from a_1 (otherwise a_1 could be replaced by either a_2 or a_3). Assume, w.l.o.g. $a_2 = w_j$ for some $1 \leq j \leq k-1$. Then, W' obtained from W by replacing a_1 by a_3 is also a k-witness structure and implies a k-splitting of F' (since W' does not contain a_1). ($\Leftarrow$) Assume that $[X_0, X_1]$ is a k-splitting for F' then it is also a k-splitting for F. □

Theorem 1. *Let (X, F, k) be any reduced problem instance. If $|F| \geq 2k$ then $\Pi_{\mathcal{SSP}}(X, F, k) = TRUE$.*

Theorem 1 follows from Lemma 2, below.

Lemma 2. *"Boundary Lemma"*
If (X, F, k) is reduced and $\Pi_{\mathcal{SSP}}(X, F, k) = TRUE$ and $\Pi_{\mathcal{SSP}}(X, F, k+1) = FALSE$ then $|F| \leq 2k$.

Proof. Assume there exists a counter example to Lemma 2, that is, a reduced problem instance (X, F, k) with $\Pi_{\mathcal{SSP}}(X, F, k) =$ TRUE and $\Pi_{\mathcal{SSP}}(X, F, k+1) =$ FALSE but $|F| > 2k$. The following Claims 2 to 2 show that this leads to a contradiction.

Since $\Pi_{\mathcal{SSP}}(X, F, k) =$ TRUE, there exists a k-splitting $\mathcal{SSP}(X, F, k) = [X_0, X_1]$ which splits k sets $S_1, \ldots, S_k \in F$ and, by Lemma 1(a), there exists a k-witness structure $W = (b_1, \ldots, b_k, w_1, \ldots, w_k)$ such that $\{b_i, w_i\} \subset S_i$ for all $i = 1, \ldots, k$.

A set $S \in F$ is called *chosen* if $S = S_i$ for some $i \in \{1, \ldots, k\}$, otherwise S is called *not chosen*. Recall that any element $b_1, \ldots, b_k$ is called *black* and any $w_1, \ldots, w_k$ is called *white*. All other elements $a \in X - W$ are called *grey*.

Claim. If a set $S \in F$ is not chosen then it cannot contain both a black and a white element.

Proof. Otherwise, $\Pi_{\mathcal{SSP}}(X, F, k+1) =$ TRUE.

Claim. If a set $S \in F$ is not chosen then it consists entirely of black elements or entirely of white elements.

Proof. Assume, w.l.o.g, that S contains a black element b_i and a grey element $a \in X - W$. If we color a white, i.e. add $b_{k+1} = b_i$ and $w_{k+1} = a$ to W, then we obtain a $k+1$-witness and, hence, $\Pi_{\mathcal{SSP}}(X, F, k+1) =$ TRUE. Assume that S contains only grey elements. Due to Rule 2, S contains at least two elements a and b. If we color a white and b black, then we obtain a $k+1$-witness and, hence, $\Pi_{\mathcal{SSP}}(X, F, k+1) =$ TRUE.

Two chosen sets S_i, S_j are called *connected* if $b_i = b_j$ or $w_i = w_j$. Let $C_1, \ldots, C_r \subset F$ be a partitioning of $\{S_1, \ldots, S_k\}$ into maximal collections of connected chosen sets (i.e. connected components with respect to the above *connected* relation). We will refer to $C_1, \ldots, C_r$ as *connected components*.

A set $S \in F$ *intersects* a component C_t if there exists a chosen set $S_i \in C_t$ with $b_i \in S$ or $w_i \in S$.

Claim. If $S \in F$ is not chosen then it does not intersect two different components C_t and $C_{t'}$. (See Figure 2 for an illustration.)

Proof. Consider a set $S \in F$ that is not chosen and assume that S intersects two different components C_t and $C_{t'}$. Hence, there exists a chosen set $S_i \in C_t$ with $b_i \in S$ or $w_i \in S$ and there exists a chosen set $S_j \in C_{t'}$ with $b_j \in S$ or $w_j \in S$. If $b_i \in S$ and $w_j \in S$ then S is split by $[X_0, X_1]$ and, hence, $\Pi_{\mathcal{SSP}}(X, F, k+1) =$ TRUE. Thus, assume w.l.o.g. that $b_i \in S$ and $b_j \in S$. However, if we now invert the colors of all witnesses for chosen sets in C_t then all chosen sets in C_t are still split and S is split as well. Thus, $\Pi_{\mathcal{SSP}}(X, F, k+1) =$ TRUE; a contradiction.

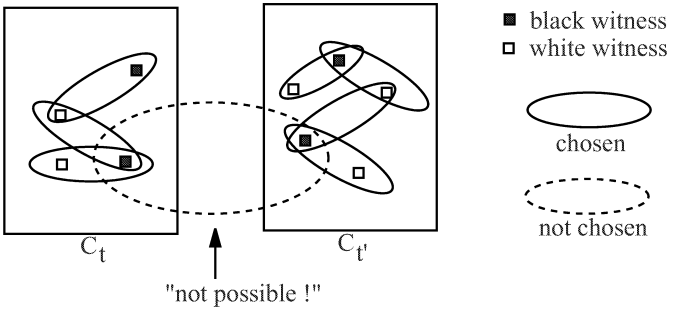


Fig. 2. Illustration of Claim 2

Claim. For any components C_t, the number of sets which intersect C_t and are chosen is larger than (or equal to) the number of sets which intersect C_t and are not chosen.

Proof. Assume that the number of sets which intersect C_t and are not chosen is larger than the number of sets which intersect C_t and are chosen. Consider the following undirected graph $G = (V, E)$ where V is the union of the witnesses $\{b_i, w_i\}$ of all sets S_i which intersect C_t and are chosen and $E = E_1 \cup E_2$ defined as follows. For each chosen set S_i which intersect C_t, E_1 contains an edge $e(S_i) = (b_i, w_i)$. For each set $S \in F$ which intersects C_t and is not chosen, E_2 contains an edge $e(S) = (a_1, a_2)$ where a_1, a_2 are two arbitrary (but different) elements of S. Note that, since (X, F, k) is reduced, $|S| \geq 2$. Furthermore, it follows from Claim 2 that S consists entirely of black elements or entirely of white elements.

Hence, a_1 and a_2 are either both black or white elements. See Figure 3 for an illustration. For any $v \in V$ let $deg_1(v)$ be the number of edges in E_1 that are incident to v and let $deg_2(v)$ be the number of edges in E_2 that are incident to v. From the assumption at the beginning of this proof it follows that $|E_2| > |E_1|$. Hence, there exists a vertex $v_0 \in V$ such that $deg_2(v_0) > deg_1(v_0)$. As a consequence, if we invert the color of the witness corresponding to v_0, we obtain a new witness structure which splits at least one more set, which implies $\Pi_{\mathcal{SSP}}(X, F, k+1) =$ TRUE; a contradiction.

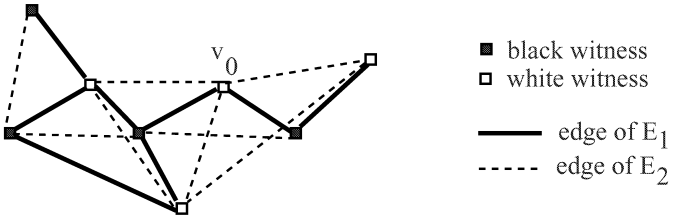


Fig. 3. Illustration of Claim 2

Claim. The number of sets which are chosen is larger than (or equal to) the number of sets which are not chosen.

Proof. Follows from Claims 2 and 2.

Since the number of chosen sets is k, it follows from Claim 2 that $F \leq 2k$; a contradiction. This concludes the proof of Lemma 2 and Theorem 1.

Let (X, F, k) be a *reduced* problem instance. If $|F| \geq 2k$ then $\Pi_{\mathcal{SSP}}(X, F, k) =$ TRUE by Theorem 1. Hence, for the remainder let us assume that $|F| < 2k$.

Since, by Rule 3 of Algorithm 1, every $S \in F$ is of size smaller than $2k$, it follows that X is of size at most $4k^2$. Thus, there are at most 2^{4k^2} possible k-witness structures which implies an $O(n^4 + n2^{4k^2})$ algorithm for set k-splitting. In the following we will show how to reduce the 2^{4k^2} term to $2^{O(k)}$.

Consider a *reduced* problem instance (X, F, k) with $|F| < 2k$. If $\Pi_{\mathcal{SSP}}(X, F, k) =$ TRUE then there exists a k-witness structure $W = (b_1, \ldots, b_k, w_1, \ldots, w_k)$. Assume that W has the largest number of private elements among all possible k-witness structures for (X, F, k). We shall call such a k-witness structure *maximal*. Let $X(W)$ denote the set of elements $a \in X$ that are contained in W. Consider the collection $F(W)$ of sets $S_i \in F$ which are split by W. Each set $S_i \in F(W)$ has a black witness b_i and a white witness w_i in W. We partition $F(W)$ into sets $A(W)$, $B(W)$, $C(W)$, $D(W)$ where $S_i \in A(W)$ if S_i has a private black witness and a private white witness, $S_i \in B(W)$ if S_i has a private black witness and a shared white witness, $S_i \in C(W)$ if S_i has a private white witness and a shared black witness, and $S_i \in D(W)$ if S_i has a

shared black witness and a shared white witness. See Figure 4 for an illustration. Let X_A be the set of all elements $a \in X$ that are contained in at least one set $S_i \in A(W)$, and let X_{BCD} be the set of all elements $a \in X$ that are contained in at least one set $S_i \in B(W) \cup C(W) \cup D(W)$.

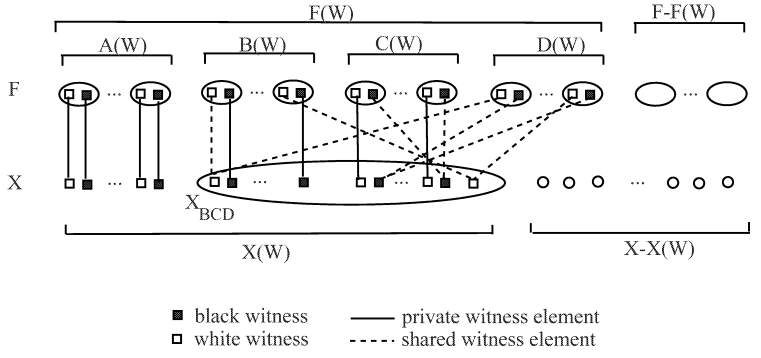


Fig. 4. Illustration of $A(W)$, $B(W)$, $C(W)$, $D(W)$ and X_{BCD}.

Lemma 3. *Consider a reduced problem instance (X, F, k) with $|F| < 2k$ and a maximal k-witness structure W, and let X_{BCD} and $X(W)$ be defined as above, then $X_{BCD} \subset X(W)$.*

Proof. Consider an element $a \in X_{BCD}$ which is not contained in $X(W)$. By assumption, a is contained in a set $S_i \in B(W) \cup C(W) \cup D(W)$. Note that, by definition, S_i has at most one private witness in W. However, since a is not in $X(W)$, we can add a to W as a witness for S_i, replacing a shared witness of S_i. This increases the number of private witnesses in W; a contradiction to the assumption that W is maximal.

For any $F' \subset F$ and $X' \subset X$ let $\hat{G}(F', X')$ be a bipartite graph with vertex set $V_{F'} \cup X'$, where $V_{F'}$ contains two elements b_S and w_S for each $S \in F'$. For every set $S \in F'$ and each element $a \in S \subset X'$, the graph $\hat{G}$ contains two edges: one edge between b_S and a, and one edge between w_S and a. A maximum matching [11] in the bipartite graph $\hat{G}(F', X')$ is called *complete* if every vertex in $V_{F'}$ is matched. For such a matching, we call those elements in X' matched to a $b_S \in V_{F'}$ the black elements and those elements in X' matched to a $w_S \in V_{F'}$ the white elements.

Algorithm 2 SET k-SPLITTING

(1) *Kernelization*

Using Algorithm 1, convert the given problem instance into an equivalent *reduced* problem instance (X, F, k). IF $|F| \geq 2k$ THEN report that $\Pi_{\mathcal{SSP}}(X, F, k) =$ TRUE and STOP.

(2) *Search*
FOR ALL collections $\{S_1, \ldots, S_k\}$ of k sets of F
FOR ALL bi-partitions of $\{S_1, \ldots, S_k\}$ into $A(W)$ and $\bar{A}(W) = B(W) \cup C(W) \cup D(W)$ such that $|X_{BCD}| \leq 2k$
FOR ALL tri-partitions of X_{BCD} into X^0_{BCD}, X^1_{BCD} and X^2_{BCD} for which $[X^0_{BCD}, X^1_{BCD}]$ splits all sets in $B(W) \cup C(W) \cup D(W)$
(Note: X^0_{BCD} represents those elements $a \in X_{BCD}$ that are black witnesses for sets in $B(W) \cup C(W) \cup D(W)$, X^1_{BCD} represents those elements $a \in X_{BCD}$ that are white witnesses for sets in $B(W) \cup C(W) \cup D(W)$, and X^2_{BCD} represents the remainder of X_{BCD}.)
IF there exists a complete matching in $\hat{G}(A(W), X_A - (X^0_{BCD} \cup X^1_{BCD}))$ with black elements X^0_A and white elements X^1_A THEN report that $[X^0_{BCD} \cup X^0_A, X^1_{BCD} \cup X^1_A]$ splits k sets in F and $\Pi_{\mathcal{SSP}}(X, F, k) =$ TRUE and STOP.
Report that $\Pi_{\mathcal{SSP}}(X, F, k) =$ FALSE.
— End of Algorithm —

Theorem 2. *Algorithm 2 solves the* SET k-SPLITTING *problem in time* $O(n^4 + 2^{O(k)}n^{2.5})$.

Proof. The time for Step 1 is bounded by $O(n^4)$. For Step 2, observe that the number of collections $\{S_1, \ldots, S_k\}$ of k sets of F is at most $4^k = 2^{2k}$ and the number of bi-partitions of $\{S_1, \ldots, S_k\}$ is at most 2^k. Since $X_{BCD} \subset X(W)$ due to Lemma 3, and $|X(W)| \leq 2k$, the number of tri-partitions of X_{BCD} is at most $3^{2k} = 2^{\frac{2\log 3}{\log 2}k}$. The computation of a maximum matching requires time $O(n^{5/2})$[11]. Thus, the time for Step 2 is bounded by $O(n^{5/2}2^{(3+\frac{2\log 3}{\log 2})k})$.

3 Optimality

In this section we employ newly developed methods of parameterized complexity analysis to prove "exponential optimality" for our main FPT result in Section 2.

To illustrate the issue, consider some recent results for the k-VERTEX COVER problem. An FPT algorithm of the form $2^{O(k)}$, based on search trees, was first described by Mehlhorn [14]. In [2] it was shown that the PLANAR k-VERTEX COVER problem can be solved in time $2^{O(\sqrt{k})}n$. This raises two natural questions: (1) Is there an FPT algorithm for the general k-VERTEX COVER problem with running time $2^{O(\sqrt{k})}n^c$? (2) Can the FPT algorithm for the PLANAR k-VERTEX COVER problem be further improved? For example, is an algorithm with time $2^{O(k^{1/3})}n^c$ possible? It has been shown that the answers to both questions is "no" [5]. There is no FPT algorithm with a running time of the form $2^{o(k)}n^c$ for the general k-VERTEX COVER PROBLEM, and there is no FPT algorithm with a running time $2^{o(\sqrt{k})}n^c$ for the PLANAR k-VERTEX COVER problem unless

there is an unlikely collapse $FPT = MINI[1]$ in the hierarchy of parameterized complexity classes

$$FPT \subseteq MINI[1] \subseteq W[1].$$

Such a collapse is considered unlikely because because $FPT = MINI[1]$ if and only if n-variable 3SAT can be solved in time $2^{o(n)}$ [7,5].

In the remainder of this section we establish a similar result for the SET k-SPLITTING problem $\mathcal{SSP}(X, F, k)$.

Theorem 3. *There is no FPT algorithm for* SET k-SPLITTING *with running time* $2^{o(k)}n^c$ *unless* $FPT = MINI[1]$.

Proof. It is sufficient to show that the following problem is hard for $MINI[1]$:

MINI SET SPLITTING
Input: Integers k, n in unary format, a family $\mathcal{F} \subseteq 2^X$ where $|\mathcal{F}| \leq k \log n$, and an integer r. Parameter: k. Question: Is there a bipartition of X that splits at least r sets of F.

Proving that MINI SET SPLITTING is hard for $MINI[1]$ is sufficient to establish our theorem because (1) If there is a $2^{o(s)}n^c$ algorithm to determine if s sets can be split then this algorithm can be used to determine in time $2^{o(k \log n)}n^c$ if $r \leq k \log n$ sets can be split. (2) If $g(n, k) = o(k \log n)$ for any fixed k, i.e. $lim_{n\to\infty} \frac{f(k,n)}{k \log n} = 0$ for any fixed k, then $2^{f(k,n)}n^c$ is bounded by $g(k)n^{c'}$ for appropriately chosen constant c' and function $g(k)$.

To show that MINI SET SPLITTING is $MINI[1]$ hard, we reduce from the $MINI[1]$-complete problem MINI-3SAT [7]

MINI 3SAT
Input: Integers k, n in unary format, a 3SAT expression $\mathcal{E}$ where $\mathcal{E}$ has at most $k \log n$ variables and $k \log n$ clauses. Parameter: k. Question: Is $\mathcal{E}$ satisfiable.

The standard reduction from 3SAT to the variant NOT-ALL-EQUAL 3SAT FOR ALL POSITIVE LITERALS is, in fact, a linear-size reduction. That is, the expression $\mathcal{E}'$ to which $\mathcal{E}$ is transformed satisfies $|\mathcal{E}'| \leq c|\mathcal{E}|$ for some constant c. This yields immediately an FPT reduction from MINI 3SAT to MINI NOT-ALL-EQUAL 3SAT FOR ALL POSITIVE LITERALS. The latter is a special case of MINI SET SPLITTING (where all sets in the family $\mathcal{F}$ have size 3).

4 Conclusion

In this paper, we have presented the first exploration of the parameterized complexity of one of the classic problems about families of sets, SET SPLITTING, parameterized by the number of sets to be split. We have presented an FPT algorithm with a running time of $2^{O(k)}n^{2.5}$ and shown that there can be no FPT algorithm for this problem with a running time of the form $2^{o(k)}n^c$ unless the satisfiability of n-variable 3SAT instances can be decided in time $2^{o(n)}$.

The "final" goal of FPT methods is to increase the size of solvable problem instances for NP-complete problems like Set Splitting. In this context, an important open question is whether the $2^{(3+\frac{2\log 3}{\log 2})k}$ term in the running time of Algorithm 2 can be further reduced to some 2^{ck} with $c < 3 + \frac{2\log 3}{\log 2}$. We emphasize that this is a first study of the parameterized complexity of Set Splitting. For a practical implementation, our algorithm will probably also require more reduction rules in order to "shrink" problem instances as much as possible during kernelization and thereby further increase the size of solvable problem instances.

References

1. A.A. Ageev and M.I. Sviridenko. An approximation algorithm for hypergraph max k-cut with given sizes of parts. In *Proc. ESA 2000*, volume 1879 of *Lecture Notes in Computer Science*, pages 32–41, 2000.
2. J. Alber, H.L. Bodlaender, H. Fernau, and R. Niedermeier. Fixed parameter algorithms for planar dominating set and related problems. In *Proc. Scandinavian Workshop on Algorithm Theory (SWAT 2000)*, Lecture Notes in Computer Science, pages 97–110, 2000.
3. G. Andersson and L. Engebretsen. Better approximation algorithms and tighter analysis for set splitting and not-all-equal sat. In *ECCCTR: Electronic Colloquium on Computational Complexity*, 1997.
4. S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. In *Proc. ACM STOC*, pages 284–293, 1995.
5. L. Cai and D. Juedes. Subexponential parameterized algorithms collapse the w-hierarchy. In *Proc. ICALP 2001*, volume 2076 of *Lecture Notes in Computer Science*, 2001.
6. J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, and P. J. Taillon. Solving large FPT problems on coarse grained parallel machines. *Journal of Computer and System Sciences*. to appear.
7. R.D. Downey, V. Estivill-Castro, M.R. Fellows, E. Prieto-Rodriguez, and F.A. Rosamond. Cutting up is hard to do: The parameterized complexity of k-cut and related problems. In *Proc. Computing: The Australasian Theory Symposium (CATS 2003), to appear*, Elsevier Electronic Notes in Computer Science, Adelaide, 2003.
8. R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1998.
9. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
10. J. Hastad. Some optimal inapproximability results. In *Proc. ACM STOC*, pages 1–10, 1997.
11. J.E. Hopkroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM J.Comput.*, 2:225–231, 1973.
12. V. Kann, J. Lagergren, and A. Panconesi. Approximability of maximum splitting of k-sets and some other apx-complete problems. *Information Processing Letters*, 58(3):105–110, 1996.
13. L. Lovasz. Coverings and colorings of hypergraphs. In *Proc. 4th Southeastern Conf. on Combinatorics, Graph Theory, and Computing*, pages 3–12, 1973.

14. K. Mehlhorn. *Data Structures And Algorithms*. Springer Verlag, 1990.
15. E. Petrank. The hardness of approximation: Gap location. *Computational Complexity*, 4:133–157, 1994.
16. H. Zhang and C. X. Ling. An improved learning algorithm for augmented naive Bayes. In *Proc. Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, volume 2035 of *Lecture Notes in Computer Science*, pages 581–586, 2001.
17. U. Zwick. Approximation algorithms for constraint satisfaction problems involving at most three variables per constraint. In *Proc. SODA*, pages 201–220, 1998.
18. U. Zwick. Outward rotations: A tool for rounding solutions of semidefinite programming relaxations, with applications to max cut and other problems. In *Proc. ACM STOC*, pages 679–687, 1999.

Drawing Planar Graphs on a Curve*

E. Di Giacomo[1], W. Didimo[1], G. Liotta[1], and S.K. Wismath[2]

[1] Dipartimento di Ingegneria Elettronica e dell'Informazione,
Università degli Studi di Perugia, Perugia, Italy
{digiacomo,didimo,liotta}@diei.unipg.it
[2] Department of Mathematics and Computer Science, University of Lethbridge,
Alberta, Canada (financial support from NSERC is gratefully acknowledged)
wismath@cs.uleth.ca

Abstract. This paper introduces and studies the concept of a *curve embedding* of a planar graph. Let $\mathcal{C}$ be the family of 2D curves described by concave functions and let G be a planar graph. A curve embedding of G is a linear ordering of the vertices of G such that there exists a crossing-free 2D drawing of G where the vertices are constrained to be on any given curve of $\mathcal{C}$ and the edges are drawn as polylines with at most one bend. We prove that every planar graph has a curve embedding which can be computed in linear time. Further we present applications of the concept of curve embedding to upward drawings and point-set constrained drawings.

1 Introduction

Motivated by the many applications that impose geometric constraints on the position of the vertices, a variety of papers have been published in the graph drawing literature that investigate the problem of computing drawings where vertices are constrained to be on given curves or straight lines in 2D space (see, e.g., [2,7,10,12,15]). Further references can be found in [9,13].

In this paper we investigate the problem of drawing a graph in the plane with the additional constraint that vertices are represented as points of a given 2D curve. Since the class of graphs that can be drawn under such a strong constraint on the vertex positions is small if the edges are required to be straight-line segments (it coincides with the class of outerplanar graphs), it makes sense to consider edges drawn as polylines and to keep the number of bends along each edge as small as possible. Besides being of theoretical interest on its own right, our results have applications to upward drawability problems and to point-set embeddability problems [3,4,14]. Furthermore, the subject of this paper naturally relates with the concept of *2D-book embedding* [1] of planar graphs. A 2D-book embedding of a planar graph G can be defined as a linear ordering of its vertices such that there exists a planar drawing of G such that: (i) the vertices are points on a straight line (the *spine* of the drawing) and follow the given ordering; (ii) and

* Research partially supported by "Progetto ALINWEB: Algoritmica per Internet e per il Web", MIUR Programmi di Ricerca Scientifica di Rilevante Interesse Nazionale.

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 192–204, 2003.

the edges are polylines with at most one bend, i.e. each edge is represented by at most two consecutive straight-line segments. The family of planar graphs that have a 2D-book embedding coincides with planar sub-hamiltonian graphs [1]. The main contributions in this paper can be outlined as follows.

- We introduce and study the concept of a *curve embedding* of a planar graph (see Section 2). Let $\mathcal{C}$ be the family of 2D curves described by concave functions and let G be a graph. A curve embedding of G is a linear ordering of the vertices of G such that there exists a crossing-free 2D drawing of G where the vertices lie on any given curve of $\mathcal{C}$ according to the linear ordering and the edges are polylines with at most one bend. Informally, a curve embedding can be seen as a 2D-book embedding where the spine is "bent". Figure 1 shows a planar graph G, a linear ordering L of the vertices of G and a crossing-free drawing of G on a semi-circle where vertices appear according to L and edges have at most one bend.
- We show that every planar graph G has a curve embedding and that such an ordering for its vertices can be computed in linear time (see Section 3). Furthermore, we show that if the curve of $\mathcal{C}$ is a semi-circle then a drawing of G that corresponds to the curve embedding can also be computed in linear time. Recall that deciding whether a given planar graph has a 2D-book embedding is NP-complete [1].
- We study the interplay between drawing a planar graph with all vertices on a curve and at most one bend per edge, and drawing it with all vertices on a straight line and with at most two bends per edge (see Subsection 4.1). Namely, we show that a curve embedding can be used to compute an orientation of a planar graph that admits an upward planar drawing where all vertices are collinear and every edge has at most two bends. Both the orientation and the drawing can be computed in linear time.
- We present an application of curve embeddings to the problem of computing a drawing of a graph where the vertices are constrained to be mapped on a given set of n distinct points (see Subsection 4.2). Kaufmann and Wiese [14] showed how to solve the problem for all planar graphs and with at most two bends per edge. The algorithm in [14] first computes a hamiltonian augmentation of the input graph via results due to Chiba and Nishizeki [5, 6] by four-connecting the graph and then uses the augmented ordering to construct the drawing. Curve embeddings are used to define a variant of the algorithm by Kaufmann and Wiese where a hamiltonian augmented graph is computed without initially four-connecting the graph, and so that every edge of the drawing is monotone in one direction.

For reasons of space, some proofs are omitted.

2 Preliminary Definitions

Let G be a graph. A *drawing* Γ of G maps each vertex v of G to a distinct point $p(v)$ of the plane and each edge $e = (u, v)$ of G to a simple Jordan curve connecting $p(u)$ and $p(v)$. Drawing Γ is *planar* if no two distinct edges intersect

except at common endvertices. Graph G is *planar* if it admits a planar drawing. A planar drawing Γ of G partitions the plane into topologically connected regions called the *faces* defined by Γ. The unbounded face is called the *external face*.

An *embedding* of a planar graph G is an equivalence class of planar drawings that define the same set of faces, that is, the same set of face boundaries. A planar graph G together with the description of a set of faces F is called an *embedded planar graph*. A *maximal* embedded planar graph is such that all faces are triangles, that is, the boundary of each face has three vertices and three edges. Given any embedded planar graph G it is easy to add edges that split the faces of G, so to obtain a maximal embedded planar graph that includes G.

Definition 1. *A* concave curve Λ *is a function* $f(x) : I = [\alpha, \beta] \subset \mathbb{R} \to \mathbb{R}$ *such that* $\forall x \in I$ $f(x)$ *has a second derivative on* I *and* $f''(x) < 0$.

Let $\alpha \leq x_1 < x_2 \leq \beta$; we say that $(x_1, f(x_1))$ *is before* $(x_2, f(x_2))$ and $(x_2, f(x_2))$ *is after* $(x_1, f(x_1))$.

Definition 2. *Let* G *be a planar graph and let* Λ *be a concave curve. A* curve drawing *of* G *on* Λ *is a planar drawing of* G *such that (see Figure 1(b)):*

- *each vertex* v *of* G *is mapped to a unique point* $p(v)$ *of* Λ*;*
- *each edge* $e = (u, v)$ *of* G *is drawn as a polyline with endpoints* $p(u)$ *and* $p(v)$*; a* bend *of* e *is a common point of two consecutive segments of the polyline representing* e*.*

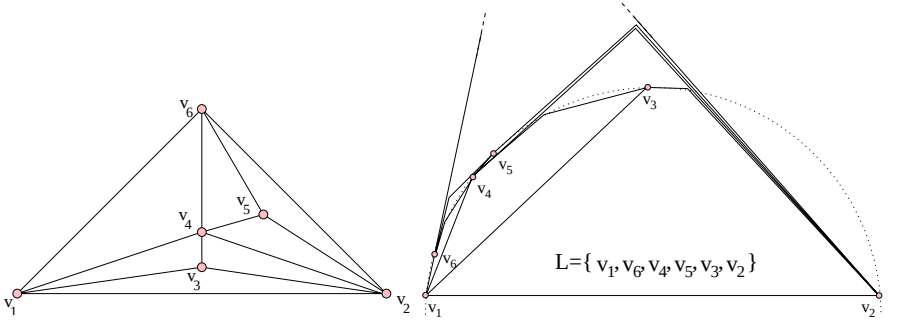


Fig. 1. (a) A planar graph G. (b) A curve drawing of G on a semi-circle. L is the curve embedding induced by the curve drawing.

Definition 3. *A* curve embedding *of* G *is a linear ordering* L *of the vertices of* G *such that:*

- G *has a curve drawing* Γ *on any concave curve* Λ*;*
- *if* u *precedes* v *in* L *then* $p(u)$ *is before* $p(v)$ *in* Γ*;*
- *each edge in* Γ *has at most one bend.*

The linear ordering L *is the curve embedding* induced *by* Γ.

Let p and q be two points in the plane. We denote by $\overline{pq}$ the straight-line segment connecting p and q. Given any point p of Λ, let $\tau(\Lambda, p)$ be the tangent to Λ in p. Let $C = \{c_1, \ldots, c_s\}$ be a set of points on Λ, such that c_{i+1} is after c_i $(1 \leq i \leq s-1)$, and let $\hat{c}_i$ denote the intersection point between $\tau(\Lambda, c_i)$ and $\tau(\Lambda, c_{i+1})$.

Definition 4. *The* tangent polyline of Λ with respect to C *is the polyline* $\tau(\Lambda, C) = \overline{c_1\hat{c}_1} \cup \overline{\hat{c}_1 c_2} \cup \overline{c_2\hat{c}_2} \cup \overline{\hat{c}_2 c_3} \cup \cdots \cup \overline{c_{s-1}\hat{c}_{s-1}} \cup \overline{\hat{c}_{s-1}c_s}$.

Definition 5. *The region $P(\Lambda, c_i)$ $(c_i \in C)$ is the closed half-plane defined by $\tau(\Lambda, c_i)$ and containing Λ. The region $P(\Lambda, C)$ is the intersection of all $P(\Lambda, c_i)$ $(1 \leq i \leq s)$.*

Since each $P(\Lambda, c_i)$ contains Λ, therefore $P(\Lambda, C)$ contains Λ, too. Also the following properties hold.

Property 1. $\tau(\Lambda, C)$ is contained in the boundary of $P(\Lambda, C)$.

Property 2. $\overline{c_1 c_s}$ is contained in $P(\Lambda, C)$.

Definition 6. *A point p on Λ is* externally visible *if the straight line orthogonal to $\tau(\Lambda, p)$ at p does not intersect any point of Γ in the region $\mathbb{R}^2 - P(\Lambda, p)$. An interval $I = [\alpha, \beta]$ on Λ is* externally visible *if all points of I are externally visible.*

Definition 7. *Let y_M be the maximal y-coordinate of any point of Γ, and let e and e' be any two distinct edges of G. We say that e is* externally covered *by e' in Γ if for each point p with y-coordinate greater than y_M and for each point q of e, the segment $\overline{pq}$ intersects e'.*

In Section 3 we describe an algorithm that computes a curve embedding of a planar graph. This algorithm makes use of a particular ordering of the vertices introduced by de Fraysseix *et al.*, which is known as *canonical ordering* [8]. We recall here its definition. An example of canonical ordering of a maximal plane graph is shown in Figure 1(a).

Definition 8 (Canonical Ordering.). [8] *Let G be a maximal embedded planar graph with external boundary u, v, w. A* canonical ordering *of G with respect to u, v is an ordering of the vertices $v_1 = u, v_2 = v, v_3, \ldots, v_n = w$ of G with the following properties for every $4 \leq k \leq n$:* **(i)** *The subgraph $G_{k-1} \subseteq G$ induced by $v_1, v_2, \ldots, v_{k-1}$ is biconnected and the boundary C_{k-1} of its external face contains edge (u, v);* **(ii)** *v_k is in the external face of G_{k-1}, and its neighbors in G_{k-1} form a subinterval of the path $C_{k-1} - (u, v)$.*

In [8] it is also proved that a canonical ordering of a maximal embedded planar graph can be computed in $O(n)$ time. We introduce some further notation that will be subsequently used. Let $l(p, q)$ denote the straight line passing through p and q. The *perpendicular bisector* of the segment $\overline{pq}$ is the straight line orthogonal to $\overline{pq}$ and passing through the middle point of $\overline{pq}$. Given a straight line or a segment l that forms an angle θ with the x-axis, we denote by $\sigma(l) = \tan(\theta)$ the slope of l.

3 Computing Curve Embeddings

In this section we give an algorithm that finds a curve embedding of a planar graph G in linear time. We first describe a strategy to compute a curve drawing of G on a concave semi-circle with at most one bend per edge, and then we show how the same strategy can be used to compute a curve drawing on any concave curve, maintaining the linear ordering of the vertices along the curve. This ordering is the desired curve embedding. Without loss of generality, we concentrate on maximal planar graphs and first outline the invariant properties to be maintained by our algorithm.

Let G be a maximal embedded planar graph with external boundary u, v, w, and let $u = v_1, v = v_2 \ldots, v_n = w$ be a canonical ordering of G with respect to u, v. Let G_k be the subgraph of G induced by $v_1, \ldots, v_k$ and let $C_k : u = c_1, \ldots, c_s = v$ be the boundary of the external face of G_k. Let Γ_k be a curve drawing of G_k on a concave curve Λ, such that no vertex is drawn on the endpoints of Λ and the following properties hold:

- P1 $\forall c_i \in C_k$, all edges of G_k incident on c_i are drawn in Γ_k inside $P(\Lambda, c_i)$;
- P2 The edges of the external boundary of Γ_k are the same as those of C_k and they appear in the same circular clockwise order as in C_k;
- P3 Edge (u, v) has 0 bends. Each edge $e = (a, b) \neq (u, v)$ of Γ_k has either 0 or 1 bend, according to the following rules: **(i)** if e has 0 bends then $|e \cap \Lambda| = 2$, i.e., e intersects Λ only at its endpoints a and b; **(ii)** if e has 1 bend then $|e \cap \Lambda| = 4$. Denote by $a = z_0, z_1, z_2, z_3 = b$ the four intersection points between Λ and e; it must be that z_{i+1} is after z_i $(i = 0, \ldots, 2)$. We call z_1 and z_2 the *first crossing* and the *second crossing* of e, respectively.

Lemma 1. *Γ_k is contained in $P(\Lambda, C_k)$.*

Lemma 2. *For each vertex c_i on the external face of Γ_k, there exists on Λ a neighborhood of c_i, $I_{c_i} = (\alpha_{c_i}, \beta_{c_i})$ that is externally visible.*

Corollary 1. *The edges of the external face of Γ_k define a convex polygon $P(\Gamma_k)$ such that: (i) Γ_k is inside $P(\Gamma_k)$ and (ii) $P(\Gamma_k)$ is contained in $P(\Lambda, C_k)$.*

3.1 Drawing on a Semi-circle

Lemma 3. *Let G be a maximal embedded planar graph and let Λ be a concave semi-circle. There exists a curve drawing of G on Λ with at most one bend per edge.*

Proof. Let $f(x)$ be the function describing Λ, and let $f(x)$ be defined in an interval $I = [\alpha, \beta]$. Let u, v, and w be the vertices on the external face of G, and let $v_1 = u, v_2 = v, \ldots, v_n = w$ be a canonical ordering of G with respect to u and v. We construct the drawing iteratively by adding one vertex per step. In step k $(1 \leq k \leq n)$ vertex v_k is added, along with all the edges connecting v_k to

vertices of C_{k-1}. The drawing obtained at the end of step k is denoted by Γ_k. We prove by induction on k ($1 \leq k \leq n$) that v_k can be added to Γ_{k-1} so that Γ_k is planar and maintains Properties P1, P2, and P3.

Base case: Vertices v_1 and v_2 are placed on two points $(x_1, f(x_1))$ and $(x_2, f(x_2))$ such that $\alpha < x_1 < x_2 < \beta$. Edge (v_1, v_2) is drawn as a straight-line segment. Clearly, at this step the drawing is planar (it only consists of one edge) and Properties P1, P2, and P3 hold.

Inductive case: Suppose by induction that Γ_{k-1} ($k > 2$) is planar and verifies Properties P1, P2, and P3. Let $w_1, \ldots, w_h$ be the vertices of C_{k-1} that are connected to v_k, in the clockwise order they appear in C_{k-1}.

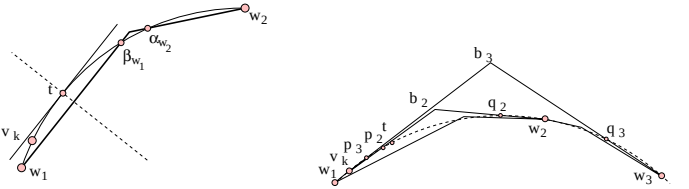


Fig. 2. (a) Placement of vertex v_k. (b) Drawing of the edges incident on v_k

Construction of Γ_k: By Lemma 2 vertex w_1 has a neighborhood $I_{w_1} = (\alpha_{w_1}, \beta_{w_1})$ that is externally visible. Namely, by Property P2, β_{w_1} coincides with w_2 if the edge (w_1, w_2) is straight-line, otherwise β_{w_1} coincides with the first crossing of (w_1, w_2). Vertex v_k is placed in I_{w_1} as follows. Denote with t the point in Λ between w_1 and β_{w_1} such that the tangent to Λ in t is parallel to $\overline{w_1\beta_{w_1}}$ (see Figure 2(a)). This point is the intersection point between Λ and the perpendicular bisector of the segment $\overline{w_1\beta_{w_1}}$. Draw v_k anywhere on Λ between w_1 and t. We now add edges $e_1 = (v_k, w_1)$, $e_2 = (v_k, w_2)$, $\ldots$, $e_h = (v_k, w_h)$. Edge e_1 is drawn as a straight-line segment connecting w_1 and v_k. Let $p_2, p_3, \ldots, p_h$ be $h-1$ points between v_k and t on Λ, with p_{i-1} after p_i ($3 \leq i \leq h$). By Lemma 2 each vertex w_i ($2 \leq i \leq h$) has a neighborhood $I_{w_i} = (\alpha_{w_i}, \beta_{w_i})$ that is externally visible. Namely, α_{w_i} coincides with w_{i-1} if edge (w_{i-1}, w_i) is straight-line, otherwise α_{w_i} coincides with the second crossing of (w_{i-1}, w_i). For each w_i ($2 \leq i \leq h$), let q_i be any point on Λ between α_{w_i} and w_i. Let b_i be the intersection point between the straight lines $l(v_k, p_i)$ and $l(w_i, q_i)$ ($2 \leq i \leq h$). Edge e_i ($2 \leq i \leq h$) is drawn as the polyline $\overline{v_k b_i} \cup \overline{b_i w_i}$, where b_i is the bend of e_i (see Figure 2(b)).

Proof of the planarity of Γ_k: All edges $e_1, \ldots, e_h$ are outside $P(\Gamma_{k-1})$ except for their end-vertices, which lie on the boundary of $P(\Gamma_{k-1})$. Namely, each edge e_i ($2 \leq i \leq h$) consists of two segments $\overline{v_k b_i}$ and $\overline{b_i w_i}$. Since $\sigma(\overline{v_k b_i}) > \sigma(\tau(\Lambda, t))$ then $\sigma(\overline{v_k b_i}) > \sigma(\overline{w_1 \beta_{w_1}})$ (see Figure

3(a)). Also the straight line $l(w_1, \beta_{w_1})$ contains a side of $P(\Gamma_{k-1})$ and $P(\Gamma_{k-1})$ is convex; hence $\overline{v_k b_i}$ is outside $P(\Gamma_{k-1})$, except for the point v_k. Analogously, $\sigma(\overline{b_i w_i}) < \sigma(\overline{\alpha_{w_i} w_i})$ because point α_{w_i} is before q_i. Since the straight line $l(\alpha_{w_i}, w_i)$ contains a side of $P(\Gamma_{k-1})$ and $P(\Gamma_{k-1})$ is convex therefore $\overline{b_i w_i}$ is outside $P(\Gamma_{k-1})$, except for the point w_i. Finally, edge e_1 is a chord between w_1 and v_k, and v_k is before w_2; it follows that $\sigma(\overline{w_1 v_k}) > \sigma(l(w_1, \beta_{w_1}))$, and hence e_1 is outside $P(\Gamma_{k-1})$, except for its end-vertices. Therefore each edge e_i $(i = 1, \ldots, h)$ does not intersect any edge of Γ_{k-1}, except at common end-vertices.

Furthermore, all edges $e_1, \ldots, e_h$ do not intersect each other. In fact, for each pair e_i, e_{i+1} $(i = 1 \ldots, h)$ we have that: (i) $\sigma(\overline{v_k b_i}) < \sigma(\overline{v_k b_{i+1}})$; (ii) $\sigma(\overline{b_i w_i}) > \sigma(\overline{b_{i+1} w_{i+1}})$; (iii) w_i is before w_{i+1}. These conditions imply that edge e_i is externally covered by e_{i+1} (see Figure 3(b)).

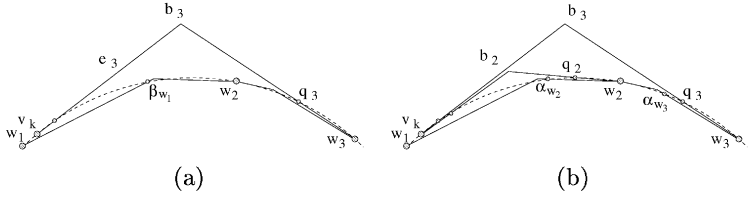


Fig. 3. (a) Edge e_3 does not cross any existing edge. (b) Edge e_2 and e_3 do not cross each other.

Proof of Properties P1, P2, and P3 for Γ_k: Denoting $C_{k-1} : c_1, c_2, \ldots, c_l = w_1, c_{l+1} = w_2, \ldots, c_r = w_h, \ldots, c_s$, observe that $C_k : c_1, c_2, \ldots, c_l = w_1, v_k, c_r = w_h, \ldots, c_s$. Since none of the edges added in step k is incident to vertices $c_1, \ldots, c_{l-1}$ and $c_{r+1}, \ldots, c_s$, then Property P1 holds for these vertices by the inductive hypothesis. We prove Property P1 for vertices $c_l = w_1$, v_k, and $c_r = w_h$. Observe that given any two points z and z_1 on Λ, each segment containing the chord $\overline{z z_1}$ and having z as endpoint lies in $P(\Lambda, z)$. All the edge segments that are incident on $z \in \{w_1, v_k, w_h\}$ have z as endpoint and contain a chord $\overline{z z_1}$. Thus Property P1 holds for vertices w_1, v_k, and w_h. About Property P2 we observe that the portion of the boundary of the external face of Γ_k from vertex c_1 to vertex $c_l = w_1$ (in clockwise order) is the same as Γ_{k-1}. The next edge of C_k encountered in the clockwise order is (w_1, v_k). This edge is on the external face of Γ_k. Namely, edge (w_1, v_k) is outside the polygon $P(\Gamma_{k-1})$ and it is not externally covered by any other edge e_i $(2 \leq i \leq h)$ because all the first crossings of these edges are after v_k. The next edge of C_k encountered in the clockwise order is (v_k, w_h). Also this edge is on the external face of Γ_k since it is outside $P(\Gamma_{k-1})$ and all the other edges e_i $(2 \leq i \leq h-1)$ are externally covered by it. Finally the portion of the

boundary of the external face of Γ_k from vertex $c_r = w_h$ to vertex c_s (in clockwise order) is the same as Γ_{k-1}. Property P3 holds as an immediate consequence of the construction.

□

3.2 Curve Embedding

The drawing procedure described in the proof of Lemma 3 can be extended to general concave curves. Indeed, the proof of Lemma 3 does not rely upon the fact that Λ is a semi-circle, except when point t is computed in order to establish where v_k must be placed. However, a point with the same property as t exists also on any concave curve. Namely, let $f(x)$ be the function representing Λ and consider a chord $\overline{(a, f(a))(b, f(b))}$. By Lagrange's Theorem there exists a point c in $[a, b]$ such that $f'(c) = (f(b) - f(a))/(b - a)$; that is, the tangent in $(c, f(c))$ on Λ is parallel to $\overline{(a, f(a))(b, f(b))}$. Therefore we can compute t by assuming $(a, f(a)) = w_1$, and $(b, f(b)) = \beta_{w_1}$, and by observing that $f'(x)$ is invertible since it is a monotone decreasing function. Hence, Lemma 3 is easily extended as follows.

Lemma 4. *Let G be a maximal planar graph and let Λ be a concave curve. There exists a curve drawing of G on Λ with at most one bend per edge.*

Since the linear ordering of the vertices along the curve computed by the above procedure does not depend on the choice of the curve, we can conclude that such an ordering is a curve embedding of G. It is easy to see that the Algorithm described in the proof of Lemma 3 can be implemented to run in $O(n)$ time when Λ is a semi-circle. The following theorem summarizes the main contribution of this section.

Theorem 1. *Let G be a planar graph with n vertices. There exists an $O(n)$-time algorithm that computes a curve embedding of G. Also, a curve drawing of G on a semi-circle can be computed in $O(n)$ time.*

The time complexity of computing a curve drawing of G on a general concave curve Λ depends on the time complexity of computing point t in the construction of Γ_k. In particular, if the inverse function of $f'(x)$ is known, the curve drawing can also be computed in $O(n)$ time.

In the following we call `curveDrawer` the algorithm of Theorem 1 that computes a curve drawing of G on a semi-circle.

4 Upward and Point-Set Constrained Drawings

In this section we study the interplay between a curve drawing and a drawing where all vertices are collinear and each edge has at most two bends. The relationship between the two types of drawings is used to prove new results on two well-studied graph drawing topics.

4.1 Upward Drawability

Let G be a planar graph and let L be a linear ordering of its vertices. Ordering L is a *spine embedding* of G (also called *two-page topological book embedding*[11]) if there exists a crossing-free drawing Σ of G with the following properties: (i) The vertices of G are represented in Σ as points that lie on a straight line and that respect the ordering L; and (ii) each edge of G is represented in Σ as a polyline with at most two bends. Drawing Σ is a *spine drawing* of G and the straight line on which the vertices of Σ lie is the *spine*. A spine drawing is *nice* if all its edges are monotone in a common direction. A spine embedding is said to be *nice* if it gives rise to a nice spine drawing. Figure 4(b) shows a nice spine embedding of the graph of Figure 1(a).

The following lemma shows the relationship between curve embeddings and spine embeddings.

Lemma 5. *Let G be a planar graph. The curve embedding computed by Algorithm* `curveDrawer` *is a nice spine embedding of G. A nice spine drawing of G can be computed in $O(n)$ time.*

Proof. Let Λ be a semi-circle, let Γ be the curve drawing constructed by Algorithm `curveDrawer` on Λ, and let L be the corresponding curve embedding. In order to prove that L is a nice spine embedding of G, we describe an algorithm that computes a nice spine drawing Σ of G where the linear ordering of the vertices along the spine coincides with L. We assume, without loss of generality, that the spine is horizontal and that its y-coordinate is zero. Also, we denote by $x(p)$ and $y(p)$ the x-coordinate and the y-coordinate of a point p in 2D.

We recall that each edge $e = (v, w)$ with one bend in Γ has two crossings with Λ distinct from its end-vertices, called the first and second crossings and denoted by z_1 and z_2, respectively. We say that z_1 is an *unavoidable crossing* if there is some vertex between v and z_1 and another vertex between z_1 and w along Λ. Let $p_1, p_2, \ldots, p_m$ be the sequence of vertices and unavoidable crossings as they appear in Γ along Λ. Each portion of edge δ_{ij} connecting p_i to p_j $(1 \leq i \neq j \leq m)$ is called *subedge*. Observe that a subedge may coincide with the whole edge. The algorithm is as follows.

Step 1. Draw $p_1, p_2, \ldots, p_m$ as points on the spine in this order, so that $x(p_{i+1}) - x(p_i) = 1$, $(1 \leq i < m)$

Step 2. For all i, j such that $(1 \leq i \neq j \leq m)$, draw subedge δ_{ij} as the polyline $\overline{p_i, p} \cup \overline{p, p_j}$, where $x(p) = (x(p_i) + x(p_j))/2$ and:

- If δ_{ij} is straight-line in Γ (δ_{ij} is either an edge or the subedge connecting vertex p_i to the unavoidable crossing p_j), let $y(p) = -(x(p_j) - x(p_i))/2$;
- If δ_{ij} is a polyline with one bend in Γ and p_i is an unavoidable crossing then let $y(p) = (x(p_j) - x(p_i))/2$;
- If δ_{ij} is a polyline with one bend in Γ and p_i is not an unavoidable crossing (δ_{ij} is an edge) then: **(i)** let $y(p) = (x(p_j) - x(p_i))/2$ if there is no vertex or crossing between p_i and the first crossing of the edge (p_i, p_j) on Λ in Γ; **(ii)** let $y(p) = -(x(p_j) - x(p_i))/2$ otherwise.

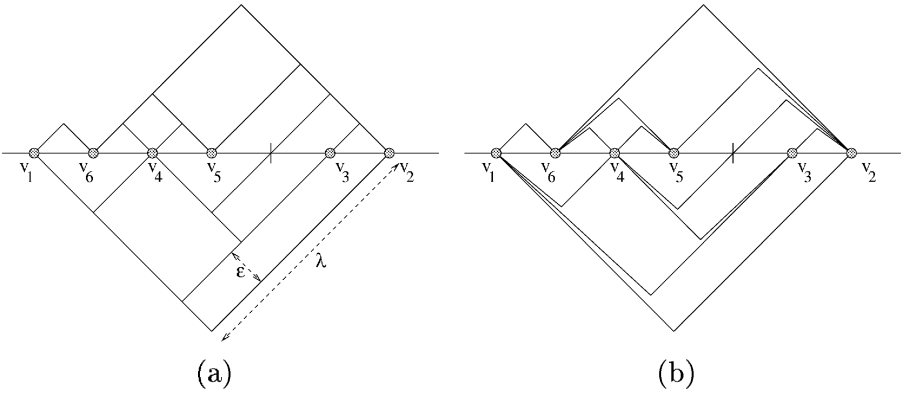


Fig. 4. (a)A nice spine drawing of the graph in Figure 1(a), with overlapping. (b) The removal of the overlapping.

At the end of this step some edges incident on a same vertex may overlap. See Figure 4(a).

Step 3. Remove the overlapping. This step is accomplished by using a technique due to Kaufmann and Wiese [14] that we adapt to our case. The minimum distance between any two non-overlapping parallel segments is at least $\varepsilon = 1/\sqrt{2}$. Let λ be the maximum length of a segment and Δ be the maximum degree of the vertices of G. For each point p_i representing a vertex of G sort the pairwise overlapping adjacent segments according to their length in decreasing order. Let δ_{ij} $(i \neq j)$ be a subedge represented by the polyline $\overline{p_i, p} \cup \overline{p, p_j}$ and let $\overline{p_i, p}$ be the h-th overlapping segment adjacent to p_i. Rotate the straight line l containing $\overline{p_i, p}$ by $h\varepsilon/\lambda\Delta$ towards the spine. The new polyline representing δ_{ij} is $\overline{p_i, p'} \cup \overline{p', p_j}$, where p' is the intersection point of the rotated line l and $\overline{p, p_j}$. See also Figure 4(b).

The proof of the correctness of the algorithm above is omitted due to space limitations. □

An *upward planar drawing* of a directed planar graph is such that each edge is monotonically increasing in a common direction, for example the left-right direction. A directed planar graph is upward planar if it admits an upward planar drawing. By Lemma 5 a curve embedding can be used to compute a nice spine drawing. Therefore one can orient the edges of the nice spine drawing from left to right so to obtain an upward drawing of the graph where all vertices are on a straight line and each edge has at most two bends. From this reasoning the following theorem holds.

Theorem 2. *Let G be a planar graph with n vertices. There exists an $O(n)$-time algorithm that orients the edges of G so that the resulting directed graph is upward planar and admits an upward planar drawing where all vertices are*

collinear and each edge has at most two bends. Furthermore, such an upward planar drawing can be computed in $O(n)$ time.

4.2 Point-Set Constrained Drawings

In [14] Kaufmann and Wiese present an elegant $O(n \log n)$-time algorithm to draw any planar graph by mapping the vertices to a given set of points, and with at most two bends per edge.

Their algorithm is based on finding a hamiltonian cycle in the input graph. Since not all planar graphs are guaranteed to have such a cycle, a preliminary step of the algorithm by Kaufmann and Wiese is to augment the input graph by adding vertices and edges so to make it hamiltomian. Firstly, the separation triplets are found using an algorithm by Chiba and Nishizeki [5]. Then vertices and edges are added to the graph, to create an augmented graph which is four-connected, and therefore admits a hamiltonian cycle. Every added dummy vertex splits an edge at most once. After the graph has been augmented another result by Chiba and Nishizeki [6] can be used to find a hamiltonian cycle. The overall time complexity of this augmentation step is linear.

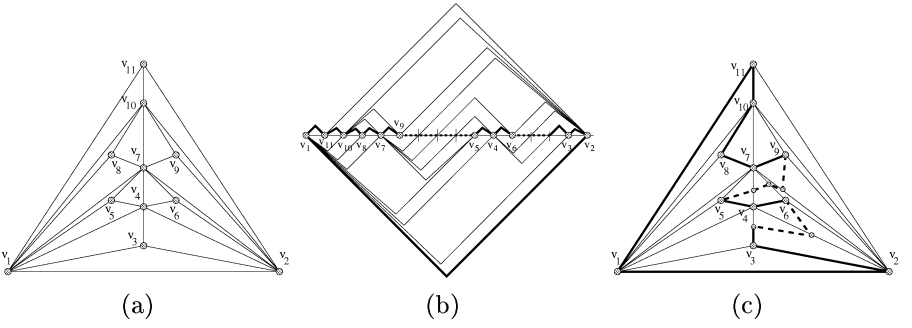


Fig. 5. (a) A non-hamiltonian graph G. (b) A nice spine embedding of G. (c) An augmentation of G to a hamiltonian graph.

A curve embedding on the other hand could be used to directly produce an augmented hamiltonian circuit of a graph G and without necessarily four-connecting the graph. Namely, Step 1 of the algorithm described in the proof of Lemma 5, defines an augmentation of G to a hamiltonian graph. Each first crossing can be seen as a dummy vertex that splits the corresponding edge once. The hamiltonian cycle is obtained by connecting consecutive vertices (dummy or not) along the spine of the nice spine drawing computed in Lemma 5. Figure 5 shows a non-hamiltonian graph G, a nice spine embedding of G, and its

augmentation; note that the augmented graph is not four-connected.

The following lemma provides an alternative strategy for the preliminary step of the algorithm by Kaufmann and Wiese.

Lemma 6. *Let G be a planar non-hamiltonian graph with n vertices. There exists a $O(n)$-time algorithm that adds vertices and edges to G to create a hamiltonian augmented graph G' which is not necessarily four-connected.*

Furthermore, by applying the drawing technique of Kaufmann and Wiese to the nice embedding computed by Lemma 5 we can map the input graph to any given set of points in such a way that all edges have at most two bends and are monotone in the x-direction. The following theorem extends the result in [14].

Theorem 3. *Let G be a planar graph with n vertices and let P be an arbitrary set of points in the plane. There exists an $O(n \log n)$-time algorithm that computes a drawing of G by mapping its vertices to the elements of P and such that each edge is a polyline with at most two bends and monotone in the x-direction.*

References

1. F. Bernhart and P. C. Kainen. The book thickness of a graph. *J. Combin. Theory*, Ser. B 27:320–331, 1979.
2. T. C. Biedl. Drawing planar partitions i: LL-drawings and LH-drawings. In *Symposium on Computational Geometry*, pages 287–296, 1998.
3. P. Bose. On embedding an outer-planar graph on a point set. In *Graph Drawing (Proc. GD '97)*, volume 1353 of *LNCS*, pages 25–36. Springer-Verlag, 1997.
4. P. Bose, M. McAllister, and J. Snoeyink. Optimal algorithms to embed trees in a point set. *J. of Graph Alg. and Appl.*, 2(1):1–15, 1997.
5. N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14:210–223, 1985.
6. N. Chiba and T. Nishizeki. The hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs. *Journal of Algorithms*, 10:189–211, 1989.
7. S. Cornelsen, T. Schank, and D. Wagner. Drawing graphs on two and three lines. In *Graph Drawing (Proc. GD '02)*, volume 2528 of *LNCS*, pages 31–41. Springer-Verlag, 2002.
8. H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.
9. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.
10. P. Eades and S. Whitesides. Drawing graphs in two layers. *Theoretical Computer Science*, 131(2):361–374, 1994.
11. H. Enomoto, M. Miyauchi, and K. Ota. Lower bounds for the number of edge-crossings over the spine in a topological book embedding of a graph. *Discrete Applied Mathematics*, 92:149–155, 1999.
12. M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: performance of exact and heuristic algorithms. *J. of Graph Alg. and Appl.*, 1(1):1–25, 1997.

13. M. Kaufmann and D. Wagner, editors. *Drawing Graphs*, volume 2025 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.
14. M. Kaufmann and R. Wiese. Embedding vertices at points: Few bends suffice for planar graphs. *J. of Graph Alg. and Appl.*, 6(1):115–129, 2002.
15. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. on Syst., Man and Cybern.*, 11(2):109–125, 1981.

Tree-Partitions of k-Trees with Applications in Graph Layout*

Vida Dujmović[1] and David R. Wood[2]

[1] School of Computer Science, McGill University, Montréal, Canada
vida@cs.mcgill.ca
[2] School of Computer Science, Carleton University, Ottawa, Canada
davidw@scs.carleton.ca

Abstract. A *tree-partition* of a graph is a partition of its vertices into 'bags' such that contracting each bag into a single vertex gives a forest. It is proved that every k-tree has a tree-partition such that each bag induces a $(k-1)$-tree, amongst other properties. Applications of this result to two well-studied models of graph layout are presented. First it is proved that graphs of bounded tree-width have bounded *queue-number*, thus resolving an open problem due to Ganley and Heath [2001] and disproving a conjecture of Pemmaraju [1992]. This result provides renewed hope for the positive resolution of a number of open problems regarding queue layouts. In a related result, it is proved that graphs of bounded tree-width have *three-dimensional straight-line grid drawings* with linear volume, which represents the largest known class of graphs with such drawings.

1 Introduction

This paper considers two models of graph layout. The first, called a *queue layout*, consists of a total order of the vertices, and a partition of the edges into *queues*, such that no two edges in the same queue are nested [11,12,15,17,20]. The dual concept of a *stack layout* (or *book embedding*), is defined similarly, except that no two edges in the same *stack* may cross. The minimum number of queues (respectively, stacks) in a queue (stack) layout of a graph is its *queue-number* (*stack-number*). Applications of queue layouts include parallel process scheduling, fault-tolerant processing, matrix computations, and sorting networks (see [15]). We prove that graphs of bounded tree-width have bounded queue-number, thus solving an open problem due to Ganley and Heath [9], who proved that stack-number is bounded by tree-width, and asked whether an analogous relationship holds for queue-number. This result has significant implications for other open problems in the field.

The second model of graph layout considered is that of a *three-dimensional* (*straight-line grid*) *drawing* [2,3,5,8,14,20]. Here vertices are positioned at grid-points in $\mathbb{Z}^3$, and edges are drawn as straight line-segments with no crossings.

* Research supported by NSERC and FCAR.

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 205–217, 2003.

While graph drawing in the plane is well-studied, there is a growing body of research in three-dimensional graph drawing. Applications include information visualisation, VLSI circuit design, and software engineering (see [5]). We focus on three-dimensional drawings with small volume, and prove that graphs of bounded tree-width have three-dimensional drawings with $\mathcal{O}(n)$ volume, which is the largest known class of graphs admitting such drawings. The best previous bound was $\mathcal{O}(n \log^2 n)$.

To prove the above results, we employ a structure called a *tree-partition* of a graph, which consists of a partition of the vertices into 'bags' such that contracting each bag to a single vertex gives a forest. In a result of independent interest, we prove that every k-tree has a tree-partition such that each bag induces a connected $(k-1)$-tree, amongst other properties. The second tool that we use is a *track layout*, which consists of a vertex-colouring and a total order of each colour class, such that between any two colour classes no two edges cross. We prove that every graph has a track layout where the number of tracks is bounded by a function of the graph's tree-width.

The remainder of the paper is organised as follows. Section 2 recalls a number of definitions and well-known results. In Section 3 we prove the above-mentioned theorem concerning tree-partitions of k-trees. In Section 4 we establish our results for track layouts. Combining these with earlier work in the companion papers [5,20], in Section 5 we prove our theorems for queue layouts and three-dimensional drawings. We discuss ramifications of our results for a number of open problems in Section 6.

2 Preliminaries

We consider undirected, simple, and finite graphs G with vertex set $V(G)$ and edge set $E(G)$. The number of vertices and maximum degree of G are respectively denoted by $n = |V(G)|$ and $\Delta(G)$. The subgraph induced by a set of vertices $A \subseteq V(G)$ is denoted by $G[A]$. A graph H is a *minor* of G if H is isomorphic to a graph obtained from a subgraph of G by contracting edges. A family of graphs closed under taking minors is *proper* if it is not the class of all graphs.

A *graph parameter* is a function α that assigns to every graph G a non-negative integer $\alpha(G)$. Let $\mathcal{G}$ be a family of graphs. By $\alpha(\mathcal{G})$ we denote the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum, taken over all n-vertex graphs $G \in \mathcal{G}$, of $\alpha(G)$. We say $\mathcal{G}$ has *bounded* α if $\alpha(\mathcal{G}) \in \mathcal{O}(1)$. A graph parameter α is *bounded by* a graph parameter β, if there exists a function f such that $\alpha(G) \leq f(\beta(G))$ for every graph G.

A *k-tree* for some $k \in \mathbb{N}$ is defined recursively as follows. The empty graph is a k-tree, and the graph obtained from a k-tree by adding a new vertex adjacent to each vertex of a clique with at most k vertices is a k-tree. This definition is by Reed [16]. The following more common definition of a k-tree, which we call 'strict', was introduced by Arnborg and Proskurowski [1]. A k-clique is a *strict k-tree*, and the graph obtained from a strict k-tree by adding a new vertex adjacent to each vertex of a k-clique is a strict k-tree. Obviously the strict k-trees

are a proper sub-class of the k-trees. The *tree-width* of a graph G, denoted by $\mathsf{tw}(G)$, is the minimum k such that G is a subgraph of a k-tree (which equals the minimum k such that G is a subgraph of a strict k-tree [16]). Note that k-trees can be characterised as the chordal graphs with no clique on $k+2$ vertices. Graphs with tree-width at most one are the forests. Graphs with tree-width at most two are the *series-parallel* graphs, defined as those graphs with no K_4 minor.

Let G be a graph. A total order $\sigma = (v_1, v_2, \ldots, v_n)$ of $V(G)$ is called a *vertex-ordering* of G. Suppose G is connected. The *depth* of a vertex v_i in σ is the graph-theoretic distance between v_1 and v_i in G. We say σ is a *breadth-first* vertex-ordering if for all vertices $v <_\sigma w$, the depth of v in σ is no more than the depth of w in σ. Vertex-orderings, and in particular, vertex-orderings of trees will be used extensively in this paper. Consider a breadth-first vertex-ordering σ of a tree T such that vertices at depth $d \geq 1$ are ordered with respect to the ordering of vertices at depth $d-1$. In particular, if v and x are vertices at depth d with respective parents w and y at depth $d-1$ with $w <_\sigma y$ then $v <_\sigma x$. Such a vertex-ordering is called a *lexicographical* breadth-first vertex-ordering of T.

3 Tree-Partitions

Let G be a graph and let T be a tree. An element of $V(T)$ is called a *node*. Let $\{T_x \subseteq V(G) : x \in V(T)\}$ be a set of subsets of $V(G)$ indexed by the nodes of T. Each T_x is called a *bag*. The pair $(T, \{T_x : x \in V(T)\})$ is a *tree-partition* of G if:

- $\forall$ distinct nodes x and y of T, $T_x \cap T_y = \emptyset$, and
- $\forall$ edge vw of G, either
  - $\exists$ node x of T with $v \in T_x$ and $w \in T_x$ (vw is an *intra-bag* edge), or
  - $\exists$ edge xy of T with $v \in T_x$ and $w \in T_y$ (vw is an *inter-bag* edge).

The main property of tree-partitions that has been studied is the maximum size of a bag, called the *width* of the tree-partition. The minimum width over all tree-partitions of a graph G is the *tree-partition-width* of G, denoted by $\mathsf{tpw}(G)$. Ding and Oporowski [4] proved that $\mathsf{tpw}(G) \leq 24\,\mathsf{tw}(G) \cdot \max\{1, \Delta(G)\}$, and Seese [19] proved that $\mathsf{tw}(G) \leq 2\,\mathsf{tpw}(G) - 1$, for every graph G.

Theorem 1 below provides a tree-partition of a k-tree with additional features besides small width (see Figure 1). First, the subgraph induced by each bag is a connected $(k-1)$-tree. This allows us to perform induction on k. Second, in each non-root bag T_x, the vertices in the parent bag of x with a neighbour in T_x form a clique. This feature is crucial in the intended application (Theorem 2). Finally the bound on the tree-partition-width represents a constant-factor improvement over the above result by Ding and Oporowski [4] in the case of k-trees.

Theorem 1. *Let G be a k-tree with maximum degree Δ. Then G has a rooted tree-partition $(T, \{T_x : x \in V(T)\})$ such that for all nodes x of T,*

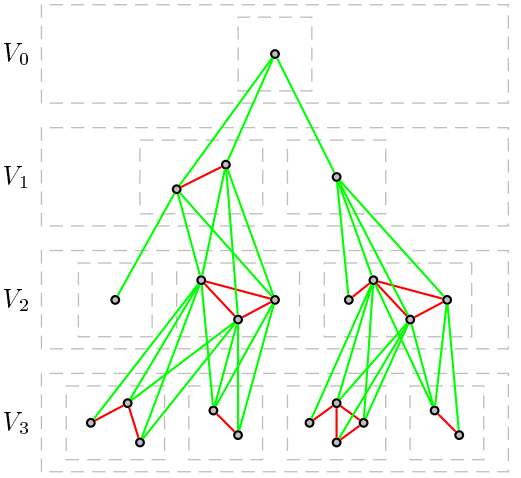


Fig. 1. Tree-partition of a 3-tree.

(a) *if x is a non-root node of T and y is the parent node of x, then the vertices in T_y with a neighbour in T_x form a clique C_x of G, and*
(b) *the induced subgraph $G[T_x]$ is a connected $(k-1)$-tree.*

Furthermore the width of $(T, \{T_x : x \in V(T)\})$ *is at most* $\max\{1, k(\Delta - 1)\}$.

Proof. We assume G is connected, since if G is not connected then a tree-partition of G that satisfies the theorem can be determined by adding a new root node with an empty bag which is adjacent to the root node of a tree-partition of each connected component of G. It is well-known[1] that for every vertex r of the k-tree G, there is a vertex-ordering $\sigma = (v_1, v_2, \dots, v_n)$ of G with $v_1 = r$, such that for all $1 \leq i \leq n$,

(i) $G^i = G[\{v_1, v_2, \dots, v_i\}]$ is connected and the vertex-ordering of G^i induced by σ is a breadth-first vertex-ordering of G^i.
(ii) the neighbours of v_i in G^i form a clique $C_i = \{v_j : v_i v_j \in E(G), j < i\}$ with $1 \leq |C_i| \leq k$ (unless $i = 1$ in which case $C_i = \emptyset$).

Let r be a vertex of minimum degree Then $\deg(r) \leq k$. Let $\sigma = (v_1, v_2, \dots, v_n)$ be a vertex-ordering of G with $v_1 = r$, and satisfying (i) and (ii). By (i), the depth of each vertex v_i in σ is the same as the depth of v_i in the vertex-ordering of G^j induced by σ, for all $j \geq i$. We therefore simply speak of *the* depth of v_i. Let V_d be the set of vertices of G at depth d.

Claim: For all $1 \leq i \leq n$, in every connected component Z of $G^i[V_d]$, the set of vertices at depth $d-1$ with a neighbour in Z form a clique of G, for all $d \geq 1$.

[1] In the language of chordal graphs, σ is a (reverse) perfect elimination vertex-ordering and can be determined, for example, by the Lex-BFS algorithm of Rose *et al.* [18].

Proof. We proceed by induction on i. The result is trivially true for $i = 1$. Suppose it is true for $i-1$. Let d be the depth of v_i. Each vertex in C_i is at depth $d-1$ or d. Let C'_i be the set of vertices in C_i at depth d, and let C''_i be the set of vertices in C_i at depth $d-1$. Thus C'_i and C''_i are both cliques with $C_i = C'_i \cup C''_i$. Furthermore, if $i > 1$ then v_i must have a neighbour at depth $d-1$, and thus $C''_i \neq \emptyset$. Let X be the vertex set of the connected component of $G^i[V_d]$ such that $v_i \in X$. By induction, for all $d' \leq d$, the claim holds for all connected components Y of $G^i[V_{d'}]$ with $Y \neq X$, since such a Y is also a connected component of $G^{i-1}[V_{d'}]$.

Case 1. $C'_i = \emptyset$: Then v_i has no neighbours in G^i at depth d; that is, $X = \{v_i\}$. Thus the set of vertices at depth $d-1$ with a neighbour in X is precisely the clique $C_i = C''_i$.

Case 2. $C'_i \neq \emptyset$: The neighbourhood of v_i in X forms a non-empty clique (namely C'_i). Thus $X \setminus v_i$ is the vertex-set of a connected component of $G^{i-1}[V_d]$. Let Y be the set of vertices at depth $d-1$ with a neighbour in $X \setminus v_i$. By induction, Y is a clique. Since $C''_i \cup C'_i$ is a clique, $C''_i \subseteq Y$. Thus the set of vertices at depth $d-1$ with a neighbour in X is the clique Y. □

Define a graph T and a partition $\{T_x : x \in V(T)\}$ of $V(G)$ indexed by the nodes of T as follows. There is one node x in T for every connected component of each $G[V_d]$, whose bag T_x is the vertex-set of the corresponding connected component. We say x and T_x are at *depth* d. Clearly a vertex in a depth-d bag is also at depth d. The (unique) node of T at depth zero is called the *root* node. Let two nodes x and y of T be connected by an edge if there is an edge vw of G with $v \in T_x$ and $w \in T_y$. Thus $(T, \{T_x : x \in V(T)\})$ is a 'graph-partition'. We now prove that in fact T is a tree. First observe that T is connected since G is connected. By definition, nodes of T at the same depth d are not adjacent. Moreover nodes of T can be adjacent only if their depths differ by one. Thus T has a cycle only if there is a node x in T at some depth d, such that x has at least two distinct neighbours in T at depth $d-1$. However, by the above claim (with $i = n$), the set of vertices at depth $d-1$ with a neighbour in T_x form a clique (called C_x), and are hence in a single bag at depth $d-1$. Thus T is a tree and $(T, \{T_x : x \in V(T)\})$ is a tree-partition of G.

We now prove that each bag T_x induces a connected $(k-1)$-tree. This is true for the root node since it only has one vertex. Suppose x is a non-root node of T at depth d. Each vertex in T_x has at least one neighbour at depth $d-1$. Thus in the vertex-ordering of T_x induced by σ, each vertex $v_i \in T_x$ has at most $k-1$ neighbours $v_j \in T_x$ with $j < i$. These neighbours induce a clique. Thus $G[T_x]$ is a $(k-1)$-tree. By definition each $G[T_x]$ is connected.

Finally, consider the size of a bag in T. We claim that each bag contains at most $\max\{1, k(\Delta-1)\}$ vertices. The root bag has one vertex. Let x be a non-root node of T with parent node y. Suppose y is the root node. Then $T_y = \{r\}$, and thus $|T_x| \leq \deg(r) \leq k \leq k(\Delta-1)$ assuming $\Delta \geq 2$. If $\Delta \leq 1$ then all bags have one vertex. Now assume y is a non-root node. The set of vertices in T_y with a neighbour in T_x forms the clique C_x. Let $k' = |C_x|$. Thus $k' \geq 1$, and since $C_x \subseteq T_y$ and $G[T_y]$ is a $(k-1)$-tree, $k' \leq k$. A vertex $v \in C_x$ has $k'-1$

neighbours in C_x and at least one neighbour in the parent bag of y. Thus v has at most $\Delta - k'$ neighbours in T_x. Hence the number of edges between C_x and T_x is at most $k'(\Delta - k')$. Every vertex in T_x is adjacent to a vertex in C_x. Thus $|T_x| \leq k'(\Delta - k') \leq k(\Delta - 1)$. This completes the proof. □

4 Track Layouts

A *colouring* of a graph G is a partition $\{V_i : i \in I\}$ of $V(G)$, where I is a set of *colours*, such that for every edge vw of G, if $v \in V_i$ and $w \in V_j$ then $i \neq j$. Each set V_i is called a *colour class*. If $<_i$ is a total order of a colour class V_i, then we call the pair $(V_i, <_i)$ a *track*. If $\{V_i : i \in I\}$ is a colouring of G, and $(V_i, <_i)$ is a track for each colour $i \in I$, then we say $\{(V_i, <_i) : i \in I\}$ is a *track assignment* of G *indexed by* I. At times it will be convenient to also refer to a colour $i \in I$ and the colour class V_i as a *track*. The precise meaning will be clear from the context. A *t-track assignment* is a track assignment with t tracks. An *X-crossing* in a track assignment consists of two edges vw and xy such that $v <_i x$ and $y <_j w$, for distinct tracks V_i and V_j. A t-track assignment with no X-crossing is called a *t-track layout*. The *track-number* of a graph G, denoted by $\mathsf{tn}(G)$, is the minimum t such that G has a t-track layout.

Dujmović *et al.* [5] first introduced track layouts[2], and proved that track-number is bounded by path-width. In particular, $\mathsf{tn}(G) \leq \mathsf{pw}(G) + 1$ for every graph G, where $\mathsf{pw}(G)$ denotes the path-width of G. In what follows we prove that track-number is bounded by tree-width. First consider the case of trees. The following result is implicit in the proof by Felsner *et al.* [8] that every outerplanar graph has a three-dimensional drawing with linear volume (see Figure 2).

Lemma 1. **[8]** *Every tree T has a 3-track layout.*

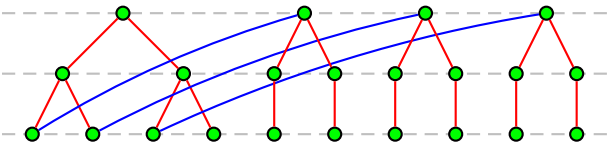


Fig. 2. A 3-track layout of a tree.

Let $\{(V_i, <_i) : i \in I\}$ be a track layout of a graph G. We say a clique C of G *covers* the set of tracks $\{i \in I : C \cap V_i \neq \emptyset\}$. Let S be a set of cliques of G. Suppose there is a total order $\preceq$ on S such that for all cliques $C_1, C_2 \in S$, if there exists a track $i \in I$, and vertices $v \in V_i \cap C_1$ and $w \in V_i \cap C_2$ with $v <_i w$, then $C_1 \prec C_2$. Then we say $\preceq$ is *nice*, and S is *nicely ordered* by the track layout. The proof of the next lemma is elementary.

[2] A track layout was called an 'ordered layering with no X-crossing and no intra-layer edges' in [5,6,20]. Similar structures are implicit in [8,11,12,17]. Note that this definition of *track-number* is unrelated to that of Gyárfás and West [10].

Lemma 2. [6] *Let $L \subseteq I$ be a set of tracks in a track layout $\{(V_i, <_i) : i \in I\}$ of a graph G. If S is a set of cliques, each of which covers L, then S is nicely ordered by the given track layout.*

Theorem 2. *Track-number is bounded by tree-width. In particular, every graph G with tree-width $\mathsf{tw}(G) \leq k$ has track-number $\mathsf{tn}(G) \leq t_k = 3^k \cdot 6^{(4^k-3k-1)/9}$.*

Proof. If G is not a k-tree then add edges to G to obtain a k-tree containing G as a subgraph. It is well-known that a graph with tree-width at most k is a *spanning* subgraph of a k-tree. These extra edges can be deleted once we are done. We proceed by induction on k with the following induction hypothesis:

For all $k \in \mathbb{N}$, there exist constants s_k and t_k, and sets I and S such that

1. *$|I| = t_k$ and $|S| = s_k$,*
2. *each element of S is a subset of I, and*
3. *every k-tree G has a t_k-track layout indexed by I, such that for every clique C of G, the set of tracks that C covers is in S.*

Consider the base case with $k = 0$. A 0-tree G has no edges and thus has a 1-track layout. Let $I = \{1\}$ and order $V_1 = V(G)$ arbitrarily. Thus $t_0 = 1$, $s_0 = 1$, and $S = \{\{1\}\}$ satisfy the hypothesis for every 0-tree. Now suppose the result holds for $k-1$, and G is a k-tree. Let $(T, \{T_x : x \in V(T)\})$ be a tree-partition of G described in Theorem 1, where T is rooted at r. By Theorem 1 each induced subgraph $G[T_x]$ is a $(k-1)$-tree. By induction, there are sets I and S with $|I| = t_{k-1}$ and $|S| = s_{k-1}$, such that for every node x of T, the induced subgraph $G[T_x]$ has a t_{k-1}-track layout indexed by I. For every clique C of $G[T_x]$, if C covers $L \subseteq I$ then $L \in S$. Assume $I = \{1, 2, \ldots, t_{k-1}\}$ and $S = \{S_1, S_2, \ldots, S_{s_{k-1}}\}$. By Theorem 1, for each non-root node x of T, if p is the parent node of x, then the set of vertices in T_p with a neighbour in T_x form a clique C_x. Let $\alpha(x) = i$ where C_x covers S_i. Let $\alpha(r) = 1$.

To construct a track layout of G we first construct a track layout of T indexed by $\{(d, i) : d \geq 0, 1 \leq i \leq s_{k-1}\}$, where the track $L_{d,i}$ consists of nodes x of T at depth d with $\alpha(x) = i$. Here the *depth* of a node x is the distance in T from the root node r to x. We order the nodes of T within the tracks by increasing depth. There is only one node at depth $d = 0$. Suppose we have determined the orders of the nodes up to depth $d-1$ for some $d \geq 1$. Let $i \in \{1, 2, \ldots, s_{k-1}\}$. The nodes in $L_{d,i}$ are ordered primarily with respect to the relative positions of their parent nodes (at depth $d-1$). More precisely, let $\rho(x)$ denote the parent node of each $x \in L_{d,i}$. For all nodes x and y in $L_{d,i}$, if $\rho(x)$ and $\rho(y)$ are in the same track and $\rho(x) < \rho(y)$ in that track, then $x < y$ in $L_{d,i}$. For x and y with $\rho(x)$ and $\rho(y)$ on distinct tracks, the relative order of x and y is not important. It remains to specify the order of nodes in $L_{d,i}$ with a common parent. Suppose P is a set of nodes in $L_{d,i}$ with a common parent node p. By construction, for every node $x \in P$, the parent clique C_x covers S_i in the track layout of $G[T_p]$. By Lemma 2 the cliques $\{C_x : x \in P\}$ are nicely ordered by the track layout of $G[T_p]$. Let the order of P in track $L_{d,i}$ be specified by a nice ordering of $\{C_x : x \in P\}$, as

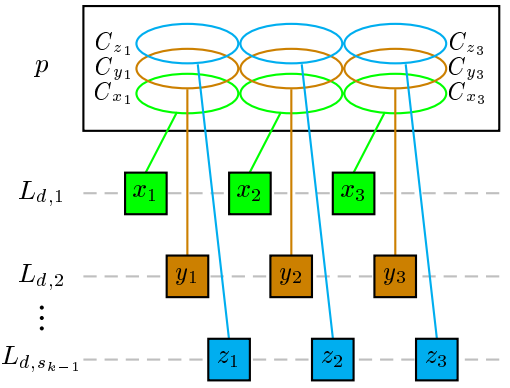


Fig. 3. Nodes with a common parent p.

illustrated in Figure 3. This construction defines a partial order on the nodes in track $L_{d,i}$ that can be arbitrarily extended to a total order. Hence we have a track assignment of T. Since the nodes in each track are ordered primarily with respect to the relative positions of their parent nodes in the previous tracks, there is no X-crossing, and hence we have a track layout of T.

To construct a track assignment of G from the track layout of T, replace each track $L_{d,i}$ by t_{k-1} 'sub-tracks', and for each node x of T, insert the track layout of $G[T_x]$ in place of x on the sub-tracks corresponding to the track containing x. More formally, the track assignment of G is indexed by $\{(d,i,j) : d \geq 0, 1 \leq i \leq s_{k-1}, 1 \leq j \leq t_{k-1}\}$. Each track $V_{d,i,j}$ consists of those vertices v of G such that, if T_x is the bag containing v, then x is at depth d in T, $\alpha(x) = i$, and v is on track j in the track layout of $G[T_x]$. If x and y are distinct nodes of T with $x < y$ in $L_{d,i}$, then $v < w$ in $V_{d,i,j}$, for all vertices $v \in T_x$ and $w \in T_y$ on track j. If v and w are vertices of G on track j in bag T_x at depth d, then the relative order of v and w in $V_{d,\alpha(x),j}$ is the same as in the track layout of $G[T_x]$.

Clearly adjacent vertices of G are in distinct tracks. Thus we have defined a track assignment of G. We claim that there is no X-crossing. Clearly an intra-bag edge of G is not in an X-crossing with an edge not in the same bag. By induction, there is no X-crossing between intra-bag edges in a common bag. Since there is no X-crossing in the track layout of T, inter-bag edges of G which are mapped to edges of T without a common parent node, are not involved in an X-crossing. Consider a parent node p in T. For each child node x of p, the vertices in T_p adjacent to a vertex in T_x forms the clique C_x. Thus there is no X-crossing between a pair of edges both from C_x to T_x, since the vertices of C_x are on distinct tracks. Consider two child nodes x and y of p. For there to be an X-crossing between an edge from T_p to T_x and an edge from T_p to T_y, the nodes x and y must be on the same track in the track layout of T. Suppose $x < y$ in this track. By construction, C_x and C_y cover the same set of tracks, and $C_x \preceq C_y$ in the corresponding nice ordering. Thus for any track containing vertices $v \in C_x$ and $w \in C_y$, $v \leq w$ in that track. Since all the vertices in T_x are to the left of the vertices in T_y (on a common track), there is no X-crossing

between an edge from T_p to T_x and an edge from T_p to T_y. Therefore there is no X-crossing, and hence we have a track layout of G.

We now 'wrap' the track layout of G. Define a track assignment of G indexed by $\big\{(d', i, j) : d' \in \{0, 1, 2\}, 1 \leq i \leq s_{k-1}, 1 \leq j \leq t_{k-1}\big\}$, where each track $W_{d',i,j} = \bigcup\{V_{d,i,j} : d \equiv d' \pmod 3\}$. If $v \in V_{d,i,j}$ and $w \in V_{d+3,i,j}$ then $v < w$ in the order of $W_{d',i,j}$ (where $d' = d \bmod 3$). The order of each $V_{d,i,j}$ is preserved in $W_{d',i,j}$. The tracks $\{W_{d',i,j} : d' \in \{0, 1, 2\}, 1 \leq i \leq s_{k-1}, 1 \leq j \leq t_{k-1}\}$ forms a track assignment of G. For every edge vw of G, the depths of the bags in T containing v and w differ by at most one. Thus in the wrapped track assignment of G, adjacent vertices remain on distinct tracks, and there is no X-crossing. The number of tracks is $3 \cdot s_{k-1} \cdot t_{k-1}$. Every clique C of G is either contained in a single bag of the tree-partition or is contained in two adjacent bags. Let $S' = \big\{\{(d', i, h) : h \in S_j\} : d' \in \{0, 1, 2\}, 1 \leq i, j \leq s_{k-1}\big\}$. For every clique C of G contained in a single bag, the set of tracks containing C is in S'. Let $S'' = \big\{\{(d', i, h) : h \in S_j\} \cup \{((d' + 1) \bmod 3, p, h) : h \in S_q\} : d' \in \{0, 1, 2\}, 1 \leq i, j, p, q \leq s_{k-1}\big\}$. For every clique C of G contained in two bags, the set of tracks containing C is in S'. Observe that $S' \cup S''$ is independent of G. Hence $S' \cup S''$ satisfies the hypothesis for k. Now $|S'| = 3s_{k-1}^2$ and $|S''| = 3s_{k-1}^4$, and thus $|S' \cup S''| = 3s_{k-1}^2(s_{k-1}^2 + 1)$. Therefore any solution to the recurrences $\{s_0 \geq 1,\ t_0 \geq 1,\ s_k \geq 3s_{k-1}^2(s_{k-1}^2 + 1),\ t_k \geq 3s_{k-1} \cdot t_{k-1}\}$ satisfies the theorem. It is easily verified that $s_k = 6^{(4^k-1)/3}$ and $t_k = 3^k \cdot 6^{(4^k-3k-1)/9}$ is such a solution. □

A number of refinements to the proof of Theorem 2 that result in improved bounds are possible [6]. For example, in the case of $\mathsf{tw}(G) = 2$, we prove that $\mathsf{tn}(G) \leq 18$, whereas Theorem 2 proves that $\mathsf{tn}(G) \leq 54$. One such refinement uses strict k-trees. From an algorithmic point of view, the disadvantage of using strict k-trees is that at each recursive step, extra edges must be added to enlarge the graph into a strict k-tree, whereas when using (non-strict) k-trees, extra edges need only be added at the beginning of the algorithm.

If maximum degree as well as tree-width is bounded then the dependence on the tree-width in our track-number bound can be substantially reduced.

Theorem 3. *Every graph G with maximum degree $\Delta(G)$, tree-width $\mathsf{tw}(G)$, and tree-partition-width $\mathsf{tpw}(G)$, has track-number $\mathsf{tn}(G) \leq 3\,\mathsf{tpw}(G) \leq 72\,\mathsf{tw}(G) \cdot \max\{1, \Delta(G)\}$.*

Proof. Let $(T, \{T_x : x \in V(T)\})$ be a tree-partition of G with width $\mathsf{tpw}(G)$. By Lemma 1, T has a 3-track layout. Replace each track by $\mathsf{tpw}(G)$ 'sub-tracks', and for each node x in T, place the vertices in T_x on the sub-tracks replacing the track containing x, with at most one vertex in T_x on a single track. The total order of each sub-track preserves the total order in each track of the track-layout of T. There is no X-crossing, since in the track layout of T, adjacent nodes are on distinct tracks and there is no X-crossing. Thus we have a track layout of G with $3\,\mathsf{tpw}(G) \leq 72\,\mathsf{tw}(G) \cdot \max\{1, \Delta(G)\}$ tracks [4]. □

5 Queue Layouts and 3D Graph Drawings

A *queue layout* of a graph G consists of a vertex-ordering σ of G, and a partition of $E(G)$ into *queues*, such that no two edges in the same queue are *nested* with respect to σ. That is, there are no edges vw and xy in a single queue with $v <_\sigma x <_\sigma y <_\sigma w$. A similar concept is that of a *stack layout* (or *book embedding*), which consists of a vertex-ordering σ of G, and a partition of $E(G)$ into *stacks* (or *pages*) such that there are no edges vw and xy in a single stack with $v <_\sigma x <_\sigma w <_\sigma y$. The minimum number of queues (respectively, stacks) in a queue (stack) layout of G is called the *queue-number* (*stack-number* or *page-number*) of G, and is denoted by $\mathsf{qn}(G)$ ($\mathsf{sn}(G)$). Ganley and Heath [9] proved that stack-number is bounded by tree-width, and asked whether queue-number is also bounded by tree-width? The bound of $\mathsf{sn}(G) \leq \mathsf{tw}(G) + 1$ by Ganley and Heath [9] has recently been improved to $\mathsf{sn}(G) \leq \mathsf{tw}(G)$ by Lin and Li [13].

A 1-tree has queue-number at most one, since in a lexicographical breadth-first vertex-ordering of a tree no two edges are nested [12]. Rengarajan and Veni Madhavan [17] proved that 2-trees have queue-number at most three. Wood [20] proved that queue-number is bounded by path-width and tree-partition-width. In particular, $\mathsf{qn}(G) \leq \mathsf{pw}(G)$ and $\mathsf{qn}(G) \leq \frac{3}{2}\mathsf{tpw}(G)$ for every graph G. Hence $\mathsf{qn}(G) \leq 36\,\mathsf{tw}(G) \cdot \max\{1, \Delta(G)\}$ by the result of Ding and Oporowski [4]. Wood [20] also proved that $\mathsf{qn}(G) \leq \mathsf{tn}(G)-1$ for every graph G. Thus Theorem 2 implies the following result, which answers the above question of Ganley and Heath [9] in the affirmative. Further consequences are discussed in Section 6.

Theorem 4. *Queue-number is bounded by tree-width. In particular, every graph G with tree-width $\mathsf{tw}(G) \leq k$ has queue-number $\mathsf{qn}(G) < 3^k \cdot 6^{(4^k-3k-1)/9}$.*

A *three-dimensional straight-line grid drawing* of a graph, henceforth called a *3D drawing*, represents the vertices by distinct points in $\mathbb{Z}^3$ (called *grid-points*), and represents each edge as a line-segment between its end-vertices, such that edges only intersect at common end-vertices, and an edge only intersects a vertex that is an end-vertex of that edge. In contrast to the case in the plane, it is well known that every graph has a 3D drawing. We therefore are interested in optimising certain measures of the aesthetic quality of a drawing. If a 3D drawing is contained in an axis-aligned box with side lengths $X - 1$, $Y - 1$ and $Z - 1$, then we speak of a $X \times Y \times Z$ drawing with *volume* $X \cdot Y \cdot Z$. We study 3D drawings with small volume.

Cohen *et al.* [2] proved that every graph has a 3D drawing with $\mathcal{O}(n^3)$ volume, and this bound is asymptotically tight for K_n. It is therefore of interest to identify fixed graph parameters that allow for 3D drawings with $o(n^3)$ volume. Pach *et al.* [14] proved that graphs of bounded chromatic number have 3D drawings with $\mathcal{O}(n^2)$ volume, and that this bound is asymptotically optimal for $K_{n,n}$. The first non-trivial $\mathcal{O}(n)$ volume bound was established by Felsner *et al.* [8] for outerplanar graphs. Dujmović *et al.* [5,20] proved that track layouts, queue layouts, and 3D drawings with small volume are inherently related.

Theorem 5. [5,20] *Every n-vertex graph G has a $\mathcal{O}(\mathsf{tn}(G)) \times \mathcal{O}(\mathsf{tn}(G)) \times \mathcal{O}(n)$ drawing. Let $\mathcal{F}(n)$ be a family of functions closed under multiplication, such as $\mathcal{O}(1)$ or $\mathcal{O}(\text{polylog}\, n)$. Then for any graph family $\mathcal{G}$, every graph $G \in \mathcal{G}$ has a $\mathcal{F}(n) \times \mathcal{F}(n) \times \mathcal{O}(n)$ drawing if and only if the track-number $\mathsf{tn}(\mathcal{G}) \in \mathcal{F}(n)$. Moreover, if $\mathcal{G}$ is proper minor-closed then $\mathcal{G}$ has track-number $\mathsf{tn}(\mathcal{G}) \in \mathcal{F}(n)$ if and only if $\mathcal{G}$ has queue-number $\mathsf{qn}(\mathcal{G}) \in \mathcal{F}(n)$.*

Applying Theorem 5, Dujmović *et al.* [5] proved that every graph G has a 3D drawing with $\mathcal{O}(\mathsf{pw}(G)^2 \cdot n)$ volume, which is $\mathcal{O}(n \log^2 n)$ for graphs of bounded tree-width. Using the result of Rengarajan and Veni Madhavan [17] discussed in Section 5, Wood [20] proved that series-parallel graphs have 3D drawings with $\mathcal{O}(n)$ volume, but with a constant of at least 10^{16}. For particular sub-classes of series-parallel graphs, improved constants have been obtained [3].

Wood [20] proved that graphs of bounded tree-partition-width have 3D drawings with $\mathcal{O}(n)$ volume, although the actual volume bound is approximately $\mathcal{O}(\mathsf{tw}(G)^4(\mathsf{tw}(G)^2\, \mathsf{tpw}(G))^{\mathsf{tw}(G)^2} \cdot n)$. Theorems 3 and 5 together prove the following result, which represents a substantial improvement in the dependence on $\mathsf{tpw}(G)$ compared with the above-mentioned result.

Theorem 6. *Every n-vertex graph G with bounded tree-partition-width, which includes graph of bounded tree-width and bounded degree, has a 3D drawing with $\mathcal{O}(n)$ volume. In particular, the drawing is $\mathcal{O}(\mathsf{tpw}(G)) \times \mathcal{O}(\mathsf{tpw}(G)) \times \mathcal{O}(n)$, which is $\mathcal{O}(\mathsf{tw}(G)\, \Delta(G)) \times \mathcal{O}(\mathsf{tw}(G)\, \Delta(G)) \times \mathcal{O}(n)$.*

Theorems 2 and 5 together prove our main result of this section.

Theorem 7. *Every n-vertex graph G with bounded tree-width has a 3D drawing with $\mathcal{O}(n)$ volume. In particular, the drawing is $\mathcal{O}(6^{4^{\mathsf{tw}(G)}}) \times \mathcal{O}(6^{4^{\mathsf{tw}(G)}}) \times \mathcal{O}(n)$.*

As well as providing many new classes of graphs that admit 3D drawings with $\mathcal{O}(n)$ volume, Theorem 7 dramatically improves the constant in the bound for series-parallel graphs. As mentioned in Section 4, such graphs have 18-track layouts. It follows that every series-parallel graph has a $36 \times 37 \times 37 \lceil \frac{n}{18} \rceil$ drawing.

6 Open Problems

Consider the following open problems: (1) Do planar graphs have bounded queue-number? (2) Is queue-number bounded by stack-number? Since planar graphs have bounded stack-number, the second question is more general than the first. Heath *et al.* [11] conjectured that both of these questions have an affirmative answer. More recently however, Pemmaraju [15] conjectured that the 'stellated K_3', a planar 3-tree, has $\Theta(\log n)$ queue-number, and provided evidence to support this conjecture (also see [9]). This suggested that the answers to the above questions were both negative. In particular, Pemmaraju [15] and Heath [private communication, 2002] conjectured that planar graphs have $\mathcal{O}(\log n)$ queue-number. However, Theorem 4 provides a queue-layout of *any* 3-tree, and thus

the stellated K_3, with $\mathcal{O}(1)$ queues. Hence our result disproves the first conjecture of Pemmaraju [15] mentioned above, and renews hope in an affirmative answer to the above open problems. By Theorem 5, question (1) is equivalent to the question of whether planar graphs have bounded track-number, which was asked by H. de Fraysseix [private communication, 2000] in the context of graph drawing. If planar graphs have bounded track-number then such graphs would also admit 3D drawings with $\mathcal{O}(n)$ volume, which is an open problem due to Felsner *et al.* [8]. The authors recently proved that planar graphs and graphs of bounded degree have 3D drawings with $\mathcal{O}(n^{3/2})$ volume [7].

References

1. S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Appl. Math.* **23**(1):11–24, 1989.
2. R. F. Cohen, P. Eades, T. Lin, and F. Ruskey. Three-dimensional graph drawing. *Algorithmica* **17**(2):199–208, 1996.
3. E. Di Giacomo, G. Liotta, and S. Wismath. Drawing series-parallel graphs on a box. In *Proc. 14th Canadian Conf. on Computational Geometry* (CCCG '02), pages 149–153. The Univ. of Lethbridge, Canada, 2002.
4. G. Ding and B. Oporowski. Some results on tree decomposition of graphs. *J. Graph Theory* **20**(4):481–499, 1995.
5. V. Dujmović, P. Morin, and D. R. Wood. Path-width and three-dimensional straight-line grid drawings of graphs. In *Proc. 10th Symp. on Graph Drawing* (GD '02), LNCS **2628**:42–53, Springer, 2002.
6. V. Dujmović and D. R. Wood. Tree-partitions of k-trees with applications in graph layout. Technical Report TR-02-03, School of Computer Science, Carleton Univ., Ottawa, Canada, 2002.
7. V. Dujmović and D. R. Wood. Three-Dimensional Grid Drawings with Sub-Quadratic Volume. In *Proc. 11th Symp. on Graph Drawing* (GD '03), LNCS, Springer, to appear. Also in: Towards a Theory of Geometric Graphs, *Contemporary Mathematics*, Amer. Math. Soc., to appear.
8. S. Felsner, G. Liotta, and S. Wismath. Straight-line drawings on restricted integer grids in two and three dimensions. In *Proc. 9th Symp. on Graph Drawing* (GD '01), LNCS **2265**:328–342, Springer, 2002.
9. J. L. Ganley and L. S. Heath. The pagenumber of k-trees is $O(k)$. *Discrete Appl. Math.* **109**(3):215–221, 2001.
10. A. Gyárfás and D. West. Multitrack interval graphs. In *26th Southeastern Conf. on Combinat., Graph Theory and Comput., Congr. Numer.* **109**:109–116, 1995.
11. L. S. Heath, F. T. Leighton, and A. L. Rosenberg. Comparing queues and stacks as mechanisms for laying out graphs. *SIAM J. Disc. Math.* **5**:398–412, 1992.
12. L. S. Heath and A. L. Rosenberg. Laying out graphs using queues. *SIAM J. Comput.* **21**(5):927–958, 1992.
13. Y. Lin, and X. Li. Pagenumber and treewidth. *Disc. Applied Math.*, to appear.
14. J. Pach, T. Thiele, and G. Tóth. Three-dimensional grid drawings of graphs. In *Proc. 5th Symp. on Graph Drawing* (GD '97), LNCS **1353**:47–51, Springer, 1997. Also in: Advances in discrete and computational geometry, *Contemporary Mathematics* **223**:251–255, Amer. Math. Soc., 1999.
15. S. V. Pemmaraju. *Exploring the Powers of Stacks and Queues via Graph Layouts.* PhD thesis, Virginia Polytechnic Institute and State Univ., Virginia, U.S.A., 1992.

16. B. A. Reed. Algorithmic aspects of tree width. In *Recent advances in algorithms and combinatorics*, pages 85–107, Springer, 2003.
17. S. Rengarajan and C. E. Veni Madhavan. Stack and queue number of 2-trees. In *Proc. 1st Conf. on Computing and Combinatorics* (COCOON '95), LNCS **959**:203–212, Springer, 1995.
18. D. J. Rose, R. E. Tarjan, and G. S. Leuker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.* **5**:266–283, 1976.
19. D. Seese. Tree-partite graphs and the complexity of algorithms. In *Proc. Conf. Fundamentals of Computation Theory*, LNCS **199**:412–421, Springer, 1985.
20. D. R. Wood. Queue layouts, tree-width, and three-dimensional graph drawing. In *Proc. 22nd Foundations of Software Technology and Theoretical Computer Science* (FST TCS '02), LNCS **2556**:348–359, Springer, 2002.

Resource Allocation Problems in Multifiber WDM Tree Networks*

Thomas Erlebach[1], Aris Pagourtzis[2**], Katerina Potika[2**], and Stamatis Stefanakos[1]

[1] Computer Engineering and Networks Laboratory (TIK),
ETH Zürich, CH-8092 Zürich, Switzerland
{erlebach,stefanak}@tik.ee.ethz.ch
[2] Department of Computer Science, School of ECE,
National Technical University of Athens, Greece
{pagour,epotik}@cs.ntua.gr

Abstract. All-optical networks with multiple fibers lead to several interesting optimization problems. In this paper, we consider the problem of minimizing the total number of fibers necessary to establish a given set of requests with a bounded number w of wavelengths, and the problem of maximizing the number of accepted requests for given fibers and bounded number w of wavelengths. We study both problems in undirected tree networks $T = (V, E)$ and present approximation algorithms with ratio $1 + 4|E| \log |V| / OPT$ and 4 for the former and ratio 2.542 for the latter. Our results can be adapted to directed trees as well.

1 Introduction

Optical communication networks are the most promising technology for satisfying the ever-increasing demand for communication bandwidth. A single optical fiber allows transmission of data at rates up to several Terabits per second. *Wavelength-division multiplexing* (WDM) is used to partition the huge optical bandwidth into channels that operate at different wavelengths. Each channel can carry data at the rate of several Gigabits per second, a speed that can still be handled by electronic network elements. If optical switches (crossconnects) are employed at the nodes of the network, signals can remain in optical form on the whole path from transmitter to receiver, without any need for intermediate conversion to or from electrical form. Such networks are called *all-optical* networks. Advantages of all-optical networks include transparent transport of data in arbitrary formats, low delays, low bit-error rates, and high transmission rates [14].

If a connection is to be established in an all-optical network, a path in the network from the transmitter to the receiver must be determined (*routing*) and a certain wavelength must be reserved for the connection on all links of that path (*wavelength assignment*)

* Partially supported by the Swiss National Science Foundation (project AAPCN) and by EU Thematic Network APPOL II, IST-2001-32007, with funding by the Swiss Federal Office for Education and Science.

** Work mainly done while the authors were visiting ETH Zürich (A. Pagourtzis) and the University of Konstanz (K. Potika), both supported by the Human Potential Programme of EU under contract no HPRN-CT-1999-00104 (project AMORE).

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 218–229, 2003.

[24]. If wavelength conversion is not available, the reserved wavelength must be the same on all links of the path. Since wavelength converters are expensive devices that are not yet commonly available, we will consider only all-optical networks without wavelength converters in this paper.

As the number of wavelengths supported in an all-optical network is a scarce resource, graph-theoretical optimization problems modeling routing and wavelength assignment have been studied intensively over the last 5–10 years. Wavelengths are conveniently represented as colors and the routing and wavelength assignment problem is often treated as a path coloring problem: Given a set of terminal pairs in a graph, connect each terminal pair by a path and assign a color to each path such that no two paths with the same color pass through the same edge and such that the number of colors is minimized. We refer to [3,15,12] for surveys of known results for this problem.

More recently, researchers have begun to investigate optical networks with multiple parallel fibers between adjacent nodes [22,18,17]. In such networks, two paths on the link from u to v can use the same wavelength if they are carried on different fibers. At each node, the signal arriving on any fiber of an incoming link can be forwarded to an arbitrary fiber of an outgoing link, but without changing the wavelength. We model an all-optical network with multiple fibers as a simple graph $G = (V, E)$ with a function $\mu : E \to \mathbb{N}$, where $\mu(e)$ specifies the number of fibers on the link $e \in E$.

We consider all-optical tree networks with multiple fibers. Since the path for a terminal pair is uniquely determined in this case, we will mostly talk about paths instead of terminal pairs in the following. Three optimization problems arise naturally: minimizing the total number of fibers needed (for given paths in G and given number of available colors), minimizing the number of colors (for given paths in G and given μ), and maximizing the number of accepted paths (for given paths in G and given μ and number of available colors). We refer to these problems as MinCollisions-PMC, MinColors-PMC, and Max-PMC, respectively, where PMC is short for "path multicoloring." In this paper, we study MinCollisions-PMC and Max-PMC for tree networks, with the goal of devising efficient approximation algorithms. For each of the problems, we consider also the *uniform* variant where all edges have the same number of fibers and indicate this by using "UPMC" instead of "PMC" in the problem name.

Our Results. In Section 2, we consider the problem MinCollisions-PMC. First, we give an algorithm achieving a total number of fibers that is at most $OPT+3|E|$ for undirected trees in which all paths touch the same node. Then we extend the algorithm to general sets of paths in trees, using at most $OPT + 4|E| \log |V|$ fibers in total. These algorithms perform well if the optimal solution is large. Furthermore, we show how to derive a 4-approximation algorithm for MinCollisions-PMC from an algorithm for MinColors-PMC given in [4]. In addition, we give an efficient algorithm for MinCollisions-UPMC that uses at most $\lceil 3OPT/2 \rceil$ fibers on every edge. We also obtain similar approximation results for the directed case.

In Section 3, we show that a ρ-approximation algorithm for Maximum Path Packing (Max-PP) implies a $1/(1 - e^{-1/\rho})$-approximation algorithm for Max-PMC. Using a known 2-approximation for Max-PP in trees, this implies a 2.542-approximation algorithm for Max-PMC in undirected and directed trees.

A longer version of this paper is available as technical report [9].

Definitions and Preliminaries. A multifiber network is modeled as a graph $G = (V, E)$ with edge weights $\mu : E \to \mathbb{N}$ that represent the number of fibers on each link of the network. The weight function μ may be given in advance, meaning that the number of fibers available on each link is pre-determined, or it may be sought, representing the number of fibers that should be reserved or installed on each link in order to establish a set of connections.

Connections in the network are represented as paths on G. (In general networks, the connections would be represented as terminal pairs, but since we deal with trees, we consider paths instead of terminal pairs.) We will mainly consider the case that G is an undirected graph and that the connections are undirected paths, but we will also discuss how some of our results extend to the directed case (for which we assume that G is a bidirected graph, i.e., a directed graph in which $(u, v) \in E$ implies $(v, u) \in E$, and the connections are directed paths).

We say that a path *touches* a node v if it contains an edge that has v as an endpoint. For a set of paths $\mathcal{P}$ on G the load $L(e, \mathcal{P})$ of edge e is the number of paths in $\mathcal{P}$ that go over e. The load $L(\mathcal{P}) = \max_{e \in E} L(e, \mathcal{P})$ of G is the maximum number of paths that go over the same edge of G. We will omit $\mathcal{P}$ and simply write $L(e)$ and L whenever it is clear from the context which set $\mathcal{P}$ we mean. A wavelength assignment for a set of connections in a multifiber network corresponds to a *multicoloring* of the corresponding paths; in a multicoloring, color collisions are allowed among paths that share an edge (meaning that the corresponding connections will use the same wavelength on different fibers). A path multicoloring is *valid with respect to* μ, if for each edge e the number of times any color appears on paths through e (we refer to this number also as the *maximum number of color collisions* on e) is at most $\mu(e)$. The following optimization problems arise naturally:

Minimum Collisions Path Multicoloring (MinCollisions-PMC). *Given a graph $G = (V, E)$, a set of paths $\mathcal{P}$ on G and a number w, find a path multicoloring of $\mathcal{P}$ with w colors such that $\sum_{e \in E} \mu(e)$ is minimized, where $\mu(e)$ denotes the maximum number of times that any color appears on edge e.*

Minimum Colors Path Multicoloring (MinColors-PMC). *Given a graph $G = (V, E)$ with edge weights $\mu : E \to \mathbb{N}$ and a set of paths $\mathcal{P}$ on G, find a valid, with respect to μ, path multicoloring of $\mathcal{P}$ such that the number of colors used is minimized.*

Maximum Path Multicoloring (Max-PMC). *Given a graph $G = (V, E)$ with edge weights $\mu : E \to \mathbb{N}$, a set of paths $\mathcal{P}$ on G and a number w, find a valid, with respect to μ, path multicoloring of a subset $\mathcal{P}' \subseteq \mathcal{P}$ with w colors such that $|\mathcal{P}'|$ is maximized.*

For all three problems we also consider the case in which the weight function μ is (required to be) a constant, corresponding to situations where the number of fibers has to be the same over all links of the network. We thus obtain three new problems, Minimum Collisions Uniform Path Multicoloring (MinCollisions-UPMC), Minimum Colors Uniform Path Multicoloring (MinColors-UPMC), and Maximum Uniform Path Multicoloring (Max-UPMC). Note that for the problem MinCollisions-UPMC, the objective reduces to minimizing the unique number μ. All these problems are $\mathcal{NP}$-hard for undirected and directed trees, as follows easily from the $\mathcal{NP}$-hardness of path coloring (MinColors-UPMC with $\mu = 1$) [13,7].

For bidirected networks, the edges (u, v) and (v, u) are considered to be separate edges, i.e., each color can appear at most $\mu(u, v)$ times on directed paths containing (u, v) and at most $\mu(v, u)$ times on directed paths containing (v, u). For the problem MinCollisions-PMC, the objective value is the sum of $\mu(e)$ over all directed edges of the graph.

An algorithm A for a minimization problem Π is a ρ-approximation if for every instance I of Π, A runs in time polynomial in $|I|$ and delivers a solution with objective value at most $\rho \cdot OPT(I)$, where $OPT(I)$ denotes the objective value of an optimal solution for I. Similarly, if Π is a maximization problem, A is a ρ-approximation if, for every instance I of Π, it delivers a solution with objective value at least $1/\rho \cdot OPT(I)$. If the problem and the instance under consideration are clear from the context, we will simply use OPT to denote the objective value of an optimal solution for the instance.

Previous Work. Wavelength assignment problems in optical networks have been studied extensively in the literature. In this section we review the known results in the area with emphasis on those related to multifiber networks.

To the best of our knowledge, the MinCollisions-PMC problem was first studied in [22]. It was shown that it can be solved optimally in polynomial time in chain networks, while for star and ring networks 2-approximation algorithms were given. A different algorithm for MinCollisions-PMC in chain networks was presented in [26]. Nomikos et al. [20,21] study a generalization of MinCollisions-PMC with different fiber costs and give constant approximation algorithms for rings and spiders.

Li and Simha [17] and Margara and Simon [18] study the MinColors-UPMC problem in ring and star networks. In [17] it is shown that MinColors-UPMC in rings remains $\mathcal{NP}$-hard for every fixed number of available fibers. An upper bound of approximately $(\mu + 1)L/\mu^2$ wavelengths for ring networks with μ available fibers per link and load L is given. Several restrictions of MinColors-UPMC in ring and star networks are considered as well. Similar results are shown independently in [18]. The latter paper also studies the (multiplicative) gap between the optimal number of colors and the lower bound $\lceil L/\mu \rceil$. For example, it is proved for every $\mu \geq 1$ that there exists an undirected tree where this gap can be at least $1 + 1/(2\mu^2)$. This line of research is continued in [19] where it is proved that for any network G there exists a μ such that any set $\mathcal{P}$ of paths on G can be colored with $\lceil L(\mathcal{P})/\mu \rceil$ colors with respect to μ. In [10], MinCollisions-UPMC and MinColors-UPMC are studied for general graphs and a connection of these problems with hypergraph coloring is established.

If $\mu = 1$, MinColors-UPMC reduces to the well studied path coloring problem (PC). An algorithm using at most $3L/2$ colors for undirected trees was given in [23]. In directed trees, an algorithm using at most $5L/3$ colors is due to Erlebach et al. [8].

Li and Simha [16] studied MinColors-UPMC for undirected trees and showed that a valid multicoloring using at most $\lceil 3L/(2\mu) \rceil$ colors can be computed efficiently. For the general problem MinColors-PMC in trees, a 4-approximation algorithm for undirected and directed trees was given recently by Chekuri, Mydlarz and Shepherd [4]. Motivated by an integer multicommodity flow problem on trees they show that a set $\mathcal{P}$ of paths on a (directed or undirected) tree with $\mu(e)$ available fibers on each link e can be colored using at most $4 \max_{e \in E} \lceil L(e, \mathcal{P})/\mu(e) \rceil$ colors. Nomikos et al. [21] present constant approximation algorithms for MinColors-PMC in rings and spiders,

with approximation ratios 2 (for directed and undirected rings as well as for directed spiders) and $5/2$ (for undirected spiders).

As far as we know, MAX-PMC has not been studied in the literature. For its single-fiber variant, however, known as MAX-PC, a 1.58 approximation is known for the undirected case [25] while for directed trees there exists a slightly worse 2.22-approximation [6]. These algorithms follow from a reduction from MAX-PC with an arbitrary number of wavelengths to MAX-PC with one wavelength [1].

2 Minimizing the Number of Fibers (MINCOLLISIONS-PMC)

In the setting of this problem we are given a graph G, a set of paths $\mathcal{P}$ on G, and w colors. Recall that the objective is to minimize the maximum number of color collisions on each edge (in fact, we want to minimize the sum of these maximum numbers over all edges but, as we will see, local optimization implies total optimization). For an edge e we denote by $OPT(e, \mathcal{P})$ the minimum $\mu(e)$ over all possible multicolorings of $\mathcal{P}$ with w colors; when referring to an algorithm, we denote by $SOL(e, \mathcal{P})$ the number $\mu(e)$ that corresponds to a multicoloring returned by the algorithm.

A lower bound for $OPT(e, \mathcal{P})$ is $OPT(e, \mathcal{P}) \geq \lceil L(e, \mathcal{P})/w \rceil$. For an instance $I = (G, \mathcal{P}, w)$ we have $OPT(I) \geq \sum_{e \in E} OPT(e, \mathcal{P})$ and $OPT(I) \geq |E|$ (we assume w.l.o.g. that every edge is used by at least one path).

Before proceeding to the algorithm for trees with arbitrary path sets we will present an algorithm for a special class of instances: trees with centered path sets.

2.1 Trees with Centered Path Set

Here we deal with trees with a set $\mathcal{P}$ of paths such that all paths touch the same node, called the *center* of $\mathcal{P}$. We consider the path center as root of the tree and denote it by r.

MINCOLLISIONS-PMC is $\mathcal{NP}$-complete in trees with centered path set, since the path set of a star has always a center and it is known [22] that MINCOLLISIONS-PMC in stars is $\mathcal{NP}$-hard. We present an approximation algorithm for MINCOLLISIONS-PMC in trees with centered path set.

Algorithm 1 returns a correct path multicoloring of $\mathcal{P}$ with w colors. This is because H is a w-degree bipartite graph, hence it can be edge-colored with w colors. Since each path corresponds to an edge $e \in A$, after the execution of the edge coloring algorithm all paths have been colored. The most costly part of the algorithm is bipartite edge coloring (step 4), therefore its time complexity is $O(|\mathcal{P}| \log w)$ [5].

We next show that this algorithm is a 4-approximation for MINCOLLISIONS-PMC in trees with centered path set.

Approximation Ratio. In step 1 the subset of paths in $\mathcal{P}$ that pass through edge e is partitioned in two sets: one called $\mathcal{P}_{in}(e)$, containing paths that have direction towards the root, and one called $\mathcal{P}_{out}(e)$, containing the remaining paths (with opposite direction). Let $L_i(e, \mathcal{P}) = |\mathcal{P}_i(e)|$, $i \in \{in, out\}$.

Lemma 1. *For each edge e and each direction i, $i \in \{in, out\}$, the algorithm for* MINCOLLISIONS-PMC *in trees with centered path set colors paths that use e in direction i with at most $\lceil L_i(e, \mathcal{P})/w \rceil + 1$ color collisions.*

Algorithm 1 MINCOLLISIONS-PMC in trees with centered path set.
Input: tree $G = (V, E)$, set of paths $\mathcal{P}$ with center r, number of colors w.
Output: path multicoloring of $\mathcal{P}$ with w colors.

1. Assign an arbitrary direction to every path in $\mathcal{P}$.
2. Create two (initially empty) lists of paths Q_s, Q_f. Traverse the nodes of the tree using postorder traversal and update Q_s, Q_f as follows:
   For each node visited do
   append all paths that start at this node to list Q_s and all paths that finish at this node to list Q_f.
3. Create a set S of starting groups of paths and a set F of finishing groups of paths:
   While Q_s, Q_f not empty do
   delete w elements from Q_s and group them together to form a new group s and add s to S; delete w elements from Q_f and group them together to form a new group f and add f to F.
   Construct a bipartite graph $H(S, F, A)$, where there is an edge $e \in A$ for each path $p \in \mathcal{P}$ (connecting the groups that contain p).
4. Color the edges of H with w colors using the algorithm in [5]. Assign to each path the color of the corresponding edge e of H.

Proof. Consider an edge $e = \{u, v\}$; w.l.o.g. assume that v is the parent of u (w.r.t. the root r). We will prove the claim for paths in direction in ($\mathcal{P}_{in}(e)$); the proof for direction out is completely symmetric.

Notice that paths in $\mathcal{P}_{in}(e)$ are exactly the paths that start at u or at some descendant of u. Hence, these paths are consecutive in the list Q_s because postorder visits the subtrees of u and u itself consecutively (of course, any other standard tree traversal would do, as long as it visits a node and its subtrees in a consecutive manner). Therefore the algorithm puts paths in $\mathcal{P}_{in}(e)$ in several full groups (i.e. consisting of w paths each) and at most two semi-full groups. Let the number of paths in the first group (which can be semi-full or full) be k, and let the total number of paths in the remaining groups be k'. The number of groups is clearly $1 + \lceil k'/w \rceil = \lceil k/w \rceil + \lceil k'/w \rceil \leq \lceil (k + k')/w \rceil + 1 = \lceil L_{in}(e, \mathcal{P})/w \rceil + 1$. The number of color repetitions, for any color, is at most the number of groups. Thus, we have $SOL_{in}(e, \mathcal{P}) \leq \lceil L_{in}(e, \mathcal{P})/w \rceil + 1$. □

Proposition 1. *The algorithm for* MINCOLLISIONS-PMC *in trees with centered path set gives a path multicoloring such that on each edge e the maximum number of color collisions is at most $\lceil L(e, \mathcal{P})/w \rceil + 3 \leq OPT(e, \mathcal{P}) + 3$.*

Proof. In step 1 the algorithm partitions the load on each edge e in two parts: $L_{in}(e, \mathcal{P})$ paths that go towards the root and $L_{out}(e, \mathcal{P})$ paths in opposite direction. The number of collisions of any color on edge e is at most $SOL_{in}(e, \mathcal{P})$ for paths in $\mathcal{P}_{in}(e)$ and at most $SOL_{out}(e, \mathcal{P})$ for paths in $\mathcal{P}_{out}(e)$. Thus, the total number of collisions of any color on e is $SOL(e, \mathcal{P}) \leq SOL_{in}(e, \mathcal{P}) + SOL_{out}(e, \mathcal{P}) \leq \lceil L_{in}(e, \mathcal{P})/w \rceil + 1 + \lceil L_{out}(e, \mathcal{P})/w \rceil + 1 \leq \lceil L(e, \mathcal{P})/w \rceil + 3 \leq OPT(e, \mathcal{P}) + 3$. □

Theorem 1. *The algorithm for* MINCOLLISIONS-PMC *in trees with centered path set is a $(1 + \frac{3|E|}{OPT})$-approximation in the undirected case.*

Proof. Adding the solutions for all edges and using Proposition 1 we have for any instance $I = (G, \mathcal{P}, w)$: $SOL(I) \leq \sum_{e \in E} SOL(e, \mathcal{P}) \leq \sum_{e \in E} (\lceil \frac{L(e,\mathcal{P})}{w} \rceil + 3) \leq OPT(I) + 3|E|$. □

Remark: The above algorithm is a 4-approximation in the worst case, since $OPT \geq |E|$; however, the approximation ratio improves as the load increases.

A useful observation is that in trees with centered path set where the center is of degree 2 the above algorithm achieves a better ratio:

Corollary 1. *The algorithm for* MinCollisions-PMC *in trees with centered path set where the center is of degree 2, is a* $(1 + \frac{|E|}{OPT})$*-approximation in the undirected case, that is a* 2*-approximation in the worst case.*

Sketch of Proof. This is due to an improvement of step 1 of Algorithm 1. Since there are only two subtrees of the root, assigning to each path a direction from the left subtree to the right subtree we end up with edges that have only paths in one direction, thus for each edge it holds that $SOL(e, \mathcal{P}) \leq OPT(e, \mathcal{P}) + 1$. □

MinCollisions-PMC in Directed Trees with Centered Path Set. For the directed case of the problem we take advantage of the fact that directions of paths are given, therefore there is no loss due to arbitrary selection of direction. This leads to the following improved approximation results (we omit the proof of Proposition 2).

Proposition 2. *The algorithm for* MinCollisions-PMC *in directed trees with centered path set* $\mathcal{P}$ *gives a path multicoloring such that on each edge* e *the maximum number of color collisions is at most* $\lceil L_{in}(e, \mathcal{P})/w \rceil + \lceil L_{out}(e, \mathcal{P})/w \rceil + 2 \leq OPT(e, \mathcal{P}) + 2$ *if* e *is used by paths in both directions and at most* $OPT(e, \mathcal{P}) + 1$ *if* e *is used in only one direction.*

Theorem 2. *The algorithm for* MinCollisions-PMC *in directed trees with centered path set is a 2-approximation.*

Proof. Let E_1 be the set of edges which have paths in only one of the two directions and E_2 be the set of edges with paths in both directions. An obvious lower bound is $OPT(I) \geq |E_1| + 2 \cdot |E_2|$. By Proposition 2, we have $SOL(I) \leq \sum_{e \in E_1} (OPT(e, \mathcal{P}) + 1) + \sum_{e \in E_2} (OPT(e, \mathcal{P}) + 2) \leq OPT(I) + |E_1| + 2 \cdot |E_2| \leq 2 \cdot OPT(I)$. □

2.2 MinCollisions-PMC in Trees with Arbitrary Path Set

Now we present an algorithm for trees with arbitrary path sets. It is based on a partitioning of the path set $\mathcal{P}$ into $O(\log |V|)$ disjoint sets $\mathcal{P}_i$ using a separator-based approach (as in [2]), such that each $\mathcal{P}_i$ consists of disjoint sets of paths with common center. Then the algorithm MinCollisions-PMC for trees with centered path set can be used for each component of $\mathcal{P}_i$. Combining the solutions, we will obtain a $(1 + O(\frac{|E| \log |V|}{OPT}))$-approximation for MinCollisions-PMC in trees.

The Algorithm. It is easy to see that every tree $T = (V, E)$ contains a vertex v such that $T - \{v\}$ is a forest of trees with at most $|V|/2$ nodes each. We call such a vertex

a $\frac{1}{2}$-separator. We assign levels to the vertices of T as follows: find a $\frac{1}{2}$-separator v in T and assign to it level 1; then find a $\frac{1}{2}$-separator in each tree of $T - \{v\}$ and assign to each of them level 2; repeat this step (remove all vertices of the current level i, find a $\frac{1}{2}$-separator in each of the remaining trees, and assign to all of the newly found separators the level $i + 1$) until all vertices have been assigned levels. Now let the level of a path in $\mathcal{P}$ be the minimum level of all vertices that it touches. For $1 \leq i \leq \log |V|$, let $\mathcal{P}_i$ be the set of paths in $\mathcal{P}$ with level i. This partitions $\mathcal{P}$ into $t \leq \log |V|$ subsets $\mathcal{P}_i$, where each subset $\mathcal{P}_i$ is a disjoint union of centered path sets that do not interfere with each other. Thus, Algorithm 1 can be applied to $\mathcal{P}_i$ by applying it to each centered path set in $\mathcal{P}_i$ separately, and Proposition 1 still holds. Thus we get the following algorithm:

Algorithm 2 MINCOLLISIONS-PMC in trees
Input: tree G, set of paths $\mathcal{P}$, number of colors w.
Output: path multicoloring of $\mathcal{P}$ with w colors.

1. Partition the path set $\mathcal{P}$ into subsets $\mathcal{P}_i$, $1 \leq i \leq t$, as described in the text.
2. Find a path multicoloring with w colors for each subset $\mathcal{P}_i$ using the algorithm for trees with centered path set (Algorithm 1).

Approximation Ratio. The total load of each edge is $L(e, \mathcal{P}) = \sum_{i=1}^{t} L(e, \mathcal{P}_i)$. For the multicoloring returned by Algorithm 2 let $SOL(e, \mathcal{P}_i)$ denote the maximum number of color collisions on edge e among paths that belong to the subset $\mathcal{P}_i$. Since each $\mathcal{P}_i$ is colored by using Algorithm 1 we have $SOL(e, \mathcal{P}_i) \leq \lceil L(e, \mathcal{P}_i)/w \rceil + 3$ by Proposition 1. Hence, in each edge e, for any color, the total number of color collisions is at most $SOL(e, \mathcal{P}) \leq \sum_{i=1}^{t} SOL(e, \mathcal{P}_i) \leq \sum_{i=1}^{t} (\lceil L(e, \mathcal{P}_i)/w \rceil + 3) \leq \lceil \sum_{i=1}^{t} L(e, \mathcal{P}_i)/w \rceil + 4 \cdot t - 1 = \lceil L(e, \mathcal{P})/w \rceil + 4 \cdot t - 1 \leq OPT(e, \mathcal{P}) + 4 \cdot t - 1$. Therefore, for any instance $I = (G, \mathcal{P}, w)$, $SOL(I) \leq \sum_{e \in E} SOL(e, \mathcal{P}) \leq \sum_{e \in E} (OPT(e, \mathcal{P}) + 4 \cdot t - 1) \leq OPT(I) + (4 \cdot t - 1) \cdot |E| < (1 + \frac{4|E| \log |V|}{OPT(I)}) \cdot OPT(I)$. We have thus proved the following:

Theorem 3. *The algorithm for* MINCOLLISIONS-PMC *in trees is a* $(1 + \frac{4|E| \log |V|}{OPT})$*-approximation in the undirected case.*

Remark: If $OPT(I) = \Omega(|E| \log |V|)$, then Algorithm 2 is a constant approximation; for example, if $OPT(I) \geq 4|E| \log |V|$, Algorithm 2 achieves approximation ratio 2. This renders the algorithm particularly useful for heavily loaded instances. However, in the worst case the approximation guarantee is not better than $O(\log |V|)$; an algorithm that always achieves a constant approximation is presented in Subsection 2.3.

The Directed Case. By Proposition 2, for each centered path set we have $SOL(e, \mathcal{P}_i) \leq \lceil L_{in}(e, \mathcal{P}_i)/w \rceil + \lceil L_{out}(e, \mathcal{P}_i)/w \rceil + 2$. The overall solution is thus: $SOL(e, \mathcal{P}) \leq \sum_{i=1}^{t} (\lceil L_{in}(e, \mathcal{P}_i)/w \rceil + \lceil L_{out}(e, \mathcal{P}_i)/w \rceil + 2) \leq OPT(e, \mathcal{P}) + 4 \cdot t - 2$. For edges $e \in E_1$ it holds that $SOL(e, \mathcal{P}) \leq OPT(e, \mathcal{P}) + 2 \cdot t - 1$. Therefore, $SOL(I) \leq \sum_{e \in E_1} (OPT(e, \mathcal{P}) + 2 \cdot t - 1) + \sum_{e \in E_2} (OPT(e, \mathcal{P}) + 4 \cdot t - 2) \leq OPT(I) + (2 \cdot t - 1)(|E_1| + 2 \cdot |E_2|) < OPT(I) + 2|E'| \log |V|$, where E' is the set of active directed edges (we call a directed edge *active* if there are paths that use it). We get:

Theorem 4. *The algorithm for* MinCollisions-PMC *in trees is a* $(1 + \frac{2|E'|\log|V|}{OPT(I)})$*-approximation in the directed case, where* E' *is the set of active directed edges.*

2.3 A 4-Approximation for MinCollisions-PMC in Trees

The algorithm is based on a very recent 4-approximation algorithm for the problem MinColors-PMC. For MinColors-PMC, it is clear that $\max_{e\in E}\lceil L(e,\mathcal{P})/\mu(e)\rceil$ is a lower bound on the number of colors used by the optimal solution. Chekuri, Mydlarz, and Shepherd [4] gave a polynomial algorithm that comes within a factor of 4 of this lower bound. We can state their result as follows:

Theorem 5 (Chekuri, Mydlarz and Shepherd, 2003). *There is a* 4*-approximation for* MinColors-PMC *in trees. It uses at most* $4\max_{e\in E}\lceil L(e,\mathcal{P})/\mu(e)\rceil$ *colors.*

Using this algorithm as a subroutine, we can obtain a 4-approximation algorithm for MinCollisions-PMC.

Corollary 2. *There exists a* 4*-approximation algorithm for* MinCollisions-PMC *in undirected trees.*

Proof. Let an instance of MinCollisions-PMC be given. Set $\mu'(e) = \lceil L(e,\mathcal{P})/w\rceil$ for every edge $e \in E$. Next, compute a valid path multicoloring with respect to μ' using the algorithm mentioned in Theorem 5. This will need at most $4\max\lceil L(e,\mathcal{P})/\mu'(e)\rceil \le 4w$ colors. Now, we identify four colors at a time (i.e., assuming that the colors are numbered $1, 2, \ldots$, we map each color i to color $\lceil i/4\rceil$). This reduces the number of colors by a factor of 4 and increases the number of color repetitions on every edge at most by a factor of 4. Thus, we have a path multicoloring with at most w colors that is feasible with respect to μ, where $\mu(e) = 4\mu'(e)$ for every $e \in E$. Since $OPT(e,\mathcal{P}) \ge \mu'(e)$, the corollary follows. □

The result extends to the directed case as well, since the algorithm in [4] applies both to undirected and directed trees.

2.4 Uniform Number of Fibers: MinCollisions-UPMC

In this section we consider the problem MinCollisions-UPMC, where the number of fibers has to be the same over all links of the network. The goal of MinCollisions-UPMC is to minimize the uniform μ (equivalently, to compute a path multicoloring that minimizes $\max_{e\in E}\mu(e)$). Algorithm 3 uses as a subroutine the algorithm for path coloring (PC) in undirected trees by Raghavan and Upfal [23] that colors a path set $\mathcal{P}$ on a tree with no more than $3L(\mathcal{P})/2$ colors.

Theorem 6. *The algorithm for* MinCollisions-UPMC *in trees computes a multicoloring with at most* $\lceil 3\cdot OPT/2\rceil$ *color collisions on any edge in the undirected case.*

Proof. Given an instance $I = (G,\mathcal{P},w)$, for any path multicoloring of $\mathcal{P}$ it holds $\max_{e\in E}\mu(e) \ge \lceil L(\mathcal{P})/w\rceil$, hence $OPT(I) \ge \lceil L(\mathcal{P})/w\rceil$. After using the algorithm for PC in trees it holds that all paths using an edge have different colors in the range $0,\ldots,s-1$, $s \le 3\cdot L(\mathcal{P})/2$. Hence, the number of color collisions for any color is at most $\lceil \frac{3}{2}L(\mathcal{P})/w\rceil \le \lceil \frac{3}{2}\cdot OPT(I)\rceil$ in the resulting multicoloring. □

Algorithm 3 MINCOLLISIONS-UPMC in trees.
Input: tree G, set of paths $\mathcal{P}$, number of colors w.
Output: path multicoloring of $\mathcal{P}$ with w colors minimizing $\mu = \max_{e \in E} \mu(e)$.

1. Color all paths in $\mathcal{P}$ using colors in the range $0, \ldots, s-1$, $s \leq 3L(\mathcal{P})/2$.
2. Replace each color x with color $x \bmod w$.

The approximation ratio of the above algorithm is thus $\frac{3}{2} + \frac{1}{2OPT} \leq 2$.

For MINCOLLISIONS-UPMC in directed trees, we follow a similar approach using an algorithm for PC in directed trees that colors all paths with at most $5 \cdot L(\mathcal{P})/3$ colors [8]. This leads to at most $\lceil \frac{5}{3} \cdot OPT \rceil$ collisions and approximation ratio $\frac{5}{3} + \frac{2}{3OPT}$ which is at most 2 for $OPT \geq 2$.

3 Maximizing the Number of Connections

Now we consider the MAX-PMC problem and propose a constant-ratio approximation algorithm for tree networks. Our algorithm uses as a subroutine an approximation algorithm for the MAXIMUM PATH PACKING problem, formally defined as follows.

MAXIMUM PATH PACKING (MAX-PP). *Given a graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{N}$ and a set $\mathcal{P}$ of paths on G, find a valid, with respect to c, path packing of $\mathcal{P}$, i.e., a subset $\mathcal{P}' \subseteq \mathcal{P}$ such that no more than $c(e)$ paths in $\mathcal{P}'$ go over edge e. The goal is to maximize $|\mathcal{P}'|$.*

Algorithm 4 MAX-PMC
Input: graph $G = (V, E)$ with edge weights $\mu : E \to \mathbb{N}$, set $\mathcal{P}$ of paths, number of colors w.
Output: a valid, with respect to μ, path multicoloring of a subset $\mathcal{P}' \subseteq \mathcal{P}$ with w colors.

```
set P' := ∅;
for i = 1 to w do
    use an algorithm for MAX-PP to find a valid, with respect to μ, path packing P_i of P;
    assign color i to all paths in P_i and set P' := P' ∪ P_i;
    remove all paths in P_i from P;
```

Theorem 7. *The above algorithm for* MAX-PMC *is a $1/(1 - e^{-1/\rho})$-approximation if a ρ-approximation algorithm for* MAX-PP *is used as a subroutine.*

The proof of Theorem 7 is a straightforward adaptation of the technique used by Awerbuch et al. [1] for reducing MAX-PC (the single-fiber version of MAX-PMC) with an arbitrary number of wavelengths to MAX-PC with one wavelength and is omitted. Theorem 7 applies to arbitrary graphs; for trees we obtain a 2.542-approximation algorithm by using the existing 2-approximation algorithm for MAX-PP from [11].

Corollary 3. *There exists a 2.542-approximation algorithm for* MAX-PMC *in tree networks.*

We note that the same result can be obtained for the directed case by using the MAX-PP algorithm given in [6] (note that in [6] MAX-PP is defined for uniform edge capacities; however, as the authors already note therein, their algorithm can be adapted to work in the case of arbitrary edge capacities).

4 Conclusion

We have studied several optimization problems motivated by multifiber optical networks. For the problem of minimizing the total number of fibers (MINCOLLISIONS-PMC), our algorithm computes a solution of cost at most $OPT + 4|E| \log |V|$ in undirected trees $T = (V, E)$, thus achieving approximation ratio $1 + \frac{4|E| \log |V|}{OPT}$. Using the recent 4-approximation algorithm for MINCOLORS-PMC in [4], we also derived a 4-approximation algorithm for MINCOLLISIONS-PMC in undirected or directed trees. For the problem of maximizing the number of accepted requests, we presented a general reduction from MAX-PMC to MAX-PP and obtained a 2.542-approximation algorithm for undirected and directed trees.

We remark that our results can be generalized to MINCOLLISIONS-PMC with different fiber costs on different edges (because our analysis compares the number of fibers on each individual edge with the number of optimal fibers on the edge) and to MAX-PMC with different profits for different requests (by using a constant-factor approximation for the weighted version of MAX-PP such as the 4-approximation for MAX-PP in trees given in [4]). Concerning future work, it would be interesting to see whether the approximation ratios can be improved further and whether good results can also be obtained for other network topologies.

Acknowledgments. We would like to thank Chandra Chekuri, Marcelo Mydlarz and Bruce Shepherd for the permission to use a preliminary version of [4] in this work. We are indebted to Chandra Chekuri for suggesting the current proof of Corollary 2. Finally, we are grateful to the anonymous referees for helpful comments.

References

1. B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Rosén. On-line competitive algorithms for call admission in optical networks. *Algorithmica*, 31(1):29–43, 2001.
2. B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. Competitive non-preemptive call control. In *Proceedings of the 5th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'94)*, pages 312–320, 1994.
3. B. Beauquier, J.-C. Bermond, L. Gargano, P. Hell, S. Perennes, and U. Vaccaro. Graph problems arising from wavelength-routing in all-optical networks. In *Proceedings of IPPS'97, Second Workshop on Optics and Computer Science (WOCS)*, 1997.
4. C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree. In *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming (ICALP'03)*, LNCS 2719, pages 410–425, 2003.
5. R. Cole, K. Ost, and S. Schirra. Edge-coloring bipartite multigraphs in $O(E \log D)$ time. *Combinatorica*, 21(1):5–12, 2001.

6. T. Erlebach and K. Jansen. Maximizing the number of connections in optical tree networks. In *Proceedings of the 9th Annual International Symposium on Algorithms and Computation (ISAAC'98)*, LNCS 1533, pages 179–188, 1998.
7. T. Erlebach and K. Jansen. The complexity of path coloring and call scheduling. *Theoretical Computer Science*, 255(1–2):33–50, 2001.
8. T. Erlebach, K. Jansen, C. Kaklamanis, M. Mihail, and P. Persiano. Optimal wavelength routing in directed fiber trees. *Theoretical Computer Science*, 221:119–137, 1999.
9. T. Erlebach, A. Pagourtzis, K. Potika, and S. Stefanakos. Resource allocation problems in multifiber WDM tree networks. TIK-Report 178, Computer Engineering and Networks Laboratory (TIK), ETH Zürich, August 2003.
10. A. Ferreira, S. Perennes, A. Richa, H. Rivano, and N. Stier. On the design of multifiber WDM networks. In *Proc. AlgoTel'02*, pages 25–32, Mèze, France, May 2002.
11. N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.
12. L. Gargano and U. Vaccaro. Routing in all-optical networks: Algorithmic and graph-theoretic problems. In I. Althöfer et al., editors, *Numbers, Information and Complexity*, pages 555–578. Kluwer Academic Publishers, 2000.
13. M. C. Golumbic and R. E. Jamison. The edge intersection graphs of paths in a tree. *J. Comb. Theory Series B*, 38(1):8–22, February 1985.
14. P. E. Green. *Fiber Optic Networks*. Prentice Hall, Englewood Cliffs, NJ, 1993.
15. R. Klasing. Methods and problems of wavelength-routing in all-optical networks. Technical Report CS-RR-348, Department of Computer Science, University of Warwick, September 1998. Presented as invited talk at the MFCS'98 Workshop on Communications.
16. G. Li and R. Simha. On the wavelength assignment problem in multifiber optical tree networks. In *Terabit Optical Networking: Architecture, Control, and Management Issues*, number 4213 in Proceedings of SPIE, pages 84–91, November 2000.
17. G. Li and R. Simha. On the wavelength assignment problem in multifiber WDM star and ring networks. *IEEE/ACM Transactions on Networking*, 9(1):60–68, 2001.
18. L. Margara and J. Simon. Wavelength assignment problem on all-optical networks with k fibres per link. In *Proceedings of the 27th International Colloquium on Automata, Languages, and Programming (ICALP 2000)*, LNCS 1853, pages 768–779, 2000.
19. L. Margara and J. Simon. Decidable properties of graphs of all-optical networks. In *Proceedings of the 28th International Colloquium on Automata, Languages, and Programming (ICALP'01)*, LNCS 2076, pages 518–529, 2001.
20. C. Nomikos, A. Pagourtzis, K. Potika, and S. Zachos. Path multi-coloring in weighted graphs. In *Proceedings of the 8th Panhellenic Conference on Informatics*, volume I, pages 178–186, Nicosia, Cyprus, November 2001.
21. C. Nomikos, A. Pagourtzis, K. Potika, and S. Zachos. Fiber cost reduction and wavelength minimization in multifiber WDM networks. Manuscript, 2003.
22. C. Nomikos, A. Pagourtzis, and S. Zachos. Routing and path multicoloring. *Information Processing Letters*, 80(5):249–256, 2001.
23. P. Raghavan and E. Upfal. Efficient routing in all-optical networks. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC'94)*, pages 134–143, 1994.
24. R. Ramaswami and K. N. Sivarajan. Routing and wavelength assignment in all-optical networks. *IEEE/ACM Transactions on Networking*, 3(5):489–500, October 1995.
25. P.-J. Wan and L. Liu. Maximal throughput in wavelength-routed optical networks. In *Multichannel Optical Networks: Theory and Practice*, volume 46 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 15–26. AMS, 1998.
26. P. Winkler and L. Zhang. Wavelength assignment and generalized interval graph coloring. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*, pages 830–831, 2003.

An Improved Upper Bound on the Crossing Number of the Hypercube

Luerbio Faria[1⋆], Celina M. Herrera de Figueiredo[2⋆⋆], Ondrej Sýkora[3⋆⋆⋆], and Imrich Vrťo[4†]

[1] Departamento de Matemática, Faculdade de Formação de Professores, Universidade do Estado do Rio de Janeiro-UERJ, São Gonçalo, RJ, Brazil
[2] Department of Computer Science, Institute of Mathematics, UFRJ Caixa Postal 68530, Rio de Janeiro, RJ, Brazil
[3] Department of Computer Science, Loughborough University Loughborough, Leicestershire LE11 3TU, The United Kingdom
[4] Department of Informatics, Institute of Mathematics Slovak Academy of Sciences Dúbravská 9, 841 04 Bratislava, Slovak Republic

Abstract. We draw the n-dimensional hypercube in the plane with $\frac{5}{32}4^n - \left\lfloor \frac{n^2+1}{2} \right\rfloor 2^{n-2}$ crossings, which improves the previous best estimation and coincides with the long conjectured upper bound of Erdös and Guy.

1 Introduction

The crossing number of a graph G, denoted by $\mathrm{cr}(G)$, is the minimum number of crossings of its edges among all drawings of G in the plane. It is a fundamental topological invariant but appears naturally in the design of VLSI circuits [9] and visualization of graph like structures [2]. The problem is NP-hard [6]. There are only a few infinite families of graphs for which exact crossing numbers are known. See e.g., surveys [10,12]. One of the most challenging problems is the crossing number of the hypercube. In 1969 Harary [8] mentioned that there does not even exist a conjecture about the crossing number of the hypercube. Then Eggleton and Guy [3] announced a drawing which implies that for $n \geq 3$

$$\mathrm{cr}(Q_n) \leq \frac{5}{32}4^n - \left\lfloor \frac{n^2+1}{2} \right\rfloor 2^{n-2}. \tag{1}$$

⋆ Partially supported by CNPq, CAPES, FAPERJ and FINEP, Brazilian research agencies and PROCIÊNCIA/FAPERJ-UERJ Project.
⋆⋆ Supported in part by CNPq, CAPES, FAPERJ and FINEP.
⋆⋆⋆ This research was supported in part by the EPSRC grant Nr. GR/R37395.
† Work supported by the VEGA grant Nr. 2/3164/23.

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 230–236, 2003.

Later a gap was found in the construction [7]. However, Erdös and Guy [4] conjectured equality in (1). Madej [11] proposed a drawing of Q_n with

$$\frac{1}{6}4^n - n^2 2^{n-3} - 3 \cdot 2^{n-4} + \frac{1}{48}(-2)^n$$

crossings and showed that $\mathrm{cr}(Q_5) \leq 56$. Sýkora and Vrt'o [13] proved that Madej's bound is asymptotically optimal, by deriving the following lower bound:

$$\mathrm{cr}(Q_n) \geq \frac{1}{20}4^n + O(n^2 2^n).$$

Then Dean and Richter [1] showed that $\mathrm{cr}(Q_4) = 8$, which is the only exact and nontrivial result in this area. Recently, Faria and Figueiredo [5] proved that $\mathrm{cr}(Q_6) \leq 352$ and decreased the Madej's upper bound to

$$\frac{165}{1024}4^n - (2n^2 - 11n + 34)2^{n-3}, \tag{2}$$

for $n \geq 7$, which coincides with the RHS of (1) for $n = 7, 8$.

In this paper we construct a new drawing of Q_n in the plane which has the conjectured number of crossings

$$\frac{5}{32}4^n - \left\lfloor \frac{n^2+1}{2} \right\rfloor 2^{n-2}.$$

2 Definitions

The hypercube Q_n is defined in the standard way. The vertices are all binary strings of length n and two vertices are adjacent iff the corresponding strings differ in precisely one position.We say that an edge belongs to the i-th dimension if its endvertices (strings) differ in the i-th position from the left. A vertex is called an *even* vertex if the number of 1's in its corresponding string is even. Let C_m denote an m-vertex cycle, $m \geq 3$. For graphs G and H let $G \times H$ denote their Cartesian product.

3 The Drawing

Theorem 1. *For $n \geq 3$,*

$$\mathrm{cr}(Q_n) \leq \frac{5}{32}4^n - \left\lfloor \frac{n^2+1}{2} \right\rfloor 2^{n-2}.$$

Proof. We consider the cases for n odd and n even separately. First we prove by induction that the result holds for all odd n. Second, we use the result for n odd to establish the corresponding result for $n + 1$.

3.1 Odd Case

Let $n = 2k - 1$, for $k \geq 2$. The main idea of the construction is the following. Assume that we have a suitable drawing D_n of Q_n in the plane with the claimed number of crossings, denoted by $\mathrm{cr}(D_n)$. We utilize the identity $Q_{n+2} = Q_n \times C_4$. We replace every vertex v of Q_n in the "small" neighbourhood of v in the drawing D_n by a 4-cycle. The small neighborhood is a circle centered at v, which contains no crossings and no other vertices. Then every edge e in D_n which started in v will be replaced by 4 "parallel" edges drawn along the original edge e. Doing this carefully we get a drawing D_{n+2} of Q_{n+2}. The total number of crossings will be the number of crossings between edges starting in the vertices of the 4-cycle times 2^n plus $16\mathrm{cr}(D_n)$. The term 16 comes from the fact that every crossing in D_n is replaced by 4 parallel edges crossed by another 4 parallel edges in D_{n+2}.

We will construct inductively a drawing D_n of Q_n satisfying the following four properties:

1. The number of crossings in the drawing is:
$$\mathrm{cr}(D_n) = \frac{5}{32}4^n - \left\lfloor \frac{n^2+1}{2} \right\rfloor 2^{n-2}.$$
2. Edges of the same dimension do not cross.
3. In any 4-cycle $abcd$, where ab, cd are of the n-th dimension and ad, bc are of the i-th dimension, $i < n$, it holds that the number of edges drawn between edges ab and ad in the counterclockwise order and between dc and da in the clockwise order is the same. See Figure 1a.

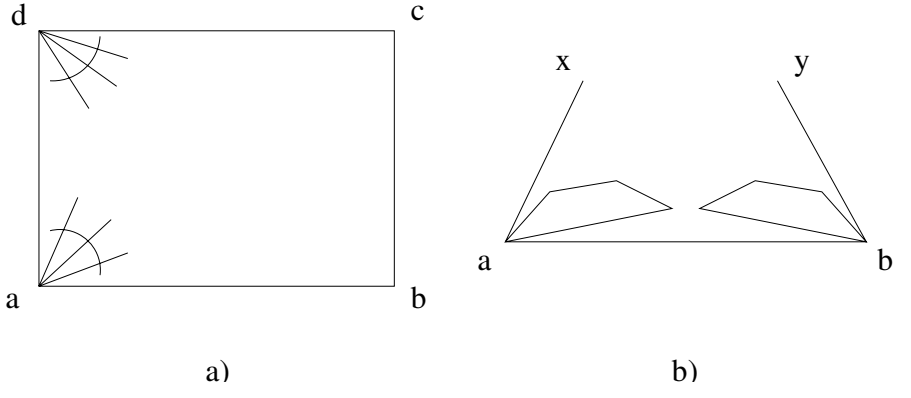


Fig. 1.

4. In the drawing of D_n take any edge ab of the n-th dimension, where a is an even vertex. In a small neighborhood of the vertex a a 4-cycle is attached to a as shown in Figure 1b, i.e. the 4-cycles do not interfere with the current

drawing and the 4-cycle is placed between the edge ab and its neighbour ax in the counterclockwise order.
Draw a symmetrical 4-cycle attached at the vertex b lying between the edge ab and its neighbour by in the clockwise order. Call these 4-cycles *new* 4-cycles. Consider again any 4-cycle $abcd$, where ab, cd are edges of the $n-$th dimension and ad, bc are of the i-th dimension, $i < n$ and a is an even vertex. Starting with the edge ab let the edge of the $i-$th dimension be the $r(i)$-th in the clockwise direction around the vertex a.

a) If $r(i) > k$ then the new four 4-cycles attached to a, b, c, d lie inside the cycle $abcd$ as one can see in Figure 2a.

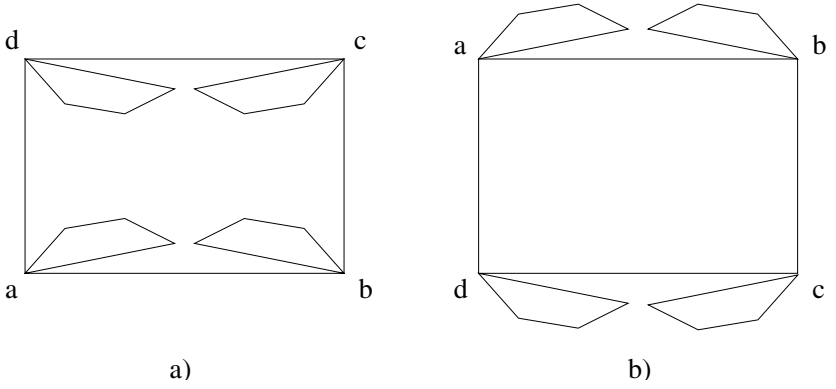


Fig. 2.

b) If $r(i) \leq k$ then the new 4-cycles attached to a, b, c, d lie outside the $abcd$ as one can see in Figure 2b.

Now we start the induction for odd n. Let $n = 3$. Then Figure 3 shows a drawing of Q_3 satisfying the above 4 properties.

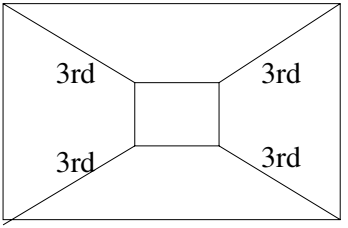


Fig. 3.

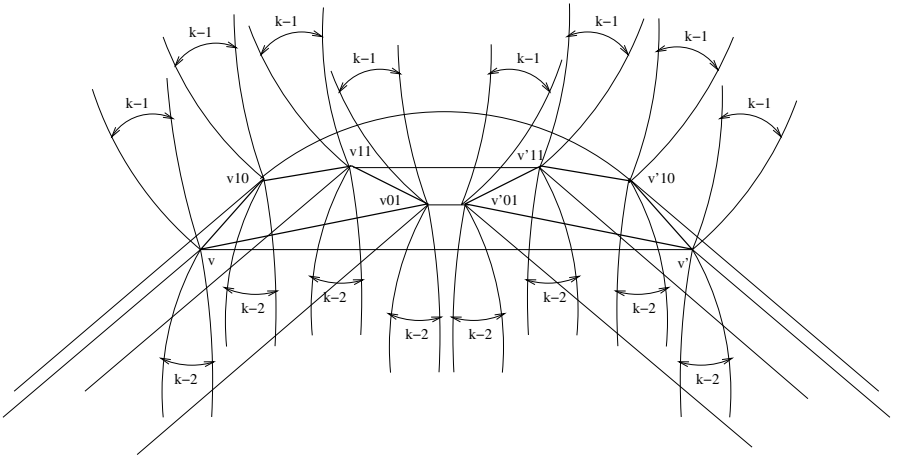


Fig. 4.

Assume we have a drawing of Q_n, for some $n \geq 3$ satisfying the four properties. In what follows, we construct a drawing of Q_{n+2} satisfying the properties. Take any even vertex v in the drawing of Q_n. Let v' be its neighbour along the n-th dimension. Without producing new crossings, draw 2 new 4-cycles on vertices $v00 = v, v01, v11, v10$ and $v'00 = v', v'01, v'11, v'10$ s.t. the edge $v00, v01$ ($v'00, v'01$) is a "neighbour" of the edge vv', in the drawing. The edge pattern in the original vertices v and v' is repeated in all vertices of the cycle corresponding to v and v', respectively. And the new edges are routed as depicted in the Figure 4.

We complete the drawing of Q_{n+2} by description of the routing of the new edges. The drawing of the edges of the i-th dimension, $i = n, n+1, n+2$, is obvious from Figure 4. Let $i < n$. Consider any 4-cycle v, v', u, u' in the drawing of Q_n, where the edges vv' and uu' belong to the n-th dimension and the edges vu' and $v'u$ belong to the i-th dimension, and v is an even vertex. Starting with the edge vv', let the edge of the i-th dimension be the $r(i)$-th in the clockwise direction around the vertex v. Consider now the corresponding cycle $v00, v'00, u00, u'00$ in the "partial" drawing of Q_{n+2}.

Distinguish 3 cases (see Figure 5):

1. Let $r(i) > k$. Note that in this case the new 4-cycles lie inside the 4-cycle $vv'uu'$. Draw the connection between vjl and $u'jl$ for $j, l, = 0, 1, j + l > 0$ parallel with the "old" edge $v00$ and $u'00$ using the pattern valid for the "upper" bunch of $k - 1$ edges.
2. Let $r(i) = k$. In this case the new 4-cycles lie outside the 4-cycle $vv'uu'$. Use the connection pattern valid for the k-th edge.
3. Let $r(i) < k$. In this case again the new 4-cycles lie outside the 4-cycle $vv'uu'$. Use the connection pattern valid for the "lower" bunch of $k - 2$ edges.

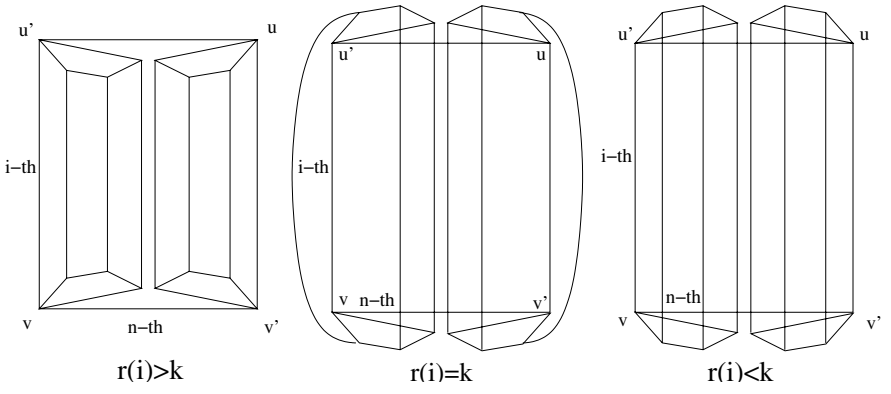


Fig. 5.

We make a similar drawing for the 4-cycles attached to $v'00$ and $u00$. The drawing immediately implies that the produced graph is Q_{n+2}.

It remains to show that the drawing of Q_{n+2} satisfies the properties 1-4. We have

Lemma 1. *The number of crossings between the edges incident to the vertices of the new 4-cycle attached to* $v00$ *is* $(3n^2 - 4n - 1)/2$.

Proof. To appear in the full version. □

Then the total number of crossings in the drawing D_{n+2} of Q_{n+2} is

$$\begin{aligned}\mathrm{cr}(D_{n+2}) &= 16\mathrm{cr}(D_n) + \frac{3n^2 - 4n - 1}{2}2^n \\ &= 16\left(\frac{5}{32}4^n - \left\lfloor\frac{n^2+1}{2}\right\rfloor 2^{n-2}\right) + (3n^2 - 4n - 1)2^{n-1} \\ &= \frac{5}{32}4^{n+2} - \left\lfloor\frac{(n+2)^2+1}{2}\right\rfloor 2^n.\end{aligned}$$

The second property is obviously fulfilled. The third and fourth properties can be checked by taking the 4-cycle $v00, v01, u'01, u'00$, formed by edges of the $(n+2)$-nd dimension and the i-th dimension, $i < n+2$ and using the inductive assumption. The inductive step for the odd case is completed.

3.2 Even Case

We start with the drawing D_n of Q_n described above, for n odd. We utilize the identity $Q_{n+1} = Q_n \times P_2$, where P_2 denotes the 2-vertex path. The construction proceeds similarly as above but in a simpler way. To every vertex u in D_n, attach a *new edge* uv in a small neighbourhood of u immediately above the edge of the n-th dimension. For every edge e, starting in u, draw a new edge which starts

in v and goes "parallel" to e. Doing this in a similar way as in the odd case, up to the case where rank of the dimension of the edge e is k (this case is treated in the same way as for the dimensions of rank less than k), we get a drawing of Q_{n+1}.

Lemma 2. *The number of crossings between edges incident to the vertices of the new edge is* $((n-1)/2)^2$.

Proof. To appear in the full version. □

Since for each edge of the drawing D_n of Q_n there are 2 edges in the drawing D_{n+1} of Q_{n+1}, then for each crossing in D_n we have 4 additional crossings in D_{n+1}. As we proved that $cr(D_n) = \frac{5}{32}4^n - \frac{n^2+1}{2}2^{n-2}$, then by Lemma 2 the drawing D_{n+1} obtained from the drawing D_n contains the following number of crossings:

$$\begin{aligned}\mathrm{cr}(D_{n+1}) &= 4\mathrm{cr}(D_n) + \left(\frac{n-1}{2}\right)^2 2^n = 4\left(\frac{5}{32}4^n - \frac{n^2+1}{2}2^{n-2}\right) + \left(\frac{n-1}{2}\right)^2 2^n \\ &= \frac{5}{32}4^{n+1} - \left\lfloor \frac{(n+1)^2+1}{2} \right\rfloor 2^{n-1}.\end{aligned}$$

□

References

1. Dean, A.M., Richter, R.B., The crossing number of $C_4 \times C_4$, *J. Graph Theory* **19** (1995), 125–129.
2. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G., Graph Drawing: Algorithms for Visualisations of Graphs, Prentice Hall, New Jersey, 1999.
3. Eggleton, R.B., Guy, R.P., The crossing number of the n-cube, *Notices AMS* **17** (1970), 757.
4. Erdös, P., Guy, R.P., Crossing number problems, *American Mathematical Monthly* **80** (1973), 52–58.
5. Faria, L., Figueiredo, C.M.H., On Eggleton and Guy's conjectured upper bound for the crossing number of the n-cube, *Mathematica Slovaca* **50** (2000), 271–287.
6. Garey, M.R., Johnson, D.S., Crossing number is NP-complete, *SIAM J. Discrete Mathematics* **4** (1983), 312–316.
7. Guy, R.P., Latest results on crossing numbers, in: Proc. *Recent Trends in Graph Theory*, Lecture Notes in Mathematics 186, Springer, New York, 1971, 143–156.
8. Harary, F., Graph Theory, Addison Wesley, Reading, 1969.
9. Leighton, F.T., Complexity Issues in VLSI, MIT Press, 1983.
10. Liebers, A., Planarizing graphs - a survey and annotated bibliography, *J. Graph Algorithms and Applications* **5** (2001), 1–74.
11. Madej, T., Bounds for the crossing number of the n-cube. *J. Graph Theory* **15** (1991), 81–97.
12. Shahrokhi, F., Sýkora, O., Székely, L.A., Vrťo, I., Crossing numbers: bounds and applications, in: Intuitive Geometry, (I. Bárány, K. Böröczky, eds.), Bolyai Society Mathematical Studies 6, Akadémia Kiadó, Budapest, 1997, 179–206.
13. Sýkora, O., Vrťo, I., On the crossing number of the hypercube and cube connected cycles, *BIT* **33** (1993), 232–237.

NCE Graph Grammars and Clique-Width

Alexander Glikson* and Johann A. Makowsky**

Department of Computer Science
Technion - Israel Institute of Technology
Haifa, Israel
{aglik,janos}@cs.technion.ac.il

Abstract. Graph grammars are widely used in order to define classes of graphs having some inductive and narrow structure. It is known, that graph classes defined by context-free graph grammars have bounded clique-width, but this general observation does not give the bound on the clique-width explicitly.
We investigate here the explicit relationship between various not necessarily context-free Neighborhood Controlled Embedding (NCE) graph grammars and the clique-width of graphs generated by them. We show that all the graphs, generated by any given NCE graph grammar, have an explicitly computable bounded clique-width (where the bound depends only on parameters of the grammar), and provide the corresponding algorithms (based on dynamic programming techniques) for finding clique-width expression based on given derivation tree.
All the results are first obtained for Node Label Controlled (NLC) grammars, but can be generalized to both NCE grammars and $edNCE$ grammars (for directed graphs with dynamic edge relabelling).

Keywords: Graph grammars, clique-width, tree-width.

1 Introduction

The purpose of this paper is to find an explicit relationship between a wide class of graph grammars, the Neighborhood Controlled Embedding (NCE) graph grammars, and bounds for the tree-width and clique-width of graph languages generated by them. We express these bounds (provided they exist) in terms of parameters of the grammar only.

The first results of this kind were proven by C. Lautemann in [Lau88], where he derived upper bounds for the tree-width of graph languages generated by a given Hyperedge-Replacement (HR) grammar. In section 4 we use similar techniques to derive bounds for the tree-width for two subclasses of NCE grammars, Apex NLC and Boundary NLC.

Courcelle and Olariu in [CO00] showed that clique-width of graphs of tree-width k, is at most $2^{k+1} + 1$. Therefore, any graph language of bounded tree-width, is automatically of bounded clique-width. Moreover, B. Courcelle, J.

* The results are part of the first author's M.Sc. thesis [Gli03].

** Partially supported by a Grant of the Fund for Promotion of Research of the Technion–Israeli Institute of Technology.

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 237–248, 2003.

Engelfriet and G. Rozenberg in [CER93] provided a complicated proof that any given context-free graph grammar based on vertex-replacement (Confluent $edNCE$, or context-free VR grammar) generates graphs of bounded clique-width. Although an upper bound for the clique-width could be derived from their proof, it is not straightforward.
In general, finding an explicit bound for the clique-width is a more complicated task than finding a bound for the tree-width.

In this paper we derive direct explicit upper-bounds for the clique-width of graphs generated by any given Node Label Controlled (NLC) grammar (which is a particular case of NCE), not necessarily confluent (context-free). We also improve the known bounds for the tree-width for several special cases of NLC grammars. For this purpose we use a modified concept of tree-width, introduced first in [FV93], which allows us to achieve more accurate bounds for the clique-width of corresponding graphs. More specifically, for a graph generated by an NLC grammar, we provide, in Theorems 12, 13 and 14, polynomial time algorithms which generate, from a derivation tree for the graph, a parse term for the clique-width of this graph, where the clique-width is independent of the graph and depends only on specific parameters of the grammar. Our main results are:

Theorem 1 *Let K be a graph language generated by an NLC grammar. Then K is of bounded clique-width.*

Using these algorithms, which produce clique-width parse term from a derivation tree, one can apply the dynamic programming techniques introduced in [CMR00] to the clique-width parse term, to obtain the following result. $CMSOL$ denotes Monadic Second Order Logic augmented with modular counting quantifiers, cf. [EF95].

Theorem 2 *Let K be a class of graphs generated by an NLC grammar. Let Φ be a graph property definable in $CMSOL$. Then checking whether a graph $G \in K$ has property Φ is polynomial time Turing reducible to the computation of a derivation tree for G.*

We note that the computation of a derivation tree can be **PSpace**-hard for NLC grammars and **NP**-hard for Linear Apex NLC grammars, cf. [F98,K01].

Analogue results like Theorems 1 and 2 can be proven for classes of graphs generated by an NCE grammar, by an $eNCE$ grammar, where one also allows edges with labels, and by an $edNCE$ grammar, where additionally the edges may be directed. But in this extended abstract we do not have enough space to develop the details.

The corresponding bounds on the clique-width are given in Theorem 14. Proofs are only sketched, due to space limitations, and may be found in [Gli03].

2 Vertex-Replacement Graph Grammars

A graph grammar is a generalized type of grammar, which mimics in a natural way the usual Chomsky hierarchy in the case of graph-like structures. One should note however, that there are context-sensitive languages of words, which,

if looked at as graphs, are context-free. In this paper we investigate graph classes generated by general, not necessarily context-free, graph grammars based on vertex replacement, the Neighborhood Controlled Embedding (NCE) graph grammars. In this extended abstract, however, we restrict the presentation to the NLC grammars, where no edge labels are allowed, and also vertices with the same label in the right-hand side graphs of productions are indistinguishable during the embedding. Surveys on graph grammars are [DKH97,Eng97,K97]. In an NLC grammar, a nonterminal vertex can be substituted by some graph according to the grammar rules, while the edges between the neighbors of the nonterminal and the inner vertices of the right hand side graph are established according to the corresponding *embedding*[1] relation. There are several restricted subtypes of NLC grammars, according to the structure of the right hand side graphs in the grammar rules, and according to the permitted embedding.

Apex NLC**:** There are no edges between nonterminal vertices in right hand side graphs. The embedding can only connect terminals.

Linear NLC**:** The right hand side graph can contain at most one nonterminal. The embedding can connect terminals, or a terminal and a nonterminal.

Boundary NLC**:** There are no edges between nonterminal vertices in the right hand side graphs. The embedding can connect terminals, or a terminal and a nonterminal.

Confluent NLC**:** Any kind of right hand side graphs or embedding is permitted, as long as the grammar is confluent (i.e. the order of applying the substitutions does not affect the resulting graph).

For a grammar Γ we denote by $L(\Gamma)$ the class of graphs generated by Γ and call it the *graph language generated by* Γ. We denote by X-Y-NLC the class of graph languages generated by an X-Y-NLC grammar, where $X \in \{Lin, \epsilon\}$ and $Y \in \{A, B, C, \epsilon\}$, ϵ is the empty word and A stands for Apex, B for Boundary and C for Confluent. C-NCE is sometimes also called **Context Free VR**, e.g. in [ER90,Cou95].

From [K97,K01] we have the following strict inclusions and complexities[2]:

Proposition 3

(i) *Lin-A-*$NLC \subset$ *Lin-*$NLC \subset$ *B-*$NLC \subset$ *C-*$NLC \subset NLC$
(i) *Lin-A-*$NLC \subset$ *A-*$NLC \subset$ *B-*$NLC \subset$ *C-*$NLC \subset NLC$
(iii) *Lin-*NLC *and A-*NLC *are incomparable.*
(iv) *Every* NLC *grammar generates a graph language in* **PSpace** *and some of them are* **PSpace***-complete.*

[1] For lack of space we skip the detailed definition of these embedding relations. They will be included in the full paper and can be found most conveniently in [K97]. Informally, for each pair of labels (α, β) in the embedding relation, every α-labelled inner vertex is connected to every β-labelled neighbor.

[2] The corresponding inclusions of grammars do not always hold as is, but for every X and Y, where the corresponding languages inclusion holds: X-$NLC \subset$ Y-NLC, every X-NLC grammar could be easily transformed into an equivalent Y-NLC grammar (in linear time and with similar grammar parameters), generating the same language.

(v) Every B-NLC grammar generates a graph language in **NP**.
(vi) There exists a Lin-A-NLC grammar which generates a **NP***-complete graph language.*

In particular, this also gives lower bounds for the complexity of finding derivation trees for NLC gramamrs.

Now we define several parameters of NLC graph grammars, which will be used later in order to express corresponding bounds of tree-width and clique-width.

Definition 4 (NLC parameters) *Let Γ be the given NLC grammar with a set of nonterminals $A_1, A_2, \ldots$ and production rules $A_i \to (H_i, emb_i)$. We define the following parameters:*

n - *maximal number of vertices in H_i*
k - *maximal number of terminals in H_i*
m - *maximal number of nonterminals in H_i*
d - *maximal degree of nonterminals in H_i*
α - *size of the terminals alphabet*
β - *size of the nonterminals alphabet*
γ - *maximal number of different terminal labels in H_i ($\gamma \leq min(k, \alpha)$)*

3 Tree-Width and Clique-Width

Here we define the notions of tree-width and clique-width. We use a refinement of the classical notion of tree-width, introduced in [FV93]. For survey on classical tree-width see [Bod98].

Definition 5 (Tree-Width) *A* (k,m)-tree decomposition *of graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$ where $\{X_i \mid i \in I\}$ - a family of subsets of V, one for each node of T, and T a tree such that*

- $\bigcup_{i \in I} X_i = V$.
- *for all edges $(v, w) \in E$ there exists an $i \in I$ with $v \in X_i$ and $w \in X_i$.*
- *for all $i, j, k \in I$: if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.*
- *for all $i \in I$, $|X_i| \leq k + 1$.*
- *for every two adjacent nodes $i, j \in I$ ($(i, j) \in F$), $|X_i \cap X_j| \leq m$.*

A graph G is of tree-width(k,m) *if there exists a (k,m)-tree decomposition of G.*

Note that the classical tree-width is equivalent to the case $m = k$ [3].

Definition 6 (Clique-Width, [CO00]) *Given graph $G = (V, E)$, the* clique-width of G (denoted by *cwd(G)) is the minimal number of colors required to obtain the given graph from colored singletons, using disjoint union ($\oplus$), recoloring ($\rho_{i \to j}$) and edge creation ($\eta_{i,j}$).*

[3] Given a graph G and $k \in \mathbb{N}$ there are efficient algorithms which determine whether G has tree-width k, and if yes, produce a tree decomposition, cf. [Bod97]. The $O(n^2)$ algorithm can be generalized for tree-width(k,m), but the existence of the linear one remains open.

A description of a graph using these operations is called a *clique-width parse term* (or *parse term*, if no confusion arises). The simplest class of graphs of unbounded tree-width but of clique-width at most 2 are the cliques. Given a graph G and $k \in \mathbb{N}$, determining whether G has clique-width k is in **NP**. A polynomial time algorithm was presented for $k \leq 3$ in [CHLetal]. It remains open whether for some $k \geq 4$ the problem is **NP**-complete. It will follow from our work in Section 6, that in the case of graph languages generated by NLC grammars, an upper bound of the clique-width of a graph can be computed in polynomial time from a derivation tree of the graph [4]. On the one hand the upper bound obtained does not depend on the particular derivation (only on the grammar); on the other hand, the upper bound may be far from optimal.

In [CM02] the classes of graphs generated by C-$edNCE$ grammars (context-free VR-grammars) are characterized as those defined as the least solution of systems of recursive set equations based on the operations used in the definition of clique-width. Also in [CM02], based on [CE95,Cou92,EvO97], a characterization of context-free Hyperedge Replacement grammars (HR-grammars) is given in similar terms adapted to the operations used in computing a graph from its tree decomposition (disjoint union, renaming and fusion).

4 *NLC* Graph Grammars of Bounded Tree-Width

4.1 Apex *NLC*

The following theorem summarizes our results concerning the tree-width and clique-width of graphs generated by A-NLC grammar (the proof is available in the full version):

Theorem 7 *Let Γ be an A-NLC grammar, and G a graph in $L(\Gamma)$. Then*

(i) G is at most of tree-width$(k + d, d)$.
(ii) The clique-width of G is bounded by $k + \gamma + 1$.

Note. Unlike the general case (see section 5), in the case of an A-NLC grammar the clique-width is *polynomial* in the tree-width, rather than exponential.

4.2 Linear/Boundary *NLC* with Bounded Nonterminal Degree

Definition 8 *An NLC grammar Γ has* bounded nonterminal degree *if there exists some constant $D \in \mathbb{N}$ depending only on the grammar Γ, such that the degree of every nonterminal in every derivation of Γ is bounded by D.*

Note. Although the maximal degree of the nonterminals in H_i is bounded by d, the degree of the nonterminals during a derivation may grow arbitrarily.

Theorem 9 *Let Γ be a B-NLC grammar of bounded nonterminal degree, and G be a graph in $L(\Gamma)$. Then the tree-width of G is bounded by tree-width$(k+D, D)$.*

[4] Moreover, for given k, there exists and $edNCE$ grammar that generates all the graphs of clique-width $\leq k$.

Note. The clique-width could be exponential in k (see Theorem 13).

Combining the results of B. Courcelle, in [Cou95] and of J. Engelfriet and G. Rozenberg in [ER90], we conclude:

Theorem 10 *Let K be a class of graphs generated by a C-edNCE grammar. Then K can be generated by a B-edNCE grammar with bounded nonterminal degree if and only if it is of bounded tree-width.*

Note. If we remove the nonterminal degree restriction, already in the Lin-NLC case, the tree-width of corresponding graphs could be arbitrarily large. For example, the following Lin-NLC grammar (fig. 1) generates the class of all complete graphs (which is known to have unbounded tree-width).

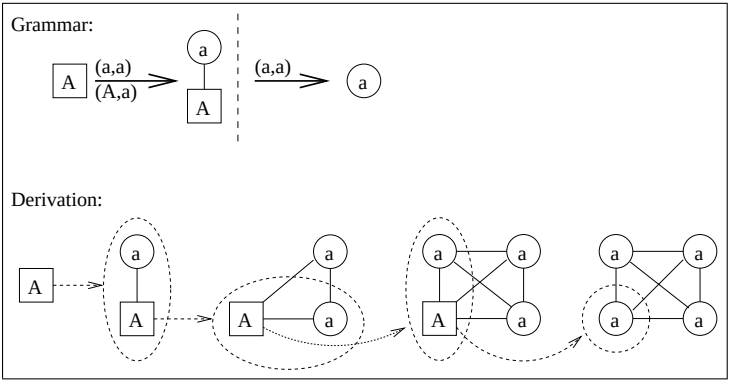


Fig. 1. Linear NLC grammar generating all cliques, and derivation of K_4

5 Clique-Width of Graphs of Bounded Tree-Width

B. Courcelle and S. Olariu in [CO00] proved that graphs of tree-width k have clique-width at most $2^{k+1} + 1$ [5].

We use the refined definition of tree-width(k,m) (see Definition 5) in order to refine the corresponding clique-width bound:

Theorem 11 *Let G be a graph of tree-width(k,m).*
Then the clique-width of G is at most $2^{m+1} + (k - m) + 1$.

[5] This result was improved to $3 \cdot 2^{k-1}$ by D. Corneil and U. Rotics in [CR01], and also an explicit algorithm for building clique-width parse term was introduced, based on the corresponding k-tree that embeds the given graph.

Notice that for the standard tree-width (m=k), we obtain exactly the same result as the one by B. Courcelle and S. Olariu[6].

Proof. The algorithms proposed here are based on dynamic programming techniques. A large set of labels is used in order to *remember*, i.e. encode with a label, the connectivity of the specific vertex to other vertices, which were not yet represented in the clique-width term.

Sketch of Algorithm. We perform two passes over the tree decomposition:

(i) first pass, top-down, while for each node we update two labelling functions:
  (i.a) *new* - new vertices introduced in this node (might appear only in the subtree rooted by the current node),
  (i.b) *old* - all the other vertices.

(ii) second pass, bottom-up (building the clique-width expression):
  (ii.a) Every vertex is inserted into the clique-width term (disjoint union) when it appears in the *new* set of the current node (i.e. at the root of the subtree induced by this vertex)
  (ii.b) All the vertices in the current *new* set are labelled with unique labels
  (ii.c) Edges are established between the new vertices if needed (η operations)
  (ii.d) For each child of the current node:
    i. Assume that all the vertices that appear in the corresponding parse term are colored using 2^m colors according to the subset of the *old* set of vertices of the child, adjacent to the vertex in question.
    ii. Proceed disjoint union with the parse term representing the child.
    iii. Additional edges are established between every new vertex and all vertices that are adjacent to it in the original graph and were already activated before, according to their labels (labels that remember that they should be connected to this specific vertex) (η operations).
    iv. Every vertex is re-labelled according to the corresponding subset of vertices in the *old* set of the current node, which are adjacent to the current vertex (ρ operations).

Since in each step we need to remember the subset of old vertices adjacent to the current vertex, the total number of colors required for this algorithm is $(k-m)+1$ (for *new* vertices) + 2^m (labels used by children) + 2^m (another set of labels used for relabelling before the next step). □

6 Clique-Width of Graphs Generated by *NLC* Graph Grammars

When a grammar derivation tree is given, we can build the corresponding clique-width parse term using dynamic programming. We are going to parse the derivation tree bottom-up, and create a clique-width term for every derivation node (representing its subtree).

[6] Optimizations similar to those introduced in [CR01] could be applied here as well.

6.1 Linear *NLC*

In the case of a Lin-NLC grammar, every introduced terminal must remember what is the subset of terminal labels, that this vertex should be connected to. Hence we get:

Theorem 12 *Let Γ be a Lin-NLC grammar and G be a graph in $L(\Gamma)$. Then the clique-width of G is bounded by $2^\alpha + k$.*

6.2 Boundary *NLC*

In case of a B-NLC grammar, we also have to remember the subset of terminal labels, that this vertex should be connected to by the embedding. However, in order to handle several children, we will need another set of $(2^\alpha - 1)$ colors.

Theorem 13 *Let Γ be a B-NLC grammar and G a graph in $L(\Gamma)$. Then the clique-width of G is bounded by $2^{\alpha+1} + k - 1$.*

Proof. The proof is given by the algorithm below. In the appendix, the algorithm is explained in details using a generic example.

Sketch of Algorithm. Given a derivation tree, proceed bottom-up, for each node:

(i) color terminals v_i of the current node with unique colors, $c_i(v_i)$, proceed disjoint union ($\oplus$), establish edges between them as needed (η_{c_i,c_j}).

(ii) For each child of the current node:

(ii.a) perform disjoint union with the graph defined by the subexpression corresponding to the current child ($\oplus$).

(ii.b) connect *new* terminals to the *old* ones: for each terminal v_i labelled y and adjacent to the nonterminal deriving the current child, establish edges between c_i and all the *old* labels $y\bar{z}$ (where $\bar{z}$ - any vector of terminal labels), that *remember* that the corresponding vertices *should be connected to y-labelled vertices*: $\eta_{c_i,y\bar{z}}$

(ii.c) update *old* colors for next step, assuming that current child is derived from nonterminal labelled Y, and the current node is H_i. Let $\bar{z} = \{z|(Y,z) \in emb_{H_i}\}$. For each color of some vertices in the current child, $\bar{y}$, recolor $\bar{y}$ to the intersection between $\bar{y}$ and $\bar{z}$ ($\rho_{\bar{y}\to\bar{y}'\cap\bar{z}'}$).

(iii) Update new colors (terminals of the current node H_i): recolor terminal vertex with label x and colored c_i, with the color indicating the following connectivity ($\rho_{c_i\to\bar{y}}$):

(iii.a) remember the connectivity (conn) $\{y|(x,y) \in emb_{H_i}\}$

The number of colors used by this algorithm is $2^\alpha - 1$ for all the nonempty combinations of terminal labels to be connected to the current vertex (plus another $2^\alpha - 1$ colors for children recoloring), plus k colors for *new* vertices, and another one color for "uncolor". The algorithm indeed produces clique-width expression for the graph, represented by the given derivation tree, since we have succeeded in encoding by the colors all the needed connectivity information in order to establish exactly the edges, produced by the derivation. Formal proof of the correctness of the algorithm can be found in [Gli03]. □

6.3 General NLC

In case of general NLC, the idea is similar to B-NLC, while we need to remember connectivity to both terminals and nonterminals, and not just terminals. In addition, we will need to remember the original label of the vertex, in order to establish correctly the edges in case of non-confluent grammar. This will enlarge the number of needed colors, and will affect the steps (6.2), (6.2) and (6.2) of the algorithm. The details will be given in the full version of the paper.

Theorem 14 *Let Γ be an NLC grammar and G be a graph in $L(\Gamma)$. Then the clique-width of G is bounded by $\alpha \cdot 2^{\alpha+\beta+1} + k - 1$.*

Sketch of Proof. In case of a non-confluent grammar Γ, we have to modify the algorithm of Theorem 13 in order to construct parse term corresponding to some specific derivation order. On the basis of the information which of the adjacent nonterminals was substituted first, we apply η operations according to the corresponding embedding relation and the vertex labels. The resulting parse term does depend on the derivation order. However, the number of colors does not. □

7 Conclusions and Further Research

In this paper we have investigated the explicit relationship between NCE graph grammars and the clique-width of the graphs generated by them. Our main result is that any given NCE grammar generates graphs with bounded clique-width, while the bound depends only on the parameters of the grammar, and not on specific derivation. Moreover, given a derivation tree of a graph, we have shown an algorithm for efficient construction of a clique-width parse term representing the given graph which gives a good but not necessarily optimal upper bound to the true clique-width. The same applies to $edNCE$ grammars. In [KJ99] a wider class of graph grammars is introduced, the $HRNCE$ grammars (Hyperedge-Replacement grammars with NCE-style embedding mechanism). There it is also shown that every recursively enumerable graph language can be generated by an $HRNCE$ grammar. Therefore, the square grids $Grid_{n,n}$ form an $HRNCE$ language, as they are clearly recursive. However, they are known to have unbounded clique-width, cf. [GR00]. This shows that our results cannot be extended to $HRNCE$ grammars.

In order to evaluate a CMSOL-property of a graph in a NCE graph language, we start therefore with a derivation tree, and compute from it the clique-width parse term using the algorithm from the proof of Theorem 14. From there on, one can proceed with the standard techniques from [CMR00].

The question arises, if one can use the derivation sequence for the graph with respect to an NCE grammar directly for the efficient evaluation of CMSOL-properties of the graph. For this, one would have to verify that each vertex replacement operation is CMSOL-smooth in the sense of [CM02,Mak03]. We shall discuss this in details in the full version of the paper.

Acknowledgments. We would like to thank B. Courcelle and E. V. Ravve for finding time to look at the extended abstract and convey us some valuable comments.

References

[Bod97] H. Bodlaender. Treewidth: Algorithmic techniques and results. In I. Privara and P. Ruzicka, editors, *Proceedings of the 22th International Symposium on the Mathematical Foundation of Computer Science, MFCS'97*, volume 1295 of *Lecture Notes in Computer Science*, pages 29–36. Springer, 1997.

[Bod98] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. Theoret. Comput. Sci. 209:1–45, 1998.

[CE95] B. Courcelle and J. Engelfriet. A logical characterization of the sets of hypergraphs defined by hyperedge replacement grammars. *Mathematical Systems Theory*, 28:515–552, 1995.

[CER93] B. Courcelle, J. Engelfriet and G. Rozenberg. Handle-rewriting hypergraph grammars. Journal of computer and system sciences 46:218–270, 1993.

[CHLetal] D. Corneil, M. Habib, J. Lanlignel,B. Read and U. Rotics. Polynomial time recogntion of clique-width ≤ 3 graphs. LNCS 1726:126–134, 2000.

[CO00] B. Courcelle and S. Olariu. Upper bounds to the clique-width of graphs. Discrete Applied Mathematics 101:77–114, 2000.

[CMR00] B. Courcelle, J.A. Makowsky and U. Rotics, Linear time solvable optimization problems on graphs of bounded clique-width, Theory Comput. Systems 33 (2000) 125–150.

[CM02] B. Courcelle and J.A. Makowsky. Fusion on relational structures and the verification of monadic second order properties. Mathematical Structures in Computer Science, 12:in print, 2002.

[CR01] D. Corneil and U. Rotics. On the relationship between clique-width and tree-width. LNCS 2204:78–90, 2001.

[Cou92] B. Courcelle. Monadic second-order logic of graphs VII: Graphs as relational structures. *Theoretical Computer Science*, 101:3–33, 1992.

[Cou95] B. Courcelle. Structural properties of context-free sets of graphs generated by vertex replacement. Information and Computation 116:275–293, 1995.

[DKH97] F. Drewes, H-J. Kreowski, H. Habel. Hyperedge replacement. In *Handbook of graph grammars and computing by graph transformations, Vol. 1: Foundations*, G. Rozenberg, editor. pages 95–162. World Scientific, 1997.

[Eng97] J. Engelfriet. Context-free graph grammars. In *Handbook of formal languages, Vol. 3: Beyond Words*, pages 125–213. Springer, 1997.

[EF95] H.D. Ebbinghaus and J. Flum. Finite Model Theory. Perspectives in Mathematical Logic. Springer, 1995.

[ER90] J. Engelfriet and G. Rozenberg. A comparison of boundary graph grammars and context-free hypergraph grammars. Information and Computation 84:163–206, 1990.

[EvO97] J. Engelfriet and V. van Oostrom. Logical description of context-free graph-languages. *Journal of Computer and System Sciences*, 55:489–503, 1997.

[F98] M. Flasiński. Power properties of NLC graph grammars with a polynomial membership problem. Theoretical Computer Science 201:189–231, 1998.

[FV93] T. Feder and M. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction. In *STOC'93*, pages 612–622. ACM, 1993.

[GR00] M. C. Golumbic and U. Rotics. On the clique-width of some perfect graph classes. *Internation Journal of Foundations of Computer Science*, 11:423–443, 2000.

[Gli03] A. Glikson. M. Sc. thesis. The Technion, Israeli Institute of Technology, 2003. http://www.cs.technion.ac.il/ admlogic/readme.html

[K97] Changwook Kim. A hierarchy of $eNCE$ families of graph languages. Theoretical Computer Science 186:157–169, 1997.

[K01] Changwook Kim. Efficient recogniztion algorithms for boundary and linear $eNCE$ graph languages. Acta Informatica 37:619–623, 2001.

[KJ99] Changwook Kim and Tae Eui Jeong. $HRNCE$ grammars - a hypergraph generating system with an $eNCE$ way of rewriting. Theoretical Computer Science 223:143–178, 1999.

[Lau88] C. Lautemann. Decomposition trees: structured graph representation and efficient algorithms, LNCS 299:28–39, 1988.

[Mak03] J.A. Makowsky. Algorithmic uses of the Feferman-Vaught theorem. submitted to the Special Issue of APAL with papers from the Tarski Centennary Conference (January 2003)

[RS86] Robertson, N. and Seymour, P. D., "Graph Minors. II. Algorithmic Aspects of Treewidth," J. Algorithms, 7(1986), pp. 309–322.

Appendix: Theorem 13 proof by Example

Let's look at the B-NLC grammar Γ at figure 2. Capital letters in squares indicate nonterminals, while small letters in circles indicate terminals. For each production rule $A_i \rightarrow (H_i, emb_i)$, the corresponding subgraph H_i is specified, together with the embedding relation emb_i, while the first label of each pair indicates the label of a vertex inside the subgraph (terminal or nonterminal), and the second - label of the neighbor (terminal).

Moreover, figure 3 shows one of possible derivation trees of Γ. We are going to use it in order to explain our algorithm for producing clique-width term of the graph (Theorem 13), derived from this derivation tree.

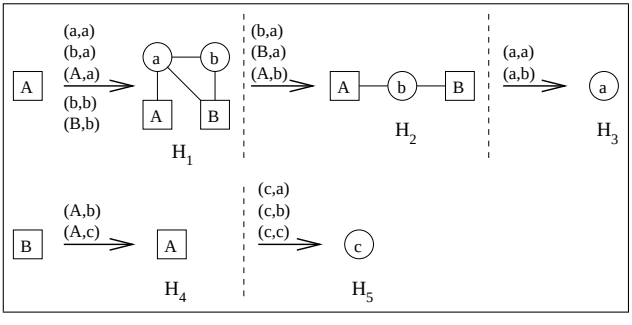


Fig. 2. Boundary NLC grammar Γ

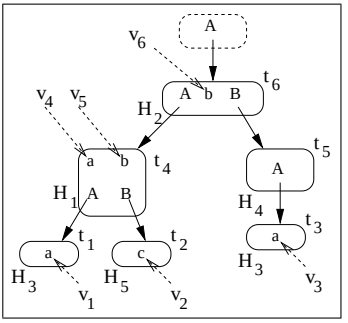


Fig. 3. Derivation tree of a graph $G \in L(\Gamma)$

We traverse the tree decomposition bottom-up, and for each node generate the corresponding t_i clique-width expression. The term t_6, after processing all the nodes except the axiom, is the resulting clique-width expression.

Colors: 0 = "uncolor"
1,**2** = *new* colors
a = should be connected to a
b = should be connected to b
ab = should be connected to both a and b
\+ a', b', a'b' for recoloring

t_1**:** (i) $\mathbf{1}(v_1)$
(ii) no children/subexpressions
(iii) v_1 is labelled "a"; $(a,a),(a,b) \in emb_{H_3} \Rightarrow \underline{\text{conn}}$ {a,b}
$\Longrightarrow t_1 = \rho_{1\to ab}(\mathbf{1}(v_1))[\Rightarrow v_1 : ab]$
...
t_4**:** (i) $t'_4 = \eta_{1,2}(\mathbf{1}(v_4) \oplus \mathbf{2}(v_5))$
(ii) • first child (derived from A):
(ii.a) $t''_4 = t'_4 \oplus t_1$
(ii.b) v_4 is labelled "a" and is adjacent to A $\Rightarrow \eta_{1,a\bar{y}}[\Rightarrow (v_4, v_1)]$
(ii.c) $(A,a) \in emb_{H_1} \Rightarrow z = \{a\} \Rightarrow \rho_{\bar{y}\to\bar{y}\cap\{a\}}[\Rightarrow v_1 : ab \to a']$
• second child (derived from B):
(ii.a) $\ldots \oplus t_2$
(ii.b) v_4 is labelled "a" and is adjacent to B $\Rightarrow \eta_{1,a\bar{y}}[\Rightarrow (v_4, v_2)]$;
v_5 is labelled "b" and is adjacent to B $\Rightarrow \eta_{2,b\bar{y}}[\Rightarrow (v_5, v_2)]$
(ii.c) $(B,b) \in emb_{H_1} \Rightarrow z = \{b\} \Rightarrow \rho_{\bar{y}\to\bar{y}\cap\{b\}}[\Rightarrow v_2 : abc \to b']$
(iii) $v_4 : (a,a) \in emb_{H_1} \Rightarrow \underline{conn}\{a\} \Rightarrow \rho_{1\to a}[\Rightarrow v_4 : a]$;
$v_5 : (b,a),(b,b) \in emb_{H_1} \Rightarrow \underline{conn}\{a,b\} \Rightarrow \rho_{1\to ab}[\Rightarrow v_5 : ab]$
(iv) reuse colors: $\rho_{\bar{x}'\to\bar{x}}[\Rightarrow v_1 : a' \to a, v_2 : b' \to b]$
...

Chordal Probe Graphs
(Extended Abstract)

Martin Charles Golumbic and Marina Lipshteyn

Caesarea Rothschild Institute of Computer Science, University of Haifa, Israel
golumbic@cs.haifa.ac.il

Abstract. In this paper, we introduce the class of chordal probe graphs which are a generalization of both interval probe graphs and chordal graphs. A graph G is chordal probe if its vertices can be partitioned into two sets P(probes) and N(non-probes) where N is a stable set and such that G can be extended to a chordal graph by adding edges between non-probes. We show that a chordal probe graph may contain neither an odd-length chordless cycle nor the complement of a chordless cycle. We give polynomial time recognition algorithms for the subfamily of weakly chordal graphs which are also chordal probe, first in the case of a fixed given partition of the vertices into probes and non-probes, and second in the more general case where no partition is given.

1 Introduction

Let $G = (V, E)$ be a finite, undirected, simple graph (i.e., without self-loops and parallel edges). The complement $\overline{G} = (V, \overline{E})$ of G has the same set of vertices and edge set defined by $\overline{E} = \{(x, y) \mid x, y \in V \text{ and } x \neq y \text{ and } (x, y) \notin E\}$. Given a subset $X \subset V$, the subgraph induced by X is $G_X = (V(G_X), E(G_X))$, where $E(G_X) = \{(v, w) \in E \mid v \in X \text{ and } w \in X\}$ and $V(G_X) = X$.

A set $X \subseteq V$ is an *independent set* or a *stable set* in G if for all $u, v \in X$, $(u, v) \notin E$. A set $Y \subseteq V$ is a *clique* in G if for all $u, v \in Y$, $u \neq v$, $(u, v) \in E$. If $Y = V$, then G is a *complete* graph.

A sequence $[v_1, \ldots, v_k]$ of distinct vertices is a *path* in G if (v_1, v_2), $\ldots$, $(v_{k-1}, v_k) \in E$. These edges are called the edges of the path. The length of the path is the number $k - 1$ of its edges. A closed path $[v_1, \ldots, v_k, v_1]$ is called a *cycle* if in addition $(v_k, v_1) \in E$. A chord of a cycle $[v_1, \ldots, v_k, v_1]$ is an edge between two vertices of the cycle that is not an edge of the cycle. A cycle is *chordless* if it contains no chords. Trivially, a triangle has no chord, so we refer to a chordless cycle in this work as having length strictly greater than 3. We denote by C_k the chordless cycle on k vertices.

Definition 1. *An undirected graph G is* chordal (triangulated) *graph, if every cycle in G of length strictly greater than 3 possesses a chord. A graph $G = (V, E)$ is a* weakly chordal *graph if neither G nor its complement $\overline{G}$ have an induced subgraph C_k, $k \geq 5$ (see [2], [3]).*

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 249–260, 2003.

We often use the notation $P_{H[v,w]}$ to denote a chordless path $[v = v_1, \ldots, v_k = w]$ in the induced subgraph H, that is, $v_1, \ldots, v_k \in V(H)$ and $\{(v_1, v_2), \ldots, (v_{k-1}, v_k)\} = E(H_{\{v_1,\ldots,v_k\}})$. If $C = [v_1, \ldots, v_k, v_1]$ is a chordless cycle in G (the labels are assigned clockwise), then we denote by $P_{C[v,w]}$ a clockwise path of the cycle C starting with vertex $v \in C$ and ending with the vertex $w \in C$.

In this paper, we introduce the family of *chordal probe graphs*, a generalization of interval probe graphs, defined as follows.

Definition 2. *An undirected graph $G = (V, E)$ is a* chordal probe graph *if its vertex set can be partitioned into two subsets, P(probes) and N(non-probes), where N is a stable set and there exists a completion $E' \subseteq \{(u, v) | u, v \in N,\ u \neq v\}$ such that $G' = (V, E \cup E')$ is a chordal graph.*

Example 1. The graph C_4 is not chordal by definition. However, it is chordal probe, since there exists a partition $P = \{v_1, v_3\}$, $N = \{v_2, v_4\}$ that can be completed into a chordal graph by adding an edge (v_2, v_4).

In general, any bipartite graph is a chordal probe graph, by filling one of the stable sets of the bipartition into a clique.

Definition 3. *An undirected graph $G = (V, E)$ is an* interval graph *if its vertices can be put into one-to-one correspondence with a set of intervals $\Im = \{I_v\}_{v \in V}$ of a linearly ordered set (like the real line) such that two vertices are adjacent in G if and only if the corresponding intervals have a non-empty intersection that is, $(u, v) \in E \Leftrightarrow I_u \cap I_v \neq \emptyset$ (see[3]).*

Definition 4. *Three vertices of G form an* asteroidal triple *of G if for every pair of them there is a path connecting these two vertices that avoids the neighborhood of the remaining vertex.*

Theorem 1. *(Lekkerkerker and Boland) A graph G is an interval graph if and only if it is chordal and does not contain an asteroidal triple [3].*

Interval probe graphs, a generalization of *interval graphs*, were introduced by Zhang [11] and used in [12] to model certain problems in physical mapping of DNA when only partial data is available on the overlap of clones (i.e., the intervals)(see [6],[10]).

Definition 5. *An undirected graph $G = (V, E)$ is a* interval probe graph *if its vertex set can be partitioned into two subsets, P(probes) and N(non-probes), where N is a stable set and there exists a completion $E' \subseteq \{(u, v) | u, v \in N,\ u \neq v\}$ such that $G' = (V, E \cup E')$ is an interval graph.*

Remark 1. Interval probe graphs are chordal probe graphs. *Any interval completion is also a chordal graph completion. The converse is not true, as follows. The even length chordless cycles greater than 4 are chordal probe but not interval probe graphs, since interval probe graphs are weakly chordal (see [6],[10]).*

The odd chordless cycles of length > 4, however, are not chordal probe graphs as shown in Section 2.

The definitions of *interval probe* and *chordal probe graphs* do not specify a particular partition of the vertices in advance. However, in the biology applications, the partition into probes and non-probes is part of the input. Hence, we may distinguish between the general case of interval probe or chordal probe graphs, where we must find both a partition and a completion for it, and the special case of *partitioned interval probe* or *partitioned chordal probe graphs*, where we are given a fixed partition and must only find a completion for it. A polynomial time algorithm for the problem of recognizing *partitioned interval probe graphs* (i.e., with respect to a fixed partition) was first reported in [8]. Their method uses PQ-trees and constructs an interval probe model in $O(|V|^2)$ time. Another method given in [9], uses modular decomposition and has complexity $O(|V| + |E|log|V|)$. In contrast to this, however, the complexity of the general problem of recognizing *interval probe graphs* (when no partition is given) is an open problem. In this paper, we give $O(m^2)$ algorithms for recognizing whether a graph is a weakly chordal, chordal probe graph, first in the partitioned case (Section 3.1) and second in the general case (Section 3.3).

In [4], we presented the hierarchy of tolerance, interval probe and interval graphs, and the restricted cases of having an interval representation where (i) intervals have unit length or (ii) no interval properly contains another interval, see also [6]. In [4], we give the complete hierarchy for the classes of chordal probe, weakly chordal, interval probe and related families of graphs.

2 Structural Results

Lemma 1. *If G is a chordal probe graph with respect to a given partition (P, N), where N is a stable set, then probes and non-probes alternate in every chordless cycle in G.*

Proof. Suppose there is a chordless cycle $C = [x_1, \ldots, x_k, x_1]$ in G (where $k \geq 4$ and the labels are assigned according to clockwise order), such that probes and non-probes don't alternate in C. Since N is a stable set, there exists a pair $\{x_1, x_2\}$ of adjacent probes in C. The probes x_1 and x_2 remain with the same neighborhood in any chordal completion graph G' of G. However, combining the path $[x_k, x_1, x_2, x_3]$ and the chordless path from x_3 to x_k would give a chordless cycle in G'. This contradicts the chordality of G'. Hence, the given partition (P, N) does not have a chordal completion, contradicting the assumption. Hence, probes and non-probes alternate along the cycle C. □

Lemma 1 implies the following Theorem.

Theorem 2. *If G is a chordal probe graph, then G has no induced subgraph C_{2k+1}, for $k \geq 2$.*

Theorem 3. *If G is a chordal probe graph, then G has no induced subgraph $\overline{C_k}$, for $k \geq 5$.*

Proof. Let $G = (V, E)$ be a chordal probe graph. By Theorem 2, G has no induced $\overline{C_5}$, since $\overline{C_5}$ is isomorphic to C_5. Suppose there exists $k \geq 6$, such that $\overline{C_k} = [x_1, \ldots, x_k, x_1]$ is an induced subgraph of G (ordered according to clockwise direction). The cycle $C' = [x_2, x_4, x_1, x_5, x_2]$ is chordless in G. In any partition of $V(G)$ into probes and non-probes, which has chordal completion, either x_1 and x_2 are non-probes or x_4 and x_5 are non-probes. Without loss of generality, assume that x_1 and x_2 are non-probes. The vertex x_1 is adjacent to all the vertices in $\overline{C_k}$, except for x_2 and x_k. Hence, the vertices $x_3, \ldots, x_{k-1}$ are probes. But $C'' = [x_3, x_5, x_2, x_6, x_3]$ is also a chordless cycle of length 4 in G. In any partition (P, N) of $V(G)$, which has chordal completion, either x_2 and x_3 are both non-probes or both probes. Contradiction. □

The complete hierarchy of chordal probe graphs and other well-studied families of graphs are summarized in [5].

Definition 6. *A graph $G = (V, E)$ is an* even-chordal *graph if it has no induced even chordless cycle > 4. (This is called a (6,1)-even-chordal in [2]).*

Combining Theorem 2 and Theorem 3 we obtain the following:

Theorem 4. *If G is a chordal probe graph, then G is even-chordal if and only if G is weakly chordal.*

Definition 7. *A partition of a vertex set of a graph into two subsets P (probes) and N (non-probes) is* valid *if N is a stable set and probes and non-probes alternate in every chordless cycle in G.*

Definition 8. *Let $C = [v_1, v_2, v_3, v_4, v_1]$ be a chordless cycle in G, such that probes and non-probes alternate in C. Any chordal completion of G has an edge (v_2, v_4), which we call an* enhanced *edge and the probes $\{v_1, v_3\}$ are called the* creator pair *of the enhanced edge (v_2, v_4). The* enhanced graph *$G^* = (P \cup N, E^*)$ is the graph G together with all enhanced edges*[1].

Theorem 5. *Let $G = (V, E)$ be an even-chordal graph. If there exists a valid partition of $V(G)$, then G is a chordal probe graph and the enhanced graph G^* is a chordal completion.*

Proof. Let $G = (P, N, E)$ be an even-chordal graph with a given valid partition of $V(G)$ into P and N. By the definition of valid partition, there are no chordless cycles of odd length in G, hence all chordless cycles in G are 4-cycles.

We show by contradiction that G^* has no induced chordless cycle. Suppose C is a chordless cycle in G^*, with smallest possible number t of enhanced edges.

If C has no enhanced edge ($t = 0$), then C is a 4-cycle in G. Since probes and non-probes alternate in C, it must have enhanced edge. Contradiction.

[1] The notion of the enhanced graph was first introduced for interval probe graphs in [11], (see [10]). We apply it more generally.

Thus, $t \geq 1$ and there is no chordless cycle in G^* with less than t enhanced edges. Let $\{(a_1, b_1), \ldots, (a_t, b_t)\}$ be the enhanced edges in C. The two following claims needed to complete the proof are proved in [5].

Claim 1. Let $\{c, d\}$ be a creator pair of an enhanced edge (a, b) in C. At most one of c or d may have a neighbor $x \in C$, such that $x \neq a$, $x \neq b$.

Claim 2. No creator pair creates more than one enhanced edge in C. In other words there are no two equal creator pairs $\{c_i, d_i\} = \{c_j, d_j\}$, $i \neq j$, which create (a_i, b_i) and (a_j, b_j) respectively.

We now complete the proof of the theorem. By Claim 1, there exists a creator d_i for every enhanced edge (a_i, b_i), such that d_i is adjacent only to vertices a_i and b_i in C. All the elements of $X = \{d_1, \ldots, d_t\}$ are different because their neighborhoods are different.

If X is an independent set of G, then by replacing each enhanced edge (a_i, b_i) by the two edges $(a_i, d_i), (d_i, b_i) \in E(G)$, we would obtain a chordless cycle C' of G of length > 4. Contradiction!

Otherwise, (by renumbering if necessary) there exist probes d_1 and $d_i, i > 1$, such that $(d_1, d_i) \in E(G)$. If $a_1 = b_i$, then $b_1 \neq a_i$ due to Claim 2. If $a_1 = b_i$, then assign $C' = [(b_1, d_1), (d_1, d_i), (d_i, a_i), E(P_{C[a_i, b_1]})]$, else assign $C' = [(a_1, d_1), (d_1, d_i), (d_i, b_i), E(P_{C[b_i, a_1]})]$. The cycle C' is a chordless cycle in G^* with number of enhanced edges less than t. Contradiction!

Thus, we have shown that the enhanced graph G^* of G with respect to the given valid partition is a chordal graph. Therefore, it is a chordal completion and G is a chordal probe graph. □

Theorem 6. *Let $G = (P, N, E)$ be a chordal probe graph with respect to a given partition (P, N), where N is a stable set. If G is an even-chordal graph, then the enhanced graph G^* is chordal.*

Proof. According to Lemma 1, probes and non-probes alternate in every cycle greater than 3 in G. Since G is an even-chordal graph, then according to Theorem 5 the enhanced graph G^* is chordal. □

By Remark 1, every interval probe graph is a chordal probe graph with respect to the same partition and is also an even-chordal graph. Therefore, we obtain an alternate proof of the following:

Corollary 1 (Zhang [11]). *Let $G = (V, E)$ be an interval probe graph with respect to the partition of its vertex set into P and N, where N is a stable set. The enhanced graph G^* is chordal.*

3 Algorithmic Aspects

This section deals with recognition of even-chordal graphs which are chordal probe graphs. This class properly contains the class of interval probe graphs. We consider both the case with respect to a given partition (section 3.3) and the more challenging case without being given the partition (section 3.1).

Remark 2. A chordless cycle of length 4 (i.e., 4-cycle) has exactly two valid partitions where either pair of non-adjacent vertices could be the non-probes. Moreover, assigning any one of its four vertices to be a probe (respectively non-probe) forces its neighbors in the cycle to be non-probes (resp. probes) and its non-neighbor to be a probe (resp. non-probe).

Remark 3. The number of 4-cycles in a graph is at most $O(|E(G)|^2)$ and generating them can be done in $O(|E(G)|^2)$ time.

3.1 Recognition of Even-Chordal Graphs Which Are Chordal Probe Graphs with Respect to a Given Partition

According to Theorem 4, a chordal probe graph is weakly chordal if and only if it is even-chordal. The recognition of weakly chordal graphs can be done in $O(|E(G)|^2)$ ([1] or [7]). We use this method at the first stage of our algorithm. At the second stage, for each 4-cycle, we verify that probes and non-probes alternate on this cycle. The algorithm may fail at first stage, meaning that the graph is not weakly chordal. The algorithm may fail at the second stage, then the graph is weakly chordal, but is not chordal probe, since probes and non-probes do not alternate in every chordless cycle in G. The overall time complexity of the algorithm is $O(|E(G)|^2)$ by Remark 3.

3.2 The C_4-Connectivity Relation

Definition 9. *Let $G = (V, E)$ be a connected graph and let $S(G)$ denote the set of all 4-cycles in G. We define the sets $S_x(G) = \{C \in S(G) | x \in V(C)\}$ for each $x \in V(G)$. A path $[v_1, \ldots, v_i, \ldots, v_n]$ in G is a* C_4-path *if there exists $C_i \in S(G)$ such that $(v_i, v_{i+1}) \in E(C_i)$ for each $i = 1, \ldots, n-1$. A pair of vertices is* C_4-connected *if there exists a* C_4-path *that connects the vertices. A graph is* C_4-connected graph *if each pair of its vertices is* C_4-connected.

The C_4-connectivity *is an equivalence relation on V, so it partitions the set V into vertex disjoint maximal C_4-connected components, which we call* C_4-components. *A C_4-component which has only one vertex is called a* singleton C_4-component.

Let $H_1, \ldots, H_t$ be the C_4-components of G. The edges $\{(x, y) | x \in H_i$ and $y \in H_i,\ i = 1, \ldots, t\}$ are called the internal *edges of the graph. For each $x \in H_i$ we define the set $N_j(x) = \{y \in H_j | (x, y) \in E(G),\ j \neq i\}$ and each such edge (x, y) is called an* external *edge. If $N_j(x) = \{y\}$ and $N_i(y) = \{x\}$, then (x, y) is an* exclusive external edge *and the vertices x and y are called* exclusive endpoints. *See Figure 1.*

Lemma 2. *If $(P_1, N_1), (P_2, N_2)$ are two different partitions of a C_4-connected graph G, such that probes and non-probes alternate in every chordless cycle in G, then $P_1 = N_2$ and $N_1 = P_2$.*

Proof. Suppose there exists $v \in V(G)$, which is either assigned to be a probe in both partitions or a non-probe in both partitions.

We will now prove by induction on the length of a C_4-path $\overrightarrow{P} = [v = v_0, \ldots, v_i, \ldots, v_m]$ that the vertex v_m has the same assignment in both partitions.

In the case that $\overrightarrow{P} = [v_0, v_1]$, the edge (v_0, v_1) is an edge set of a 4-cycle. Therefore, v_1 has the same assignment in both partitions due to Remark 2.

Assume that for every path of length $i < m$, the vertex v_i has the same assignment in both partitions.

Let $\overrightarrow{P} = [v = v_0, \ldots, v_m]$ be a C_4-path of length m. By induction, each vertex v_i $(1 \leq i < m)$ must have the same assignment in both partitions. Consequently, the vertex v_m must have the same assignment in both partitions, since the length of the c_4-path $[v_i, \ldots, v_m]$ is less than m.

Therefore, all the vertices in G have the same assignment in both partitions and hence $P_1 = P_2$ and $N_1 = N_2$. Contradiction! Thus, the assignment of each vertex in (P_1, N_1) is different than its assignment in (P_2, N_2), so $P_1 = N_2$ and $P_2 = N_1$. □

Lemma 3. *Let G be a weakly chordal graph, and H_i be a C_4-component of G. Then the induced subgraph $G_{N_j(x)}$ is connected for any $x \in H_i$ $(i \neq j)$.*

Lemma 4. *There is at most one exclusive external edge that connects two C_4-components in G.*

Proof. Suppose (u, v) and (a, b) are exclusive external edges that connect H_i and H_j, where $u, a \in H_i$ and $v, b \in H_j$, by definition $u \neq a$ and $v \neq b$. Let x be the first vertex on $P_{H_i[a,u]}$, which has neighbors on $P_{H_j[b,v]}$, and let y be the first vertex on $P_{H_j[b,v]}$ which is adjacent to the vertex x ($y \neq b$ since (a, b) is an exclusive external edge). The cycle $C' = [a, P_{H_i[a,x]}, x, y, P_{H_j[y,b]}, b, a]$ is chordless of length > 3 and has vertices from different C_4-components. Contradiction. □

We use the *FindC4Comps* procedure to find all the C_4-components in a graph. The procedure is a variant of breadth first search, and in $O(|E(G)|^2)$ time it combines those 4-cycles that are not vertex disjoint into a C_4-component. Those vertices which are not in a vertex set of a 4-cycle are singleton C_4-components. See [5] for more details.

Example 2. $H_1, \ldots, H_8$ are the C_4-components in the graph shown in Figure 1, where H_1, H_2, H_3, H_6 are singleton C_4-components. The C_4-component H_5 has only one valid partition.

3.3 Recognition of Even-Chordal Graphs Which Are Chordal Probe Graphs without Being Given the Partition

In Stage 1 of our algorithm, we test whether G is weakly chordal, using the method in [1] or [7]. In Stage 2, for each vertex x, we find the set $S_x(G)$ of 4-cycles containing x, and in Stage 3, we construct the C_4-components $H_1, \ldots, H_t$

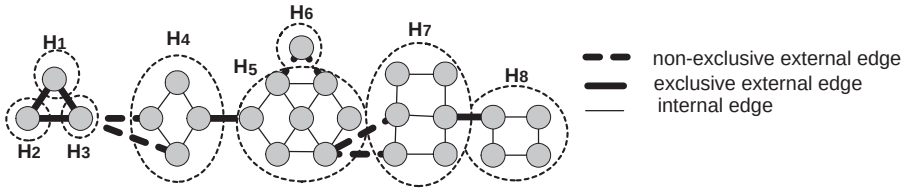


Fig. 1. Example

of G. Stage 4 finds all valid partitions of each H_i. If there exists a C_4-component H_i, which does not have a valid partition, then H_i is not a chordal probe graph by Lemma 1; hence G is not a chordal probe graph by the hereditary property.

At Stage 5, we extend the partitions from the C_4-components to the entire graph, and upon success we construct a graph $G_1 = (V_1, E_1)$, whose vertices correspond to the vertices of G and edges correspond to the internal edges and a certain subset of exclusive external edges of G. We will prove that if stages 1-5 succeed, then G_1 is a chordal probe graph. Finally, we will show that G_1 is a chordal probe graph if and only if G is a chordal probe graph.

Algorithm II. Recognition of even-chordal graphs which are chordal probe graphs without being given the partition

1. *verify that $G = (V, E)$ is weakly chordal using the algorithm in [1] or [7], otherwise return 'failure';*
2. *construct the set $S(G)$ of all 4-cycles in G and the sets $S_x(G)$ for each $x \in V(G)$;*
3. *find the set $H = H_1, \ldots, H_t$ of C_4-components by calling* FindC4Comps*;*
4. *for each $H_i \in H$ do*
   if H_i is a singleton C_4-component then $P_i = V(H_i)$, $N_i = \emptyset$, $l_i = 1$;
   else
   find a valid partition (P_i, N_i) and the number l_i of valid partitions of H_i by calling the FindPartitions *procedure;*
   if it fails, then return 'failure';
5. *build the graph $G_1 = (V_1, E_1)$ by calling the* PropagateConstrainedGraph *procedure. If it fails, then return 'failure', otherwise return 'success';*

We now give the details of the procedures that are used in the algorithm.

The *FindPartitions* procedure finds all valid partitions of a weakly chordal C_4-component H_i and assigns a label l_i, namely 1 or 2.

If H_i does not have a valid partition, then the *FindPartitions* procedure fails. The procedure first finds a partition of vertices, such that probes and non-probes alternate on every chordless cycle. It makes an arbitrary vertex v to be a probe and then propagates the assignment of all the other vertices accordingly.

Applying Lemma 2, there exist at most two such partitions of H_i, where in the case of two valid partitions, the set of probes in one is exactly the set of non-probes in the other. Then the *FindPartitions* procedure checks if non-probes are a stable set and if probes are a stable set.

Procedure: *FindPartitions*
Input: *A C_4-component H_i and the set $S_x(G)$ for each $x \in V(G)$.*
Output: *A valid partition (P_i, N_i) of H_i and the label l_i, or failure.*

step a: *$(P_i, N_i) \leftarrow (\{v\}, \emptyset)$, where v is an arbitrary vertex in H_i;*
$l_i \leftarrow 0$;
insert the vertex v into a queue Q;
while the queue Q is not empty do
remove vertex u from Q;
for each cycle $C_i \in S_u(G)$ do
if the non-neighbor x of u in C_i has a different assignment than u, then return 'failure';
if x is not yet assigned then
assign x to have the same assignment as u and insert x into Q;
if either neighbor y_1 or y_2 of u in C_i has the same assignment as u, then return 'failure';
if either y_i is not yet assigned then
assign y_i to have different assignment than u and insert y_i into Q;
step b: *if N_i is a stable set then*
if P_i is a stable set then $l_i \leftarrow 2$, else $l_i \leftarrow 1$;
else if P_i is a stable set, then swap the sets P_i and N_i and $l_i \leftarrow 1$;
else return 'failure';

Lemma 5. *Let H_i be a non-singleton C_4-component of a weakly chordal graph G. The* FindPartitions *procedure finds all the valid partitions of H_i.*

Proof. Each partition that is found by the *FindPartitions* procedure is valid, since probes and non-probes alternate in every 4-cycle (this is checked in step (a)) and non-probes are a stable set (this is checked in step (b)). According to Lemma 2, there exists at most two valid partitions, while the set of probes in one of the partitions is exactly the set of non-probes in the other partition. The *FindPartitions* procedure checks if both partitions are valid, by checking if the probes and the non-probes are stable sets. □

The *PropagateConstrainedGraph* procedure identifies the external edges of G and builds the graph G_1, enforcing the set of non-probes to be a stable set. In the process, a vertex can be forced to be a probe using the *ForceToBeAProbe* procedure, which assigns a valid partition to the C_4-component, such that the vertex is a probe and reduces its label to 1, or fails.

Procedure: *PropagateConstrainedGraph*
Input: *The set $H = H_1, \ldots, H_t$ of C_4-components in a graph $G = (V, E)$, and a valid partition $\{P_i, N_i\}$ of H_i together with the label l_i, for all i.*
Output: *Finds the graph $G_1 = (V_1, E_1)$, or 'failure'*

Step I: *for each H_i, mark the internal edges $E(H_i)$;*
Step II: */* marking of non-exclusive external edges */*
for each $v \in V(G)$, let $v \in H_i$ do
  for each unmarked edge $(u,v) \in E(G)$, $u \in N_j(v)$, $|N_j(v)| > 1$ do
    mark the edge (u,v);
    if there is at least one non-probe in $N_j(v)$ then call ForceToBeAProbe*(v);*
Step III: */* marking of some exclusive external edges */*
insert all C_4-components with label equal to 1 into the queue Q;
while Q is not empty do
  remove component H_i from Q;
  for each $v \in H_i$ do
    if there exists an unmarked edge $(u,v) \in E(G)$, $u \in N_j(v)$ then
      mark the edge (u,v);
      if u,v are both non-probes then
        call ForceToBeAProbe*(u);*
        insert H_j into Q;

let $G_1 = (V_1, E_1)$ be the graph with $(V_1 = V)$ and edges correspond to the remaining unmarked edges of G together with all internal edges $\cup E(H_i)$.

Lemma 6. *If the* PropagateConstrainedGraph *procedure succeeds, and e is an external edge of G, then $e \in E_1$ if and only if it is an exclusive external edge that connects two C_4-components both having two valid partitions remaining at the end of the procedure.*

Lemma 7. *Every chordless cycle in G_1 is a subgraph of a C_4-component of G.*

Definition 10. *A vertex is called* simplicial *if its adjacency set is a clique. Let $G = (V, E)$ be an undirected graph and let $\sigma = [v_1, \ldots, v_n]$ be an ordering of the vertices. We say that σ is a* perfect elimination ordering*(PEO) if each v_i is a simplicial vertex of the induced subgraph $G_{\{v_i,\ldots,v_n\}}$.*

Theorem 7. *(Fulkerson and Gross) A graph is chordal iff it has a PEO [3].*

Let $G_2 = (V_2, E_2)$ be the quotient graph of G_1, where each $h_i \in V_2$ corresponds to the C_4-component H_i in G and $E_2 = \{(h_i, h_j) | \exists (u,v) \in E_1$ such that $u \in H_i, v \in H_j\}$.

Remark 4. Obviously G_2 is a chordal graph, since every chordless cycle in G_1 is contained in a C_4-component due to Lemma 7. Therefore, there exists a PEO $\sigma = [h_1, \ldots, h_t]$ of V_2, which corresponds to an ordering $\nu = [H_1, \ldots, H_t]$ of the C_4-components in G.

Lemma 8. *Let G be a weakly chordal graph and $\nu = [H_1, \ldots, H_t]$ the ordering of the C_4-components in G, which corresponds to a PEO in G_2. There exists at most one exclusive endpoint of an edge in H_i that connects H_i with any of the C_4-components $[H_{i+1}, \ldots, H_t]$ in G_1.*

Proof. Suppose there exists exclusive external edges (x_1, y_1) and (x_2, y_2), $x_1 \neq x_2$, such that $x_1, x_2 \in H_i$, $y_1 \in H_j$, $y_2 \in H_k$ and $j, k > i$. Then $j \neq k$ according to Lemma 4. Since h_i is a simplicial vertex in $G_{2[h_{i+1},\ldots,h_t]}$, there exists an exclusive external edge (z_1, z_2), $z_1 \in H_j$, $z_2 \in H_k$ (possibly $y_1 = z_1$ or $y_2 = z_2$). The chordless cycle $C = [x_1, P_{H_i[x_1,x_2]}, x_2, y_2, P_{H_k[y_2,z_2]}, z_2, z_1, P_{H_j[z_1,y_1]}, y_1, x_1]$ in G has vertices in different C_4-components. Contradiction.□

Theorem 8. *The* PropagateConstrainedGraph *procedure succeeds if and only if G is a chordal probe graph.*

Proof. ($\Rightarrow$) If the *PropagateConstrainedGraph* procedure succeeds, then by Remark 4, let σ be PEO of G_2 that corresponds to an ordering ν of the C_4-components in G. If h_i is an isolated vertex in G_2, then (P_i, N_i) is a valid partition of H_i. Otherwise, let x be the exclusive endpoint in H_i that connects H_i to H_j, for all $j > i$ in G_1, as in Lemma 8. There exist two valid partitions of H_i due to Lemma 6. Let (P_i, N_i) be the valid partition of H_i, such that $x \in P_i$.

Consider the partition $(P = \cup(P_i), N = \cup(N_i))$ of $V(G_1)$. Probes and non-probes alternate in every cycle of length > 3 in G, since such a cycle is an induced subgraph in a C_4-component by Lemma 7. Suppose there exists an edge $(u, v) \in E_1$, such that $u \in N_i$, $v \in N_j$ and $j > i$. Then u is the exclusive endpoint of H_i and hence $u \in P_j$, a contradiction. Thus N is a stable set and (P, N) is a valid partition of G_1. Therefore, G_1 is a chordal probe graph by Theorem 5.

Let $(P = \cup(P_i), N = \cup(N_i))$ be a valid partition of G_1, where (P_i, N_i) is a valid partition of H_i. We will prove that (P, N) is a valid partition of G and hence G is a chordal probe graph due to Theorem 5.

Since G is a weakly chordal graph, each chordless cycle of length > 3 in G is an induced subgraph of a C_4-component and hence probes and non-probes alternate in the cycle. Thus, we only need to prove that N is a stable set. Suppose for a contradiction that there exists an edge $(u, v) \in E(G)$, such that $u, v \in N$. Moreover, $u \in H_i$ and $v \in H_j$, since N_i is a stable set for all i. In case that $|N_j(v)| > 1$, the procedure either fails or forces the vertex u to be a probe. Therefore $|N_j(v)| = 1$ and similarly $|N_i(v)| = 1$. Thus, (u, v) is an exclusive external edge. Neither H_i nor H_j were inserted into Q, since otherwise the procedure would remove H_i (or H_j) from Q and force v (or u) to be a probe. Thus (u, v) is unmarked by the procedure, meaning that $(u, v) \in E_1$ and N is not a stable set in G_1. This is a contradiction, since G_1 is a chordal probe graph with respect to (P, N).

($\Leftarrow$) We prove that if the procedure fails, then G is not a chordal probe graph. In the case that the *PropagateConstrainedGraph* procedure fails at Step II, there exists an edge $(u, v) \in E(G)$, $v \in H_i$, $u \in N_j(v)$, $|N_j(v)| > 1$, such that there is at least one non-probe in $N_j(v)$ in a given partition (P_j, N_j), v is a non-probe in a given partition (P_i, N_i) and $l_i = 1$. Since $N_j(v)$ is a connected set by Lemma 3 and N_j is a stable set, u has a probe neighbor in $N_j(v)$ in the given partition (P_j, N_j). Thus, there is also a non-probe in $N_j(v)$ in the opposite partition of H_j. Therefore, there exists a non-probe in $N_j(v)$ in any valid partition of H_j. Suppose G is a chordal probe graph. Now, (P_i, N_i) is the only valid partition of

H_i, since v must be a non-probe in any valid partition of G. Thus there would be a pair of adjacent non-probes in any valid partition of G, a contradiction.

In the case that the *PropagateConstrainedGraph* procedure fails at Step III, there exists an edge $(u, v) \in E(G)$, $u \in N_j(v)$, $|N_j(v)| = 1$, where u,v are both non-probes in the given partitions (P_i, N_i), (P_j, N_j) and $l_j = l_i = 1$. Suppose G is a chordal probe graph. Since both H_i and H_j have only one valid partition, u and v are both non-probes in any valid partition of G. Thus there would be a pair of adjacent non-probes in any valid partition of G, a contradiction. □

Remark 5. The time complexity of the Algorithm II is $O(|E(G)|^2)$.

Proof. Stage 1 of the algorithm has time complexity $O(|E(G)|^2)$, using the algorithm in [1] or [7]. By Remark 3, Stage 2 also has time complexity $O(|E(G)|^2)$. At Stage 3, we call *FindC4Comps*, which has complexity $O(|E(G)|^2)$. At Stage 4, *FindPartitions* is called for each C_4-component H_i of the graph, and the time complexity is $O(|E(H_i)|^2)$ (see [5]); hence Stage 4 has complexity $\sum_i O(|E(H_i)|^2) = O(|E(G)|^2)$. The time complexity of Stage 5 is $O(|E(G)|)$ (see [5] for details). □

References

1. A. Berry, J. Bordat, P. Heggernes, *Recognizing weakly triangulated graphs by edge separability*, Nordic Journal of Computing 7(3), Fall(2000), 164–177.
2. A. Brandstädt, V. Le, J.P. Spinrad, *Graph Classes: A Survey*, SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 1999.
3. M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York (1980).
4. M.C. Golumbic, M. Lipshteyn, *On the hierarchy of tolerance, probe and interval graphs*, Congressus Numerantium 153 (2001), 97–106.
5. M.C. Golumbic, M. Lipshteyn, *Chordal probe graphs*, manuscript (2003). URL: http://www.cri.haifa.ac.il/TechReport/
6. M.C. Golumbic, A.N. Trenk, *Tolerance Graphs*, Cambridge Univ. Press (2003).
7. R.B. Hayward, J. Spinrad, R. Sritharan, *Weakly chordal graph algorithms via handles*, Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (2000), 42–49.
8. J.L. Johnson, J. Spinrad, *A polynomial time recognition algorithm for probe interval graphs*, Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (2001), 477–486.
9. R.M. McConnell, J. Spinrad, *Construction of probe interval models*, Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (2002), 866–875.
10. F.R. McMorris, C. Wang, P. Zhang, *On probe interval graphs*, Discrete Applied Mathematics 88 (1998), 315–324.
11. P. Zhang, *Probe interval graph and its application to physical mapping of DNA*, manuscript 1994.
12. P. Zhang, E.A. Schon, S.G. Fischer, E. Cayanis, J. Weiss, S. Kistler, and P.E. Bourne, *An algorithm based on graph theory for the assembly of contigs in physical mapping of DNA.*, CABIOS 10 (1994), 309–317.

Subgraph Induced Planar Connectivity Augmentation
(Extended Abstract)

Carsten Gutwenger[2], Michael Jünger[1], Sebastian Leipert[2], Petra Mutzel[3], Merijam Percan[1], and René Weiskircher[3]

[1] Universität zu Köln, Institut für Informatik,
Pohligstraße 1, 50969 Köln, Germany
{mjuenger,percan}@informatik.uni-koeln.de
Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).
[2] caesar research center,
Ludwig-Erhard-Allee 2, 53175 Bonn, Germany
{gutwenger,leipert}@caesar.de
[3] Technische Universität Wien E186, Favoritenstraße 9–11, 1040 Wien, Austria,
{mutzel,weiskircher}@ads.tuwien.ac.at

Abstract. Given a planar graph $G = (V, E)$ and a vertex set $W \subseteq V$, the subgraph induced planar connectivity augmentation problem asks for a minimum cardinality set F of additional edges with end vertices in W such that $G' = (V, E \cup F)$ is planar and the subgraph of G' induced by W is connected. The problem arises in automatic graph drawing in the context of c-planarity testing of clustered graphs. We describe a linear time algorithm based on SPQR-trees that tests if a subgraph induced planar connectivity augmentation exists and, if so, constructs a minimum cardinality augmenting edge set.

1 Introduction

For an undirected graph $G = (V, E)$, a subset of vertices W of V, and E_W the subset of E that contains only edges with end vertices in W let $G_W = (W, E_W)$ be the subgraph of G induced by W. If G is planar, a *subgraph induced planar connectivity augmentation* for W is a set F of additional edges with end vertices in W such that the graph $G' = (V, E \cup F)$ is planar and the graph G'_W is connected.

We present a linear time algorithm based on the SPQR data structure that tests if a subgraph induced planar connectivity augmentation exists and, if so, constructs a minimum cardinality augmenting edge set. The difficulty of the subgraph induced planar connectivity augmentation problem arises from the fact that the computation of an appropriate planar embedding is part of the problem. Once the embedding is fixed, the decision becomes trivial.

The subgraph induced planar connectivity augmentation problem arises for example in the context of c-planarity testing of clustered graphs. A *clustered*

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 261–272, 2003.

graph is an undirected graph together with a nested family of vertex subsets whose members are called *clusters*. The concept of c-planarity is a natural extension of graph planarity for clustered graphs and plays an important role in automatic graph drawing. While the complexity status of the general problem is unknown, c-planarity can be tested in linear time if the graph is c-connected, i.e., all cluster induced subgraphs are connected [3,2]. In approaching the general case, it appears natural to augment the clustered graph by additional edges in order to achieve c-connectivity without losing c-planarity.

The results presented in this paper are the basis for a first step towards this goal. Namely, the algorithm presented here leads to a linear time algorithm that tests "almost" c-connected cluster graphs, i.e., cluster graphs in which at most one cluster is disconnected, for c-planarity [6,8].

In general, connectivity augmentation problems consist of adding a set of edges in order to satisfy certain connectivity constraints such as k-connectivity or k-edge-connectivity. The first results in this area are due to Lovász [12]. Since then, augmentation results for many different connectivity properties have been proved. For more information on connectivity augmentation problems see the surveys, e.g., by Frank [5] and Khuller [11].

Planar connectivity augmentation problems have been introduced by Kant and Bodlaender [10]. They have shown that the problem of augmenting a planar graph to a planar biconnected graph with the minimum number of edges is NP-hard. Moreover, they have given a 2-approximation algorithm for the planar biconnectivity augmentation problem. Fialko and Mutzel [4] have given a $\frac{5}{3}$-approximation algorithm. Polyhedral investigations of this problem have been conducted, e.g., in [13] and [14].

To our knowledge, this is the first time that subgraph induced planar connectivity augmentation is investigated. It is a generalization of planar connectivity augmentation. The subgraph induced connectivity augmentation problem is a special case of a certain connectivity augmentation problem considered, e.g., by Stoer [15] in the context of network survivability.

This paper is organized as follows. Section 3 describes the easy case in which the embedding of the given graph is fixed for example, when the graph is triconnected. The most interesting case is the treatment of biconnected graphs in Section 4 with the help of the SPQR-tree data structure and the results of the previous section. Finally, Section 5 is concerned with the non-biconnected case in which we construct the block-cut tree of G and use the algorithm for biconnected graphs for each block along with a planarity testing step. All sections are illustrated by an application to an example graph (see Figure 2 - Figure 5). The proofs omitted in this extended abstract and the information about the implementation will be given in the full version [7].

2 Preliminaries

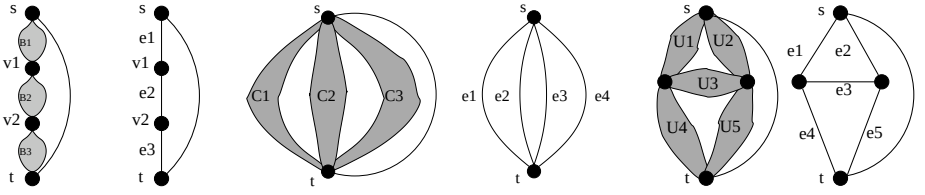
SPQR-trees have been introduced by Di Battista and Tamassia [1]. They represent a decomposition of a planar biconnected graph according to its split pairs

(pairs of vertices whose removal splits the graph or vertices connected by an edge). The construction of the SPQR-tree works recursively (see Figure 2). At every node v of the tree, we split the graph into the split components of the split pair associated with that node. The first split pair of the decomposition is an edge of the graph and is called the *reference edge* of the SPQR-tree. We add an edge to each of the split components to make sure that they are biconnected and continue by computing their SPQR-tree and making the resulting trees the subtrees of the node used for the splitting. Every node of the SPQR-tree has two associated graphs:

- The *skeleton* of the node associated with a split pair p is a simplified version of the whole graph where some split-components are replaced by single edges.
- The *pertinent graph* of a node v is the subgraph of the original graph that is represented by the subtree rooted at v.

The two vertices of the split pair that are associated with a node v are called the *poles* of v. There are four different node types in an SPQR-tree (S-,P-,Q- and R-nodes) that differ in the number and structure of the split components of the split pair associated with the node (see Figure 1). The Q-nodes form the leaves of the tree, and there is one Q-node for each edge in the graph. The skeleton of a Q-node consists of the poles connected by two edges. The skeletons of S-nodes are cycles, while the skeletons of R-nodes are triconnected graphs. P-node skeletons consist of the poles connected by at least three edges. Figure 1 shows examples for skeletons of S-, P- and R-nodes.

Skeletons of adjacent nodes in the SPQR-tree share a pair of vertices. In each of the two skeletons, one edge connecting the two vertices is associated with a corresponding edge in the other skeleton. These two edges are called *twin edges*. The edge in a skeleton that has a twin edge in the parent node is called the *virtual edge* of the skeleton.



(a) S-node (b) P-node (c) R-node

Fig. 1. Decomposition of biconnected graphs and the skeletons of the corresponding nodes in the SPQR-tree. Here the three cases are illustrated.

Each edge e in a skeleton represents a subgraph of the original graph. This graph together with e is the *expansion graph* of e.

All leaves of the SPQR-tree are Q-nodes and all inner nodes S-, P- or R-nodes. When we regard the SPQR-tree as an unrooted tree, then it is unique for every biconnected planar graph. Another important property of these trees is that their size (including the skeletons) is linear in the size of the original graph and that they can be constructed in linear time [1,9]. As described in [1, 9], SPQR-trees can be used to represent the set of all combinatorial embeddings of a biconnected planar graph. Every combinatorial embedding of the original graph defines a unique combinatorial embedding for a skeleton of each node in the SPQR-tree. Conversely, when we define an embedding for the skeleton of each node in the SPQR-tree, we define a unique embedding for the original graph. The skeletons of S- and Q-nodes are simple cycles, so they have only one embedding. But the skeletons of R-and P-nodes have at least two different embeddings. Therefore, the embeddings of the R- and P-nodes determine the embedding of the graph and we call these nodes the *decision nodes* of the SPQR-tree.

3 An Easy Case: Fixed Embedding

We consider the case that the planar graph G is given together with a fixed embedding. In the following, we call the vertices belonging to W *blue vertices*. Our task is to insert edges so that the induced subgraph G_W is connected and G is still planar after edge insertion.

Our algorithm looks at each face f in G that has at least two non-adjacent blue vertices on its boundary. We start at an arbitrary blue vertex v on the boundary of f and introduce a new edge through f that connects it to the next blue vertex on the boundary. Thus we step through the blue vertices on the boundary of f, connecting each to its successor on the boundary until we come back to v. We call the resulting graph G'. Note that we only introduce a linear number of edges in this step and that G' is planar. Another important property of G' is that G'_W is connected if and only if there is a planar augmentation for W in G.

Then we compute the graph G'' by deleting all vertices from G' that are not blue. We assign value 1 to all edges introduced in the first step and 0 to all other edges. We can find a minimum spanning tree in G'' in linear time because there are only two different weights on the edges. One way to do this is to use Prim's Algorithm where we use two lists instead of the priority queue. The algorithm may now determine that G'' is not connected. Then we know that there is no planar augmentation for G. Otherwise, the edges of weight 1 in the minimum spanning tree are our solution. Thus, we can solve the problem in linear time.

4 The Algorithm for the Biconnected Case

In this section we present an algorithm for the case that the given graph G is biconnected. First, we compute the SPQR-tree $\mathcal{T}$ of G. In Section 4.1, we

present a recursive algorithm for coloring all edges in all skeletons of the SPQR-tree. This coloring stores information about the position of the blue vertices in each skeleton of the SPQR-tree by assigning three different colors to the edges. The coloring enables us to test if an augmentation is possible by examining the colors in each skeleton. If an augmentation is possible, we compute an embedding of the graph that allows an augmentation (see Section 4.2). Then we can apply the algorithm of the previous section to this fixed embedding in order to compute the list of edges needed to solve the augmentation problem. Algorithm 1 gives an overview of the algorithm for biconnected graphs.

Algorithm 1: The algorithm `BiconnectedAugmenter` computes a planar connectivity augmentation for a planar biconnected graph G and a subset W of the vertices, if it exists.

Input: A biconnected planar graph G and a subset W of its vertices,
Result : `true` if and only if there is a planar augmentation for W; in the positive case an embedding Π and a minimum cardinality augmenting edge set will be computed.
Calculate the SPQR-tree $\mathcal{T}$ of G;
Make an arbitrary node r which is not a Q-node the root of $\mathcal{T}$;
`MarkEdgesPhase1`(r, W);
`MarkEdgesPhase2`(r, W);
`BiconnectivityFeasibilityCheck`$(\mathcal{T})$;
Embedding $\Pi =$ `CalculateEmbedding`;
return `FixedEmbeddingAugmenter`(Π, W);

4.1 The Coloring Algorithm

Again we call a vertex blue, if it is contained in W and black otherwise. Figure 2 shows an example graph, in which the set W is given by $\{3, 8, 10\}$ (shown by bold circles). The skeletons are shown in Figure 3, which displays essentially the SPQR-tree without the Q-nodes. The blue vertices in the tree are marked by circles.

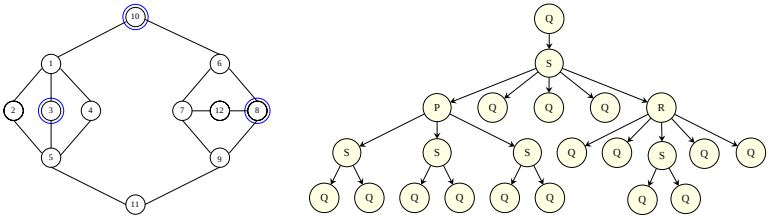


Fig. 2. Example: A graph G and its SPQR-tree.

We assign one of two colors to each edge in each skeleton: blue or black. We call an edge in a skeleton blue, if its expansion graph contains blue vertices and black otherwise. For example, in Figure 5 in the skeleton of the S-node of the SPQR-tree, the edge between the vertices 1 and 5 is blue because the blue vertex 3 is contained in its expansion graph. It is represented by a dashed line.

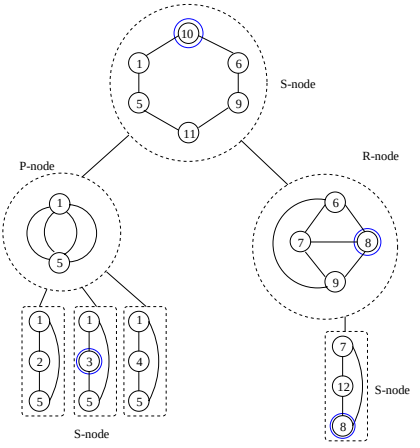


Fig. 3. Continuation of Figure 2: Q-nodes are omitted; assigning blue to the vertices of the subset W.

Additionally, we assign the attribute *permeable* to some blue edges. Intuitively, an edge is permeable if we can construct a path connecting only blue vertices through its expansion graph. Therefore, in Figure 5 the edge between the vertices 1 and 5 in the skeleton of the root node does not get the attribute permeable. But we assign for example the attribute permeable to the edge between the vertices 1 and 5 in the skeleton of the P-node whose expansion graph contains the vertex 10. In Figure 5, the permeable edges are represented by a dotted line. Let $G(e)$ be the expansion graph of edge e in skeleton $\mathcal{S}$. In any planar embedding $G(e)$, there are exactly two faces that have e on their boundary. This follows from the fact that in a planar biconnected graph, every edge is on the boundary of exactly two faces in every embedding. We call the edge e in $\mathcal{S}$ permeable with respect to W, if there is an embedding Π of $G(e)$ and a list of at least two faces $L = (f_1, \ldots, f_k)$ in Π that satisfies the following properties:

1. The two faces f_1 and f_k are the two faces with e on their boundary.
2. For any two faces f_i, f_{i+1} with $1 \leq i < k$, there is a blue vertex on the boundary between f_i and f_{i+1}.

We call a skeleton $\mathcal{S}$ of a node v of $\mathcal{T}$ permeable if the pertinent graph of v together with the virtual edge of $\mathcal{S}$ have the two properties stated above. So $\mathcal{S}$

is permeable if the twin edge of its virtual edge is permeable. For example, in Figure 5 the skeleton that contains the vertex 3 is permeable.

We develop an algorithm that marks each edge in every skeleton of the SPQR-tree $\mathcal{T}$ of G with the colors black or blue and assigns the attribute permeable depending on the expansion graph of the edge. The algorithm works recursively. We assume that $\mathcal{T}$ is rooted at node r and r is not a Q-node.

First we mark all the edges of the skeletons of the children of r recursively black or blue and assign the permeable attribute by treating them as the roots of subtrees. Each edge in the skeleton $\mathcal{S}$ of r except the reference edge corresponds to a child of r. Let e be such an edge in $\mathcal{S}$, v the corresponding child of r and $\mathcal{S}'$ the skeleton of v. If $\mathcal{S}'$ contains a blue edge or vertex, we mark e blue and otherwise black. The permeability of e depends on the type of v:

Q-node: We mark e permeable if the skeleton $\mathcal{S}'$ contains a blue vertex.

S-node: If the skeleton $\mathcal{S}'$ contains a blue vertex or a permeable edge, we mark e permeable.

P-node: If the skeleton $\mathcal{S}'$ contains only permeable edges or a blue vertex, we mark e permeable.

R-node: We consider a graph H where the vertices are the faces of $\mathcal{S}'$ and there is an edge between two vertices if there is a permeable edge or a blue vertex on the boundary that separates the two faces. Let s and t be the two faces left and right of the virtual edge of $\mathcal{S}'$. If there is a path in H connecting s and t, we mark e permeable (see Figure 4).

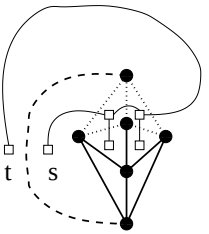


Fig. 4. A permeable R-node with the graph H: permeable edges are represented by dotted lines, the virtual edge of the skeleton by a dashed line.

After executing this algorithm, which we call `MarkEdgesPhase1`, all edges of the skeleton of the root node r are marked, because we have seen all the blue vertices of the graph. All other skeletons except the skeleton of r contain one edge that is not yet marked: the virtual edge of the skeleton.

The algorithm `MarkEdgesPhase2` works top down by traversing $\mathcal{T}$ from the root to the leaves. The edges of the skeleton of the root r of $\mathcal{T}$ are already marked in the first step, therefore we can proceed to the children and mark the virtual edges of its children. Let v be a node in $\mathcal{T}$ where the skeleton $\mathcal{S}'$

of the parent node is already completely marked. We mark the virtual edge e in the skeleton $\mathcal{S}$ of v blue if there is a blue edge or vertex in the skeleton of the parent. The permeability of e again depends on the type of the skeleton in exactly the same way as in the algorithm `MarkEdgesPhase1` (see Figure 5). Note that the case Q-node is irrelevant here because the Q-nodes form the leaves of the tree. In Figure 5 we see a result of the two algorithms `MarkEdgesPhase1` and `MarkEdgesPhase2` for our example graph. Permeable edges are represented by dotted lines, blue edges that are not permeable by dashed lines and blue vertices by a circle around them.

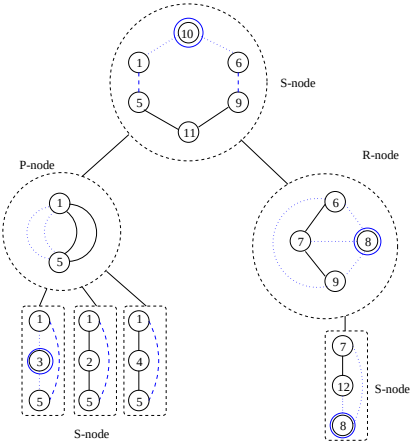


Fig. 5. Continuation of Figure 3: After calling the algorithms `MarkEdgesPhase1` and `MarkEdgesPhase2`.

The two algorithms `MarkEdgesPhase1` and `MarkEdgesPhase2` can both be implemented in linear time because the size of the SPQR-tree of a planar biconnected graph including all skeletons is linear in the size of the graph [1].

Lemma 1. *Let e be an edge in a skeleton of an inner node and $G(e)$ its expansion graph. Then the coloring algorithm marks e blue if and only if $G(e)$ contains a vertex of W. Furthermore, e is marked permeable if an only if there is an embedding Π of $G(e)$ together with a sequence of faces $f_1, \ldots, f_k$ with the following property:*

(*) The two faces f_1 and f_k are the two faces with e on their boundary and for any two faces f_i, f_{i+1} with $1 \leq i < k$, there is a blue vertex on the boundary between f_i and f_{i+1}.

4.2 The Embedding Algorithm

Let $\mathcal{S}$ be a skeleton of a P-node. We call the embedding of $\mathcal{S}$ *admissible* if all blue edges are consecutive and the blue edges that are not permeable (if they

exist) are at the beginning and at the end of the sequence (see Figure 6 for three examples of admissible orderings). For example, in Figure 5 the skeleton of the P-node has already an admissible embedding.

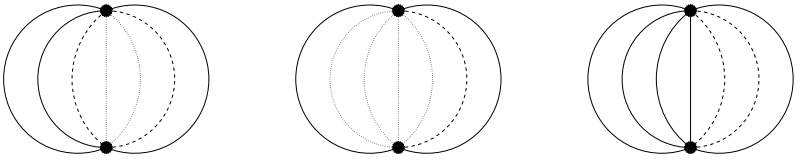


Fig. 6. Admissible embeddings of a P-node skeleton (permeable edges are dotted, blue edges that are not permeable are dashed).

Our algorithm for finding an augmentation or proving that no augmentation exists works in two phases:

1. Using the colors and attributes of the edges in each skeleton, we fix an embedding for every P- and R-node skeleton and thus determine an embedding for G.
2. We use the algorithm of Section 3 for fixed embeddings to determine whether an augmentation is possible.

The embedding computed in the first step has the property that it allows an augmentation if and only if there is an embedding of G that allows an augmentation.

We set the embedding for the skeletons of the R- and P-nodes recursively using the structure of the SPQR-tree. We assume that the vertices in G are numbered and that all edges are directed from the vertex with lower number to the vertex with higher number.

For simplicity, we consider whether a special case is present where a planar connectivity augmentation cannot exist.

Theorem 1. *Let G be a biconnected series-parallel planar graph and W a subset of its vertices. There exists a planar connectivity augmentation for W in G if and only if all P-nodes of the SPQR-tree of G contain at the most two edges that are blue but not permeable.*

The conclusion of the theorem is that if there exists a P-node that contains a skeleton with more than two blue edges in a biconnected graph G, there cannot exist a planar connectivity augmentation. For example, in Figure 5 there are no edges that are blue but not permeable in the skeleton of the P-node.

First, we test whether the biconnected graph contains only P-nodes with skeletons that have at the most two edges. Then we can sort the two blue edges as mentioned in Figure 6 to obtain an admissible embedding. Note that the two blue but not permeable edges are not consecutive in an admissible embedding if

there are additionally black and permeable edges. Therefore, there is nothing to do for the skeleton of the P-node in Figure 5.

Next, we construct an algorithm that marks edges in the skeletons with a new attribute that can have three different values: `left`, `right` and `nil`. If the virtual edge (the edge whose twin edge is in the parent of v) in a skeleton of node v is marked `left`, then the pertinent graph of v must be embedded in such a way that there is a blue vertex on the boundary of the face left of the virtual edge. If the edge is marked `right`, a blue vertex must be on the boundary of the face right of the virtual edge. If the virtual edge is marked `nil`, there is no restriction on the embedding of the pertinent graph of v.

For each node v in the SPQR-tree, where the embedding of the parent node has already been fixed, we perform two steps:

1. We determine an embedding using the attribute of the virtual edge and the colors and attributes of the other edges.
2. We determine the attribute for the virtual edge in the skeleton of each child of v.

Only the skeletons of R- and P-nodes have more than one embedding, so the first step is only important for these node types. First we consider the case where v is an R-node. In this case, its skeleton has two embeddings. If the attribute on the virtual edge is `nil`, we can choose any of the two embeddings. Otherwise, we choose an embedding where there is a blue edge or vertex on the face left (right) of the virtual edge if the attribute was `left` (`right`). If none of the two embeddings has this property, no augmentation is possible. If v is a P-node, we can only choose among the admissible embeddings of the skeleton. Again, we choose an embedding according to the attribute and if no suitable embedding exists, there can be no augmentation.

Now we have to determine the attribute for the virtual edge of each child of v. Let $\mathcal{S}$ be the skeleton of v. For all edges in $\mathcal{S}$ that are either black or permeable, we pass `nil` to the corresponding child. Let L be the set of edges in $\mathcal{S}$ that are blue but not permeable.

First we consider the case that v is an S-node. If the attribute of the virtual edge is `nil`, we pass `nil` to every child that corresponds to an edge in L. Otherwise, the attribute on the virtual edge designates one of the two faces of the S-node skeleton as the one that must have a blue vertex on the boundary. For each edge e in L, we pass `left` to the corresponding child if this face is right of e and `right` otherwise.

To make the following descriptions more concise, we call a face *permeable*, if it has a permeable edge or a blue vertex on its boundary. If v is a P-node, L contains at most two edges. For each of edge e in L, there are three possible cases:

1. Exactly one of the faces with e on its boundary contains a blue edge. If this is the face on the right of e, we pass `left` to the child corresponding to e and `right` otherwise.

2. One of the faces with e on its boundary is permeable and the other is not. If the permeable face is left of e, we pass `right` to the child corresponding to e and `left` otherwise.
3. The faces left and right of e are both permeable or both contain only black edges and vertices. In this case, we pass `nil` to the child.

If v is an R-node, we also have to consider the faces left and right of each edge e in L. The same cases as for P-nodes apply, but there is one additional case that cannot occur in a P-node: Both faces left and right of e are not permeable but both contain a blue edge except e (if this happens in a P-node, there can be no augmentation). In this case, we pass `nil` to the corresponding child. For example, in Figure 5 the attributes of all virtual edges are `nil`.

To start the process, we choose an arbitrary P- or R-node as the root of the SPQR-tree. If we choose an arbitrarily R-node, we select one of the two embeddings of the skeleton. If we select a P-node, we choose an arbitrary admissible embedding. Now we can compute the attributes we pass to the children of the node as stated above and compute an embedding for each skeleton of the SPQR-tree by applying the algorithm in depth first or breadth first sequence to all inner nodes of the tree.

This algorithm defines an embedding for each R- and P-node in the SPQR-tree and thus for the graph G. Since we touch each skeleton only once and the operations we perform for each skeleton can be done in time linear in the size of the skeleton, the embedding can be computed in linear time. Then we apply the algorithm from Section 3 to the fixed embedding. This algorithm either computes the list of edges that constitutes the planar connectivity augmentation or it signals that no augmentation is possible for this embedding.

Theorem 2. *Let Π be the embedding of G determined by the algorithm described above. G has a planar connectivity augmentation with respect to W if and only if there exists a planar connectivity augmentation for G with respect to the embedding Π.*

The proof uses structural induction over the SPQR-tree. We first show that the claim holds for graphs whose SPQR-tree has only one inner node and then use induction to show that it holds for graphs whose SPQR-tree has more than one inner node. Because of space considerations we show the proof in [7].

Theorem 3. *Given a biconnected planar graph $G = (V, E)$ and a subset of vertices $W \subseteq V$. The algorithm* `BiconnectedAugmenter` *tests correctly whether a subgraph induced planar connectivity augmentation exists and, if so, constructs a minimum cardinality augmenting edge set. It runs in time $O(|V|)$.*

5 The Algorithm for the Connected Case

Using BC-trees, this algorithm can be extended for connected graphs. Because of space considerations we omit a detailed description that is available in [7]. Using this result for non-connected planar graphs, the problem becomes easy.

References

1. Di Battista, G., Tamassia, R. (1996) On-line planarity testing. Journal on Computing **25** (5), 956–997
2. Cohen, R.-F., Eades, P., Feng, Q.-W. (1995) Planarity for clustered graphs. In P. Spirakis (ed.) Algorithms – ESA '95, Third Annual European Symposium, Lecture Notes in Computer Science 979, Springer-Verlag, 213–226
3. Dahlhaus, E. (1998) Linear time algorithm to recognize clustered planar graphs and its parallelization (extended abstract). In C. L. Lucchesi (ed.) LATIN '98, 3rd Latin American symposium on theoretical informatics, Campinas, Brazil, Lecture Notes in Computer Science 1380, 239–248
4. Fialko, S., Mutzel, P. (1998) A new approximation algorithm for the planar augmentation problem. In Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '98), San Francisco, California, ACM Press, 260–269
5. Frank, A. (1992) Augmenting graphs to meet edge-connectivity requirements. SIAM J. Discrete Mathematics **5** (1), 25–53
6. Gutwenger, C., Jünger, M., Leipert, S., Mutzel, P., Percan, M., Weiskircher, R. (2002) Advances in c-planarity testing of clustered graphs. Technical report, Institut für Informatik, Universität zu Köln, zaik2002–436.
7. Gutwenger, C., Jünger, M., Leipert, S., Mutzel, P., Percan, M., Weiskircher, R. (2002) Subgraph induced planar connectivity augmentation. Technical report, Institut für Informatik, Universität zu Köln, zaik2002–435.
8. Gutwenger, C., Jünger, M., Leipert, S., Mutzel, P., Percan, M., Weiskircher, R. (2003) Advances in c-planarity testing of clustered graphs. In M. T. Goodrich and S. G. Kobourov (eds.) Graph Drawing (Proc. 2002), Lecture Notes in Computer Science 2528, Springer-Verlag, 220–235
9. Gutwenger, C., Mutzel, P. (2001) A linear time implementation of SPQR-trees. In J. Marks (ed.) Graph Drawing (Proc. 2000), Lecture Notes in Computer Science 1984, Springer-Verlag, 77–90
10. Kant, G., Bodlaender, H. L. (1991) Planar graph augmentation problems. In Proc. 2nd Workshop Algorithms Data Struct., Lecture Notes in Computer Science 519, Springer-Verlag, 286–298
11. Khuller, S. (1997) Approximation Algorithms for Finding Highly Connected Subgraphs. In Hochbaum, D. S. (ed.), PWS Publishing Company, Boston, 236–265
12. Lovász, L. (1979) Combinatorial Problems and Exercises. North-Holland.
13. Mutzel, P. (1995) A polyhedral approach to planar augmentation and related problems. In P. Spirakis (ed.) Algorithms – ESA '95, Third Annual European Symposium, Lecture Notes in Computer Science 979, Springer-Verlag, 494–507
14. De Simone, C., Jünger, M. (1997) On the two-connected planar spanning subgraph polytope. Discrete Applied Mathematics **80** (229), 223–229
15. Stoer, M. (1992) Design of Survivable Networks, Lecture Notes in Mathematics 1531. Springer-Verlag, Berlin

On the Recognition of General Partition Graphs

Ton Kloks[1], Chuan-Min Lee[2⋆], Jiping Liu[1⋆⋆], and Haiko Müller[3]

[1] Department of Mathematics and Computer Science
The University of Lethbridge
Alberta, T1K 3M4, Canada
{kloks,liu}@cs.uleth.ca
[2] Department of Computer Science and Information Engineering
Chung Cheng University
Chiayi, Taiwan
cmlee@cs.ccu.edu.tw
[3] School of Computing
University of Leeds
Leeds LS2 9JT, United Kingdom
hm@comp.leeds.ac.uk

Abstract. A graph G is a general partition graph if there is some set S and an assignment of non-empty subsets $S_x \subseteq S$ to the vertices of G such that two vertices x and y are adjacent if and only if $S_x \cap S_y \neq \varnothing$ and for every maximal independent set M the set $\{S_m \mid m \in M\}$ is a partition of S. For every minor closed family of graphs there exists a polynomial time algorithm that checks if an element of the family is a general partition graph.
The triangle condition says that for every maximal independent set M and for every edge (x, y) with $x, y \notin M$ there is a vertex $m \in M$ such that $\{x, y, m\}$ induces a triangle in G. It is known that the triangle condition is necessary for a graph to be a general partition graph (but in general not sufficient). We show that for AT-free graphs this condition is also sufficient and this leads to an efficient algorithm that demonstrates whether or not an AT-free graph is a general partition graph.
We show that the triangle condition can be checked in polynomial time for planar graphs and circle graphs. It is unknown if the triangle condition is also a sufficient condition for planar graphs to be a general partition graph. For circle graphs we show that the triangle condition is *not* sufficient.

Introduction and Preliminaries

Definition 1. *A graph is a* general partition graph *if there exists a set S and an assignment of a non-empty subset $S_x \subseteq S$ to each of its vertices x such that two vertices x and y are adjacent if and only if $S_x \cap S_y \neq \varnothing$ and such that for every maximal independent set M, the collection $\{S_m \mid m \in M\}$ partitions S.*

⋆ Corresponding author
⋆⋆ Partially supported by NSERC of Canada

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 273–283, 2003.

For a graph $G = (V, E)$ we write $N(x)$ for the set of neighbors of a vertex $x \in V$ and $N[x] = N(x) \cup \{x\}$ for the *closed* neighborhood of x.
The following theorem was proved in [15].

Theorem 1. *A graph is a general partition graph if and only if there exists a clique cover* $\mathbb{C}$ *for* the edges *of* G *such that every maximal independent set meets every clique in* $\mathbb{C}$.

Definition 2. *A graph* $G = (V, E)$ *satisfies the* triangle condition *if for every maximal independent set* M *and every edge* (x, y) *with* $x, y \notin M$, *there exists a vertex* $m \in M$ *such that* $\{x, y, m\}$ *is a triangle in* G.

It follows immediately from Theorem 1 that a general partition graph satisfies the triangle condition [15]. However, this condition is not sufficient. A computer search has found all connected graphs on 10 or fewer vertices which satisfy the triangle condition but are not general partition graphs [7]. One of these is a graph on 9 vertices and 9 others have 10 vertices [1]. It was noted in [1] that all these graphs have the 3-sun (also called Hajós graph, see, e.g. [2]) as an induced subgraph.

Remark 1. Notice that any graph G can be embedded in a general partition graph by adding a new vertex for every edge in the graph and making this adjacent to the two ends of this edge. The new graph has only one possible clique cover with maximal cliques, and clearly every maximal independent set hits every clique of this cover.

Notice that all *cographs* (i.e., those graphs without induced P_4) are general partition graphs since every maximal independent set meets *every* maximal clique in such a graph. The converse is not true: the 3-sun gives counterevidence. Notice that all cographs are permutation graphs, hence AT-free (Definition 3 below).

Chordal graphs were considered in [1]. In this paper it is proved that chordal graphs are general partition graphs if and only if they obey the triangle condition. Furthermore, it is shown that these graphs can be general partition graphs only if every edge is either incident with a simplicial vertex or constitutes a triangle with some simplicial vertex.

The following lemma was shown in [1]:

Lemma 1. *If a graph* G *satisfies the triangle condition but is* not *a general partition graph then the 3-sun is an induced subgraph of* G.

Definition 3. *Three vertices* x, y, z *form an* asteroidal triple *(AT) of* G *if for every pair of them there is a path connecting the two vertices that avoids the neighborhood of the remaining vertex. A graph* G *is AT-free if no three vertices form an AT.*

Remark 2. Notice that the vertices of an AT are pairwise non-adjacent. The 3-sun is an example of a graph containing an AT. Lekkerkerker and Boland [14] characterized interval graphs as those chordal graphs which are AT-free. Recently AT-free graphs (and generalizations) have drawn renewed attention [2,3,6].

Corollary 1. *An AT-free graph G is a general partition graph if and only if it satisfies the triangle condition.*

Proof. Let G be an AT-free graph. If G is a general partition graph then it satisfies the triangle condition [15]. Conversely, if a graph G satisfies the triangle condition and is *not* a general partition graph, then it has a 3-sun (thus an AT) as an induced subgraph by Lemma 1. □

In this paper we show that the triangle condition can be checked in polynomial time for AT-free graphs, planar graphs and for circle graphs. Thus far it is unknown if the triangle condition is also sufficient for planar graphs. However, we show that there is a polynomial time algorithm to check if a planar graph is a general partition graph.

1 Graphs with Bounded Clique Numbers

In this section we show that for every minor closed class of graphs, there exists a polynomial time algorithm that checks if a graph is a general partition graph.

Theorem 2. *Assume $\mathcal{G}$ is a class of graphs with clique number at most k for some constant k. Then there exists a polynomial time algorithm to check if a graph $G \in \mathcal{G}$ is a general partition graph.*

Proof. We use Theorem 1. Clearly, we may assume that the clique cover mentioned in this theorem contains only maximal cliques.

Since the graph has clique number at most k, there are at most $O(n^k)$ maximal cliques in G. Of course we can obtain a list of all the maximal cliques in polynomial time (for more or less efficient algorithms see, e.g., [4,20]).

Let C be a maximal clique for which there exists a maximal independent set M such that $C \cap M = \varnothing$. Then clearly C cannot be in any clique cover as specified by Theorem 1. We can recognize these *intolerable* cliques as follows. Let $C = \{x_1, \ldots, x_k\}$. For each choice of $x'_1 \in N(x_1) - C, \ldots, x'_k \in N(x_k) - C$ check if $\{x'_1, \ldots, x'_k\}$ (with possible repetitions) is an independent set in G. If there is such an independent set C° then clearly there exists a maximal independent set M containing C° which does not intersect C since every vertex of C has a neighbor in M.

Let $\mathbb{C}$ be the set of *tolerable* maximal cliques (i.e., those that are not intolerable) in G. By Theorem 1 it now follows immediately that G is a general partition graph if and only if $\mathbb{C}$ is a clique cover for the edges of G. This can clearly be tested in polynomial time. □

Corollary 2. *For every minor closed class of graphs $\mathcal{G}$ there exists an efficient algorithm to check if a graph in the class is a general partition graph.*

Proof. According to [18] (see also, e.g., [12]) there exists a *finite obstruction set* $\mathcal{O}$ that demarcates the graphs in $\mathcal{G}$. Let k be the smallest order of the graphs in $\mathcal{O}$. Then the graphs $G \in \mathcal{G}$ cannot have cliques of order at least k, since otherwise every graph with at most k vertices would be a subgraph of G and hence the smallest graph in $\mathcal{O}$ would be a minor. By Theorem 2, there exists a polynomial algorithm to test whether a graph of $\mathcal{G}$ is a general partition graph. □

Corollary 3. *There exists an efficient algorithm to check if a planar graph is a general partition graph.*

2 AT-Free Graphs

In this section we show that there is a polynomial time algorithm to test whether an AT-free graph is a general partition graph. We use Corollary 1 and show that the triangle condition can be tested in polynomial time for AT-free graphs.

Definition 4. *Let x and y be two non-adjacent vertices in an AT-free graph G. Let $C_x(y)$ be the component of $G - N[x]$ that contains y. The* interval $I(x, y)$ *is* $I(x, y) = C_x(y) \cap C_y(x)$.

Definition 5. *Let G be an AT-free graph. Two vertices x and y are* close *if they are non-adjacent and $I(x, y) = \varnothing$.*

Theorem 3. *Assume G is AT-free. Then G is a general partition graph if and only if no close pair $\{x, y\}$ are the end-vertices of an induced P_4.*

Proof. Let G be a general partition graph. Assume x and y are close and assume that there is an induced P_4, say $P = [x, a, b, y]$. Consider a maximal independent set M containing x and y. Clearly, M does not contain a or b. Hence, by the triangle condition, a and b must have a common neighbor z in M. We claim that $z \in I(x, y)$.

Indeed, since $z \in N(b)$, $b \in N(y)$, and $z \notin N(x)$, we have that $z \in C_x(y)$. Of course also $z \in C_y(x)$ and hence $z \in I(x, y)$. But this is a contradiction since $I(x, y) = \varnothing$ by definition.

Now assume that no close pair are the end-vertices of an induced P_4, and that nevertheless G is not a general partition graph. Hence, by Corollary 1 there exist a maximal independent set M and an edge (a, b) with $a, b \notin M$ such that a and b do not have a common neighbor in M. Let $x \in N(a) \cap M$ and $y \in N(b) \cap M$. These vertices must exist since M is maximal and hence $P = [x, a, b, y]$ forms an induced P_4.

We choose x and y in M such that $[x, a, b, y]$ is an induced P_4 and $|I(x, y)|$ is minimal. If $|I(x, y)| = 0$ then $\{x, y\}$ forms a close pair and by assumption these are not the end-vertices of an induced P_4.

Let $z \in I(x,y) \cap M$. Such a vertex exists because otherwise the vertices in M would belong to components of $G - N[x]$ that do not contain y and in components of $G - N[y]$ that do not contain x. But a vertex $z \in I(x,y)$ must be adjacent to some vertex in M, which is a contradiction.

If $z \notin N(a)$ and $z \notin N(b)$ then we claim that $\{x,y,z\}$ is an AT. Indeed, P is an x,y-path avoiding $N(z)$. Since $z \in C_x(y)$, there is a z,y-path avoiding $N(x)$ and of course, for the same reason there is also an x,z-path avoiding $N(y)$. Hence z is adjacent to at least one of a, b, but not both since $z \in M$ and a and b have no common neighbor in M. Assume z is adjacent to a. Notice that $[z,a,b,y]$ is now an induced P_4.

Consider $I(z,y)$. It is easy to see (e.g., [3]) that $I(z,y) \subset I(x,y)$, and hence $|I(z,y)| < |I(x,y)|$, since $z \in I(x,y) - I(z,y)$. This is a contradiction since we chose x and y in M such that $|I(x,y)|$ is as small as possible. □

Theorem 4. *There exists a polynomial algorithm to check if an AT-free graph is a general partition graph.*

Proof. By Theorem 3 we only have to check for every close pair $\{x,y\}$ if there is no induced P_4 with x and y as end-vertices. Clearly this can be done in polynomial time. □

Remark 3. Notice that not all AT-free general partition graphs are perfect. An example is given in Figure 1.

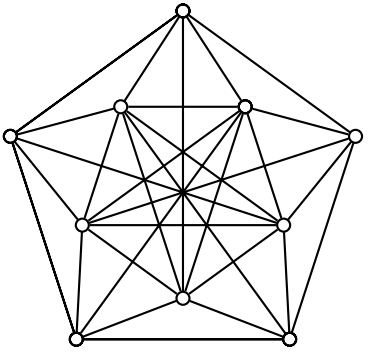


Fig. 1. An AT-free general partition graph containing a C_5

3 Circle Graphs

In this section we show that the triangle condition can be tested in polynomial time for graphs for which RED MAXIMAL INDEPENDENT SET can be solved in

polynomial time. We show that *circle graphs* fall in this category. *Furthermore*, if a graph has only a polynomial number of maximal cliques the RED MAXIMAL INDEPENDENT SET problem can be used to test if a graph is a general partition graph.

Definition 6. *The* RED MAXIMAL INDEPENDENT SET *problem is the following:*
Instance: *Graph G and an assignment of a color* red *or* blue *to each vertex of G*
Question: *Does there exist a maximal independent set in G containing only red vertices?*

Remark 4. Obviously, the problem is equivalent to asking if there is an independent set of red vertices dominating all blue vertices. RED MAXIMAL INDEPENDENT SET can be solved for graph classes for which WEIGHTED INDEPENDENT DOMINATING SET can be solved in polynomial time. (Give the blue vertices a weight ∞ and the red vertices a weight 1. Then there exists a red maximal independent set if and only if there exists an independent dominating set with finite weight.)

RED MAXIMAL INDEPENDENT SET is NP-complete even restricted to planar graphs [21]. The NP-completeness can be seen by observing that the following DISJOINT MAXIMAL INDEPENDENT SET (DMIS) problem is NP-complete.

Instance: Graph $G = (V, E)$ and an edge $(x, y) \in E$
Question: Does there exist a maximal independent set M such that $M \cap N[x] \cap N[y] = \varnothing$?

The fact that DMIS is NP-complete can be seen by a reduction from SAT similar to the one that was used in [5] to show NP-completeness of NOT WELL-COVERED.

Recently it was shown that RED MAXIMAL INDEPENDENT SET is solvable in $O(5^k k^3 n)$ for graphs with cliquewidth at most k [13].

Theorem 5. *Let $\mathcal{G}$ be a class of graphs for which* RED MAXIMAL INDEPENDENT SET *can be resolved in polynomial time for every partition of the vertex set into red and blue vertices. Then the triangle condition can be tested in polynomial time for graphs in $\mathcal{G}$ as well.*

If furthermore the number of maximal cliques is polynomial, then there exists a polynomial time algorithm to test if a graph in $\mathcal{G}$ is a general partition graph.

Proof. Assume RED MAXIMAL INDEPENDENT SET can be solved in polynomial time. Let (x, y) be an edge in G. The algorithm we describe checks if there is a maximal independent set M in G with $N[x] \cap N[y] \cap M = \varnothing$. Clearly, this is the case if and only if there is a maximal independent set $M \not\ni x, y$ which, together with (x, y) violates the triangle condition.

Let $A = N[x] \cap N[y]$. Color the vertices of A blue and all other vertices red. By assumption we can check in polynomial time if there is a maximal independent set M in this subgraph that contains solely red vertices.

Now assume furthermore that $\mathcal{G}$ is a class of graphs with only a polynomial number of maximal cliques. Let $G \in \mathcal{G}$ and consider a list of all maximal cliques

in G [4,20]. For each clique C in this list, we color the vertices of C blue and all other vertices of G red. We use the algorithm for RED MAXIMAL INDEPENDENT SET to test if there exists a maximal independent set which does not intersect C. If there is one, we remove this maximal clique from the list. Finally, the graph is a general partition graph if and only if the remaining maximal cliques form a clique cover for the edges in the graph. This proves the theorem. □

Remark 5. WEIGHTED INDEPENDENT DOMINATING SET (hence also RED MAXIMAL INDEPENDENT SET) can be solved in polynomial time for AT-free graphs using methods from [3]. This gives an auxiliary proof that the triangle condition can be tested in polynomial time for the class of AT-free graphs. On the other hand, recall that CLIQUE remains NP-complete for AT-free graphs [3]. Indeed, $K_{2n} - \{\text{a perfect matching}\}$ is an AT-free graph with 2^n maximal cliques.

Definition 7. *A graph G is a* circle graph *if G is the intersection graph of chords in a circle.*

Remark 6. Circle graphs can be recognized in $O(n^2)$ time [16,19]. The class contains all permutation graphs but not all elements are perfect. (For example all cycles are circle graphs.)

Remark 7. In this section we describe an algorithm to solve RED MAXIMAL INDEPENDENT SET for circle graphs. This algorithm can be used to solve TRIANGLE CONDITION. Unfortunately, the triangle condition is *not* a sufficient condition for a circle graph to be a general partition graph. Indeed, the graph on 9 vertices, pictured in [1,7] which satisfies the triangle condition but which is *not* a general partition graph is a circle graph. The graph is the intersection graph of the chords pictured in Figure 2.

For the description of our algorithm it is convenient to use another intersection model for circle graphs. Consider a collection of intervals on the real line. Two intervals with a non-empty intersection are said to *overlap* if they have a non-empty intersection, but neither is fully contained in the other. Given a set of chords in a circle, it is easy to obtain an overlap model by projecting each chord in the circle from the north pole upon a line below the circle. In [11] the following observation was made:

Theorem 6. *A graph G is a circle graph if and only if G is the overlap graph of a collection of intervals on the real line.*

Consider an overlap model for a circle graph $G = (V, E)$. We identify a vertex x with the interval in the model representing it and we write $\ell(x)$ and $r(x)$ for the left- and right-endpoint of x.

Our algorithm checks if there is a red independent set such that every blue vertex has a neighbor in it. We say that such a red independent set *covers* the

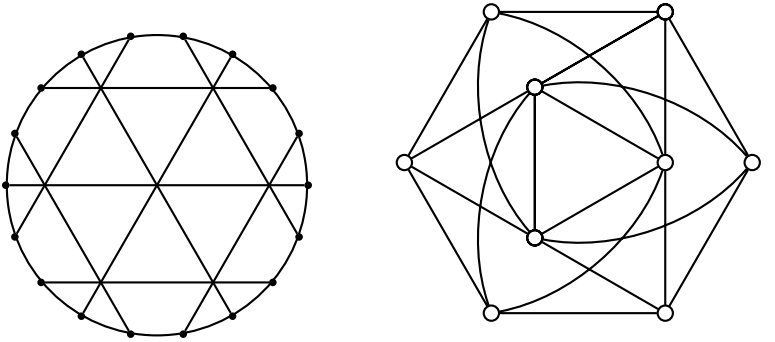


Fig. 2. Non-general partition circle graph satisfying the triangle condition

blue vertices. We describe a dynamic programming procedure $P_{\mathrm{I}}(x)$ that checks, for a given red interval x, if the blue intervals fully contained in x can be covered by an independent collection of red intervals which are also fully contained in x. Notice that we can add a new red interval ω to our model covering all other intervals, and then a **call** $P_{\mathrm{I}}(\omega)$ solves our problem.

First sort all the intervals according to their length. (Of course we may assume that no two intervals have the same length and that no two end-points of different intervals coincide.) We deal with the red intervals one by one according to increasing length.

If a red interval x contains no other red intervals, then $P_{\mathrm{I}}(x) = \textsc{true}$ if x contains no blue intervals. Otherwise, $P_{\mathrm{I}}(x) = \textsc{false}$.

Consider a red interval x and assume that for all red intervals y that are contained in x, $P_{\mathrm{I}}(y)$ is correctly determined. In order to determine $P_{\mathrm{I}}(x)$, we now consider only intervals (red and blue) that are fully contained in x and furthermore, we only consider those red intervals y in x with $P_{\mathrm{I}}(y) = \textsc{true}$. We scan the red intervals y (that are contained in x) according to *increasing* right end-point $r(y)$.

For a red interval y in x **define** $P_x(y)$ as a boolean variable indicating whether or not there is a collection of independent red intervals in x containing y that *cover* all the blue intervals b in x with $r(b) < r(y)$ (i.e., each of these blue vertices must have a red neighbor in this collection).

If there is no red interval y' such that $r(y') < \ell(y)$, then $P_x(y) = \textsc{true}$ if and only if there is no blue interval b in x with $r(b) < \ell(y)$.

Now consider another red interval y and let y' be the red interval with the *largest* right end-point $r(y') < \ell(y)$ and with $P_x(y') = \textsc{true}$. Then $P_x(y) = \textsc{true}$ if and only if there is no blue interval b *between* y' and y (i.e., with $r(y') < \ell(b) \leq r(b) < \ell(y)$).

Finally, consider the red interval y in x with largest right end-point for which $P_x(y) = \textsc{true}$. If there is a blue interval b with $\ell(b) > r(y)$, then $P_{\mathrm{I}}(x) =$

FALSE. On the other hand, if there is no blue interval b in x with $r(b) > r(y)$, then all blue vertices in x can be covered and hence $P_1(x) =$ TRUE.
If there are blue intervals with $\ell(b) < r(y) < r(b)$, then it is unclear whether there is a satisfying cover. But if there is one, then the *left* end-point of such a blue vertex must be covered. Hence we may *shrink* the blue interval to its left end-point and **re-run** the whole algorithm described above computing $P_1(x)$. Of course, this time the shrunken blue intervals are covered if they are contained in some red interval in the cover, i.e., if they don't fall between two consecutive red intervals in the cover. Notice that the description above correctly checks this.

Theorem 7. *There exists a polynomial time algorithm to solve the* RED MAXIMAL INDEPENDENT SET *problem for circle graphs.*

Proof. A moment's reflection shows the correctness of the procedure described above. Notice that for each re-run of the part computing $P_1(x)$, at least one blue interval is shrunken to its left end-point. Hence the number of re-runs is bounded by the number of blue vertices, since each blue vertex can be shrunken only once. Each step can clearly be performed in polynomial time. □

By Theorem 5 we obtain:

Theorem 8. *The* TRIANGLE CONDITION *problem can be solved in polynomial time for circle graphs.*

4 The Triangle Condition for Planar Graphs

We now show that the TRIANGLE CONDITION problem can be checked in polynomial time for planar graphs. Let G be a *plane* graph, i.e., a planar graph G given together with an embedding in the plane. Consider an edge (x, y) in G. Without loss of generality we assume that (x, y) is an edge of the outerface in G. Let $x' \in N(x) - N[y]$ and $y' \in N(y) - N[x]$ be non-adjacent private neighbors of x and y respectively. The algorithm we describe checks if there is a maximal independent set $M \ni x', y'$ such that $N[x] \cap N[y] \cap M = \varnothing$. Let $N(x) \cap N(y) = \{x_1, \ldots, x_k\}$. We choose this numbering of the common neighborhood such that for each i, $\{x_1, \ldots, x_{i-1}\}$ is contained in the interior of the area of G spanned by the triangle $\{x, y, x_i\}$.

For each i we color a vertex x_i *white* if there exist pairwise non-adjacent neighbors $x'_t \in N(x_t) - N(\{x', y'\})$ [1] for each $t = 1, \ldots, i$ such that x'_i is in the interior of the triangle $\{x, y, x_i\}$. We color a vertex x_i *red* if we cannot choose these neighbors.

We can color the vertices $x_1, x_2, \ldots, x_k$ in the following way. If there is an $x'_1 \in N(x_1) - N(\{x', y'\})$, such that x'_1 is in the interior of the triangle $\{x_1, x, y\}$, we color x_1 *white*, otherwise, we color x_1 *red*. Suppose that x_i has been colored for

[1] Here we use $N(\{x', y'\})$ to indicate the union of the two neighborhoods.

$i \geq 1$. If x_i is white, then check if there is a vertex $x'_{i+1} \in N(x_{i+1}) - N(\{x', y'\})$ in the interior of the triangle $\{x, y, x_{i+1}\}$. If there is such a vertex then color x_{i+1} white. If not, then color x_{i+1} red.

Now assume x_i is red. First check if there are neighbors $x'_i \in N(x_i) - N(\{x', y'\})$ and $x'_{i+1} \in N(x_{i+1}) - N(\{x', y'\})$ which are non-adjacent, and which both lie in the interior of the square $\{x, y, x_i, x_{i+1}\}$. If there are such neighbors, then color x_{i+1} white. Otherwise, check if there is a neighbor $x'_i \in N(x_i) - N(\{x', y'\})$ inside the square $\{x, y, x_i, x_{i+1}\}$. If so, then color x_{i+1} red. If this is also not the case, then a maximal independent set containing x' and y' and that misses $N[x] \cap N[y]$ cannot exist. (The modification necessary for the case $i = k$ is clear.)

Clearly, this test can be performed in polynomial time, and since the correctness is obvious, we obtain:

Theorem 9. TRIANGLE CONDITION *is polynomial for planar graphs.*

5 Final Remarks

Let u and v be two *twins* (true or false) in G (see, e.g., [2]). Then G is a general partition graph iff $G-u$ is a general partition graph. Also, G satisfies the triangle condition iff $G-u$ does so [15,1]. Clearly, the class of circle graphs is closed under adding a new vertex and making it a twin with an existing vertex. We conjecture that the only non-general partition circle graphs satisfying the triangle condition are obtained from the graph depicted in Figure 2 by a series of this operation. This conjecture is true for the case of 10 vertices. (For a list of these graphs see [1]. It is easy to check which of these 9 graphs with 10 vertices are circle graphs and which are not. Exactly 4 of them are circle graphs.) Clearly, if the conjecture is true then it can be tested in polynomial time if a circle graph is a general partition graph.

We end with the following conjecture:

Conjecture 1. The TRIANGLE CONDITION problem is co-NP-complete.

References

1. Anbeek, Carreen, Duane DeTemple, Kevin McAvaney, and Jack Robertson, When are chordal graphs also partition graphs?, *Australas. J. Combin.* **16**, (1997), pp. 285–293.
2. Brandstädt, A., Van Bang Le, and Jeremy P. Spinrad, *Graph classes–A Survey*, SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 1999.
3. Broersma, H. J., T. Kloks, D. Kratsch, and H. Müller, Independent sets in AT-free graphs, *Proceedings of the 24th International Colloquium on Automata, Languages, and Prgramming, ICALP'97*, Springer Verlag Lecture Notes in Computer Science 1256, (1997), pp. 760–770.
4. Bron, S. and J. Kerbosch, Algorithm 457–Finding all cliques of an undirected graph, *Comm. of ACM* **16**, (1973), pp. 575.

5. Chvátal, Václav, and Peter J. Slater, A note on well-covered graphs, In: *Quo Vadis, Graph Theory?* (J. Gimbel, J. W. Kennedy & L. V. Quintas eds.), Annals of Discrete Mathematics **55**, 1993, pp. 179–182.
6. Corneil, D. G., S. Olariu, and L. K. Stewart, Asteroidal triple-free graphs, *SIAM Journal on Discrete Mathematics* **10**, (1997), pp. 399–430.
7. DeTemple, D. W., M. J. Dineen, J. M. Robertson, and K. L. McAvaney, Recent examples in the theory of partition graphs, *Discrete Mathematics* **113**, (1993), pp. 255–258.
8. DeTemple, Duane, Frank Harary, and Jack Robertson, Partition graphs, *Soochow J. Math.* **13**, (1987), pp. 121–129.
9. DeTemple, Duane W. and Jack M. Robertson, Constructions and the realization problem for partition graphs, *J. Combin. Inform. System Sci.* **13**, (1988), pp. 50–63.
10. DeTemple, Duane, Jack Robertson, and Frank Harary, Existential partition graphs, *J. Combin. Inform. System Sci.* **9**, (1984), pp. 193–196.
11. Gavril, F., Algorithms for a maximum clique and a maximum independent set of a circle graph, *Networks* **3**, (1973), pp. 261–273.
12. Kloks, Ton, *Treewidth–Computations and Approximations*, Springer Verlag Lecture Notes in Computer Science 842, 1994.
13. Kloks, Ton, Dieter Kratsch, C. M. Lee, and Jiping Liu, Improved bottleneck domination algorithms. Manuscript 2003.
14. Lekkerkerker, C. and D. Boland, Representation of finite graphs by a set of intervals on the real line, *Fund. Math.* **51**, (1962), pp. 45–64.
15. McAvaney, Kevin, Jack Robertson, and Duane DeTemple, A characterization and hereditary properties for partition graphs, *Discrete Mathematics* **113**, (1993), pp. 131–142.
16. Naji, W., Reconnaissance des graphes de cordes, *Discrete Mathematics* **54**, (1985), pp. 329–337.
17. Nebeský, Ladislav, On partition graphs and generalizations of line graphs, *Časopis Pěst. Mat.* **102**, (1977), pp. 203–205.
18. Robertson, N. and P. D. Seymour, Graph minors–A Survey. In: (I. Anderson ed.), *Surveys in Combinatorics*, Cambridge University Press, Cambridge, 1985, pp. 153–171.
19. Spinrad, J. P., Recognition of circle graphs, *J. Algorithms* **16**, (1994), pp. 264–282.
20. Tsukiyama, S., Algorithms for generating all maximal independent sets, *Electronics & Communications* **59**, (1976), pp. 1–8.
21. Yen, William Chung-Kung, Bottleneck domination and bottleneck independent domination on graphs, *Journal of Information Science and Engineering***18**, (2002), pp. 311–331.

Short Cycles in Planar Graphs

Łukasz Kowalik*

Institute of Informatics, Warsaw University
Banacha 2, 02-097 Warsaw, Poland
kowalik@mimuw.edu.pl

Abstract. We present new algorithms for finding short cycles (of length at most 6) in planar graphs. Although there is an $O(n)$ algorithm for finding *any* fixed subgraph H in a given n-vertex planar graph [5], the multiplication constant hidden in "O" notation (which depends on the size of H) is so high, that it rather cannot be used in practice even when $|V(H)| = 4$. Our approach gives faster "practical algorithms" which are additionally much easier to implement.
As a side-effect of our approach we show that the maximum number of k-cycles in n-vertex planar graph is $\Theta(n^{\lfloor \frac{k}{2} \rfloor})$.

1 Introduction

The subgraph isomorphism problem consists in deciding whether a given graph G, called "text", contains another graph H, called "pattern", as a subgraph. This is one of the most important and natural problems in the algorithmic graph theory, often arising in applications. In this paper we focus on the situation when G is planar and we consider one of the most natural class of patterns – cycles. It is easy to show that such restricted version of the problem is NP-complete, because of the reduction from the Hamiltonian cycle problem. However, treating the pattern as a fixed graph one can consider polynomial algorithms. Eppstein [5] presents a linear algorithm finding any fixed pattern in a planar graph. Unfortunately, for a given w-vertex pattern graph H his algorithm has to generate as much as $O(w^{3w+9})$ combinatorial objects. Therefore, in spite of great theoretical importance, the algorithm of Eppstein cannot be used in practice even for 4-vertex patterns. Thus constructing effective algorithms for particular patterns is still a challenging research area.

Related Work. The problem of finding cycles of specified lengths in a planar graphs attracted many researchers. Papadimitriou et al. [9] gave first linear, but complicated algorithm for finding C_3's. Two simple linear algorithms for finding triangles were developed by Chiba et al. [3] and Chrobak et al. [4]. The first of that papers describes also simple linear algorithm for finding C_4's. Both algorithms from that paper can be also applied to d-degenerate graphs containing m edges with $O(d \cdot m)$ time complexity. Richards [10] gave $O(n \log n)$ algorithms

* Research supported by KBN grant 4T11C04425

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 284–296, 2003.

for finding C_5's and C_6's. Alon et al. presented $O(d^2 \cdot m)$ time algorithm for finding C_5 (only one occurrence) in d-degenerate graph of m edges.

Apart from Eppstein's result [5] there were also a few other algorithms for finding cycles of *arbitrary* fixed length in planar graphs. Monien [8] presented an algorithm working in $O(m \cdot n)$ time for an arbitrary graph containing n vertices and m edges. Alon et al. designed an algorithm for planar graphs working in $O(n)$ expected time or $O(n \log n)$ worst case time. In our recent paper with M. Kurowski [7] we presented a data structure answering short paths queries in planar graphs. The structure can be also adapted to construct a linear time algorithm for finding cycles of any fixed length. However, similarly as in the Eppstein's algorithm, the constant hidden in the asymptotic notation describing the time complexity of the algorithm is very high.

New Results. We present a new approach to finding short cycles in planar graphs. We are able to apply our methods to cycles of lengths from 3 to 6. We obtain algorithms listing all $\#_c$ occurrences of cycles of a given length in n-vertex graph in time $O(n + \#_c)$. Each of the algorithms detects the first cycle in time $O(n)$. Moreover, the algorithms for cycles of length 3, 4, 5 work also for d-degenerate graphs. The running time in this case is $O(d^2 \cdot m + \#_c)$ (resp. $O(d^2 \cdot m)$ time when we search for only one cycle), where m denotes the number of edges and $\#_c$ is the number of cycles.

In section 6 we show how to extend the approach of Chiba and Nishizeki [3] to the case of 5-cycles obtaining a very simple algorithm. It can be also easily modified to count 5-cycles in d-degenerate graph of m edges in time $O(d^2 \cdot m)$. Both our algorithms for finding 5-cycles in d-degenerate graphs match the time complexity of the algorithm of Alon et al. being much simpler at the same time.

As a side-effect of our approach in section 4 we show that the maximum number of k-cycles in an n-vertex planar graph is $\Theta(n^{\lfloor \frac{k}{2} \rfloor})$. This fact apart from being interesting combinatorial result is used for time complexity analysis of our algorithms.

None of our algorithms requires a plane embedding of an input graph.

Applications in Graph Coloring. The problem of finding cycles in planar graphs is related to classical coloring and list coloring of planar graphs. Deciding whether a given planar graph is 3-colorable or 3-choosable is NP-complete. Nevertheless, we know that planar graphs without triangles are 3-colorable and planar graphs without C_3's and C_4's are 3-choosable [11]. Moreover, for any k, $3 \leq k \leq 6$ an arbitrary planar graph without k-cycles is 4-choosable [6]. Thus, algorithms for finding short cycles in planar graphs allow to recognize wide classes of 3-colorable, 3-choosable and 4-choosable planar graphs.

Our Approach. We start from transforming an n-vertex input graph G into new graphs G_2 and G_3. Each edge in G_2 (resp. G_3) corresponds to a certain path of length 2 (resp. 3) in G. Obviously, since the number of all paths of length 2 in planar graph can be $\Omega(n^2)$, we have to choose only some of the paths to guarantee a linear size of $E(G_2)$. On the other hand, we cannot lose information on any cycle that we search for. In the second phase our algorithms

search for shorter cycles in graphs G_2 and G_3 or for pairs of internally disjoint paths corresponding to the same edges in G_2 or G_3.

2 Preliminaries

We assume the reader is familiar with standard terminology concerning graph theory and planar graphs in particular. In this section we recall some notions that are not so widely used.

Let G be an undirected graph. We say that G is *d-degenerate* when an arbitrary subgraph of G contains a vertex of degree at most d. A directed graph is said to be *k-oriented* if its every vertex has the out-degree at most k. If one can orient edges of graph G obtaining k-oriented graph G' we say that G is *k-orientable*. The *arboricity* $a(G)$ of graph G is the minimal number of forests needed to cover all the edges of G. The three defined notions are closely related. It is easy to show that $d = \Theta(k)$ and $d = \Theta(a(G))$. It is also well known that any planar graph is 5-degenerate, 3-orientable [4] and has arboricity at most 3.

3 Graphs of Paths of Length 2 and 3

Algorithm 1 described below transforms an input graph G into two graphs G_2 and G_3. The algorithm successively removes vertices of low degrees from graph G, storing in graphs G_2 and G_3 information about paths crossing the deleted vertices. $N(v)$ denotes the set of all neighbors of vertex v in current graph G and $N_0(v)$ denotes the set of neighbors of v in the input graph G. For every edge added to G_2 or G_3 we store the corresponding path of length 2 or 3 respectively. An edge x–$y \in G_2$ associated with a path xvy in G is denoted by $x \overset{v}{—} y$. Similarly, $x \overset{vw}{—} y$ denotes an edge in G_3 associated with path $xvwy$ in G. Observe that several edges joining the same vertices can be added to G_2 (resp. G_3). The graphs G_2 and G_3 are transformed to simple ones in step 12.

Theorem 1. *Every cycle in the input graph consists of edge disjoint paths corresponding to certain edges in G_2 or G_3, treated as multigraphs.*

Proof. Let G_0 denote the input graph. We will show by the induction on the number of deleted vertices that whenever the algorithm starts or ends the execution of the while loop every cycle in G_0 consists of edge disjoint paths corresponding to certain edges in G_2 or G_3 and of edges in the current graph G.

Before any vertex was deleted the invariant is trivially satisfied, since then $G_0 = G \cup G_2 \cup G_3$. Now we assume that the invariant holds at the beginning of the while loop (statement 4) and we will show that it is still satisfied at the end (after statement 11). Let $C_0 \subseteq G_0$ be an arbitrary cycle. From the induction hypothesis there exists a cycle $C \subseteq G \cup G_2 \cup G_3$ corresponding to C_0 (as a union of edge disjoint paths). We assume that the vertex v chosen in statement 4 is incident with an edge v–$a \in G \cap C$ (otherwise C is not affected and there is nothing to prove). Let $b \in V(C)$ be the neighbor of v in C different from a. There are three cases to consider.

Algorithm 1 Transforming G to G_2 and G_3

1: $G_2 \leftarrow \emptyset$
2: $G_3 \leftarrow \emptyset$
3: **while** $G \neq \emptyset$ **do**
4: $v \leftarrow$ a vertex with the lowest degree in G
5: **for all** $x \in N(v)$ **do**
6: **for all** $y \in N_0(v) - N(v)$ **do**
7: $G_2 \leftarrow G_2 \cup \{x \overset{v}{-} y\}$
8: **end for**
9: **for all** $v \overset{w}{-} y \in G_2$ **do**
10: $G_3 \leftarrow G_3 \cup \{x \overset{vw}{-} y\}$
11: **end for**
12: delete the edge $v - x$ from G
13: **end for**
14: delete the vertex v from G
15: **end while**
16: Replace multiple edges in G_2 and G_3 by single ones. For every single edge store all the paths corresponding to it.

When v–$b \in G$ the edge $a \overset{v}{-} b$ is added to G_2 in statement 7. It is clear that $C' = C - \{v-a, v-b\} \cup \{a \overset{v}{-} b\}$ corresponds to C_0. Similarly when $v - b \in G_2$ the edge a–b is added to G_3 in statement 9. Again we see that $C'' = C - \{v-a, v-b\} \cup \{a - b\}$ corresponds to C_0. Finally, when $v \overset{cd}{-} b \in G_3$ the edge $a \overset{v}{-} c$ is added to G_2 in statement 7. It is easy to observe that both $v \overset{c}{-} d$ and $c \overset{d}{-} b$ are in G_2. Then we see that at the end of the while loop $C''' = C - \{v-a, v \overset{cd}{-} b\} \cup \{a \overset{v}{-} c, c \overset{d}{-} b\} \subseteq G \cup G_2 \cup G_3$ corresponds to C_0. That ends the proof. □

Proposition 1. *Let G be an arbitrary d-degenerate graph and let G_2, G_3 be graphs generated by Algorithm 1. Then at most $2d \cdot |E(G)|$ paths are stored in G_2, and at most $2d \cdot |E(G_2)|$ paths are stored in G_3.*

Proof. Consider a path xvy in G corresponding to an edge e in G_2. Assume that in the moment of adding the path to G_2, $x \in N(v)$ and $y \in N_0(v) - N(v)$. We say that the edge v–y *generates* e. Let us consider an arbitrary edge u–v in graph G. It generates at most $2d$ edges in G_2. Subsequently, at most $2d \cdot |E(G)|$ paths are stored in G_2. Similarly, at most $2d \cdot |E(G_2)|$ paths are stored in G_3. □

Corollary 1. *Let G be an arbitrary d-degenerate graph and let G_2, G_3 be graphs generated by Algorithm 1. Then $|E(G_2)| \leq 2d \cdot |E(G)|$ and $|E(G_3)| \leq 2d \cdot |E(G_2)| \leq 4d^2 \cdot |E(G)|$.*

Corollary 2. *For any d-degenerate graph G, Algorithm 1 can be implemented to work in time $O(d^2 \cdot |E(G)|)$.*

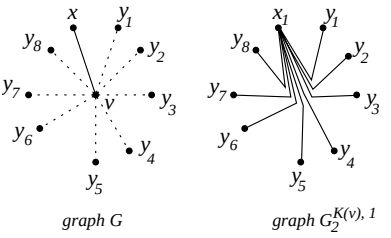


Fig. 1. Drawing edges incident with x.

Proof. It suffices to show that step 12. of the algorithm can be implemented in linear time. It can be achieved by radix-sorting all the edges of G_2 and G_3. Assume that the vertices of G form a linear order. Every edge $x - y \in G_2$ with $x < y$, corresponding to a path xvy in G, is transformed into the sequence (x, y, v). All the sequences are sorted in $O(|V(G)| + d \cdot |E(G)|)$ time using radix-sort. Then we create the simple graph G_2 and for every edge we store the set of all paths corresponding to it. Similarly we sort all the 3-paths corresponding to edges in multigraph G_3 and we build a simple graph G_3. □

Corollary 3. *If G is planar then $|E(G_2)| \leq 10|E(G)|$ and $|E(G_3)| \leq 10|E(G_2)|$.*

Lemma 1. *For any planar graph G, graph G_2 generated by Algorithm 1 is a union of at most 20 planar graphs.*

Proof. In the proof we consider multigraph G_2 generated by the algorithm before the execution of step 12.

Let $G = (V, E)$. It is well known that every planar graph is 4-colorable. Let $\mathcal{K} : V \longrightarrow \{1, 2, 3, 4\}$ be a 4-coloring of G. Let us consider a vertex v of degree ≤ 5 chosen by the algorithm in line 4. We can assign different numbers from set $\{1, \ldots, 5\}$ to its neighbors. For a neighbor x of v let $no_v(x) \in \{1, \ldots, 5\}$ denote its number. We define a partition of G_2,

$$G_2 = \bigcup_{\substack{i \in \{1,\ldots,4\} \\ j \in \{1,\ldots,5\}}} G_2^{i,j}$$

in which graph $G_2^{i,j}$ contains an edge $x \overset{v}{—} y$ considered in line 7 of the Algorithm 1 when $\mathcal{K}(v) = i$ and $no_v(x) = j$.

Now it suffices to show that each graph $G_2^{i,j}$ is planar. Let us take any plane embedding of G. Consider one of graphs $G_2^{i,j}$. We will show that it has a plane embedding. Every vertex in $G_2^{i,j}$ is embedded in exactly the same point as the corresponding vertex in G. An edge $x - y \in G_2^{i,j}$ corresponding to path $p = xvy$ in G is drawn *close* to the embedding of p in such a way that for given $x, v \in V$ none of the edges corresponding to paths xvy crosses any other edge (see Fig. 1).

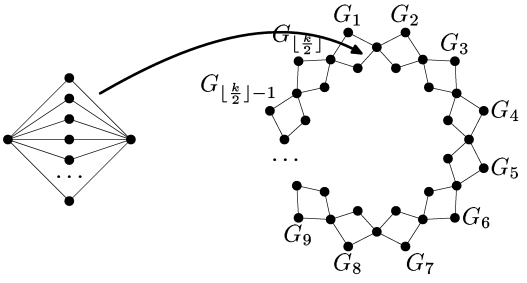


Fig. 2. Construction of an n-vertex graph containing $O(n^{\lfloor \frac{k}{2} \rfloor})$ k-cycles

Consider an arbitrary edge $e = x \overset{v}{—} y \in G_2^{i,j}$. Assume that e crosses another edge $e' = x' \overset{v'}{—} y' \in G_2^{i,j}$. If $v = v'$ either the edges were drawn in such a way that they do not cross or the edges cannot belong to the same graph $G_2^{i,j}$. Subsequently $v \neq v'$ and it remains to consider the case when $v = x'$ (the other ones are symmetric). Then v is adjacent to v' in G. We see that such a case cannot happen since v and v' are assigned different colors implying that both edges e and e' cannot belong to $G_2^{i,j}$.

□

Corollary 4. *Let G_2 be a graph generated by Algorithm 1. Then G_2 has arboricity 60, G_2 is 60-orientable and 120-degenerate.*

4 The Maximum Number of Given Length Cycles

In this short section we show how to use properties of graphs G_2 and G_3 to prove the following theorem.

Theorem 2. *Let $k \geq 3$ be a fixed integer constant. The maximum number of k-cycles in an n-vertex planar graph is $\Theta(n^{\lfloor \frac{k}{2} \rfloor})$.*

Proof. We start from the lower bound. We construct an n-vertex planar graph with $\Omega(n^{\lfloor \frac{k}{2} \rfloor})$ cycles of length k as follows. Assume that k is even. We start from a cycle C of length $\frac{k}{2}$. Then each edge u–v of cycle C is replaced by l paths of length 2 joining u and v. The construction is shown on Fig. 2. Note that $n = (l+1) \cdot \frac{k}{2}$ and $l = \Omega(n)$. It is clear that the resulting graph has $\Omega(l^{\frac{k}{2}}) = \Omega(n^{\lfloor \frac{k}{2} \rfloor})$ cycles of length k as required. When k is odd the construction is similar. We start from a cycle C of length $\lfloor \frac{k}{2} \rfloor + 1$ and we choose an edge $e \in C$. Every edge except e is replaced by l paths of length 2, as before. The resulting graph has $\Omega(l^{\lfloor \frac{k}{2} \rfloor}) = \Omega(n^{\lfloor \frac{k}{2} \rfloor})$ cycles of length k.

The upper bound is an immediate conclusion from Theorem 1 and Proposition 1. Let G be an arbitrary planar graph. It suffices to observe that a cycle of

length k in G can consist of at most $\lfloor \frac{k}{2} \rfloor$ paths of length 2 and 3 stored in G_2 and G_3. Since there is $O(n)$ such paths (Prop. 1) and every cycle in G consists of these paths, there can be no more than $O(n^{\lfloor \frac{k}{2} \rfloor})$ cycles of length k in G. □

5 Finding Cycles of Lengths from 3 to 6

In this section we show how to use graphs G_2 and G_3 to find short cycles. Algorithms for cycles of length 3, 4 and 5 apply also to d-degenerate graphs and have linear complexity, provided that $d = O(1)$. In the algorithm for cycles of length 6 we need Lemma 1. Therefore this algorithm apply only to planar graphs. In the rest of the section we assume that we search cycles in d-degenerate graph G of n vertices and m edges.

5.1 Length 3

As an immediate consequence of Theorem 1 we see that it suffices to find self-loops in G_3. Every such self-loop corresponds to certain cycle in G. The algorithm works in time $O(d^2 m)$.

5.2 Length 4

Using Theorem 1 we get that every cycle $xvyw$ of length 4 in G is formed by two paths xvy and xwy corresponding to certain edge $x - y$ in G_2. To find a cycle of length 4 it suffices to find an edge in G_2 that corresponds to more than one path in G. The algorithm works in time $O(d^2 \cdot m)$.

Similarly, if we want to list all 4-cycles in G it suffices to consider, for every edge $x - y \in G_2$, all pairs of paths corresponding to this edge. The algorithm works in time $O(d^2 \cdot m + \#_c)$, where $\#_c$ denotes the number of C_4's in the graph ($\#_c = O(n^2)$).

5.3 Length 5

Theorem 1 implies that every cycle of length 5 in G is formed by two paths of length 2 and 3 corresponding to certain edges x–$y \in G_2$ and x–$y \in G_3$ respectively. We can easily compute $E(G_2) \cap E(G_3)$ in linear time using radix-sort. Observe that two paths of length 2 and 3, corresponding to edges $x - y \in E(G_2)$ and $x - y \in E(G_3)$ respectively, do not necessarily form a cycle. For a path xvy of length 2 there can be even $\Theta(n)$ paths of length 3 joining x and y and crossing vertex v. Luckily the following proposition holds:

Proposition 2. *Any x-y path of length 3 can internally intersect with at most 2 x-y paths of length 2.*

To find one cycle of length 5, for every edge u-v in G_3 and for every path of length 3 in G corresponding to that edge we check at most 3 paths corresponding to edge u-v in G_2. The time will be $O(d^2 \cdot m)$ due to Proposition 1.

In the case when we want to find all 5-cycles, for every two edges, x-$y \in G_2$ and x-$y \in G_3$, we check all the pairs of corresponding paths. At most $O(d^2 m)$ pairs of paths do not form a cycle. Therefore the algorithm works in time $O(d^2 m + \#_c)$, where $\#_c$ denotes the number of C_5's in the graph ($\#_c = O(n^2)$).

It is possible that the algorithm returns one cycle more than once. In order get rid of unnecessary copies without increasing the time complexity it suffices to sort the cycles using radix-sort. The cycle C is transformed to a sequence $(v_1, v_2, v_3, v_4, v_5)$ such that $v_1 v_2 \ldots v_5$ are successive vertices of C, $v_1 = \min V(C)$ and $v_2 = \min N_C(v_1)$ ($N_C(v_1)$ denotes the set of neighbors of v_1 in C).

5.4 Length 6

In this section we assume that G is an n-vertex planar graph. From Theorem 1 we know that each cycle of length 6 in G is formed either from two paths corresponding to the same edge in G_3 or it is formed from 3 paths of length 2 such that corresponding edges in G_2 form a triangle in G_2. The algorithm finding one occurrence of C_6 is linear. If we want to list all C_6's our algorithm will work in time $O(n + \#_c)$, where $\#_c$ denotes the number of C_6's in the graph ($\#_c = O(n^3)$). In the latter version of algorithm it is possible that the same cycle is found twice: either as a triple of 2-paths and as two paths corresponding to the same edge in G_3. To have every cycle printed exactly once it suffices to radix-sort all the cycles found in the last phase of the algorithm, similarly as it was shown is Section 5.3.

Finding Triangles in G_2. Since G_2 is $O(1)$-degenerate (see Corollary 4), we can use the algorithm from section 5.1 to find all triangles in linear time. Alternatively we can use any other algorithm finding C_3's in $O(1)$-degenerate graphs in linear time, e. g. [3,4]. Subsequently there are $O(n)$ triangles in G_2. For every such triangle we need to find 2-paths corresponding to its edges that form a 6-cycle in G. Consider a triangle $T = v_1 v_2 v_3$ in G_2. Observe that we can ignore 2-paths $v_1 v_2 v_3$, $v_1 v_3 v_2$, $v_2 v_1 v_3$. None of them can be a part of 6-cycle corresponding to T. All the other 2-paths corresponding to the edges of T will be called *needed*. For an edge e in triangle T let $Need(e)$ denote the set of needed paths corresponding to e. We check the triples of 2-paths using algorithm 2.

Proposition 3. *Every $O(1)$ steps Algorithm 2 either returns a 6-cycle or terminates.*

Proof. Let $T = v_1 v_2 v_3$ be a triangle in G_2. Consider needed paths corresponding to the edges v_1-v_2 and v_2-v_3. Let 2-paths $v_1 x v_2$ and $v_2 y v_3$ correspond to the edges v_1-v_2 and v_2-v_3 respectively. Assume that the paths are not internally disjoint. Since they are needed, $x = y$. Observe that there can be at most 2 pairs of such

Algorithm 2 Searching for triples of 2-paths that form 6-cycles

```
1: Sort edges of triangle T so |Need(e1)| ≤ |Need(e2)| ≤ |Need(e3)|.
2: for all p ∈ Need(e1) do
3:   for all q ∈ Need(e2) do
4:     if p and q are internally disjoint then
5:       for all r ∈ Need(e3) do
6:         if p, q and r form a 6-cycle in G then
7:           Print the cycle.
8:         end if
9:       end for
10:    end if
11:  end for
12: end for
```

paths that are not internally disjoint because every 3 such pairs of paths form $K_{3,3}$ in G.

Thus, there can be at most 2 pairs of paths $p \in Need(e_1)$ and $q \in Need(e_2)$ that are not internally disjoint. Consider a pair of internally disjoint paths p and q checked by the algorithm. We can assume that $|Need(e_3)| \geq 3$, because otherwise there would be $O(1)$ triples to check and nothing remains to prove. Subsequently the algorithm can check at most 2 paths $r \in Need(e_3)$ that do not form a 6-cycle with p and q. Thus, when $|Need(e_3)| \geq 3$, among every 3 successively checked triples of paths at least one forms a cycle. □

Corollary 5. *One can find all 6-cycles in G corresponding to triangles in G_2 in time $O(n+\#_c)$ where $\#_c$ denotes the number of all such 6-cycles. The algorithm finds the first cycle in time $O(n)$.*

Checking Paths Corresponding to the Same Edge in G_3. Among all k paths corresponding to one edge in G_3 we need to find internally disjoint ones. If $k < 8$ there is $O(1)$ possible pairs and we can verify each of them. Otherwise we need the following proposition:

Proposition 4. *Let e be an edge in G_3 corresponding to $k \geq 8$ different 3-paths in G. There is an algorithm working in $O(k)$ time which either decides that there is no pair of paths corresponding to 6-cycle or returns $\Theta(k)$ cycles of length 6 and at least two paths such that there is no more cycles containing any of that paths.*

Proof. Assume that there is a cycle C compound of two internally disjoint 3-paths, p and q, each corresponding to e. Obviously there can be no more that 8 paths corresponding to e that are neither internally disjoint with p nor with q. For at most 9 paths $p_1, p_2, \ldots p_9$ the algorithm verifies whether they form a cycle with all the other paths corresponding to e. Thus, after checking at most $9k$ pairs of paths, the algorithm finds a cycle compound of one of the paths p_i

and a path p. Then for every path r corresponding to e, the algorithm checks whether r forms a cycle with p. Observe that the total number of cycles found by the algorithm is at least $k-9$ (at least $k-10$ paths form a cycle either with p_i or with p). Among these cycles are all the cycles containing paths $p_1, p_2, \ldots p_i$ and p.

In the case when there is no pair of paths corresponding to e that form a 6-cycle, the algorithm checks $9k$ pairs of cycles and reports that there is no cycle formed by paths corresponding to e. □

Corollary 6. *There is an algorithm which lists all the cycles formed by paths corresponding to the same edges in G_3. Its time complexity is $O(n + \#_c)$. The algorithm finds the first cycle in $O(n)$ time.*

5.5 Longer Cycles

From Theorem 1 we see that all cycles of length 7 are formed by two 2-paths and one 3-path corresponding to edges $e_1, e_2 \in G_2$ and to $e_3 \in G_3$, respectively. To find those edges we should search for triangles in graph $G_2 \cup G_3$. Unfortunately we were unable to show that G_3 is $O(1)$-degenerate.

6 Another Algorithm for Finding C_5 in Planar Graphs

In this section we show another linear algorithm for finding C_5's in planar graphs. This algorithm is more simple than the algorithm from section 5. It also seems to be faster in practice (see Section 7). Another important feature is that the new algorithm returns each cycle exactly once (additional sorting is not needed). The algorithm can be easily modified to count the number of 5-cycles in linear time. We extend the approach of Chiba and Nishizeki [3] to the case of cycles of length 5. The algorithm is presented below.

Lemma 2 (Chiba, Nishizeki [3]). *Let G be a graph of m edges with arboricity a. Then*

$$\sum_{u-v \in E} \min\{d(u), d(v)\} \leq 2am$$

Proposition 5. *Algorithm 3 correctly lists all 5-cycles in d-degenerate n-vertex graph G in $O(d^2 \cdot m + \#_c)$ time, where $\#_c$ denotes the number of 5-cycles in G. Moreover, the algorithm finds first cycle (or reports that there is no one) in $O(d^2 \cdot m)$ time.*

Proof. It is straightforward that for every vertex v_i the algorithm lists all the 5-cycles containing v_i. This is achieved by finding edges joining vertices distant by 2 from v_i (statement 11). We will focus on the time complexity.

Algorithm 3 Listing all cycles of length 5 in d-degenerate graph

```
1: Direct the edges of G producing d-oriented directed graph G'.
2: Sort the vertices of G in such a way that d(v_1) ≥ d(v_2) ≥ ... d(v_n).
3: for all v ∈ V do U(v) ← ∅
4: for i ← 1 to n do
5:    {finding all 5-cycles containing v_i}
6:    for all u ∈ N(v_i) do
7:       for all w' ∈ N(u) − {v_i} do
8:          U(w') ← U(w') ∪ {u}
9:       end for
10:   end for
11:   for all u ∈ N(v_i) do
12:      for all w ∈ N(u) − {v_i} do
13:         for all w → x ∈ E(G') do
14:            if x ≠ u then
15:               for all y ∈ U(x) do
16:                  if y ∉ {u, w} then
17:                     Print out the cycle v_i u w x y.
18:                  end if
19:               end for
20:            end if
21:         end for
22:      end for
23:   end for
24:   for all w such that U(w) ≠ ∅ do
25:      U(w) ← ∅
26:   end for
27:   Delete v_i from G.
28: end for
```

Statement 1 of the algorithm can be easily implemented to work in linear time. It suffices to successively choose a vertex v of degree at most d, make the edges incident with v outgoing from v in G' and mark them as deleted in G.

Consider a vertex v_i and its neighbor u at the beginning of the "for" loop. Let G_0 be the input graph and let G be the current version of G_0. Then $d_G(u) \leq d_{G_0}(u) = \min\{d_{G_0}(u), d_{G_0}(v)\}$. Subsequently, from Lemma 2, the variable w is assigned at most $4dm$ times in statement 10 (G has arboricity at most $2d$). Since G' is d-oriented the variable x is assigned at most $4d^2m$ times (statement 11). Moreover, among all checked sequences of vertices (v, u, w, x, y) at most $8d^2m$ sequences does not form a cycle. That ends a proof. □

Algorithm 3 can be easily adapted to count the number of 5-cycles in linear time. In this case $U(w)$ will denote the number of 2-paths from v_i to w. In algorithm 3 whenever a 3-path $p = v_i$–u–$w \to x$ is found we have to find those 2-paths v_i–x stored in $U(x)$ which do not intersect with p (statement 14). We can easily get rid of this time-consuming fragment. Before checking all the 3-paths of the form v_i–$u\ldots$ we decrease the number $U(x)$ for all neighbors of u. As a

Algorithm 4 Counting cycles of length 5 in d-degenerate graph

```
1: #_c ← 0
2: Direct the edges of G producing d-oriented directed graph G'.
3: Sort the vertices of G in such a way that d(v_1) ≥ d(v_2) ≥ ... d(v_n).
4: for all v ∈ V do U(v) ← 0
5: for i ← 1 to n do
6:    for all u ∈ N(v_i) do
7:       for all w' ∈ N(u) − {v_i} do U(w') ← U(w') + 1
8:    end for
9:    for all u ∈ N(v_i) do
10:      for all w ∈ N(u) − {v_i} do U(w) ← U(w) − 1
11:      for all w ∈ N(u) − {v_i} do
12:         for all w → x ∈ E(G') do
13:            if x ≠ u then
14:               if w → v_i ∈ E(G') or v_i → w ∈ E(G') then
15:                  #_c ← #_c + U(x) − 1
16:               else
17:                  #_c ← #_c + U(x)
18:               end if
19:            end if
20:         end for
21:      end for
22:      for all w ∈ N(u) − {v_i} do U(w) ← U(w) + 1
23:   end for
24:   for all w such that U(w) ≠ 0 do U(w) ← 0
25:   Delete v_i from G.
26: end for
```

result, when we find a 3-path $p = v_i$–u–$w \to x$ there is exactly $U(x)$ 2-paths from v_i to x that do not intersect with u. Then if w is adjacent to v_i exactly one of these paths intersects p (in vertex w). If w is not adjacent to v_i none of these paths intersects p. Thus in the first case we find $U(x) - 1$ paths that form 5-cycles with p and in the latter case $U(x)$ ones. See Algorithm 4 for details.

Proposition 6. *Algorithm 4 correctly counts cycles of a given n-vertex planar graph in $O(n)$ time.*

7 Experimental Results

Since the motivation of this paper was to give algorithms that can be used in practice in this section we show some of the experimental results concerning presented methods. The algorithms were tested on random 10^5-vertex triangulation generated by Donald Knuth's *Stanford GraphBase*. The computations were done on a Pentium II 500 MHz computer with 2 GB of memory. Below you can see time results for almost all the algorithms presented in the paper as well as the number of cycles found ($\#_c$). The algorithm for 6-cycles searches for triangles in graph G_2 using the algorithm by Chiba and Nishizeki [3].

algorithm	C_3	C_4	C_5	C_6	C_5 (Alg. 3)	counting C_5 (Alg. 4)
time [s]	28.03	20.70	51.55	93.43	26.94	5.89
$\#_c$	201,136	315,333	689,705	2,065,391	689,705	689,705

Acknowledgments. We would like to thank Krzysztof Diks for valuable comments on the preliminary version of this paper. Thanks go also to other members of Algorithms and Complexity Group for fruitful discussions on these results during seminar.

References

1. N. Alon, R. Yuster, U. Zwick, *Color-coding: a new method for finding simple paths, cycles and other small subgraphs within large graphs*, Proc. 26th ACM Symp. on Theory of Computing, 326–335, 1994.
2. N. Alon, R. Yuster, U. Zwick, *Finding and counting given length cycles*, Proc. European Symp. Algorithms, 354–364, 1994.
3. N. Chiba, T. Nishizeki, *Arboricity and subgraph listing algorithms*, SIAM J. Computing, **14**, 210–223, 1985.
4. M. Chrobak, D. Eppstein, *Planar orientations with low out-degree and compaction of adjacency matrices*, Theoretical Computer Science, **86**, 243–266, 1991.
5. D. Eppstein, *Subgraph isomorphism in planar graphs and related problems*, J. Graph Algorithms and Applications **3(3)**, 1–27, 1999.
6. G. Fijavz, M. Juvan, B.Mohar, R. Skrekovski, *Planar graphs without cycles of specified lengths*, European J. Combin., **23**, 377–388, 2002.
7. £. Kowalik, M. Kurowski, *Short path queries in planar graphs in constant time*, Proc. 35th ACM Symp. on Theory of Computing, 143–148, 2003.
8. B. Monien, *How to find long paths efficiently*, Ann. Disc. Math., **25**, 239–254, 1985.
9. C. H. Papadimitriou, M. Yannakakis, *The clique problem for planar graphs*, Inform. Proc. Lett. **13**, 131–133, 1981.
10. D. Richards, *Finding short cycles in planar graphs using separators* J. Algorithms, **7**, 382–394, 1986.
11. C. Thomassen, *3-list coloring planar graphs of girth 5*, Journal of Combinatorial Theory, Series B, **64**, 101–107, 1995.

Complexity of Hypergraph Coloring and Seidel's Switching

Jan Kratochvíl

Department of Applied Mathematics and
Institute for Theoretical Computer Science*
Charles University
Malostranské nám. 25
118 00 Praha 1
Czech Republic
honza@kam.mff.cuni.cz

Abstract. Seidel's switching of a vertex in a given graph results in making the vertex adjacent to precisely those vertices it was nonadjacent before, while keeping the rest of the graph unchanged. Two graphs are called switching equivalent if one can be transformed into the other one by a sequence of Seidel's switchings. We consider the computational complexity of deciding if an input graph can be switched into a graph having a desired graph property. Among other results we show that switching to a regular graph is NP-complete. The proof is based on an NP-complete variant of hypergraph bicoloring that we find interesting in its own.

1 Seidel's Switching

We consider undirected graphs without loops or multiple edges. The vertex set and the edge set of a graph G are denoted by V_G and E_G, respectively. Edges are considered as two-element subsets of the vertex set. A graph is called r-regular if every vertex has degree r. A bipartite graph is called (k,r)-biregular if every vertex in one bipartition class has degree k and every vertex in the other bipartition class has degree r. A hypergraph is a family of subsets (called hyperedges) of its vertex set. A hypergraph is called k-regular if every vertex belongs to exactly k hyperedges, and it is called k-uniform if every hyperedge has size k.

Let G be a graph. *Seidel's switching* of a vertex $v \in V_G$ results in a graph called $S(G,v)$ whose vertex set is the same as of G and the edge set is the symmetric difference of E_G and the full star centered in v, i.e.,

$$V_{S(G,v)} = V_G$$

$$E_{S(G,v)} = (E_G \setminus \{xv : x \in V_G, xv \in E_G\}) \cup \{xv : x \in V_G, x \neq v, xv \notin E_G\}.$$

Graphs G and H are called *switching equivalent* if G can be transformed into a graph isomorphic to H by a sequence of Seidel's switches. It can be easily

* Project LN00A056 supported by the Ministry of Education of the Czech Republic.

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 297–308, 2003.

seen that only the parity of the number of times a particular vertex is switched matters. Denote $A \subseteq V_G$ the set of vertices which are switched odd number of times. The resulting switched graph is then

$$S(G, A) = (V_G, E_G \div \{xy : x \in A, y \in V_G \setminus A\}),$$

and G is switching equivalent to H if and only if H is isomorphic to $S(G, A)$ for some $A \subseteq V_G$ ($\div$ denoting the symmetric difference of sets).

The concept of Seidel's switching was introduced by the Dutch mathematician J.J. Seidel in connection with symmetric structures, often of algebraic flavor, such as systems of equiangular lines, strongly regular graphs, or the so called two-graphs. For more structural properties of two-graphs cf. [13,14,15].

Colbourn and Corneil [1] (and independently but later Kratochvil et al. [10]) proved that deciding if two graphs are switching equivalent is an isomorphism complete problem. This observation can be extended to the fact that deciding if an input graph is switching equivalent to its complement is also isomorphism complete [11].

Several authors asked the question of how difficult it is to decide if a given graph is switching equivalent to a graph having some prescribed property (this property becomes the parameter of the problem). It was noted in [10], and later also in [3], that there is no correlation between the complexity of this problem and the complexity of the property itself. For instance, it is shown that every graph is switching equivalent to a graph containing a hamiltonian path, and it is polynomial to decide if a given graph is switching equivalent to a graph containing a hamiltonian cycle (while both deciding if the graph itself contains a hamiltonian path or cycle are NP-complete problems). This result was extended in [2] to graph pancyclicity. Hage et al. further show in [5] that switching equivalence to Euler graphs and to bipartite graphs are polynomially solvable problems.

An elegant characterization by forbidden induced subgraphs for the property "not being perfect" is proven in [7,8]. This characterization yields a polynomial time decision algorithm. (Equivalently, this provides a characterization of graphs such that every switching equivalent graph is perfect.) Also graphs such that all switching equivalent ones contain perfect dominating sets (perfect codes) can be characterized by a finite set of forbidden induced subgraphs and hence are recognizable in polynomial time [9].

Another direction is avoiding forbidden induced subgraphs. It is proved in [10] that deciding if a given input graph can be switched to a P_3-free graph (i.e., a graph not containing and induced copy of the path on 3 vertices) is polynomially solvable. This means deciding if the input graph is switching equivalent to the disjoint union of complete graphs. R. Hayward [6] (and later Hage et al. [5]) showed that deciding switching equivalence to triangle-free graphs is also polynomial. Somewhat surprisingly, this result is a core of the polynomial algorithm for recognizing P_3-structures of graphs [6]. (The H-structure of a graph is the hypergraph on the same vertex set whose hyperedges are those subsets of vertices that induce graphs isomorphic to H.) The question of recognizing P_3-structures

was motivated by the class of perfect graphs and P_4-structures, since induced P_4's play an important role for perfectness. Recognition of P_4-structures is still an open problem.

It was also announced (but not proved) in [10] that deciding switching equivalence to a regular graph is NP-complete. Given that recognizing regular graphs is linear, this result may seem somewhat unexpected. In this paper, we present a proof significantly simpler than the original (and so far unpublished) one. The proof is based on NP-hardness of a special hypergraph bicoloring problem (Theorem 1) which is proven in the next section. We believe that this result is interesting in its own, also for the consequence of balancing biregular bipartite graphs. In the last section we conclude with observations on switching to graphs of bounded minimum degree.

2 Bicoloring Uniform Regular Hypergraphs

Satisfiability of Boolean formulas is considered one of the basic and most useful (for further consequences) NP-complete problems. Special variants as 3-SAT, 1-in-3-SAT etc. are known to be NP-complete, and are often used in deriving other hardness results. A remarkable theorem of Schaefer [12] gives a complete characterization of predicates that determine polynomially solvable or NP-complete instances. Schaefer, however, does not take into account further regularity requirements on the input formula.

Let us have a closer look at formulas without negative literals. We can view such a formula as a hypergraph – vertices being the variables and hyperedges being the clauses. A truth valuation of the variables is then nothing else than a coloring of the vertices by two colors. A clause is not-all-equal satisfied if and only if the corresponding hyperedge is colored so that it is not monochromatic (contains at least one vertex colored true and at least one vertex colored false). Hence Not-All-Equal-SAT of formulas without negations is exactly the hypergraph bicoloring problem, well known to be NP-complete.

Another well known NP-complete problem is 1-in-3-SAT where we require that every clause (all of which are of size 3) contains exactly one true variable. In the hypergraph setting this problem asks if the vertices can be colored by two colors so that every hyperedge contains exactly one vertex colored by a favored color. In the dual setting this is equivalent to the question if the hyperedges can be colored by two colors so that every vertex belongs to exactly one hyperedge of a favored color, ergo, if one can choose a set of hyperedges which cover every vertex exactly once, the problem known as EXACT COVER. This problem is NP-complete even if every hyperedge has size 3 and every vertex belongs to exactly 3 hyperedges [4].

The aim of this section is to further explore regular instances of the hypergraph bicoloring problem. Recall that a hypergraph is called *q-regular* if every vertex belongs to exactly q hyperedges, and *m-uniform* if every hyperedge has size m. Thus the bipartite incidence graph of a q-regular m-uniform hypergraph is (q, m)-biregular. We say that an m-uniform hypergraph is *(k-in-m)-colorable*

if the vertices can be colored by two colors (say black or white) so that every hyperedge contains exactly k white vertices. We prove the following result.

Theorem 1. *For every $q \geq 3$, $m \geq 3$ and $1 \leq k \leq m-1$, deciding (k-in-m)-colorability of q-regular m-uniform hypergraphs is NP-complete.*

Proof. The case $q = 3, k = 1, m = 3$ is the variant of EXACT COVER mentioned above. We use it to show NP-completeness of the case $q = 3$ (k, m arbitrary). Assume that we have a 3-regular 3-uniform hypergraph $H = (V, E)$ and we ask if it is 1-in-3-colorable. Let $n = |E|$ be the number of hyperedges of H. Without loss of generality we may assume that n is divisible by 3 (if not, we simply reject H as not 1-in-3-colorable).

First we construct a hypergraph B called *a block* as follows:

$$V_B = \{a_i : i = 1, 2, \ldots, m-1\} \cup \{x_1, x_2, x_3\}$$

$$E_B = \{R_1, R_2, R_3\}$$

where

$$R_i = (a_1, a_2, \ldots, a_{m-1}, x_i).$$

Straightforwardly, if B is (k-in-m)-colored then $x_i, i = 1, 2, 3$ have the same color, and vice versa, B can be (k-in-m)-colored with all x_i's having the same color.

Now blocks are combined into chains: Take $r = \frac{2}{3}n(k-1)+2k$ copies of B, say $B^j, j = 1, 2, \ldots, r$, with the x vertices denoted by x_i^j, $i = 1, 2, 3$, $j = 1, 2, \ldots, r$. Then identify $x_1^j \cong x_3^{j+1}$, $j = 1, 2, \ldots, r$ (counting in subscripts modulo r) and $x_2^{2j} \cong x_2^{2j-1}$ for $j = 1, 2, \ldots, \frac{r}{2}$ (note that r is even). These newly created x vertices must receive the same color in every k-in-m-coloring. Similarly, take $s = \frac{2}{3}n(m-k-2) + 2(m-k)$ copies of B, say $C^j, j = 1, 2, \ldots, s$, with the x vertices denoted by y_i^j, $i = 1, 2, 3$, $j = 1, 2, \ldots, s$. Then identify $y_1^j \cong y_3^{j+1}$, $j = 1, 2, \ldots, s$ and $y_2^{2j} \cong y_2^{2j-1}$ for $j = 1, 2, \ldots, \frac{s}{2}$ (note that s is again even). Again these newly created y vertices must receive the same color in every k-in-m-coloring. Since each of the x or y vertices is contained in exactly 2 hyperedges of $\bigcup_{j=1}^{r} B^j \cup \bigcup_{j=1}^{s} C^j$, we have created $n(k-1)+3k$ x-vertices and $n(m-k-2)+3(m-k)$ y-vertices. (The a vertices of the blocks B^j, C^j are all mutually distinct.)

For every hyperedge $e \in E$, add $k-1$ distinct x vertices and $m-k-2$ distinct y vertices to this edge, thus creating a hyperedge $\widetilde{e}$. Distinct hyperedges are completed by mutually distinct vertices. From the remaining $3k$ x-vertices and $3(m-k)$ y-vertices, create 3 auxiliary hyperedges A_1, A_2, A_3, each containing exactly k x-vertices and $m-k$ y-vertices. Denote this hypergraph $\widetilde{H} = (\widetilde{V}, \widetilde{E})$.

Obviously, every hyperedge of $\widetilde{E}$ has exactly m vertices and every vertex of $\widetilde{V}$ is contained in exactly 3 hyperedges. We claim that $(\widetilde{V}, \widetilde{E})$ is (k-in-m)-colorable if and only if (V, E) is (1-in-3)-colorable.

Suppose $\phi : \widetilde{V} \to \{\mathsf{black},\mathsf{white}\}$ is a coloring such that in every $\widetilde{e} \in \widetilde{E}$ there are exactly k white vertices. Since all x-vertices have the same color, and so do all the y-vertices, and since A_1 should have exactly k white vertices, necessarily $\phi(x) =$white and $\phi(y) =$black (or possibly $\phi(x) =$black and $\phi(y) =$white when $m = 2k$). If $\phi(x) =$white then in every clause $\widetilde{e}$ there are exactly $k-1$ white vertices among those added with respect to e, and hence e has exactly 1 white vertex, i.e., H is (1-in-3)-colorable. (If $m = 2k$ and $\phi(x) =$black then $\widetilde{e}$ would have $m-k-2 = k-2$ white vertices among the new ones, and e would have to contain exactly 2 white vertices, i.e. (V, E) would be (2-in-3)-colorable, and hence also (1-in-3)-colorable.)

The opposite implication, i.e., that $(\widetilde{V}, \widetilde{E})$ is (k-in-m)-colorable provided (V, E) is (1-in-3)-colorable, is obvious.

For the case of $q > 3$, we reduce from $q = 3$. Let $H = (V, E)$ be a hypergraph with every hyperedge of size m and every vertex occurring in 3 hyperedges. We construct $H' = (V', E')$ by taking q copies of H with vertex $v \in V$ called v^i in the i-th copy, for $i = 1, 2, \ldots, q$. Then for each vertex $v \in V$, we add $(q-3)$ copies of the block $B(q)$, with vertices $x_1, x_2, \ldots, x_q$ being identified with $v^1, v^2, \ldots, v^q$ (respectively). The block $B(q)$ has vertices $\{a_1, a_2, \ldots, a_{m-1}, x_1, x_2, \ldots, x_q\}$ and hyperedges $e_i = \{a_1, a_2, \ldots, a_{m-1}, x_i\}$ for $i = 1, 2, \ldots, q$. Every vertex a_j has degree q, every vertex x_j has degree one and in any (k-in-m)-coloring, the vertices $x_1, x_2, \ldots, x_q$ receive the same color, and both colors are feasible. This construction is just the garbage collection, since each copy of H already has to be (k-in-m)-colored. Now H' is q-regular and an (k-in-m)-coloring of H' is obtained by taking the same (k-in-m)-coloring of each copy of H.

Corollary 1. *For every $q \geq 3$, $m \geq 3$ and $1 \leq k \leq m-1$, k-in-m-SAT is NP-complete for formulas without negations with every variable occurring in exactly q clauses.*

Note that the bounds $q \geq 3$ and $m \geq 3$ are sharp, since the case $m = 2$ is a special instance of 2-SAT and hence polynomially solvable, while in the case of $q = 2$ the variables can be regarded as edges of an m-regular graph and the question is equivalent to deciding if an m-regular graph contains a spanning k-regular subgraph, a problem well known to be polynomially solvable by matching techniques.

Next we prove an auxiliary result on bipartite graphs which we will use in the proof of Theorem 2. We say that a graph is *balanced* if its vertices can be colored by two colors (say black and white) so that every vertex has the same number of white and black neighbors (obviously, a necessary condition is that every vertex has even degree). Deciding if a graph is balanced is a hard problem, even in a very restricted case.

Proposition 1. *For all $p, q \geq 2$, it is NP-complete to decide if a $(2p, 2q)$-biregular bipartite graph is balanced.*

Proof. Assume first that one of p, q is greater than 2, say $q \geq 3$. Consider a q-regular $2p$-uniform hypergraph H as an instance of (p-in-$2p$)-colorability, which has been just shown NP-complete. Duplicate every hyperedge, obtaining a $2q$-regular $2p$-uniform hypergraph H' whose (p-in-$2p$)-colorability status is the same as of H. We can color the hyperedges of H white and their duplicates black to balance the vertices of V_H, and hence the incidence graph of H' is balanced if and only if H' is (p-in-$2p$)-colorable.

Also in the case $p = q = 2$ we reduce from (2-in-4)-colorability of 3-regular hypergraphs, just the construction is slightly more technical. Given a 3-regular 4-uniform hypergraph $H = (V, E)$, we construct a hypergraph $H' = (V', E')$ by replacing every hyperedge of H by four new hyperedges and four new vertices as follows:

$$V' = V \cup \{v_e : v \in e \in E\}$$

$$E' = \{e_v : v \in e \in E\}$$

with the newly introduced hyperedges being

$$e_v = \{v\} \cup \{x_e : x \in e, x \neq v\}.$$

We claim that H' is (2-in-4)-colorable if and only if H is, and that the hyperedges of H' can be bicolored so that every vertex belongs to at least one hyperedge of each color.

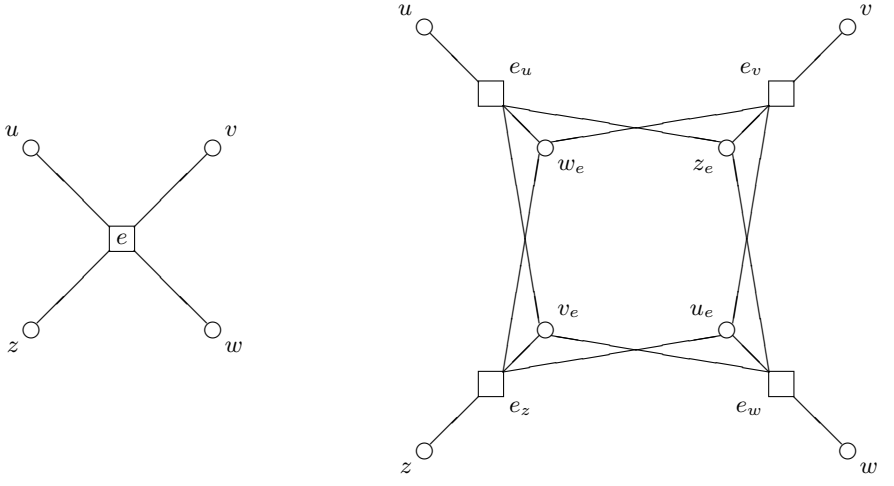


Fig. 1. Construction of H' from H, depicted as the incidence graphs B and B'.

Suppose H is (2-in-4)-colored by a coloring $\varphi : V \longrightarrow \{\mathsf{black}, \mathsf{white}\}$. We extend it to a (2-in-4)-coloring φ' of H' by setting

$$\varphi'(x_e) = \varphi'(x) = \varphi(x)$$

for every $x \in e \in E$. Consider a hyperedge $e \in E$. It has two black and two white vertices, say $e = \{u, v, w, z\}$ with $\varphi(u) = \varphi(v) =$ black and $\varphi(w) = \varphi(z) =$ white. Then e_v has two black vertices v, u_e and two white vertices w_e, z_e. Similarly for the other three edges e_u, e_w, e_z.

Next suppose that the vertices of H' are bicolored by a bicoloring φ so that every hyperedge has two white and two black vertices. We argue that the restriction of φ to V is a (2-in-4)-coloring of H. Consider a hyperedge $e \in E$, say again $e = \{u, v, w, z\}$. For any three of u_e, v_e, w_e, z_e, E' contains a hyperedge containing all these three vertices, and hence no three of u_e, v_e, w_e, z_e may have the same color. It follows that exactly two of u_e, v_e, w_e, z_e are black and exactly two are white. But then $\varphi(x_e) = \varphi(x)$ for every $x \in e$, and e has two black and two white vertices as claimed.

Now we show how to color the hyperedges of H'. Let $B = (V \cup E, \{xe : x \in e \in E\})$ be the bipartite incidence graph of H. We orient the edges of B so that every V-vertex is incident with at least one ingoing and at least one outgoing edge, and each E-vertex has indegree and outdegree two. (To find such an orientation, add to B a dummy vertex adjacent to all V-vertices. This enriched graph has all vertices of even degrees and hence allows an Eulerian walk. Orienting the edges along this walk yields an orientation in which every vertex of $V \cup E$ has indegree and outdegree two.) Color $e_v \in E'$ black if the edge ve of B is oriented from v to e, and color it white otherwise. If e, f, g are the hyperedges of H that contain a vertex $v \in V$, at least one of the edges ve, vf, vg is oriented out of v and at least one is oriented toward v, and hence at least one of the hyperedges e_v, f_v, g_v of H' is colored white and at least one is colored black. For the new vertices of $V' \setminus V$, consider $e = \{u, v, w, z\} \in E$. Since e has indegree and outdegree two (in the orientation of B), exactly two of the hyperedges e_u, e_v, e_w, e_z are black and exactly two are white. Thus each of the vertices v_e, u_e, w_e, z_e is incident with at least one black and at least one white hyperedge.

Finally we take four disjoint copies of H' with corresponding vertices and hyperedges indexed $v^i, e^i, i = 1, 2, 3, 4$, and we add transversal hyperedges

$$t_v = \{v^1, v^2, v^3, v^4\}$$

for all $v \in V'$. Obviously, the hypergraph $\widetilde{H}$ constructed in this way is 4-regular and 4-uniform, and hence its bipartite incidence graph $\widetilde{B}$ is 4-regular. Since $\widetilde{H}$ contains H' as an induced subhypergraph, $\widetilde{H}$ may be (2-in-4)-colorable only if H' is. On the other hand, using a (2-in-4)-coloring of H' on two copies of H' and its reverse on the other two copies gives a (2-in-4)-coloring of $\widetilde{H}$. The hyperedges of $\widetilde{H}$ can always be colored so that every vertex is incident with two black and white hyperedges. Use the same coloring of edges (guaranteed by the previous claim) on each copy of H', and color the transversal hyperedges as forced (e.g., if v belongs to two black and one white hyperedge in the coloring of the edges of H', color the transversal hyperedge t_v white).

The bipartite incidence graph $\widetilde{B}$ of $\widetilde{H}$ is then balanced if and only if H is (2-in-4)-colorable, which concludes the proof.

Also in Proposition 1 the bounds on p and q are tight. For one can decide in polynomial time if a $(2, 2q)$-biregular bipartite graph is balanced for every q. The neighbors of the vertices of degree 2 can be properly colored if and only if the graph does not contain a cycle of length 2 modulo 4. And if this is satisfied, we contract these vertices into edges and decide via matching algorithm if the resulting $2q$-regular graph contains a spanning q-regular subgraph.

3 Complexity of Switching to Regular Graphs

Theorem 2. *Deciding if an input graph is switching equivalent to a regular graph is NP-complete.*

The theorem follows directly from Proposition 1 and the following result.

Proposition 2. *Let G be a $(2p, 2q)$-biregular bipartite graph with $n > 2(p + q)$ vertices. If $p \neq q$, then G is switching equivalent to a regular graph if and only if G is balanced. The resulting switched graph is then $\frac{n}{2}$-regular.*

Proof. Let $G = (V_1 \cup V_2, E)$ be a bipartite $(2p, 2q)$-biregular graph with bipartition classes V_1 and V_2, such that $\deg_G u = 2p$ for every $u \in V_1$ and $\deg_G u = 2q$ for every $u \in V_2$. Suppose that switching a subset A of vertices of G results in a D-regular graph. Denote

$$X = A \cap V_1, \quad Y = A \cap V_2,$$

$$Z = V_1 \setminus A, \quad W = V_2 \setminus A$$

and

$$x = |X|,\ y = |Y|,\ z = |Z|,\ w = |W|.$$

We will refer to X, Y, Z, W as *blocks*.

Denote by $\deg_G(u, S) = |N_G(u) \cap S|$ the number of neighbors of a vertex u in a set S. Consider a vertex $u \in X$. Its degree in the switched graph $S(G, A)$ is $\deg_{S(G,A)} u = \deg_G(u, Y) + z + w - \deg_G(u, W) = w + z + 2\deg_G(x, Y) - 2p$, since $\deg_G(u, Y) + \deg_G(u, W) = 2p$. If $S(G, A)$ is D-regular, $\deg_G(u, Y) = \frac{1}{2}(D + 2p - w - z)$ and hence all vertices in X have the same number of neighbors in Y. Similarly for the number of neighbors in W and for vertices in other blocks. It follows that the partition $V_G = X \cup Y \cup Z \cup W$ is regular, i.e., any two vertices from the same block have the same number of neighbors in each other particular block (adjacency considered in the original graph G). It follows that there exist numbers a, b, c, d, $-p \leq a, b \leq p$, $-q \leq c, d \leq q$, such that

$$\deg_G(u, Y) = p - a, \quad \deg_G(u, W) = p + a, \quad \deg_G(u, Z) = 0, \quad \text{for } u \in X,$$

$$\deg_G(u, Y) = p - b, \quad \deg_G(u, W) = p + b, \quad \deg_G(u, X) = 0, \quad \text{for } u \in Z,$$

$$\deg_G(u, X) = q - c, \quad \deg_G(u, Z) = q + c, \quad \deg_G(u, W) = 0, \quad \text{for } u \in Y,$$

$$\deg_G(u, X) = q - d, \quad \deg_G(u, Z) = q + d, \quad \deg_G(u, Y) = 0, \quad \text{for } u \in W.$$

This is illustrated in Figure 2, where the symbol close to each set denotes the size of the set and the symbol at the beginning of an edge ST close to S denotes the number of neighbors of a vertex from S in T, with $S, T \in \{X, Y, Z, W\}$.

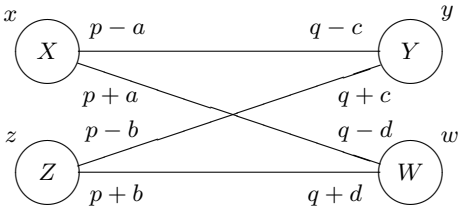


Fig. 2. Schematic diagram of the regular partition $V_G = X \cup Y \cup Z \cup W$.

The subgraph $G[X \cup Y]$ is $(p - a, q - c)$-biregular, and hence (by counting the number of its edges)

$$x(p - a) = y(q - c), \tag{1}$$

and similarly

$$w(q - d) = x(p + a), \tag{2}$$

$$y(q + c) = z(p - b), \tag{3}$$

$$z(p + b) = w(q + d). \tag{4}$$

Multiplying these four equations we get

$$xyzw(p - a)(p + b)(q + c)(q - d) = xyzw(p + a)(p - b)(q - c)(q + d). \tag{5}$$

We will show that all four blocks X, Y, Z, W are nonempty, i.e., that we can divide this equation by $xyzw \neq 0$. But first we list the equations for the degrees of vertices from each block in the switched graph, which should by assumption equal D. For instance, for $u \in X$, we have

$$\deg_{S(G,A)} u = p - a + z + w - (p + a) = z + w - 2a, \tag{6}$$

and similarly

$$u \in Y: \quad \deg_{S(G,A)} u = z + w - 2c, \tag{7}$$

$$u \in Z: \quad \deg_{S(G,A)} u = x + y + 2b, \tag{8}$$

$$u \in W: \quad \deg_{S(G,A)} u = x + y + 2d. \tag{9}$$

Once we prove that $x \neq 0 \neq y$, i.e., that X and Y are nonempty, it will follow from (6) and (7) that $a = c$. Similarly, from $zw \neq 0$ will follow that $b = d$.

To prove that $xyzw \neq 0$, assume that one of the blocks is empty. Say $X = \emptyset$ (by symmetry, this will cover all cases). Since $X \cup Z = V_1$, this means that $Z \neq \emptyset$. Also Y must be nonempty, since otherwise the switching set $A = X \cup Y$ would be empty and the switched graph $S(G, A) = G$ would not be regular. Hence $c = d = q$ and we deduce from (7,8) that

$$z + w - 2q = y + 2b. \tag{10}$$

From (3,4) we get

$$y = \frac{z(p-b)}{2q}, \quad w = \frac{z(p+b)}{2q}.$$

Substituting into (10) yields

$$z + \frac{z(p+b)}{2q} - 2q = \frac{z(p-b)}{2q} + 2b$$

which implies

$$(z - 2q)(b + q) = 0.$$

Now $z = 2q$ implies $y + w = 2p$ and $n = |V_G| = 2(p + q)$, contradicting the assumption of the theorem. Also $b = -q$ leads to a contradiction, since in this case $p + b = p - q \neq 0$ implies that $W \neq \emptyset$ and hence (9,8) give $b = q$.

From now on we will assume without loss of generality that $p < q$. We further note that $-p < a, b < p$. For $a = p$ would imply $c = q$, which is impossible since $c = a = p < q$, while $a = -p$ would imply $b = q > p$. Similarly, $b \neq -p, p$.

Dividing (5) by $xyzw$ and using the fact that $c = a$ and $d = b$ we get

$$(p - a)(p + b)(q + a)(q - b) = (p + a)(p - b)(q - a)(q + b)$$

which yields

$$\frac{(p-a)(q+a)}{(p+a)(q-a)} = \frac{(p-b)(q+b)}{(p+b)(q-b)}.$$

Consider the function $f(b) = \frac{(p-b)(q+b)}{(p+b)(q-b)}$ as a function of b. Its derivative is

$$f'(b) = \frac{2(p-q)(pq+b^2)}{(p+b)^2(q-b)^2} < 0$$

and so $f(b)$ is decreasing in the interval $(-p, q)$. Hence $b = a$ is the only solution of $f(a) = f(b)$ in the interval $b \in (-p, p)$.

Therefore, (1–4) yield

$$y = x\frac{p-a}{q-a}, \quad z = x\frac{q+a}{q-a}, \quad w = x\frac{p+a}{q-a}$$

and substituting these into

$$w + z - x - y = 4a$$

(which is obtained by comparing (7) and (8)) gives

$$4a(x - q + a) = 0.$$

Now $x = q - a$ would imply $y = p - a$, $z = q + a$ and $w = p + a$, hence $n = x+y+z+w = 2(p+q)$ contradicting the assumption. Thus we derive $a = 0$ as the only possibility, yielding

$$\deg_G(u, A) = \deg_G(u, V_G \setminus A) = p \text{ for every } u \in V_1 = X \cup Z$$

and

$$\deg_G(u, A) = \deg_G(u, V_G \setminus A) = q \text{ for every } u \in V_2 = Y \cup W.$$

This shows that G is balanced by coloring the vertices of A white and the vertices of $V_G \setminus A$ black.

On the other hand, if G is balanced by a coloring $\phi : V_G \to \{\mathsf{black}, \mathsf{white}\}$, then switching the set of white vertices yields an $\frac{n}{2}$-regular graph. To see this, denote $A = \{u \in V_G : \phi(u) = \mathsf{white}\}$ and set $X = A \cap V_1, \quad Y = A \cap V_2, \quad Z = V_1 \setminus A, \quad W = V_2 \setminus A$. Since $G[X \cup Y]$ and $G[Z \cup Y]$ are both (p, q)-biregular, we have $p|X| = q|Y| = p|Z|$ and hence $|X| = |Z|$. Similarly, $|Y| = |W|$. For any vertex $u \in X$, we have $\deg_{S(G,A)} u = p + |Z| + |W| - p = |Z| + |W| = \frac{|V_G|}{2}$, and by symmetry, $\deg_{S(G,A)} u = \frac{|V_G|}{2}$ for every $u \in V_G$.

4 Switching to Graphs of Bounded Minimum Degree

One can easily see that for every fixed k, one can decide switchability to a k-regular graph in polynomial time. We frame this fact in a more general observation.

Proposition 3. *Let $\mathcal{A}$ be an isomorphism-closed class of graphs such that every graph of $\mathcal{A}$ contains a vertex of degree at most k, for some fixed number k. If $\mathcal{A}$ can be recognized in polynomial time, then it can also be decided in polynomial time if an input graph is switching equivalent to a graph belonging to $\mathcal{A}$. More precisely, this can be decided in time $O(n^{k+3}p(n))$, where $p(n)$ is the worst case time complexity of recognizing graphs of $\mathcal{A}$.*

Proof. If the input graph is switched to a graph of $\mathcal{A}$, one of its vertices is switched to a vertex of degree at most k. There are n possible choices of which vertex will this be. For each such vertex, there are $1+(n-1)+\binom{n-1}{2}+\ldots+\binom{n-1}{k} = O(n^k)$ possible sets of at most k neighbors for that vertex (in the switched graph). So for each vertex u and for every choice of a set S of at most k other vertices, we switch the set $N_G(u) \div S$ and check if the resulting graph is in $\mathcal{A}$. Each switching can only affect $O(n^2)$ edges, hence the upper bound.

Corollary 2. *Switching equivalence to a tree, acyclic graph, planar graph, outerplanar graph, graph of bounded genus, graph of bounded tree-width, k-regular graph (for fixed k) are all polynomially decidable problems.*

One can argue about the necessity of the assumption that graphs of $\mathcal{A}$ are polynomial time recognizable as follows. It is NP-complete to decide if an input graph of maximum degree 4 is 3-colorable. For any graph G, the disjoint union of G and a triangle is switchable to a 3-colorable graph if and only if G itself is 3-colorable. Hence deciding if an input graph is switchable to a 3-colorable graph of maximum degree at most 4 (as well as deciding switchability to 3-colorable graphs) is NP-complete.

References

1. C.J. Colbourn, D.G. Corneil: On deciding switching equivalence of graphs, Discrete Appl. Math. 2 (1980) 181–184
2. A. Ehrenfeucht, J. Hage, T. Harju, G. Rozenberg: Pancyclicity in switching classes, Inform. Process. Letters 73 (2000) 153–156
3. A. Ehrenfeucht, J. Hage, T. Harju, G. Rozenberg: Complexity Issues in switching of graphs, in: Theory and Application to Graph Transformations, LNCS 1764, Springer-Verlag, 2000, pp. 59–70
4. T.F. Gonzales: Clustering to minimize the maximum intercluster distance, Theoret. Comput. Sci. 38 (1985), 293–306
5. J. Hage, T. Harju, E. Welzl: Euler graphs, triangle-free graphs and bipartite graphs in switching classes, in: Proceedings ICGT 2002, LNCS 2505, Springer-Verlag, 2002, pp. 148–160
6. R.B. Hayward: Recognizing P-3-structure: A switching approach, J. Combin. Th. Ser. B 66 (1996) 247–262
7. A.Hertz: On perfect switching classes, Discr. Appl. Math. 89 (1998) 263–267
8. A.Hertz: On perfect switching classes, Discr. Appl. Math. 94 (1999) 3–7
9. J. Kratochvil: Perfect codes and two-graphs, Comment. Math. Univ. Carolin. 30 (1989), 755–760
10. J. Kratochvíl, J. Nešetřil, O. Zýka: On the computational complexity of Seidel's switching, Combinatorics, graphs and complexity, Proc. 4th Czech. Symp., Prachatice 1990, Ann. Discrete Math. 51 (1992) 161–166
11. J. Kratochvil, M. Rosenfeld: unpublished
12. T. J. Schaefer: The complexity of satisfiability problems, Proc. of the Tenth Annual ACM Symposium on Theory of Computing (STOC), 1978, 216–226
13. J.J. Seidel: A survey of two-graphs, Teorie combinatorie, Atti Conv. Lincei, Vol 17, Academia Nazionale dei Lincei, Rome, 1973, pp. 481–511
14. J.J. Seidel, D.E. Taylor: Two-graphs, a second survey, Algebraic methods in graph theory, Vol. II, Conf. Szeged 1978, Colloq. Math. Janos Bolyai 25 (1981) 689–711
15. J.J. Seidel: More about two-graphs, Combinatorics, graphs and complexity, Proc. 4th Czech. Symp., Prachatice 1990, Ann. Discrete Math. 51 (1992) 297–308

Feedback Vertex Set and Longest Induced Path on AT-Free Graphs

Dieter Kratsch[1], Haiko Müller[2], and Ioan Todinca[3]

[1] LITA, Université de Metz, 57045 Metz Cedex 01, France
[2] School of Computing, University of Leeds, Leeds, LS2 9JT, United Kingdom
hm@comp.leeds.ac.uk
[3] LIFO, Université d'Orléans, BP 6759, 45067 Orléans Cedex 2, France

Abstract. We give a polynomial time algorithm to compute a minimum (weight) feedback vertex set for AT-free graphs, and extending this approach we obtain a polynomial time algorithm for graphs of bounded asteroidal number.
We also present an $O(nm^2)$ algorithm to compute a longest induced path in AT-free graphs.

1 Introduction

A *feedback vertex set* (fvs) of an undirected graph $G = (V, E)$ is a set W containing at least one vertex of each cycle of G. Hence $W \subseteq V$ is a feedback vertex set of $G = (V, E)$ if and only if $G - W$ is a forest. The FEEDBACK VERTEX SET (FVS) problem is, given a graph G and an integer k, whether G contains a feedback vertex set of size at most k.

An *induced path* $P = (W, F)$ of a graph G is a tree $G[W] = P$ of maximum degree two. The LONGEST INDUCED PATH (LIP) problem asks whether the input graph contains an induced path of given length.

Both FVS and LIP are NP-complete [11]. Both problems are definable in a logical language called LinEMSOL($\tau_{1,L}$). Thus their weighted versions are linear time solvable on graphs of bounded clique-width [9]. Hence both problems are linear-time solvable for graphs of bounded treewidth, cographs, distance-hereditary graphs, etc.

There are very few papers on algorithms for LIP. A very interesting one by Gavril studies the weighted version of LIP and obtains polynomial time algorithms for various graph classes among them k-bounded hole families of graphs, interval-filament graphs, polygon-circle graphs, circle graphs and circular-arc graphs [12].

On the contrary, there is a lot of research on algorithms for FVS. The problem is fixed-parameter tractable [10]. There has been a series of papers on approximation algorithms on minimum (weight) FVS resulting in a 2-approximation algorithm for minimum weight FVS [1]. Clearly FVS had been and still is a benchmark covering problem. It is also important since FVS approximation algorithms are used as subroutines in approximation algorithms for other typically network analysis problems. See e.g. [14] where a PTAS for FVS on planar graphs is constructed as a by-product.

Concerning the complexity of FVS on special graph classes the following is known. FVS is NP-complete for bipartite and for planar graphs. A minimum (weight) fvs can be computed by a polynomial time algorithm for interval graphs ($O(n + m)$) [18],

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 309–321, 2003.

permutation graphs ($O(nm)$) [16], cocomparability graphs ($O(n^2m)$) [17] and convex bipartite graphs ($O(n^2m)$) [17]. Furthermore FVS is mentioned as polynomial time solvable for chordal graphs and circular-arc graphs in [19]. J. Spinrad points out in [19] that the complexity of FVS on AT-free graphs in unknown.

Three independent vertices form an *asteroidal triple* (AT for short) if any two of them are connected by a path avoiding the neighbourhood of the third vertex. Graphs without asteroidal triples are called *AT-free*. AT-free graphs contain cocomparability graphs, permutation graphs and interval graphs. In an important paper Corneil, Olariu and Stewart were the first to study structural properties of AT-free graphs [8]. In the last ten years various papers on the complexity of NP-complete problems for AT-free graphs have been published (see e.g. [5,6,7,15]). Inspecting their results one finds out that there are LinEMSOL($\tau_{1,L}$)-definable problems that remain NP-complete on AT-free graphs. CLIQUE is NP-complete on AT-free graphs [6] and MINIMUM DOMINATING CLIQUE remains NP-complete even on cocomparability graphs [3]. This provides another motivation to study FVS on AT-free graphs.

Our paper is organised as follows. In Section 2 we summarise the main ingredients of an approach developed in [6] to design polynomial time algorithms for problems on AT-free graphs (see also [5,7]). In Section 3 we demonstrate the approach for LIP on AT-free graphs (which is much easier than for FVS) and obtain an $\mathcal{O}(nm^2)$ time algorithm. In Section 4 we present a polynomial time algorithm for FVS on AT-free graphs. In Section 5 we show how to extend the approach to graphs of bounded asteroidal number and we obtain a polynomial time algorithm for FVS.

2 Preliminaries

We use standard graph theory notation throughout. For nonadjacent vertices u and v we denote by $C_G(u,v)$ the connected component of $G - N[v]$ containing u. As usual we omit the index if this does not cause confusion. By a connected component we mean a maximal connected induced subgraph of a graph. Thus we denote a connected component of a graph G with vertex set C by $G[C]$. Note that $w \in C(u,v)$ if and only if $u \in C(w,v)$. Using this notation, the vertices u, v and w form an AT if and only if $u \in C(v,w)$, $v \in C(w,u)$ and $w \in C(u,v)$.

2.1 Blocks and Intervals

Let $G = (V,E)$ be a connected graph and $v \in V$. The pair (v,C) is called a *block* of G if $G[C]$ is a connected component of $G - N[v]$. Thus removing a closed neighbourhood $N[v]$ for some vertex v from a graph G this graph decomposes into the connected components of $G - N[v]$ and it also decomposes into blocks.

Let u and v be two nonadjacent vertices of G and let $B = C(v,w) \cap C(w,v)$. The triple (v,B,u) is called an *interval* of G. Obviously (v,B,u) is an interval if and only if (u,B,v) is an interval. Notice that B may be empty.

Removing a closed neighbourhood from a block decomposes it into blocks and exactly one interval. Formally, consider a block (v,C) and let $u \in C$. Removing $N[u]$

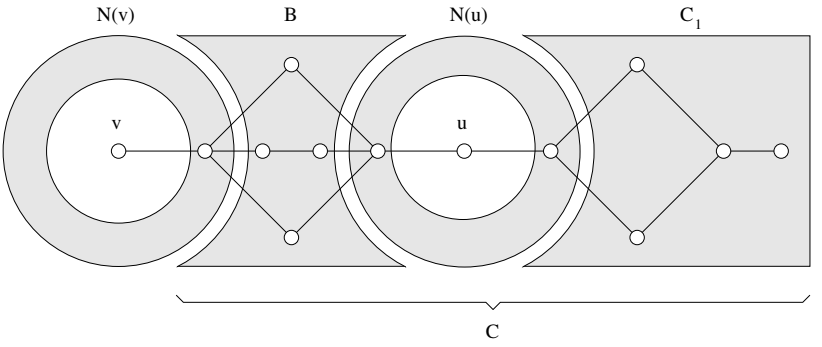


Fig. 1. Blocks and intervals

from $G[C]$ partitions C into $B, C_1, \ldots, C_p$, where (v, B, u) is an interval and (u, C_i) are blocks of G (see Figure 1).

The LIP and FVS problems are solved by dynamic programming on blocks.

2.2 Representing Neighbourhoods

Definition 1. *Consider a vertex v of the graph G and let S be an independent set contained in $N(v)$. We say that a set $R \subseteq S$ is a* representation of S w.r.t. v *if $N(S) \setminus N(v) = N(R) \setminus N(v)$ and R is of minimum size for this property.*

The following lemma shows that in AT-free graphs, each representation of an independent set $S \subseteq N(v)$ contains at most four vertices.

Lemma 1. *Let $G[C]$ be a connected component of $G - N[v]$ where $G = (V, E)$ is an AT-free graph, $v \in V$ and $C \subset V$. The neighbourhood of an independent set $S \subseteq N(v)$ in C can be represented by a subset of size at most two, i.e. there is a set $R_C \subseteq S$ with $|R_C| \leqslant 2$ and $N(R_C) \cap C = N(S) \cap C$. The outside neighbourhood of S can be represented by a subset of size at most four, i.e. there is a set $R \subseteq S$ with $|R| \leqslant 4$ and $N(R) \setminus N(v) = N(S) \setminus N(v)$.*

Proof. We consider three pairwise nonadjacent vertices $r, s, t \in N(v)$ with private neighbours $a, b, c \in C$. That is, $a \in (C \cap N(r)) \setminus (N(s) \cup N(t))$, $b \in (C \cap N(s)) \setminus (N(r) \cup N(t))$ and $c \in (C \cap N(t)) \setminus (N(r) \cup N(s))$. If a, b and c induce a triangle in G then r, s and t form an AT of G—a contradiction. Otherwise let $ab \notin E$. Now a, b and v form an AT of G because $G[C]$ is connected—a contradiction again. Hence an independent triple of vertices in $N(v)$ cannot have private neighbours in the same component of $G - N[v]$.

Next let a, b and c be vertices in three different connected components of $G - N[v]$ having private neighbours $r, s, t \in S$. That is $r \in (S \cap N(a)) \setminus (N(b) \cup N(c))$, $s \in (S \cap N(b)) \setminus (N(a) \cup N(c))$ and $t \in (S \cap N(c)) \setminus (N(a) \cup N(b))$. Clearly $\{a, b, c\}$ is an independent set of G and hence these vertices form an AT of G. Since G is AT-free at most two components of $G - N[v]$ contain vertices having private neighbours in $N(v)$. □

3 Longest Induced Path

3.1 Analysis

Lemma 2. *Let $P = (W, F)$ be an induced path of an AT-free graph G. Then for every vertex $v \in W$ and every component $G[C]$ of $G - N[v]$ the graph $P[C]$ is connected.*

Proof. Conversely, let (t, u, v, w, x) be a subpath of P with $t, x \in C$. Since P is induced the vertices t and x are non-adjacent. Since $G[C]$ is connected there is a path in $G[C]$ joining t and x. This is, t, v and x form an AT of G—a contradiction. □

Thus any longest induced path passing through v will enter at most two blocks (v, C) and (v, C'), each of them at most once. Let $\ell(u, v, w, x)$ denote the length of a longest induced path $(u, v, w, x, \dots)$ in $G[N[v] \cup C(x, v)]$. These values represent partial solutions for the block $(v, C(x, v))$ of G. Note that we store several partial solutions for each block (v, C), namely the longest extensions of all possible paths (u, v, w).

By $\mathcal{P}_l(G)$ we denote the set of induced paths of length l in G and define

$$\operatorname{lip}(G) = \max\{l : \mathcal{P}_l(G) \neq \varnothing\}.$$

If $G = (V, E)$ contains at least one path of length three the values $\ell(u, v, w, x)$ fulfil the following recursion for all $(u, v, w, x) \in \mathcal{P}_3(G)$.

$$\ell(u, v, w, x) = \begin{cases} 3 & \text{if there is no } y \in C(x, v) \text{ such that } (u, v, w, x, y) \in \mathcal{P}_4(G) \\ 1 + \max\{\ell(v, w, x, y) : & \\ \quad y \in C(x, v), u \notin C(y, w), (u, v, w, x, y) \in \mathcal{P}_4(G)\} & \text{otherwise} \end{cases} \tag{1}$$

$$\operatorname{lip}(G) = \max\{\ell(u, v, w, x) : (u, v, w, x) \in \mathcal{P}_3(G)\} \tag{2}$$

3.2 Algorithm

The above recurrence translates to the following recursive algorithm on $G = (V, E)$.

Step 1 Compute $\mathcal{P}_3(G)$. If $\mathcal{P}_3(G) = \varnothing$ then if $E = \varnothing$ then $\operatorname{lip}(G) = 0$ else if each component of G is complete then $\operatorname{lip}(G) = 1$ else $\operatorname{lip}(G) = 2$.

Step 2 Create a list of all blocks (v, C) of G and sort by $|C|$.

Step 3 For each block (v, C) and each $(u, v, w, x) \in \mathcal{P}_3(G)$ with $x \in C$ compute $\ell(u, v, w, x)$ according to Equation (1) using the values $\ell(v, w, x, y)$ computed earlier in this step.

Step 4 Determine $\operatorname{lip}(G)$ by Equation (2) using the values $\ell(u, v, w, x)$ computed in step 3.

Theorem 1. *There is an $\mathcal{O}(nm^2)$ algorithm to compute the maximum length of an induced path in AT-free graphs.*

Proof. First we show that our algorithm indeed computes $\mathrm{lip}(G)$for every graph $G = (V, E)$. This is obviously true if $\mathcal{P}_3(G) = \varnothing$. We can modify our algorithm such that it stores a path $(u, v, w, x, \dots)$ of length $\ell(u, v, w, x)$ for each such value computed. If one of these paths is not an induced one it contains a subpath corresponding to an induced cycle. The shortest such cycle has length at most 5 because longer induced cycles contain ATs. But in Equation (1) we check that every sequence of five consecutive vertices induces a path and not a cycle. Hence G contains an induced path of length l if our algorithm computes $\mathrm{lip}(G) = l$.

Now let (t, u, v, w, x) be a subpath of any induced path in G. By Lemma 2 the vertices t and x belong to different connected components of $G - N[v]$. In other words, our algorithm detects a longest induced path in G.

It remains to bound the running time. As usual let $n = |V|$ and $m = |E|$.

Step 1 For each pair of edges $vw, xy \in E$ we check whether $(v, w, x, y) \in \mathcal{P}_3(G)$. This can be done in $O(m^2)$ time.

Step 2 The number of blocks is $\mathcal{O}(n^2)$ and for fixed v we find all blocks (v, C) in time $\mathcal{O}(n + m)$. Hence this step needs time $\mathcal{O}(nm + n^2)$.

Step 3 Here we compute $\mathcal{O}(m^2)$ values $\ell(u, v, w, x)$. For each one we minimise over $\mathcal{O}(n)$ vertices y. Consequently this step takes $\mathcal{O}(nm^2)$ time.

Step 4 In the last step we minimise over all $(u, v, w, x) \in \mathcal{P}_3(G)$ in $\mathcal{O}(m^2)$ time.

Hence the overall running time can be bounded by $\mathcal{O}(nm^2)$. □

Gavril [12] considers maximum weighted induced paths in graphs in so-called k-bounded hole families (k-chordal graphs in [2]). For $k = 5$ (AT-free graphs are a 5-bounded hole family) the running time of his algorithm can be bounded by $\mathcal{O}(nm^3)$. Our algorithm can be modified to run on vertex weighted graphs without slowdown. Thus a maximum weight induced path on AT-free graphs can be computed by an $\mathcal{O}(nm^2)$ time algorithm.

4 Feedback Vertex Set

4.1 Analysis

Let

$$\mathrm{fvs}(G) = \min\{|W| : G - W \text{ is a forest}\} \tag{3}$$

and let K be a set of vertices inducing a forest in the graph G. For a fixed vertex $v \in K$, denote by L the set of leaves of $G[K]$ contained in $N(v)$ and let $R' = K \cap N(v) \setminus L$. Note that in the forest $G[K]$ every vertex in L is adjacent to v only and every vertex in R' is adjacent to v and at least one vertex outside $N[v]$. Let R be a representation of L w.r.t. v. The triple (v, R, R') we call a *local map* of K. In the example of Figure 2, the forest K is represented by a bold line. The triple $(v, \{b, d\}, \{g\})$ is a local map of K.

Lemma 3. *Let (v, R, R') be a local map of $K \ni v$ inducing a forest $G[K]$ in a graph G that might as well contain ATs. The following conditions hold:*

1. $R \cup R' \subseteq N(v)$;
2. $R \cup R'$ *is an independent set of* G;
3. $N(x) \setminus N(v) \subset N(x') \setminus N(v)$ *implies* $x \in R$ *and* $x' \in R'$ *for all* $x, x' \in R \cup R'$;
4. *If* G *is AT-free then* $|R \cup R'| \leqslant 4$.

Proof. Condition 1 follows directly from the definition.

Conversely, if $G[R \cup R']$ contains an edge xy, then x, y and v would form a cycle in $G[K]$. This implies Condition 2.

For Condition 3, firstly notice that if $x, x' \in R$ and $N(x) \setminus N(v) \subset N(x') \setminus N(v)$ then $N(R \setminus \{x\}) \setminus N(v) = N(R) \setminus N(v)$, contradicting the fact that R is a representation of $(K \cap N(v)) \setminus R'$ w.r.t. v. Suppose now that $x \in R'$ and $x \in R \cup R'$. Let z be a neighbour of x in $K \setminus N[v]$. Then z is adjacent to x' too, so v, x, z and x' induce a cycle in $G[K]$. Condition 3 follows.

Finally we consider Condition 4. Lemma 1 implies $|R| \leqslant 4$ and $|R'| \leqslant 2$, hence $|R \cup R'| \leqslant 6$. The proof of $|R \cup R'| \leqslant 4$ is omitted here due to space restrictions. □

A triple (v, R, R') is called *local map* of G if it satisfies the conditions of Lemma 3. The set of all local maps of G is denoted by $\mathcal{R}(G)$ or simply $\mathcal{R}$.

Definition 2. *Let* (v, R, R') *be a local map of* $G = (V, E)$. *We say that* $K \subseteq V$ *is* compatible *with* (v, R, R') *if*

- $G[K]$ *is a forest;*
- $\{v\} \cup R \cup R' \subseteq K$;
- *Every internal vertex* $x \in N(v)$ *of* $G[K]$ *belongs to* R';
- R *is a representation of* $(K \cap N(v)) \setminus R'$, *w.r.t.* v.

Let $\mathcal{K}(v, R, R')$ *denote the set of all sets* K *compatible with* (v, R, R').

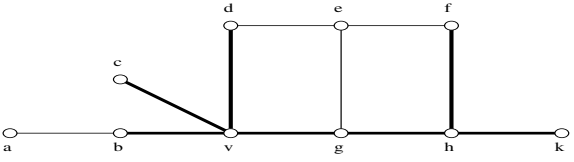


Fig. 2. Local maps

In particular, K must contain the star induced by v, R and R', and the only vertices of $K \cap N(v)$ which may have a neighbour in $K \setminus N[v]$ are the elements of R'. We point out that the vertices of R' are not required to have neighbours in $K \setminus N[v]$, they are just the only ones that *may* have neighbours in $K \setminus N[v]$. In Figure 2, the forest K is compatible with both $(v, \{b, d\}, \{g\})$ and $(v, \{d\}, \{b, g\})$. The former is a local map of the forest $K \cup \{a\}$. In other words, every induced forest containing the vertex v is compatible with its local map (v, R, R'), but some vertices in R' might be leaves of another forest compatible with (v, R, R').

Our algorithm will compute, for each local map (v, R, R') and for each block (v, C), the largest forest contained in $N[v] \cup C$, compatible with (v, R, R'). The following lemma shows in particular that it suffices to compute forests $K_1, K_2 \in \mathcal{K}(v, R, R')$ such that $K_1 \cap N(v)$ and $K_2 \cap C$ be of maximum size.

Lemma 4. *Let $K_1, K_2 \subseteq V$ be compatible with $(v, R, R') \in \mathcal{R}(G)$. For any block (v, C) of G, $K_3 = (K_1 \setminus C) \cup (K_2 \cap C)$ is compatible with (v, R, R').*

Proof. We only show that K_3 induces a forest in G because the other condition is obviously fulfilled. Conversely let $Z \subseteq K_3$ induce a shortest cycle in $G[K_3]$. Clearly $Z \cap C \neq \varnothing$ and $Z \cap N[v] \neq \varnothing$. Let $(a, x_1, \ldots, x_p, b)$ be a subpath of Z such that $x_i \in C, \forall i, 1 \leqslant i \leqslant p$ and $a, b \in N(v)$ (possibly $a = b$). Since $a, b \in K_1 \cap C$, there is a vertex a' (resp. b') of $R \cup R'$ such that $x_1 \in N(a')$ (resp. $x_p \in N(b')$). If $a' = b'$, then $(a', x_1, \ldots, x_p)$ is a cycle of G, else $(v, a', x_1, \ldots, x_p, b')$ is a cycle of G, contradicting the fact that $v, a', b', x_1, \ldots, x_p \in K_2$ and $G[K_2]$ is acyclic. □

Lemma 5. *Let (v, B, w) be an interval of $G = (V, E)$ and let $K_v, K_w \subseteq V$ be compatible with $(v, R, R'), (w, Q, Q') \in \mathcal{R}(G)$, respectively. Then $K = (K_v \setminus C(w, v)) \cup (K_w \setminus C(v, w))$ is compatible with both (v, R, R') and (w, Q, Q') if $K_0 = \{v, w\} \cup R \cup R' \cup Q \cup Q'$ is compatible with both local maps.*

Proof. We assume $K_0 \in \mathcal{K} = \mathcal{K}(v, R, R') \cap \mathcal{K}(w, Q, Q')$. Observe that $K_w \cap N(v) \cap N(w) = K_v \cap N(v) \cap N(w) = R' \cap Q'$.

Firstly, we show that $K'_v = (K_v \setminus C(w, v)) \cup K_0$ is compatible with both (v, R, R') and (w, Q, Q'). By applying Lemma 4 to the block $(v, C(w, v))$, with $K_1 = K_v$ and $K_2 = K_0$, we obtain that $K'_v \in \mathcal{K}(v, R, R')$. Since $K'_v \subseteq K_v \in \mathcal{K}(w, Q, Q')$ and K'_v contains $\{w\} \cup Q \cup Q'$, we have $K'_v \in \mathcal{K}(w, Q, Q')$. For similar reasons, $K'_w = (K_w \setminus C(v, w)) \cup K_0 \in \mathcal{K}$.

We apply Lemma 4 to the block $(v, C(w, v))$, with $K_1 = K'_v$ and $K_2 = K'_w$ and obtain that $K \in \mathcal{K}(v, R, R')$. By Lemma 4 again, applied to $(w, C(v, w))$ with $K_1 = K'_w$ and $K_2 = K'_v$, we conclude that $K \in \mathcal{K}(w, Q, Q')$. □

4.2 Synthesis

Let (v, C) be a block of an AT-free graph $G = (V, E)$, $(v, R, R') \in \mathcal{R}(G)$ and $F \subseteq V$. We define

$$\mathsf{start}(v, R, R') = \max_{K \in \mathcal{K}(v, R, R')} |K \cap N[v]|\,, \tag{4}$$

$$\mathsf{branch}(v, R, R', C) = \max_{K \in \mathcal{K}(v, R, R')} |K \cap C|\,, \tag{5}$$

$$\mathsf{accum}(F, v, R, R') = \sum_{\substack{(v,C) \text{ is block} \\ C \cap F = \varnothing}} \mathsf{branch}(v, R, R', C)\,. \tag{6}$$

Then

$$\mathrm{fvs}(G) = |V| - \max_{(v,R,R') \in \mathcal{R}(G)} \big(\mathsf{start}(v, R, R') + \mathsf{accum}(\varnothing, v, R, R')\big) \tag{7}$$

follows directly from Equations (3–6) and Lemma 4.

4.3 Computing start

As usual $\alpha(G)$ denotes the maximum size of an independent set $S \subseteq V$ of $G = (V, E)$. For a local map (v, R, R') of G we define

$$\mathsf{Inf}(v, R, R') = \{x \in N(v) : N(x) \setminus N(v) \subseteq N(R)\} \setminus (N(R) \cup N(R')) . \quad (8)$$

Lemma 6.

$$\mathsf{start}(v, R, R') = 1 + |R \cup R'| + \alpha(G[\mathsf{Inf}(v, R, R')]) \quad (9)$$

holds for all local maps $(v, R, R') \in \mathcal{R}(G)$.

Proof. Let S be a maximum independent set of $G[\mathsf{Inf}(v, R, R')]$. Then $R \cup R'$ is an independent set by definition of a local map, and $R \cup R' \cup S$ is an independent set by Equation (8). Hence $\{v\} \cup R \cup R' \cup S \in \mathcal{K}(v, R, R')$ and we conclude $\mathsf{start}(v, R, R') \geqslant 1 + |R \cup R'| + \alpha(G[\mathsf{Inf}(v, R, R')])$.

On the other hand we have $\{v\} \cup R \cup R' \subseteq K$ by Definition 2 and $|K \setminus (\{v\} \cup R \cup R')| \leqslant |S|$ for any $K \in \mathcal{K}(v, R, R')$ by Definition 2 again and our choice of the set S. This implies $\mathsf{start}(v, R, R') \leqslant 1 + |R \cup R'| + \alpha(G[\mathsf{Inf}(v, R, R')])$. □

4.4 Computing branch

Lemma 7. *Let K induce a forest in an AT-free graph G. Consider a vertex $v \in K$ and a block (v, C) such that $K \cap C \neq \varnothing$. Then there is a vertex $w \in K \cap C$ such that $K \cap B = \varnothing$ for the interval (v, B, w).*

Proof. Let w be any vertex in $K \cap C$ and assume $u \in K \cap B$. By an inductive argument it suffices to show $A \cup N[u] \subseteq B \cup N[w]$ and $w \notin A \cup N[v]$ where (v, A, u) is an interval. The inclusion follows from $u \in B$ and $w \in A \cup N[u]$ would imply that u, v and w form an AT of G. □

Lemma 8. *For every block (v, C) of an AT-free graph G and all $(v, R, R') \in \mathcal{R}(G)$ we have*

$$\begin{aligned} &\mathsf{branch}(v, R, R', C) \\ &= \max_{\substack{w \in C \text{ and} \\ \{v,w\} \cup R \cup R' \cup Q \cup Q' \in \\ \mathcal{K}(v,R,R') \cap \mathcal{K}(w,Q,Q')}} \big(\mathsf{start}(w, Q, Q') - |R' \cap S'| + \mathsf{accum}(\{v\}, w, Q, Q')\big) \end{aligned} \quad (10)$$

where $(w, Q, Q') \in \mathcal{R}(G)$ and $\mathsf{branch}(v, R, R', C) = 0$ if there is no suitable local map (w, Q, Q') exists for any vertex $w \in C$.

Proof. Let K induce a forest in G that is compatible with (v, R, R'). By Lemma 7 there is a vertex $w \in C$ such that the interval between v and w does not contain vertices in K. Let (w, Q, Q') be the corresponding local map of K. For each component $G[D]$ of $G - N[v]$ we have $|K \cap D| \leqslant \mathsf{branch}(w, Q, Q')$ by Equation (5). Summing these values up for all $D \not\ni v$ we obtain $|K \cap \bigcup_{D \not\ni v} D| \leqslant \mathsf{accum}(\{v\}, w, Q, Q')$.

Moreover Equation (4) implies $|K \cap N[w] \setminus N[v]| \leqslant \mathsf{start}(w, Q, Q') - |R' \cap S'|$. Since the interval between v and w does not contain vertices in K it follows $|K \cap C| \leqslant \max\big(\mathsf{start}(w, Q, Q') - |R' \cap S'| + \mathsf{accum}(\{v\}, w, Q, Q')\big)$.

The other way around, let the $\max$ in Equation (10) be attained for $w \in C$ and the local map (w, Q, Q'). For each block (w, D) with $v \notin D$ let $K_D \in \mathcal{K}(w, Q, Q')$ be a subset of D such that $|K_D| = \mathsf{branch}(w, Q, Q', D)$, $K_w \in \mathcal{K}(w, Q, Q')$ be a subset of $N[w]$ with $|K_w| = \mathsf{start}(w, Q, Q')$ and $K_v \in \mathcal{K}(v, R, R')$ be a subset of $N[v]$ with $|K_v| = \mathsf{start}(v, R, R')$. Then by Lemmas 4 and 5 the set $K_v \cup K_w$ is compatible with (v, R, R') and (w, Q, Q') and hence $K = K_v \cup K_w \cup \bigcup_{D \not\ni v} K_D$ is compatible with (v, R, R') and fulfils $|K \cap C| = \max\big(\mathsf{start}(w, Q, Q') - |R' \cap S'| + \mathsf{accum}(\{v\}, w, Q, Q')\big)$. □

4.5 Algorithm

Our algorithm computes $\mathrm{fvs}(G)$ for AT-free graphs G as follows:

Step 1 For all $(v, R, R') \in \mathcal{R}(G)$ compute $\mathsf{start}(v, R, R')$ using Equation (9)

Step 2 Create a list of all blocks (v, C) of G and sort by $|C|$

Step 3 For each block (v, C) and each local map (v, R, R') find $\mathsf{branch}(v, R, R', C)$ according to Equations (10) and (6) using the values $\mathsf{start}(w, Q, Q')$ computed in step 1 and values $\mathsf{branch}(w, Q, Q', D)$ computed earlier in this step.

Step 4 Compute $\mathrm{fvs}(G)$ by Equations (7) and (6) using the values $\mathsf{branch}(v, R, R', C)$ computed in step 3

Theorem 2. *There is an $\mathcal{O}(n^8 m^2)$ algorithm to compute the minimum size of a feedback vertex set in AT-free graphs.*

Proof. The correctness of this algorithm follows from Lemmas 6 and 8. We analyse the running time for an input $G = (V, E)$ where $n = |V|$, $m = |E|$ and n_v is the degree $d_G(v)$ of $v \in V$.

Step 1 For each $(v, R, R') \in \mathcal{R}$ we have $|R \cup R'| \leqslant 4$, see Condition 4 in Lemma 3. Hence we have $\mathcal{O}(n_v^4)$ local maps for each $v \in V$. This adds up to $|\mathcal{R}| = \mathcal{O}(n^3 m)$ because $\sum_{v \in V} n_v = 2m$.
We compute $\alpha(G[\mathsf{Inf}(v, R, R')])$ in time $\mathcal{O}(n_v^3)$ time because G is AT-free [15]. So the total time for this step is $\mathcal{O}(n^6 m)$.

Step 2 The number of blocks is $\mathcal{O}(n^2)$ and for fixed v we find all blocks (v, C) in time $\mathcal{O}(n + m)$. Hence this step needs time $\mathcal{O}(nm + n^2)$.

Step 3 Here we compute $\mathcal{O}(n^4 m)$ values $\mathsf{branch}(v, R, R', C)$. For each one we minimise over $\mathcal{O}(n^3 m)$ local maps (w, Q, Q'). Since $|Q \cup Q'| \leqslant 4$ we look up a single value $\mathsf{start}(w, Q, Q')$ or $\mathsf{branch}(w, Q, Q', D)$ in constant time. Consequently this step takes $\mathcal{O}(n^8 m^2)$ time.

Step 4 The last step is similar to step 3 and takes time $\mathcal{O}(n^7 m^2)$.

Hence the overall running time can be bounded by $\mathcal{O}(n^8 m^2)$. □

5 Feedback Vertex Set in Graphs with Bounded Asteroidal Number

The following definition from [13] generalises asteroidal triples.

Definition 3. *An independent set A of G is called* asteroidal set *if for every vertex $a \in A$ there is a connected component $G[C]$ of $G - N[a]$ such that $A \setminus \{a\} \subseteq C$. The* asteroidal number *of G, denoted by $\mathrm{an}(G)$, is the maximum cardinality of an asteroidal set of G.*

It is easy to see that an independent set is asteroidal if and only if each three-element subset forms an AT. The following Ramsey-type lemma will be used in the proof of Lemma 10.

Lemma 9. *Each graph on more than $2k^3$ vertices has $k+1$ pairwise nonadjacent vertices or a 2-connected subgraph on $k+1$ vertices.*

Proof. Let $G = (V, E)$ be any graph on $|V| > 2k^3$ vertices that has no 2-connected subgraph on $k+1$ vertices. By $\mathcal{B}$ we denote the set of all subsets $B \subseteq V$ such that $G[B]$ is a maximal subgraph of G without cut vertex. Two different sets $A, B \in \mathcal{B}$ are either disjoint or their intersection contains just one cut vertex of G. Clearly $|\mathcal{B}| \geqslant \lceil (2k^3+1)/k \rceil = 2k^2+1$. The set of cut vetices of G is denoted by C. We construct the forest $T = (\mathcal{B}, C, F)$ with $\{B, c\} \in F$ if $c \in B$ [20]. If T contains $k+1$ leaves we can choose a vertex from each leaf. These vertices form an independent set of G.

Otherwise we fix $B_0 \in \mathcal{B}$ and partition $\mathcal{B}$ into *layers* $L_i = \{B : d_T(B_0, B) = 2i\}$, $i \geqslant 0$. Since T has at most k leaves we have $|L_i| \leqslant k$ for all i. Hence there are more than $2k$ nonempty layers and T has a path containing at least $2k+1$ nodes in $\mathcal{B}$. Finally we choose a vertex from every other node in $\mathcal{B}$ on this path. This can easily be done such that these vertices form an independent set of G. □

Lemma 10. *For all vertices v of graph G with $\mathrm{an}(G) \leqslant k$ every independent set $S \subseteq N(v)$ has a representation of size at most $2k^3$.*

Proof. Conversely assume that S is an independent set containing more than $2k^3$ vertices such that S is its own representation. Then every vertex $s \in S$ has a private neighbour $t \in N(s) \setminus (N(v) \cup \bigcup_{s' \in S \setminus \{s\}} N(s'))$. Let T be the set of these private neighbours. Since $|T| = |S| > 2k^3$ the Lemma 9 ensures that $G[T]$ contains an independent set T' with $|T'| = k+1$ or a 2-connected subgraph $G[C]$ on $k+1$ vertices. In the latter case let $S' \subseteq S$ be the set of private neighbours of C. In both cases T' and S', respectively, form an asteroidal set of G contradicting $\mathrm{an}(G) \leqslant k$. □

Here we generalise our approach in Section 4 to graphs of bounded asteroidal number. First we reconsider Lemma 3. Let $G[K]$ be a forest in G containing the vertex v. We consider the local map (v, R, R') of K, *i.e.* R' is the set of internal vertices of $G[K]$ belonging to $N(v)$ and R is a representation of $(K \cap N(v)) \setminus R'$.

Lemma 11. *Let (v, R, R') be a local map of $K \ni v$ inducing a forest $G[K]$ in G. The following conditions hold:*

1. $R \cup R' \subseteq N(v)$;
2. $R \cup R'$ *is an independent set of* G;
3. $N(x) \setminus N(v) \subset N(x') \setminus N(v)$ *implies* $x \in R$ *and* $x' \in R'$ *for all* $x, x' \in R \cup R'$;
4. $|R| \leqslant 2k^3$ *and* $|R'| \leqslant k$ *where* $k = \mathrm{an}(G)$.

Proof. Conditions 1–3 are the same as in Lemma 3. $|R| \leqslant 2k^3$ follows from Lemma 10. It remains to prove $|R'| \leqslant k$. For each $s \in R'$ we choose a neighbour t in $K \setminus N[v]$. Let T be the set of these neighbours. Then $G[R' \cup T]$ contains exactly $|R'|$ edges, *i.e.* T is an independent set of private neighbours. In other words, T is an asteroidal set of G. Condition 4 follows. □

Now a *local map* of G is a triple (v, R, R') satisfying the conditions of Lemma 11. The set of all local maps of G is denoted by $\mathcal{R}(G)$ again. We retain the concept of *compatibility* as introduced in Definition 2.

Lemma 4 holds for all graphs. We extend Lemma 5 as follows. For an asteroidal set A of G consider the set $B(A) = \bigcap_{a,b \in A, a \neq b} C(b, a)$. We say that $(A, B(A))$ is an *interval* of G. Especially every block (v, C) is an interval $(\{v\}, C)$.

Lemma 12. *Let* (A, B) *be an interval of* $G = (V, E)$. *Let* $K_a \subseteq V$ *be compatible with* $(a, R_a, R'_a) \in \mathcal{R}(G)$ *for all* $a \in A$. *Then* $K = \bigcup_{a \in A}(K_a \setminus \bigcup_{b \in A \setminus \{a\}} C(b, a))$ *is compatible with* (a, R_a, R'_a) *for all* $a \in A$ *if* $K_0 = A \cup \bigcup_{a \in A}(R_a \cup R'_a)$ *is compatible with* (a, R_a, R'_a) *for all* $a \in A$.

Proof. omitted

We keep the definition of start, branch and accum (Equations (4), (5) and (6)). Since Lemma 6 holds for all graphs we can compute $\mathsf{start}(v, R, R')$ as before, see Equation (9). To compute $\mathsf{branch}(v, R, R', C)$ we need the following generalisation of Lemma 7.

Lemma 13. *Let* (v, C) *be a block of* G *and let* $G[K]$ *be a forest containing* v. *There is an asteroidal set* $A \subseteq K$ *of* G *with* $v \in A$ *such that the interval* $(A, B(A))$ *satisfies* $B(A) \cap K = \varnothing$.

Proof. We start with $A = \{v\}$ and $B(A) = C$. While $B(A) \cap K \neq \varnothing$,

- choose a vertex $w \in B(A) \cap K$ and let $S(v, w) = N(w) \cap N(C(v, w))$ be the unique minimal v, w-separator contained in $N(w)$;
- remove from A all the vertices a such that $S(v, w)$ separates v and a;
- add w to A.

We show that, after each step of the loop, A is an asteroidal set. At the beginning of the iteration, for any distinct vertices $a, a' \in A$, there is a a, w-path avoiding $N[a']$, by the fact that $w \in B(A) \subseteq C(a, a')$. At the end of the iteration, for any $a \in A \setminus \{v\}$, there is a a, v-path avoiding $N[w]$, otherwise a would be removed from A. Hence A is an asteroidal set.

At each step of the loop $B(A_{\mathrm{new}}) = B(A_{\mathrm{old}}) \cap C(v, w)$ holds. So we iterate at most n times. □

Lemma 14. *For every block (v, C) of a graph G and all $(v, R, R') \in \mathcal{R}(G)$ we have*

$$\mathsf{branch}(v, R, R', C) = \max_{\substack{A \subseteq C \\ A \cup \{v\} \text{ is asteroidal} \\ A \cup \{v\} \cup \bigcup_{a \in A \cup \{v\}} (R_a \cup R'_a) \in \bigcap_{a \in A \cup \{v\}} \mathcal{K}(a, R_a, R'_a)}} \Big(|\bigcup_{a \in A} R'_a \setminus R'| + \sum_{a \in A} \big(\mathsf{start}(a, R_a, R'_a) - |R'_a| + \mathsf{accum}(\{v\}, a, R_a, R'_a)\big)\Big) \quad (11)$$

where $R_v = R$, $R'_v = R'$ and $(a, R_a, R'_a) \in \mathcal{R}(G)$ for all $a \in A$.

Proof. omitted

From Lemmas 6 and 14 we derive an algorithm similar to the one given in Subsection 4.5. We claim that it runs in polynomial time whenever the asteroidal number of the input is bounded by a constant. The tricky point is to bound the size of $\mathcal{R}(G)$. Here Lemma 11 can help. Due to its Condition 4 we know $|R| \leqslant 2k^3$ and $|R'| \leqslant k$ for every local map $(v, R, R') \in \mathcal{R}(G)$ if $\mathrm{an}(G) \leqslant k$. Hence $|\mathcal{R}(G)| < n^{2k^3+k+1}$ where $k = \mathrm{an}(G)$.

References

1. V. Bafna, P. Berman, T. Fujito: A 2-approximation algorithm for the undirected feedback vertex set problem, *SIAM Journal on Discrete Mathematics* **12** (1999), pp. 289–297.
2. H. Bodlaender, D. Thilikos: Treewidth for graphs with small chordality. *Discrete Applied Mathematics* **79** (1997), pp. 45–61.
3. A. Brandstädt, D. Kratsch: On domination problems for permutation and other graphs, *Theoretical Computer Science* **54** (1987) pp. 181–198.
4. A. Brandstädt, V.V. Lozin: On the linear structure and clique width of bipartite permutation graphs, *RRR-29-2001*, Rutgers University.
5. H.J. Broersma, A. Huck, T. Kloks, O. Koppius, D. Kratsch, H. Müller and H. Tuinstra: Degree-preserving trees, *Networks* **35** (2000), pp. 26–39.
6. H.J. Broersma, T. Kloks, D. Kratsch and H. Müller: Independent sets in asteroidal triple-free graphs, *SIAM Journal on Discrete Mathematics* **12** (1999), pp. 276–287.
7. H.J. Broersma, T. Kloks, D. Kratsch and H. Müller: A generalization of AT-free graphs and a generic algorithm for solving triangulation problems, *Algorithmica* **32** (2002) pp. 594–610.
8. D.G. Corneil, S. Olariu and L. Stewart: Asteroidal triple-free graphs, *SIAM Journal on Discrete Mathematics* **10** (1997), pp. 399–430.
9. B. Courcelle, J.A. Makowsky, U. Rotics: Linear time solvable optimization problems on graphs of bounded clique-width, *Theory of Computing Systems* **33** (2000), pp. 125–150.
10. R.G. Downey, M.R. Fellows: *Parameterized complexity*, Springer, 1997.
11. M.R. Garey and D.S. Johnson: *Computers and Intractability: A guide to the Theory of NP-completeness*, Freeman, New York, 1979.
12. F. Gavril: Algorithms for maximum weight induced paths, *Information Processing Letters* **81** (2002), pp. 203–208
13. K. Kloks, D. Kratsch and H. Müller: Asteroidal sets in graphs, *Proceedings WG'97*, Lecture Notes in Computer Science N^o 1335, 229–241, Springer-Verlag, 1997.
14. J. Kleinberg, A. Kumar: Wavelength conversion in optical networks, *Journal of Algorithms* **38** (2001) pp. 25–50.

15. E. Köhler: Graphs without asteroidal triples, PhD thesis, TU Berlin 1999. `ftp://ftp.math.tu-berlin.de/pub/combi/ekoehler/diss`
16. D.Y. Liang: On the feedback vertex set problem in permutation graphs, *Information Processing Letters* **52** (1994) pp. 123–129.
17. D.Y. Liang, M.S. Chang: Minimum feedback vertex sets in cocomparability graphs and convex bipartite graphs, *Acta Informatica* **34** (1997) pp. 337–346.
18. C.L. Lu, C.Y. Tang: A linear-time algorithm for the weighted feedback vertex problem on interval graphs, *Information Processing Letters* **61** (1997) pp. 107–111.
19. J. Spinrad: *Efficient graph representations*, American Mathematical Society, Fields Institute Monograph Series 19, 2003.
20. R. Tarjan: Depth-first search and linear graph algorithms. *SIAM Journal on Computing* **1** (1972) pp. 146–160.

The Complexity of Graph Contractions

Asaf Levin[1], Daniël Paulusma[2], and Gerhard J. Woeginger[2]

[1] Department of Statistics and Operations Research,
Tel Aviv University, Tel Aviv 69978, Israel.
levinas@post.tau.ac.il
[2] Faculty of Mathematical Sciences,
University of Twente, 7500 AE Enschede, The Netherlands.
{d.paulusma,g.j.woeginger}@math.utwente.nl

Abstract. For a fixed pattern graph H, let H-CONTRACTIBILITY denote the problem of deciding whether a given input graph is contractible to H. We continue a line of research that was started in 1987 by Brouwer & Veldman, and we determine the computational complexity of H-CONTRACTIBILITY for certain classes of pattern graphs. In particular, we pin-point the complexity for all graphs H with five vertices. Interestingly, in all cases that are known to be polynomially solvable, the pattern graph H has a dominating vertex, whereas in all cases that are known to be NP-complete, the pattern graph H does not have a dominating vertex.

1 Introduction

All graphs in this paper are undirected, finite, and simple. Let $G = (V, E)$ be a graph, and let $e = [u, v] \in E$ be an arbitrary edge. The *edge contraction* of edge e in G removes the two end-vertices u and v from G, and replaces them by a new vertex that is adjacent to precisely those vertices to which u or v were adjacent. The *edge deletion* of edge e removes e from E. The *edge subdivision* of e removes e from E, and introduces a new vertex that is adjacent to the two end-vertices u and v. A graph G is *contractible* to a graph H (G is H-contractible), if H can be obtained from G by a sequence of edge contractions. A graph G contains a graph H as a *minor*, if H can be obtained from G by a sequence of edge contractions and edge deletions. A graph G is a *subdivision* of a graph H, if G can be obtained from H by a sequence of edge subdivisions.

Now let $H = (V_H, E_H)$ be some fixed connected graph with vertex set $V_H = \{h_1, \ldots, h_k\}$. There is a number of natural and elementary algorithmic problems that check whether the structure of graph H shows up as a *pattern* within the structure of some input graph G:

– PROBLEM: H-MINOR CONTAINMENT
INSTANCE: A graph $G = (V, E)$.
QUESTION: Does G contain H as a minor?

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 322–333, 2003.

- PROBLEM: H-SUBDIVISION SUBGRAPH
  INSTANCE: A graph $G = (V, E)$.
  QUESTION: Does G contain a subgraph that is isomorphic to some subdivision of H?
- PROBLEM: ANCHORED H-SUBDIVISION SUBGRAPH
  INSTANCE: A graph $G = (V, E)$; k pairwise distinct vertices $v_1, \ldots, v_k$ in V.
  QUESTION: Does G contain a subgraph that is isomorphic to some subdivision of H, such that the isomorphism maps vertex v_i of the subgraph of G into vertex h_i of the subdivision of H, for $1 \leq i \leq k$?
- PROBLEM: H-CONTRACTIBILITY
  INSTANCE: A graph $G = (V, E)$.
  QUESTION: Is G contractible to H?

1.1 Known Results

A celebrated result by Robertson & Seymour [3] states that H-MINOR CONTAINMENT can be solved in polynomial time $O(|V|^3)$ for every *fixed* pattern graph H. In fact, [3] fully settles the complexity of the first three problems on our problem list above:

Proposition 1. *(Robertson & Seymour [3])*
For any fixed pattern graph H, the three problems H-MINOR CONTAINMENT, H-SUBDIVISION SUBGRAPH, and ANCHORED H-SUBDIVISION SUBGRAPH are polynomially solvable in polynomial time. ■

What about the fourth problem on our list, H-CONTRACTIBILITY? Perhaps surprisingly, there exist pattern graphs H for which this problem is NP-complete to decide! For instance, Brouwer & Veldman [1] have shown that P_4-CONTRACTIBILITY is NP-complete. The main result of [1] is the following.

Proposition 2. *(Brouwer & Veldman [1])*
If H is a connected triangle-free graph other than a star, then H-CONTRACTIBILITY is NP-complete. If H is a star, then H-CONTRACTIBILITY is polynomially solvable. ■

Note that an equivalent way of stating Proposition 2 would be the following: H-CONTRACTIBILITY is NP-complete for every connected triangle-free graph H without a dominating vertex. H-CONTRACTIBILITY is polynomially for every connected triangle-free graph H with a dominating vertex. (A *dominating* vertex is a vertex that is adjacent to all other vertices.) Moreover, the paper [1] determines the complexity of H-CONTRACTIBILITY for all 'small' connected pattern graphs H: For $H = P_4$ and $H = C_4$, the problem is NP-complete (as implied by Proposition 2). For every other pattern graph H on at most four vertices, the problem is polynomially solvable.

The exact separating line between polynomially solvable cases and NP-complete cases of this problem (under P≠NP) is unknown and unclear.

Brouwer & Veldman [1] write at the end of their paper that they expect the class of polynomially solvable cases to be very limited.

Watanabe, Ae & Nakamura [4] consider remotely related edge contraction problems where the goal is to find the minimum number of edge contractions that transform a given input graph G into a pattern from a certain given pattern class.

1.2 New Results

We follow the line of research that has been initiated by Brouwer & Veldman [1], and we classify the complexity of H-CONTRACTIBILITY for certain classes of pattern graphs that – in particular – contain all 'small' pattern graphs H with at most five vertices. Our results can be summarized as follows:

Theorem 1. *(Main result of the paper)*
Let H be a connected graph on at most five vertices. If H has a dominating vertex, then H-CONTRACTIBILITY is polynomially solvable. If H does not have a dominating vertex, then H-CONTRACTIBILITY is NP-complete.

It is difficult for us *not* to conjecture that the presence of a dominating vertex in the pattern graph H precisely separates the easy cases from the hard cases. However, we have no evidence for such a conjecture.

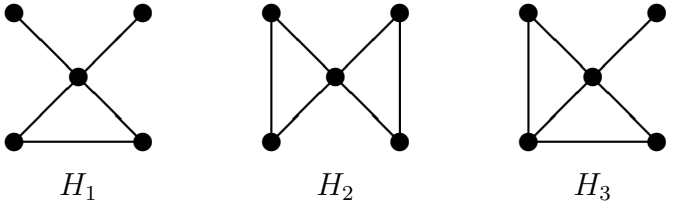


Fig. 1. The graphs $H_1 = H_1^*(2,1,0)$; $H_2 = H_1^*(0,2,0)$; and $H_3 = H_1^*(1,0,1)$.

There are fifteen graphs H on five vertices that are not covered by Proposition 2; these are exactly the connected graphs on five vertices that do contain a triangle; see Figures 1–6 for pictures of all these graphs. It turned out that ten of these fifteen graphs yield polynomially solvable H-CONTRACTIBILITY problems, whereas the other five of them yield NP-complete problems. Many of our results are actually more general: They do not only provide a specialized result for one particular five-vertex graph, but they do provide a result for an infinite family of pattern graphs, from which the result on the five-vertex graph falls out as a special case. Our main contributions may be summarized as follows:

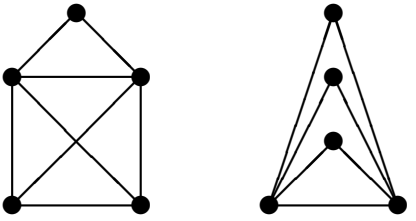


Fig. 2. The graphs $H_4 = H_2^*(1,1)$; and $H_5 = H_2^*(3,0)$.

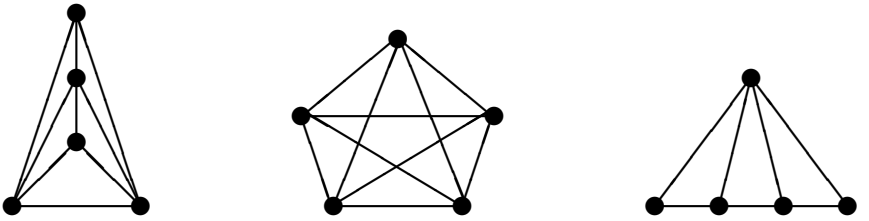


Fig. 3. The graphs $H_6 = H_3^*(2)$; $H_7 = K_5$; and $H_8 = P^+(4) = K_1 \bowtie P_4$.

(1) We analyze a class of cases where H contains one, two, or three dominating vertices, and where the set of non-dominating vertices induce a set of isolated vertices, isolated edges, and paths on three vertices. In Section 3, we prove that three subfamilies of this class yield polynomially solvable H-CONTRACTIBILITY problems. These classes contain the eight graphs H_1 thru H_8 on five vertices as depicted in Figures 1– 3.
Our structural results show that in case *some* H-contraction exists, then there also exists an H-contraction of a fairly primitive form. In our algorithmic results, we then enumerate all possibilities for these primitive pieces, and settle the remaining problems by applying the results of Robertson & Seymour [3].

(2) For the two five-vertex graphs H_9 and H_{10} as shown in Figure 4, we were not able to find 'straightforward' polynomial time algorithms. Our algorithms are based on lengthy (!) combinatorial investigations of potential contractions of an input graph to H_9 and H_{10}. They are not included in this extended abstract.

(3) In Section 4 we present a number of NP-completeness results. The NP-completeness proofs for the four (five-vertex) graphs $H_1^\#$, $H_2^\#$, $H_3^\#$, and $H_4^\#$ in Figure 5 are omitted in this extended abstract. They are done by reduction from HYPERGRAPH 2-COLORABILITY and they are inspired by a

similar NP-completeness argument of Brouwer & Veldman [1]. Theorem 5 and Theorem 6 present two generic NP-completeness constructions. As a special case, this yields NP-completeness of $H_5^{\#}$-CONTRACTIBILITY for the graph $H_5^{\#}$ in Figure 6.

The rest of this extended abstract contains the exact statements and some of the proofs for the above results.

H_9 H_{10}

Fig. 4. The graphs H_9 and H_{10}.

2 Notations, Definitions, and Preliminaries

We denote by P_n the path on n vertices, by C_n the cycle on n vertices, and by K_n the complete graph on n vertices. For a subset $U \subset V$ we denote by $G[U]$ the induced subgraph of G over U; hence $G[U] = (U, E \cap U \times U)$.

For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with $V_1 \cap V_2 = \emptyset$, we denote their *join* by $G_1 \bowtie G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup V_1 \times V_2)$, and their *disjoint union* by $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$. For the disjoint union $G \cup G \cup \cdots \cup G$ of k copies of the graph G, we write shortly kG_1; for $k = 0$ this yields the empty graph.

Consider a graph $G = (V, E)$ that is contractible to a graph $H = (V_H, E_H)$. An equivalent (and for our purposes more convenient) way of stating this fact is that

- for every vertex h in V_H, there is a corresponding connected subset $W(h) \subseteq V$ of vertices in G; we will sometimes say that $W(h)$ is the *witness* for vertex h.
- for every edge $e = [h_1, h_2] \in E_H$, there is a corresponding edge $W(e)$ in G that connects the vertex set $W(h_1)$ to the vertex set $W(h_2)$; we will sometimes say that this edge $W(e)$ in G is a *witness* for $[h_1, h_2]$.
- for every two vertices h_1, h_2 in H that are not connected by an edge in E_H, there are no edges between $W(h_1)$ and $W(h_2)$.

If for every $h \in V_H$, we contract the vertices in $W(h)$ to a single vertex, then we end up with the graph H. Note that in general, these witness sets $W(h)$

and witness edges $W(e)$ are not uniquely defined (since there may be many different sequences of contractions that lead from G to H). In our polynomial time algorithms, we will explore the structure of the witnesses, and often prove that there exists *at least one* witness with certain 'strong' and 'nice' properties.

Proposition 3. *For any fixed integer k, the following problem can be solved in polynomial time: Given a graph $G = (V, E)$ with up to k vertices that are colored by $\ell \leq k$ colors, can this coloring be extended to an ℓ-coloring of the whole vertex set V such that every color class induces a connected subgraph?*

Proof. Consider some fixed color c, and let $S_c \subseteq V$ be the set of all vertices that are pre-colored by color c. Any solution to the problem must contain a mono-chromatic tree T in color c whose leaf-set coincides with S_c. Such a tree T has at most k branch-vertices (that is, vertices of degree three or higher), and there is only a fixed number of different topologies for connecting the branch-vertices and the leaves in S_c to each other.

The strategy is as follows: For every color c, we guess the branch-vertices for a mono-chromatic tree with leaf-set S_c, and we also guess the topology of this tree. Since k is a fixed constant, this altogether only yields a polynomial number of guesses for all trees for all colors (where the degree of the polynomial depends on k). Then we are left with a special case of the ANCHORED H-SUBDIVISION SUBGRAPH problem that can be solved in polynomial time according to Proposition 1. ■

3 Some Simple Polynomially Solvable Cases

Consider a connected graph $G = (V, E)$ with a cut-vertex v, and let $C_1, \ldots, C_k$ denote the connected components of $G - v$. For $1 \leq i \leq k$ we say that the vertex subset $V - C_i$ is *induced* by the cut-vertex v,

For non-negative integers a, b, c, we let $H_1^*(a, b, c)$ be the graph $K_1 \bowtie (aK_1 \cup bK_2 \cup cP_3)$, $H_2^*(a, b)$ be the graph $K_2 \bowtie (aK_1 \cup bK_2)$, and $H_3^*(a)$ be the graph $K_3 \bowtie aK_1$.

Lemma 1. *Let u be the dominating vertex in $H_1^*(a, b, c)$. If a graph $G = (V, E)$ is contractible to $H_1^*(a, b, c)$, then there exists a witness structure with the following property: For every vertex $h \neq u$ in $H_1^*(a, b, c)$, the witness set $W(h)$ is either induced by some cut-vertex, or it consists of a single vertex.*

Proof. Consider a witness structure W for G with respect to $H_1^*(a, b, c)$ that maximizes the cardinality of $W(u)$. Let x, y, z be three vertices in $H_1^*(a, b, c) - u$ that induce a P_3 with edges $[x, y]$ and $[y, z]$. We will only show that the desired property holds for $W(x)$, $W(y)$, and $W(z)$. The arguments for the other cases are similar, but simpler.

Suppose for the sake of contradiction that there are two distinct vertices $x_1, x_2 \in W(x)$ that both have neighbors in $W(y)$. Let $[u_1, x_3] \in E$ be a witness edge for $[u, x] \in E_H$. Consider a tree T within $W(x)$ with the minimum number

of edges that connects x_1, x_2, x_3 to each other. Then we could move vertex x_3 (and part of this tree T, and possibly some other vertices) from $W(x)$ to $W(u)$ while keeping the witness structure intact. This would increase the cardinality of $W(u)$. This contradiction shows that $W(x)$ contains exactly one vertex x_1 with neighbors in $W(y)$. Next, suppose that there is some vertex $x_4 \in W(x)$ with $x_4 \neq x_1$, such that x_4 has an edge to $W(u)$. Then we could move x_4 (and part of a path from x_4 to x_1 within $W(x)$, and perhaps some other vertices) from $W(x)$ to $W(u)$. This second contradiction shows that x_1 is the unique vertex in $W(x)$ with neighbors outside $W(x)$. A symmetric argument shows that $W(z)$ contains a unique vertex z_1 with neighbors outside $W(z)$. Hence, $W(x)$ and $W(z)$ both are of the desired form.

Let us turn to $W(y)$. Consider a vertex $y_1 \in W(y)$ that has a neighbor in $W(u)$. First we discuss the case where y_1 is a cut-vertex of the subgraph induced by $W(y)$. Let $C_1, \ldots, C_k$ denote the connected components induced by $W(y) - y_1$. If some component, say C_1, is adjacent to both x_1 and z_1, then we could redefine $W(y) := C_1$ and merge all the other components together with y_1 into $W(u)$; a contradiction. Hence, we may assume that every component is adjacent to at most one of the two vertices x_1 and z_1. If x_1 has a neighbor $y_2 \in W(y)$ with $y_2 \neq y_1$, then we could redefine $W(x) := \{y_2\}$ and merge x_1 (and the rest of $W(x)$) into $W(u)$; a contradiction to the choice of $W(u)$. A symmetric argument shows that the only neighbor of z_1 in $W(y)$ is y_1. But now we are done: If $W(u)$ has a neighbor $y_3 \in W(y)$ with $y_3 \neq y_1$, then we may merge y_3 (and maybe some other vertices in $W(y)$) into $W(u)$. If $W(u)$ does not have any other neighbor but y_1 in $W(y)$, then all the edges between $W(x) \cup W(u) \cup W(z)$ and $W(y)$ are incident to vertex y_1. Hence, the witness set $W(y)$ is either induced by the cut-vertex y_1, or it consists of the single vertex y_1.

In the remaining case, vertex y_1 is not a cut-vertex of the subgraph induced by $W(y)$. If x_1 and z_1 both have neighbors other than y_1 in $W(y)$, then we could simply move y_1 into $W(u)$, and arrive at a contradiction. If y_1 is the unique neighbor of x_1 in $W(y)$ and if z_1 also has another neighbor $y_4 \in W(y)$, then we could redefine $W(z) := \{y_4\}$ and merge the old $W(z)$ into $W(u)$. A symmetric arguments settles the case where y_1 is the unique neighbor of z_1, but not the unique neighbor of x_1. Finally, the sub-case where y_1 is the unique neighbor of both x_1 and z_1 in $W(y)$ can be handled similarly as in the previous paragraph. ■

Lemma 2. *Let u_1 and u_2 be the two dominating vertices in $H_2^*(a,b)$. If a graph $G = (V,E)$ is contractible to $H_2^*(a,b)$, then there exists a witness structure with the following property:*

1. *For every vertex x in $H_2^*(a,b) - \{u_1, u_2\}$ that is only connected to u_1 and u_2, the witness set $W(x)$ is either induced by some cut-vertex or it consists of a single vertex.*
2. *For any pair y and z of adjacent vertices in $H_2^*(a,b) - \{u_1, u_2\}$, there exist two vertices $y_1 \in W(y)$ and $z_1 \in W(z)$, such that: $W(y)$ contains y_1 and (in case y_1 is a cut-vertex) a vertex subset induced by y_1. $W(z)$ contains*

z_1 and (in case z_1 is a cut-vertex) a vertex subset induced by z_1. Moreover if $\{y_1, z_1\}$ forms a cut-set of G, then with one exception, all components of $G - \{y_1, z_1\}$ that are adjacent to y_1 and z_1 belong to $W(y) \cup W(z)$.

Proof. Omitted in this extended abstract. ■

Lemma 3. *Let u_1, u_2, u_3 be the three dominating vertices in $H_3^*(a)$. If a graph $G = (V, E)$ is contractible to $H_3^*(a)$, then there exists a witness structure with the following property: For every vertex h in $H_3^*(a) - \{u_1, u_2, u_3\}$, the witness set $W(h)$ is either induced by some cut-vertex, or it consists of a single vertex.*

Proof. Omitted in this extended abstract. ■

Theorem 2. *For any fixed non-negative integers a, b, c, contractibility to $H_1^*(a, b, c)$, to $H_2^*(a, b)$, and to $H_3^*(a)$ can be decided in polynomial time.*

Proof. The proof combines the statements in Lemmas 1–3 with a lot of guessing and with an application of Proposition 3. In the following, we will use the same notation as in the statements of these lemmas.

For $H_1^*(a, b, c)$, we may guess for each of the $a + b + c$ witness sets $W(h)$ its unique element or its crucial cut-vertex. There are only $O(n^{a+b+c})$ possibilities for that. This fully defines the witness sets $W(h)$ with $h \neq u$. All remaining vertices are put into $W(u)$. It is easy to check in polynomial time, whether the guessed structure yields a feasible witness structure.

For $H_2^*(a, b)$, we guess for every vertex x (that has u_1 and u_2 as its only neighbors) the unique element or the crucial cut-vertex for $W(x)$. For every pair y and z (of adjacent vertices in $H_2^*(a, b) - \{u_1, u_2\}$), we guess the crucial vertices y_1 and y_2, and we also guess the component of $G - \{y_1, z_1\}$ that contains $W(u_1) \cup W(u_2)$. Finally, we guess two neighbors of y_1 in $W(u_1)$ and in $W(u_2)$, and two neighbors of z_1 in $W(u_1)$ and in $W(u_2)$. There are $O(n^{a+3b+1})$ possibilities for all these guesses. The guesses fully specify the witness sets for the non-dominating vertices in $H_2^*(a, b)$. They also specify $W(u_1) \cup W(u_2)$, and they specify a (fixed constant) number of vertices in $W(u_1)$ respectively $W(u_2)$ that result from the above neighbor guesses. Checking feasibility of this witness structure boils down to checking whether there exists a partition (= 2-coloring) of the vertex set $W(u_1) \cup W(u_2)$ into two connected sets $W(u_1)$ and $W(u_2)$ that respects the assignment of the guessed vertices. But that's just a special case of the problem in Proposition 3 with two colors.

For $H_3^*(a)$, we guess for every non-dominating vertex h the crucial vertex h_1 in G that specifies $W(h)$. Moreover, we guess three neighbors of h_1 in $W(u_1)$, $W(u_2)$, and $W(u_3)$. There are $O(n^{4a})$ possibilities for all these guesses. It remains to check whether $W(u_1) \cup W(u_2) \cup W(u_3)$ can be divided into three connected sets $W(u_1)$, $W(u_2)$, $W(u_3)$ that contain the appropriate guessed vertices. This is a special case of the problem in Proposition 3 with three colors. ■

Proposition 4. *(Robertson & Seymour [3])*
For any fixed integer $a \geq 1$, contractibility to the complete graph K_a can be decided in polynomial time.

Proof. A trivial consequence of Proposition 1, since K_a-MINOR CONTAINMENT and K_a–CONTRACTIBILITY are the same problem. ■

Theorem 3. *For any fixed integer $a \geq 2$, contractibility to $P^+(a) := K_1 \bowtie P_a$ can be decided in polynomial time.*

Proof. Omitted in this extended abstract. ■

Now let us apply the results of this section to the five-vertex graphs depicted in Figures 1– 3. The first three graphs H_1, H_2, and H_3 have a single dominating vertex and fall into the classes $H_1^*(*,*,*)$. The graphs H_4 and H_5 have two dominating vertices, and fall into the classes $H_2^*(*,*)$. The graph H_6 has a dominating triangle and falls into one of the classes $H_3^*(*)$. Hence, contractibility to these six graphs can be decided in polynomial time according to Theorem 2.

Graph H_7 is the clique on five vertices, and hence by Proposition 4 also contractibility to H_7 can be decided in polynomial time. Finally, graph H_8 equals $K_1 \bowtie P_4$, and hence contractibility to H_8 is polynomial by Theorem 3.

4 The NP-Complete Cases

Let $H^\#$ be the graph on 5 vertices v, w, x, y, z with edges $[w, x], [x, y], [y, z]$, and $[v, x]$. Let $H_1^\#$ be the graph $H^\#$ with the edge $[v, w]$, $H_2^\#$ be the graph $H^\#$ with the edge $[v, y]$, $H_3^\#$ be the graph $H^\#$ with edges $[v, w], [v, y]$, and $H_4^\#$ be the graph $H^\#$ with edges $[v, w], [v, y], [w, z]$. See Figure 5 for some pictures.

Theorem 4. *H-CONTRACTIBILITY is NP-complete if H is $H_1^\#$, $H_2^\#$, $H_3^\#$, or $H_4^\#$.*

Proof. Omitted in this extended abstract. ■

We now will now describe two families of pattern graphs for which the corresponding contractibility problem is NP-complete. The (five-vertex) graph $H_5^\#$ in Figure 6 belongs to the family that is analyzed in Theorem 6; $H_5^\#$ is the last one of the fifteen five-vertex graphs that are not covered by Proposition 2.

For a graph $G = (V, E)$ with $x \in V$ and $e = [x, z] \in E$ let G_{xy} denote the graph G with a new vertex y and edge $[x, y]$, and let G_{ey} denote the graph G with a new vertex y and edges $[x, y]$ and $[y, z]$.

Theorem 5. *Let H be a 2-connected graph. If H-CONTRACTIBILITY is NP-complete, then H_{xy}-CONTRACTIBILITY is NP-complete for all $x \in V_H$.*

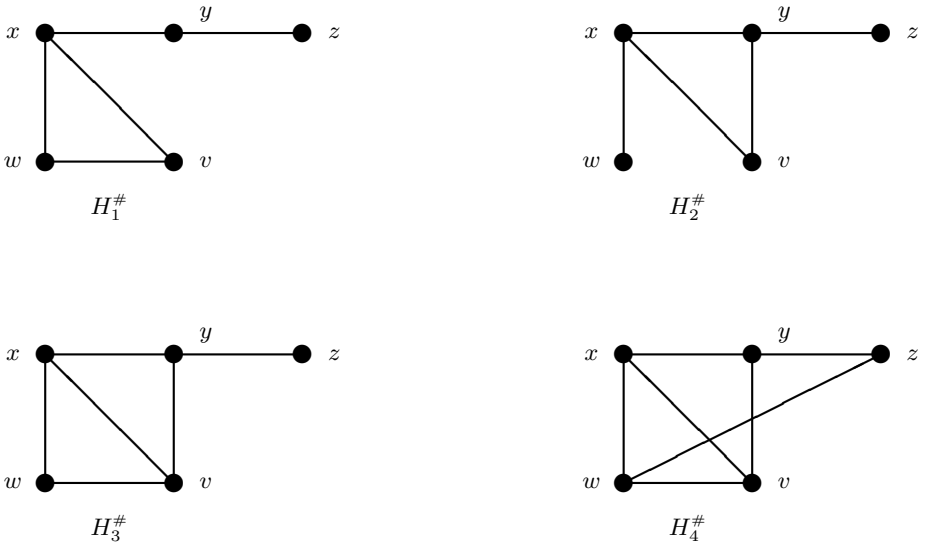


Fig. 5. The graphs $H_1^\#$, $H_2^\#$, $H_3^\#$, and $H_4^\#$ that yield NP-complete contractibility problems.

Proof. Given an instance graph G of H-CONTRACTIBILITY we construct a graph G' as follows. Let $V_G = \{r_1, \ldots, r_m\}$. First we make m disjoint copies G^i of G. We insert an extra vertex s, and for $1 \leq i \leq m$ we connect the vertex r_i of the i-th copy to s by an edge.

Our claim is that G is contractible to H if and only if G' is contractible to H_{xy} for $x \in V_H$.

If G is contractible to H, then we define $W_{G'}(y) := V_{G' \setminus G^j}$, where G^j is chosen such that r_j is a vertex in $W_{G^j}(x)$. Clearly, $W_{G'}(y)$ is connected, and G' is contractible to H_{xy}.

Conversely, suppose G' is contractible to H_{xy}. Suppose s is in $W_{G'}(v)$ for some $v \in V_H \cup y$. First assume that $v \neq x$. If vertices of more than one copy G^j of G are contained in other witness sets than $W_{G'}(v)$, then clearly $v \neq y$ but $v \neq x$ would be a cutvertex of H.

Hence, only vertices of G^j are in $V_{G'} \setminus W_{G'}(v)$. Now we remove all other copies G^i $(i \neq j)$ from G'. It is straightforward to see that the remaining part of $W_{G'}(v)$ still induces a connected subgraph.

Suppose $v = y$. Since the degree of y in H_{xy} is exactly one, we remove s and move the remaining vertices of $W_{G'}(y)$ to $W_{G'}(x)$. This way we can contract $G = G^j$ to H.

If $v \neq y$, then besides s also r_j is in $W_{G'}(v)$. Otherwise $W_{G'}(v)$ would be equal to $\{s\}$, and v would have degree one implying that H is not 2-connected. Then

we can remove s from $W_{G'}(v)$ and the remaining set $W_{G'}(v)$ induces a connected subgraph. By moving $W_{G'}(y)$ to $W_{G'}(x)$ the graph $G = G^j$ is contractible to H.

The case $v = x$ can be solved using similar arguments as above. ■

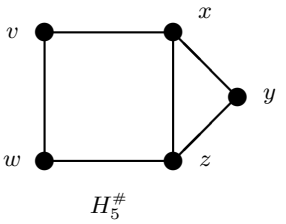


Fig. 6. The graph $H_5^{\#}$.

Theorem 6. *Let H be a 2-connected graph that does not have any vertex v with exactly two neighbors w_1, w_2 such that $[w_1, w_2] \in E_H$. If H-CONTRACTIBILITY is NP-complete, then H_{ey}-CONTRACTIBILITY is NP-complete for all $e \in E_H$.*

Proof. Given an instance graph G of H-CONTRACTIBILITY we construct a graph G' as follows. Let $E_G = \{e_1, \ldots, e_n\}$. First we make n disjoint copies G^i of G. We insert an extra vertex s, and for $1 \leq i \leq n$ we connect the end points of e_i of the i-th copy to s by an edge.

Our claim is that G is contractible to H if and only if G' is contractible to H_{ey} for $e = [x, z] \in V_E$.

If G is contractible to H, then an edge $e_j \in E_G$ exists that has one of its end points in $W_{G^j}(x)$ and the other one in $W_{G^j}(z)$. We define $W_{G'}(y) := V_{G' \setminus G^j}$. Clearly, $W_{G'}(y)$ is connected, and G' is contractible to H_{ey}.

Conversely, suppose G' is contractible to H_{ey}. Suppose s is in $W_{G'}(v)$ for some $v \in V_H \cup y$. Because H is 2-connected, also H_{ey} does not have any cutvertices. If vertices of more than one copy G^j of G are contained in other witness sets than $W_{G'}(v)$, then v would be a cutvertex of H_{ey}.

Hence, only vertices of G^j are in $V_{G'} \setminus W_{G'}(v)$. Now we remove all other copies G^i $(i \neq j)$ from G'. It is straightforward to see that the remaining part of $W_{G'}(v)$ still induces a connected subgraph.

Suppose $v = y$. We remove s and move the remaining vertices of $W_{G'}(y)$ to $W_{G'}(x)$. We can contract $G = G^j$ to H this way, because the only neighbors of y in H_{ey} are x and z and $[x, z]$ is an edge in H.

If $v \neq y$, then besides s at least one end point of e_j is in $W_{G'}(v)$. Otherwise $W_{G'}(v)$ would be equal to $\{s\}$, and v would have degree one in H or exactly two neighbors w_1 and w_2 with $[w_1, w_2] \in E_H$. Then we can remove s from $W_{G'}(v)$ and the remaining set $W_{G'}(v)$ induces a connected subgraph. By moving $W_{G'}(y)$ to $W_{G'}(x)$, we see that the graph $G = G^j$ is contractible to H. ■

References

1. A.E. Brouwer and H.J. Veldman (1987). Contractibility and NP-completeness. *Journal of Graph Theory 11*, 71–79.
2. M.R. Garey and D.S. Johnson *Computers and Intractability*. W.H. Freeman and Co., New York, 1979.
3. N. Robertson and P.D. Seymour (1995). Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B 63*, 65–110.
4. T. Watanabe, T. Ae, and A. Nakamura (1981). On the NP-hardness of edge-deletion and edge-contraction problems. *Discrete Applied Mathematics 6*, 63–78.

Tree Spanners, Cayley Graphs, and Diametrically Uniform Graphs*

Paul Manuel[1], Bharati Rajan[2], Indra Rajasingh[2], and Amutha Alaguvel[2]

[1]Department of Information and Computer Science,
King Fahd University of Petroleum and Minerals, Dhahran,
Saudi Arabia 31261
manuel@ccse.kfupm.edu.sa
[2]Department of Mathematics, Loyola College, Chennai
India 600 034
rajanbharati@rediffmail.com
indrarajasingh@yahoo.com

Abstract. In line with symmetrical graphs such as Cayley graphs and vertex transitive graphs, we introduce a new class of symmetrical graphs called diametrically uniform graphs. The class of diametrically uniform graphs includes vertex transitive graphs and hence Cayley graphs. A tree *t*-spanner of graph *G* is a spanning tree *T* in which the distance between every pair of vertices is at most *t* times their distance in *G*. The minimum tree spanner problem of a graph *G* is to find a tree *t*-spanner with *t* as small as possible. In this paper, the minimum tree spanner problem is exhaustively studied for diametrically uniform graphs, which also include 3-regular mesh of trees and generalized Petersen graphs.

1 Introduction

For a given graph $G(V, E)$ and a spanning subgraph H of G, $d_G(u, v)$ and $d_H(u, v)$ denote the shortest distance between the vertices u and v in G and H respectively. A spanning subgraph H of a graph G is a *t-spanner* of G if $d_H(u, v) \leq t\, d_G(u, v)$ for every pair of vertices u and v of G. When the spanning subgraph H is a spanning tree, it is called a *tree t-spanner*. The spanner concept has a number of applications. Much effort has been made in recent years to study this concept due to its immense applications in communication networks, distributed systems, motion planning, network design and parallel architectures [3, 4, 6].

1.1 The Tree *t*-Spanner Admissible Problem

A graph G is a *tree t-spanner admissible graph* if it admits a tree *t*-spanner. The *tree t-spanner admissible problem* is to determine the existence of a tree *t*-spanner in a given graph [4, 5].

* This research is supported by King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 334–345, 2003.

In [4], Cai showed that for $t = 2$ it can be determined in linear time whether or not a given graph has a tree t-spanner. He also showed that the problem is NP-complete for $t \geq 4$. Cai conjectured that when $t = 3$ the problem may be NP-complete on general graphs. The exact complexity status of this problem is still unresolved and it is a challenging open problem.

1.2 The Minimum Tree Spanner Problem

A *spanner* $\zeta(T,G)$ of a spanning tree T of G is defined as $\zeta(T,G) = max\{d_T(u, v) : (u, v)$ is an edge in $E(G)\}$, where $d_T(u, v)$ denotes the distance between u and v in T. A *minimum spanner* $\zeta(G)$ of G is defined as $\zeta(G) = min\{\zeta(T,G) : T$ *is a spanning tree of* $G\}$. A spanning tree T is called a *minimum tree spanner*, if $\zeta(T,G) = \zeta(G)$. Equivalently T is a minimum tree spanner if $\zeta(T,G) \leq \zeta(T',G)$, for all spanning trees T' of G. In other words, the *minimum tree spanner problem* of a graph G is to find a minimum tree spanner of G [4]. A polynomial time algorithm is available to solve this problem for digraphs [4] and directed path graphs [13]. All the NP-complete results on the tree t-spanner admissible problem [4, 5, 14] hold good for the minimum tree spanner problem. It is also shown that the minimum tree spanner problem is NP-complete for planar graphs [7].

1.3 Diametrically Uniform Graphs

For each vertex u of a graph G, the maximum distance $d(u, v)$ to any other vertex v of G is called its *eccentricity* and is denoted by $ecc(u)$. In a graph G, the maximum value of eccentricity of vertices of G is called the *diameter* of G and is denoted by λ and the minimum value of eccentricity of vertices of G is called the *radius* of G and is denoted by ρ. The set of vertices of a graph G with eccentricity equal to the radius ρ is called the *center* of G and is denoted by $Z(G)$.

Let G be a graph with diameter λ. A vertex v of G is said to be *diametrically opposite* to a vertex u of G, if $d_G(u, v) = \lambda$. A graph G is said to be a *diametrically uniform graph* if every vertex of G has *at least* one diametrically opposite vertex. The set of diametrically opposite vertices of a vertex x in G is denoted by $D(x)$.

2 Overview of the Paper

Symmetry is a fundamental virtue in all engineering designs, particularly in the domain of parallel architectures. One of the parameters of being a “good” parallel architecture is symmetry. Analysis of different types of symmetry and development of various hierarchies of symmetry in graphs has been the subject of intense study in the recent years. The symmetrical property is something conceptual and does not have a proper mathematical definition. There are a few measures of symmetry such as vertex transitivity, edge transitivity, distance transitivity and distance regularity [8, 10, 11]. Cayley graphs, edge transitive graphs, vertex transitive graphs, distance transitive graphs and distance regular graphs [8, 10, 11] are a few classes of graphs

following these measures of symmetry. Here we characterize the symmetrical property of a graph in terms of its diameter. This leads to the introduction of the diametrically uniform graphs. The concept of diametrically uniform graphs is an extension of Cayley graphs and vertex transitive graphs. More precisely, the set inclusion will be as follows.

Cayley graphs $\subset$ Vertex transitive graphs $\subset$ Diametrically uniform graphs

In this paper, $BFS(u)$ of a graph G denotes a breadth first search (*bfs*) tree of G rooted at the vertex u [1]. We observe a close relationship between breadth first search (*bfs*) trees and these classes of graphs. Given two vertices u and v of a vertex transitive graph, for every *bfs* tree $BFS(u)$, there exists a *bfs* tree $BFS(v)$ such that $BFS(u)$ and $BFS(v)$ are isomorphic. Moreover, any two *bfs* trees of a diametrically uniform graph are of equal height.

We establish sufficient conditions for diametrically uniform graphs to have the minimum spanner at least 2λ–1. We identify several examples of diametrically uniform graphs and we study the properties of those graphs. We also derive conditions under which the minimum spanner of diametrically uniform graphs is 2λ–1 or 2λ.

We solve the minimum tree spanner problem for the generalized Petersen graphs and some Cayley graphs such as hypercubes, CCC, torus, wrapped butterfly etc. Another interesting family of diametrically uniform graphs is a 3-regular mesh of trees. We show how to add a minimum number of edges to convert a mesh of trees into a diametrically uniform graph. Then we determine the minimum spanner of 3-regular mesh of trees. It is interesting to see that a 3-regular mesh of trees is a diametrically uniform graph but not a vertex transitive graph.

3 A Study of Diametrically Uniform Graphs

Most of the well-known parallel architectures are diametrically uniform graphs. For example, hypercube, wrapped butterfly, torus and cycle are diametrically uniform graphs. An even Petersen graph $P(2n, 2)$ is a diametrically uniform graph whereas an odd Petersen graph $P(2n+1, 2)$ is not a diametrically uniform graph. See Figures 4(a) and 4(b). Here we list a few characterizations of diametrically uniform graphs.

Theorem 1. A graph G is diametrically uniform if and only if $\rho = \lambda$. □

Theorem 2. A graph G is diametrically uniform if and only if $Z(G) = G$. □

3.1 A Class of Tree (2λ–1) - Spanner Admissible Graphs

We begin with a necessary condition for tree (2λ–1) - spanner admissible graphs.

Theorem 3. Let λ be the diameter of a graph G. If $\zeta(G) \geq 2\lambda$–1, then G is diametrically uniform.

Proof: Suppose that G is not diametrically uniform. Then there exists at least one vertex u_0, which has no diametrically opposite vertex. Consider the *bfs* tree $BFS(u_0)$ rooted at u_0. Then the distance, in $BFS(u_0)$, of every vertex from u_0 is at most λ–1. Thus $\zeta(BFS(u_0), G) \leq 2\lambda$–2 and hence $\zeta(G) \leq 2\lambda$–2, a contradiction. □

The next question is: Which diametrically uniform graphs have spanner at least 2λ–1?

Theorem 4. Let G be a diametrically uniform graph with diameter $\lambda > 1$. Given an edge (x, y) in $E(G)$, if for every vertex x^* of $D(x)$ there exists a vertex y^* of $D(y)$ such that (x^*, y^*) is an edge of G, then $\zeta(G) \geq 2\lambda$–1.

Proof: Suppose there exists a spanning tree T such that

$$d_T(x, y) < 2\lambda\text{–}1, \text{ for every edge } (x, y) \text{ in } E(G) \qquad (1)$$

Let us assume that T is a rooted tree. Given a vertex α and a member α^* of $D(\alpha)$ such that α^* is a descendent of α in T, we claim that the vertex α has a child β such that the subtree rooted at β contains α^* as well as a member β^* of $D(\beta)$. Let β be a child of α such that the subtree rooted at β contains α^*. This is possible since $\lambda > 1$. Since (α, β) is in $E(G)$, by the hypothesis of the theorem, there exists a vertex β^* of $D(\beta)$ such that (α^*, β^*) is in $E(G)$. Now it is enough to prove that α^* and β^* are in the same subtree rooted at β (that is, α^* and β^* are descendents of β). Suppose that this is false. Then α^* and β^* lie in two different components of $T - \alpha$ and $d_T(\alpha^*, \beta^*) \geq 2\lambda$–1. See Figure 1. This is a contradiction to (1), since (α^*, β^*) is in $E(G)$. This means that given α and a member α^* of $D(\alpha)$ such that α^* is a descendent of α in T, there exists a child β of α such that the subtree rooted at β contains α^* as well as a member β^* of $D(\beta)$.

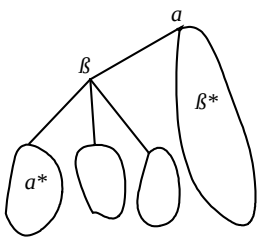


Fig.1. These α* and β* will contradict (1) since (α*, β*) is in E(G).

We start at the root of T and traverse over T in a *DFS* order. Let γ denote the root of T. Once a vertex x of T is visited, a child y of x is identified and visited inductively. Now let us start from the root γ of T. Let $\gamma^* \in D(\gamma)$. A member of $D(\gamma)$ exists since G is diametrically uniform.

Now given γ and a member γ^* of $D(\gamma)$ in the subtree rooted at γ, as discussed earlier we can find a child δ of γ such that the subtree rooted at δ contains γ^* as well as a member δ^* of $D(\delta)$. From γ, it traverses to δ. Thus the *DFS* traversal starts at γ and then it visits δ, a child of γ.

Inductively let x be the last visited vertex and x^* be a member of $D(x)$ which is a descendent of x. As we have shown above, there exists a child y of x such that the

subtree rooted at y contains x^* as well as a member y^* of $D(y)$. Hence the *DFS* traversal never reaches a leaf and does not terminate, which is not possible in a finite tree T. Hence (1) is not true. □

Remark 1. The conditions of Theorem 4 are not enough to determine $\zeta(G)$. That is, a graph satisfying the conditions of Theorem 4 may have the minimum spanner either 2λ–1 or 2λ. For example, the graphs in Figure 2 satisfy conditions of Theorem 4. But the graphs in Figures 2(a) and 2(b) have minimum spanner 2λ–1 whereas the graph in Figure 2(c) has minimum spanner 2λ.

The proof of the following theorem is analogous to that of Theorem 4.

Theorem 5. Let G be a diametrically uniform graph with diameter $\lambda > 1$. If $D(x)$ U $D(y)$ is connected for every edge (x, y) of $E(G)$, then $\zeta(G) \geq 2\lambda$–1. □

Remark 2. The conditions of Theorem 5 are not enough to determine $\zeta(G)$. Both even and odd cycles satisfy Theorem 5. The minimum spanner of an even cycle is 2λ–1 whereas the minimum spanner of an odd cycle is 2λ.

3.2 Upper Bound for the Minimum Spanner of Diametrically Uniform Graphs

Now we identify a few conditions under which the minimum spanner of a diametrically uniform graph is at most 2λ–1. A subgraph H of a bipartite graph G would be a $2k$-spanner of G if and only if it is a $(2k–1)$ - spanner of G [15]. Thus we have the following result.

Theorem 6. If G is a bipartite graph then $\zeta(G)$ is odd. In particular $\zeta(G) \leq 2\lambda$–1. □

The following result is fairly straightforward.

Theorem 7. Let G be a diametrically uniform graph. If there exists a vertex x of G and a *bfs* tree $BFS(x)$ rooted at x such that all the vertices of $D(x)$ are in a subtree rooted at some vertex y $(y \neq x)$ in $BFS(x)$, then $\zeta(G) \leq 2\lambda$–1. □

The following corollaries are direct applications of Theorem 7.

Corollary 1. Let G be a diametrically uniform graph. If for some vertex x of G, $D(x)$ is an independent set, then $\zeta(G) \leq 2\lambda$–1.□

Corollary 2. Let G be a diametrically uniform graph. If $D(x)$ is a singleton for some vertex x of G, then $\zeta(G) \leq 2\lambda$–1. □

Theorem 8. Let G be a diametrically uniform graph. The following statements are equivalent:

1. There exists a vertex x of G and a *bfs* tree $BFS(x)$ rooted at x such that all the vertices of $D(x)$ are in a subtree rooted at some vertex y $(y \neq x)$ in $BFS(x)$.

2. There exists a vertex x of G such that for every member x^* of $D(x)$, a shortest path $P(x, x^*)$ between x and x^* traverses some fixed vertex y ($y \neq x$). □

Theorems 7 and 8 yield the following result.

Theorem 9. Let G be a diametrically uniform graph. If there exists a vertex x of G such that for every member x^* of $D(x)$, a shortest path $P(x, x^*)$ between x and x^* traverses some fixed vertex y ($y \neq x$), then $\zeta(G) \leq 2\lambda - 1$. □

3.3 A Class of Tree 2λ - Spanner Admissible Graphs

Here we begin with a necessary condition for a diametrically uniform graph G with minimum spanner exactly 2λ.

Theorem 10. Let G be a diametrically uniform graph. If $\zeta(G) = 2\lambda$, then for each vertex x of G, $D(x)$ has at least two vertices y and z such that (y, z) is an edge of G. □

The next theorem provides a sufficient condition for tree 2λ - spanner admissible graphs.

Theorem 11. Let G be a diametrically uniform graph with diameter $\lambda > 1$. Suppose that (i) $D(x)$ is connected for every x of $V(G)$ and (ii) $D(x) \cap D(y)$ is non-empty for every edge (x, y) of $E(G)$. Then $\zeta(G) = 2\lambda$. □

Corollary 3. The minimum spanner of a complete graph and an odd cycle is 2λ. □

4 Classes of Graphs with Minimum Spanner $2\lambda - 1$ or 2λ

4.1 Cayley Graphs

The study of Cayley graphs has gained importance in recent times [2, 13, 10]. We look at Cayley graphs as diametrically uniform graphs. The following theorem motivates us to investigate whether vertex transitive graphs are diametrically uniform.

Proposition 1 [8, 11]. Cayley graphs are vertex transitive graphs. □

Theorem 12. Vertex transitive graphs are diametrically uniform graphs.
Proof: Let u and v be two vertices of a vertex transitive graph G such that $d(u, v) = \lambda$. Let x be any arbitrary vertex of G. We claim that $D(x)$, the set of diametrically opposite vertices of x, is non-empty. Let f be an automorphism of G such that $f(u) = x$. Then $d(u, v) = d(f(u), f(v))$ [8, 11]. That is, $d(x, f(v)) = \lambda$. Thus $f(v)$ belongs to $D(x)$ and hence $D(x)$ is non-empty. □

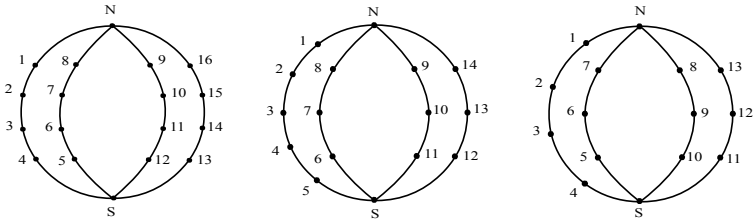


Fig.2.

Theorem 13. A graph *G* is diametrically uniform if and only if any two *bfs* trees of *G* are of equal height.
Proof: Let *G* be diametrically uniform. Then each vertex *u* of *G* has at least one diametrically opposite vertex and the height of a *bfs* tree *BFS*(*u*) is equal to the diameter of *G*. Conversely if any two *bfs* trees of *G* are of the same height say *h*, then G is diametrically uniform with diameter *h*. □

Corollary 4. Any two *bfs* trees of a vertex transitive graph are of equal height.

Lemma 1. Let *u* and *v* be two vertices of a vertex transitive graph *G*. Then for each *bfs* tree *BFS*(*u*), there exists some *bfs* tree *BFS*(*v*) such that *BFS*(*u*) and *BFS*(*v*) are isomorphic. □

The above lemma leads to the following conclusion.
Theorem 14. If *G* is a vertex transitive graph, then *D*(*u*) and *D*(*v*) have the same cardinality for any two vertices *u* and *v* of *G***.** □

Remark 3. The above theorem is useful in identifying non-vertex transitive graphs. For example, a 3-regular mesh of trees is not a vertex transitive graph. Refer to Theorem 15.

4.1.1 Wrapped Butterflies

We consider the class of wrapped butterflies $WB(k, k^r)$ [8, 11, 12].

Observation 1. A wrapped butterfly *G* is diametrically uniform. □

Observation 2. Let G be $WB(2, 2^r)$, where *r* is odd. Then
1. For every edge (*x*, *y*) of *G*, *D*(*x*) U *D*(*y*) is connected.
2. There exists a vertex *x* of *G* such that for every member *x** of *D*(*x*), a shortest path *P*(*x*, *x**) between *x* and *x** traverses some fixed vertex *y* ($y \neq x$). □

Observation 3. Let G be $WB(2, 2^r)$, where *r* is even. Then
1. For every edge (*x*, *y*) of *G*, *D*(*x*) U *D*(*y*) is connected.
2. For every vertex *x* of *G*, *D*(*x*) is a singleton. □

Proposition 2. Let G be $WB(2, 2^r)$. Then $\zeta(G) = 2\lambda-1$.
Proof: If r is odd the proof follows from the observations 1, 2, Theorems 5 and 9. If, on the other hand r is even, the proof follows from the observations 1, 3, Theorem 5 and Corollary 2. □

4.1.2 Torus

Proposition 3. [9]. The diameter of $TR(n, m, \ell)$ is $\lfloor n/2 \rfloor + \lfloor m/2 \rfloor + \lfloor \ell/2 \rfloor$.□

Observation 4. [9]. A torus $TR(n, m, \ell)$ is a diametrically uniform graph. The diametrically opposite vertices of vertex (1,1,1) of the torus are $(\lceil n/2 \rceil, \lceil m/2 \rceil, \lceil \ell/2 \rceil)$; $(\lfloor n/2 \rfloor, \lceil m/2 \rceil, \lceil \ell/2 \rceil)$;$(\lceil n/2 \rceil, \lfloor m/2 \rfloor, \lceil \ell/2 \rceil)$;$\lceil n/2 \rceil, \lceil m/2 \rceil, \lfloor \ell/2 \rfloor)$;$(\lceil n/2 \rceil, \lfloor m/2 \rfloor, \lfloor \ell/2 \rfloor)$; $(\lfloor n/2 \rfloor, \lceil m/2 \rceil, \lfloor \ell/2 \rfloor)$; $(\lfloor n/2 \rfloor, \lfloor m/2 \rfloor, \lceil \ell/2 \rceil)$; $(\lfloor n/2 \rfloor, \lfloor m/2 \rfloor, \lfloor \ell/2 \rfloor)$.□

Observation 5. Let G be $TR(2n, 2m, 2\ell)$. Then (i) $D(x) \cup D(y)$ is connected for every edge (x, y) of G and (ii) $D(x)$ is a singleton set for every vertex x of G.□

The observations 4, 5, Theorem 5 and Corollary 2 yield the following result.
Proposition 4. The minimum spanner of $TR(2n, 2m, 2\ell)$ is $2\lambda-1$. □

Observation 6. Let G be $TR(2n+1, 2m+1, 2\ell+1)$.Then (i) $D(x)$ is connected for every vertex x of G and (ii) $D(x) \cap D(y)$ is non-empty for every edge (x, y) of G. □

The observations 4, 6 and Theorem 11 give rise to the following result.
Proposition 5. The minimum spanner of $TR(2n+1, 2m+1, 2\ell+1)$ is 2λ. □

Observation 7. Let G be $TR(2n, 2m+1, 2\ell)$. Then (i) $D(x) \cup D(y)$ is connected for every edge (x, y) of G and (ii) there exists a vertex x of G such that for every member x^* of $D(x)$, a shortest path $P(x, x^*)$ between x and x^* traverses some fixed vertex y ($y \neq x$). □

The observations 4, 7, Theorems 5 and 9 give rise to the following result.
Proposition 6. The minimum spanner of $TR(2n, 2m+1, 2\ell)$ is $2\lambda-1$. □

4.1.3 Hypercubes

A hypercube Q^n [8, 11, 12] is an n- regular graph on 2^n vertices and its diameter is n.

Observation 8. A hypercube Q^n is a diametrically uniform graph. Moreover, if G is Q^n, then

1. $D(x) \cup D(y)$ is connected, for every edge (x, y) of G.
2. $D(x)$ is a singleton set, for every vertex x of G. □

Notice that an even cycle also satisfies observation 8. Hence observation 8, Theorem 5 and Corollary 2 imply the following result.

Proposition 7. The minimum spanner of the hypercube Q^n and even cycle is $2\lambda-1$. □

4.1.4 Cube-Connected Cycles

A cube-connected cycle $CCC(n)$ is a graph obtained from a hypercube Q^n on 2^n vertices by replacing each vertex by an n-cycle [8, 11, 12].

Observation 9. A cube-connected cycle $CCC(n)$ is a diametrically uniform graph and the diameter of $CCC(n)$ is $2n$. Moreover, if G is $CCC(n)$, then

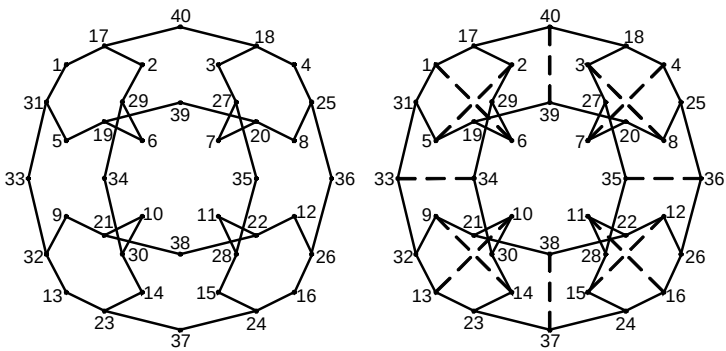
1. $D(x)$ U $D(y)$ is connected, for every edge (x, y) of G.
2. $D(x)$ is an independent set, for every vertex x of G. □

The following result is a consequence of observation 9, Theorem 5 and Corollary 1.
Proposition 8. The minimum spanner of the cube connected cycle $CCC(n)$ is $2\lambda-1$. □

4.2 The 3-Regular Mesh of Trees

Let G be a 2^n x 2^n mesh of trees [12]. See Figure 3(a). We modify G by adding new edges to G, depicted by dotted lines in Figure 3(b), so that the modified graph is 3-regular. We denote the modified graph by $MT(n)$. The diameter of a 3-regular $MT(n)$ is $4n$ and it has $2^n(2^{n+1} + 2^n - 2)$ vertices, for $n \geq 2$.

The graph in Figure 3(b) is a 3-regular 4 x 4 mesh of trees. Its diameter is 8 and $D(1) = D(2) = D(5) = D(6) = \{11,12,15,16\}$, $D(3) = D(4) = D(7) = D(8) = \{9,10,13,14\}$, $D(17) = D(19) = \{22,24\}$, $D(22) = D(24) = \{17,19\}$.



(a) Mesh of Trees (b) 3-regular Mesh of Trees

Fig. 3. A Mesh of trees

Observation 10. Let G be a 3-regular $MT(n)$. Then G is a diametrically uniform graph. Moreover, (i) given an edge (x, y) of G, for every vertex x^* of $D(x)$ there exists

a vertex y^* of $D(y)$ such that (x^*, y^*) is an edge of G and (ii) there exists some vertex x, for which $D(x)$ is an independent set. □

The following result is derived from observation 10, Theorem 4 and Corollary 1.
Proposition 9. The minimum spanner of a 3-regular $MT(n)$ is 2λ–1. □

Here we demonstrate a simple application of Theorem 14.
Theorem 15. A 3-regular mesh of trees $MT(n)$ is not vertex transitive.
Proof: By the structure of a 3-regular mesh of trees there are vertices u and v such that $|D(u)| \neq |D(v)|$. Then the conclusion follows from Theorem 14. □

4.3 Petersen Graphs

A *generalized Petersen graph* $P(n, m)$, $n \geq 3$, $1 \leq m \leq \lfloor (n\text{-}1)/2 \rfloor$ consists of an outer n-cycle $u_1, u_2 \ldots u_n$, a set of n spokes (u_i, v_i), $1 \leq i \leq n$ and n inner edges (v_i, v_{i+m}) with indices taken modulo n. For convenience, $u_1, u_2 \ldots u_n$ are represented by 1, 2 … n and $v_1, v_2 \ldots v_n$ by n+1, n+2 … 2n respectively. In this paper we consider Petersen graphs with $m = 2$ and call a generalized Petersen graph $P(n, 2)$ simply a Petersen graph.

Observation 11. A Petersen graph $P(2n, 2)$ is a diametrically uniform graph whereas $P(2n+1, 2)$ is not diametrically uniform and the diameter of $P(n, 2)$ is given by $\lambda = \lfloor (n\text{-}6)/4 \rfloor + 4$, $n \geq 8$. □

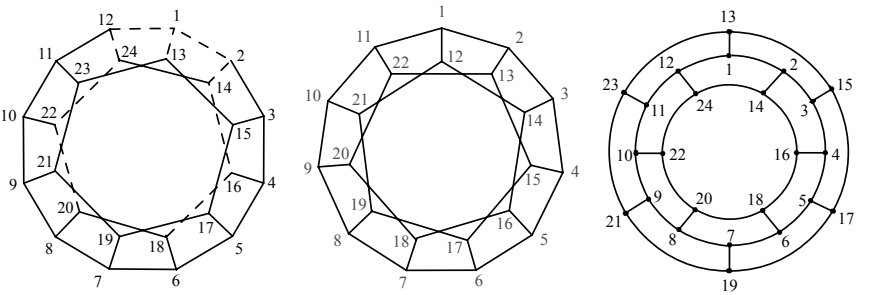
Observation 12. Let G be $P(4n, 2)$, $n \geq 2$. Then
1. For every edge (x, y) of G, $D(x) \cup D(y)$ is connected.
2. There exists a vertex x of G such that for every member x^* of $D(x)$, a shortest path $P(x, x^*)$ between x and x^* traverses some fixed vertex y $(y \neq x)$. □

For the graph in Figure 4(a), $D(1) = \{6,7,8\}$, $D(2) = \{7,8,9\}$, D(13) = {18,20}, D(14) = {19,13} and the shortest paths P(13, 20) = (13, 1, 12, 24, 22, 20) and P(13, 18) = (13, 1, 2, 14, 16, 18) traverse a fixed vertex 1. For the graph in Figure 4(b), D(1) = {6,7} but D(12) = Φ, justifying that odd Petersen graphs are not diametrically uniform.

Observation 13. Let G be $P(4n+2, 2)$, $n \geq 2$. Then
1. For every edge (x, y) of G, $D(x) \cup D(y)$ is connected.
2. For every vertex x of G, $D(x)$ is a singleton. □

The observations 11, 12, Theorems 5 and 9 yield the following result.
Proposition 10. If $G = P(4n, 2)$, $n \geq 2$, then $\zeta(G) = 2\lambda$–1. □

The next theorem follows from observations 11, 13, Theorem 5 and Corollary 2.
Proposition 11. If $G = P(4n+2, 2)$, $n \geq 2$, then $\zeta(G) = 2\lambda$–1. □
As in the case of 3-regular mesh of trees $MT(n)$, we prove that Petersen graphs $P(4n, 2)$ are not vertex transitive.



(a) P(12,2) (b) P(11,2) (c) P(12,2)

Fig.4.

Theorem 16. The Petersen graphs $P(4n, 2)$, $n \geq 2$, are not vertex transitive.
Proof: If u and v are outer and inner cycle vertices, respectively, of G, we have $|D(u)| \neq |D(v)|$. Then we have the desired conclusion from Theorem 14. □

Graph theorists agree that Petersen graphs are known for their versatility to stand as counter examples. Petersen graphs never cease to amaze the researchers for their unpredictable behaviors. Here we would like to share some of its interesting structural behaviors. The Petersen graphs $P(2n+1, 2)$, $n \geq 3$, are not diametrically uniform whereas the Petersen graphs $P(2n, 2)$, $n \geq 4$, are diametrically uniform. Petersen graph $P(5, 2)$ is vertex transitive but not Cayley [8, 19]. We just proved that $P(4n, 2)$), $n \geq 2$, is not vertex transitive. Here is the most interesting behavior of Petersen graphs. The minimum spanner of Petersen graphs $P(2n, 2)$, $n \geq 4$, is exactly $2\lambda-1$ and the minimum spanner of Petersen graphs $P(2n+1, 2)$ is strictly less than $2\lambda-1$. At this point we recall a similar property between odd and even Petersen graphs. That is, an even Petersen graph is planar (see Figure 4(c)) whereas an odd Petersen graph is non-planar. The reader may wonder whether there is any relationship between the planarity, minimum spanner property and the diametrically uniform property. This requires further investigation and is an open problem.

5 Conclusion

We have found a unified way to solve the minimum tree spanner problem for Petersen graphs, 3-regular mesh of trees, Cayley graphs such as wrapped butterfly, torus, hypercube, CCC etc. The properties of these graphs have been listed as observations. Most of the observations of these graphs need mathematical proof. There are too many observations to prove one by one mathematically. Since this will shift the focus from the main objective of the paper, we leave it to the reader to verify the observations mentioned in this paper.

References

1. Alfred V. Aho, John E. Hopcroft and Jeffry D. Ullman, Data Structures and Algorithms, Addison Wesley Longman Inc.,(2000).
2. S.B. Akers and B. Krishnamurthy, A group-theoretic model for symmetric interconnection networks, IEEE Trans Computers 38, (1989), 555–566.
3. L. Cai, Tree Spanners: Spanning trees that approximate distances, *Tech. Rept.* 260/92, Dept. of Computer Science, University of Toronto.
4. L. Cai and D.G. Corneil Tree Spanners, SIAM J. Discrete Math., 8, (1995), 359–387.
5. L. Cai and D.G. Corneil, Isomorphic tree spanner problems, Algorithmica, 14, (1995), 138–153.
6. E. Cohen, Fast Algorithms for constructing t-spanners and paths with stretch t, SIAM J. computing, 28, (1998), 210–236.
7. S.P. Fekete and J. Kremer, Tree Spanners in Planar Graphs, 24th International Workshop on Graph- Theoretic Concepts in Computer Science (WG '98) Smolenice-Castle/Slovakia, June 18 – 20, 1998.
8. M.C. Heydemann, Cayley graphs and interconnection networks, In Graph Symmetry, eds. G. Hahn and G. Sabidussi, Kluwer Academic Publishers, The Netherlands, (1997), 167–224.
9. Y. Ishigami, The wide-diameter of the n-dimensional toroidal mesh, Networks, 27, (1996), 257–266.
10. S. Lakshmivarahan, J.S. Jwo and S.K. Dhall, Symmetry in interconnection networks based on Cayley graphs of permutations groups: a survey, Parallel Computing, 19, (1993), 361–407.
11. S. Lakshmivarahan and S.K. Dhall, Rings, torus, and hypercubes architectures/algorithms for parallel computing, Parallel Computing, 25, (1999), 1877–1905.
12. F.T. Leighton, Introduction to parallel algorithms and architectures: Arrays, Trees, Hypercubes, Morgan Kaufmann Publishers, San Mateo, California, 1991.
13. Hoang-Oanh Le and Van Bang Le, Optimal tree 3-spanners in directed path graphs, Networks, 34, (1999), 81–97.
14. M. S. Madanlal, G. Venkatesan and C. Pandu Rangan, Tree 3-spanners on interval, permutation and regular bipartite graphs, *Information Processing Letters* 59, (1996), 97–102.
15. G. Venkatesan, U. Rotics, M.S. Madanlal, J.A.Makowski, C. Pandu Rangan, Restrictions of minimum spanner problem, Information and Computation, 136, (1997), 143–164.

The Probabilistic Minimum Coloring Problem
(Extended Abstract)

Cécile Murat and Vangelis Th. Paschos

LAMSADE, Université Paris-Dauphine, Place du Maréchal De Lattre de Tassigny,
75775 Paris Cedex 16, France,
{murat,paschos}@lamsade.dauphine.fr

Abstract. We study the probabilistic coloring problem (PCOLOR) under a modification strategy consisting, given an a priori solution C, of removing the absent vertices from C. We compute the objective function associated with this strategy, we give bounds on its value, we characterize the complexity of computing it and the one of computing the optimal solution associated with. We show that PCOLOR is **NP**-hard and design a polynomial time approximation algorithm achieving non-trivial approximation ratio. We then show that probabilistic coloring remains **NP**-hard even in bipartite graphs and that the unique 2-coloring in such graphs is a constant ratio approximation. We finally prove that PCOLOR is polynomial when dealing with complements of bipartite graphs.

1 Introduction

Consider a graph $G(V, E)$ of order n. In *minimum coloring problem*, we wish to color V with as few colors as possible so that no two adjacent vertices receive the same color. The *chromatic number* of a graph, denoted by $\chi(G)$, is the smallest number of colors that can feasibly color its vertices. A graph G is called *k-colorable* if its vertices can be legally colored by k colors, in other words if its chromatic number is at most k; it will be called *k-chromatic* if k is its chromatic number. Minimum coloring was shown to be NP-hard in Karp's original paper ([1]), and remains NP-complete even restricted to graphs of constant (independent on n) chromatic number at least 3 ([2]). Since adjacent vertices are forbidden to be colored with the same color, a feasible coloring can be seen as a partition of V into vertex-sets such that, for each one of these sets, no two of its vertices are mutually adjacent. Such sets are usually called *independent sets*. So, the optimal solution of minimum coloring is a *minimum-cardinality partition into independent sets*.

In this paper we deal with a probabilistic version of minimum coloring, denoted by PCOLOR in what follows. Consider any set $V' \subseteq V$ and set $G' = G[V']$, the subgraph of G induced by V'. In PCOLOR we are given: (i) a graph $G(V, E)$, and an n-vector $\mathbf{Pr} = (p_1, \ldots, p_n)$ of vertex-probabilities; in other words, an instance of PCOLOR is a pair $(G, \mathbf{Pr})$; (ii) a coloring $C = (S_1, \ldots, S_k)$ for V; (iii) a *modification strategy* M, i.e., an algorithm receiving C and G' as inputs

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 346–357, 2003.

and modifying C in order to produce a coloring C' for G'. The objective is to determine a coloring C^* of G minimizing the quantity (commonly called functional) $E(G, C, \mathtt{M}) = \sum_{V' \subseteq V} \Pr[V'] |C(V', \mathtt{M})|$ where $C(V', \mathtt{M})$ is the solution computed by $\mathtt{M}(C, V')$ (i.e., by M when executed with inputs the a priori solution C and the subgraph of G induced by V') and $\Pr[V'] = \prod_{i \in V'} p_i \prod_{i \in V \setminus V'} (1 - p_i)$ (there exist 2^n distinct sets V'; therefore, explicit computation of $E(G, C, \mathtt{M})$ is, a priori, not polynomial). The complexity of PCOLOR is the complexity of computing C^*. Using the terminology introduced in [3,4], we will call solution C (in which the modification strategy is applied) an *a priori solution*; C^* is then the optimal a priori solution.

The fact that strategy M intervenes in the formulation of the functional seems somewhat unusual with respect to standard complexity theory where no algorithm intervenes in the definition of the objective function of a problem. But *M is absolutely not an algorithm for* PCOLOR*, in the sense that it does not compute a coloring for* G; it simply fits C (no matter how it has been computed) to $G[V']$. This also implies that changing M one changes the definition of PCOLOR itself. Strictly speaking, the PCOLOR-variants induced by PCOLOR($\mathtt{M}_1$) and PCOLOR($\mathtt{M}_2$) (parameterized by two distinct modification strategies $\mathtt{M}_1$ and $\mathtt{M}_2$) are two *distinct* probabilistic combinatorial optimization problems.

A priori optimization, i.e., searching for optimal a priori solutions of probabilistic combinatorial optimization problems, has been studied in restricted versions of routing and network-design probabilistic minimization problems ([5,6, 7,3,4,8,9,10]), defined on complete graphs. Also, in [11] the minimum vertex covering problem in general and in bipartite graphs is studied. Finally, a priori optimization has been used in [12,13] to study probabilistic maximization problems, the longest path and the maximum independent set, respectively.

In what follows, we study PCOLOR(M) under the following simple but intuitive modification strategy M: `given an a priori solution C, take` $\mathtt{C} \cap \mathtt{V}'$ `as solution for` $\mathtt{G}[\mathtt{V}']$. In what follows, since M is fixed for the rest of the paper, we simplify notations using PCOLOR instead of PCOLOR(M).

In [14], we motivate the study of this problem by two real-world applications dealing with timetabling (the first one) and with satellite shots planning under meteorological uncertainty (the second one), showing that it is not simply a toy (even nice and pleasant) problem. Then, we are managed to compute the functional, to give bounds on its value, to characterize the complexity of computing it and the one of computing the optimal a priori solution associated with. We show that PCOLOR is **NP**-hard, so we try to face it with polynomial time approximation algorithms achieving non-trivial approximation ratios. We then restrict ourselves to bipartite graphs and show that PCOLOR is always **NP**-hard; we also prove that the unique 2-coloring in bipartite graphs achieves approximation ratio 2.773 when used as a priori solution. We finally show that PCOLOR is polynomial in bipartite complements of perfect matchings, under identical vertex-probabilities, and in complements of bipartite graphs, without restriction on vertex-probabilities. Complete proofs of all these results are given in appendix.

Let $\mathtt{A}$ be a polynomial time approximation algorithm for an **NP**-hard graph-problem Π, let $\mathtt{A}(G)$ be the value of the solution provided by $\mathtt{A}$ on a graph G instance of Π, and $\mathrm{OPT}(G)$ be the value of the optimal solution for G (following our notation for PCOLOR, $\mathrm{OPT}(G) = E(G, C^*, M)$). The approximation ratio $\rho_{\mathtt{A}}(G)$ of the algorithm $\mathtt{A}$ on graph G is defined as $\rho_{\mathtt{A}}(G) = \mathtt{A}(G)/\mathrm{OPT}(G)$. An approximation algorithm achieving ratio, at most, ρ on any instance of Π will be called ρ-approximation algorithm.

Since the modification strategy $\mathtt{M}$ is fixed for the rest of the paper we will simplify notations by using $E(G, C^*)$ instead of $E(G, C^*, \mathtt{M})$ and $C(V')$ instead of $C(V', \mathtt{M})$. Moreover, we shall denote by $p_{\max}$ (resp., $p_{\min}$), the maximum (resp., minimum) vertex-probability of V.

2 The Probabilistic Coloring in General Graphs

2.1 The Complexity of Probabilistic Coloring

In this section, we first analytically express the functional for PCOLOR; then, based upon it, we show that it can be computed in polynomial time[1] . Moreover, always based upon the analytical expression obtained for the functional, we give a combinatorial characterization of the optimal a priori solution.

Recall that given an a priori solution $C = (S_1, S_2, \ldots, S_k)$ of cardinality k and a subgraph G' of G induced by a subset V' of V, the modification strategy adopted in the paper consists of removing the vertices in $C \setminus V'$. Denote by $C(V')$ the coloring of G' so-obtained and set $k' = |C(V')|$. Denote also by $\mathbf{1}_{\mathsf{F}}$, the indicator function of a fact F. Then, $E(G, C) = \sum_{V' \subseteq V} \mathbf{Pr}[V']|C(V')| = \sum_{V' \subseteq V} \mathbf{Pr}[V']k'$. Consider the facts $\mathsf{F_j}$: color $\mathsf{S_j}$ has at least a vertex and $\bar{\mathsf{F}}_{\mathsf{j}}$: there is no vertex in color $\mathsf{S_j}$; then k' can be written as $k' = \sum_{j=1}^{k} \mathbf{1}_{\mathsf{F_j}} = \sum_{j=1}^{k}(1 - \mathbf{1}_{\bar{\mathsf{F}}_{\mathsf{j}}})$. Therefore, $E(G, C) = \sum_{V' \subseteq V} \mathbf{Pr}[V'] \sum_{j=1}^{k} 1 - \sum_{V' \subseteq V} \mathbf{Pr}[V'] \sum_{j=1}^{k} \mathbf{1}_{\mathsf{S_j} \cap \mathsf{V}' = \emptyset} = k - \sum_{j=1}^{k} \prod_{v_i \in S_j}(1 - p_i) = \sum_{j=1}^{k}(1 - \prod_{v_i \in S_j}(1 - p_i))$.

It is easy to see that computation of $E(G, C)$ takes at most $O(n)$ steps, consequently, PCOLOR $\in$ **NP**. On the other hand, from the final expression for $E(G, C)$, we can easily characterize the optimal a priori solution C^* for PCOLOR: if the value of an independent set S_j of G is $1 - \prod_{v_i \in S_j}(1 - p_i)$, then *the optimal a priori coloring for G is the partition into independent sets for which the sum of their values is the smallest over all such partitions.* We finally note that if we assume $p_i = 1$, $i = 1, \ldots, n$, then PCOLOR becomes the classical coloring problem. This observation immediately deduces the **NP**-hardness of PCOLOR.

Proposition 1. PCOLOR *is **NP**-hard.*

We are here faced to a graph-problem completely different from the ones studied in [13,11]. There, when strategies as $\mathtt{M}$ were used, i.e., strategies consisting of

[1] Recall that, as we have already mentioned in section 1, the functional is not a priori polynomially computable for any problem and for any modification strategy.

dropping absent vertices out of the a priori solution, the optimal a priori solutions were a maximum weight independent set, or a minimum weight vertex-covering, of the input graph, considering that its vertices receive their probabilities as weights; here, the weight of an independent set is not an additive function. Let us mention that there exist also weighted versions of the minimum coloring. For example, one can consider that the weight of a color is the maximum (or the minimum, or even, the average) weight of the vertices in the independent set representing it, and the objective is to find a coloring minimizing the sum of the weights of the colors (see, for example, [15,16] for a version of weighted coloring, where the weight of a color is the maximum over the weights of its vertices). The valued coloring dealt here is closer to the so-called chromatic sum problem ([17, 18,19]) than to the weighted coloring in [15,16].

2.2 Bounds on $E(G, C)$

This section gives some bounds for $E(G, C)$ that are used for the achievement of the approximation results presented later. Consider a coloring $C = (S_1, \ldots, S_k)$, fix a $j \in \{1, \ldots, k\}$, assume, for simplicity, $|S_j| = \ell$ and arbitrarily denote vertices in S_j by $v_1, \ldots, v_\ell$. Then, by induction in ℓ the following holds:

$$\sum_{i=1}^{\ell} p_i - \sum_{i=1}^{\ell} \sum_{j=i+1}^{\ell} p_i p_j \leqslant 1 - \prod_{i=1}^{\ell} (1 - p_i) \leqslant \sum_{i=1}^{\ell} p_i \tag{1}$$

Take the sums of members of (1) for $m = 1$ to k. It can be proved that

$$\sum_{i=1}^{n} p_i - \sum_{i=1}^{n} \sum_{j=i+1}^{n} p_i p_j \leqslant E(G, C) \leqslant \sum_{i=1}^{n} p_i \tag{2}$$

Finally, (3) gives additional bounds for $E(G, C)$.

$$\max \left\{ \sum_{j=1}^{k} \left(1 - \exp \left\{ - \sum_{v_i \in S_j} p_i \right\} \right), k p_{\min} \right\} \leqslant E(G, C) \leqslant \min \{k, n p_{\max}\} \tag{3}$$

2.3 An Approximation Algorithm for PCOLOR in General Graphs

We have already mentioned that, in the problem we deal with, the weight of an independent set is not an additive function. This makes it harder to be faced by approximation algorithms than the probabilistic problems dealt in [13,11]. In this section, we devise and analyze an approximation algorithm for general PCOLOR.

Let us, in a first time, deal with fixed vertex-probabilities, i.e., with probabilities such that $p_{\min} \geqslant t$, for some t. Then, the following lemma, that will be used later, holds.

Lemma 1. *Assume a graph of order n and with vertex-probability vector* $\mathbf{Pr}$*. If* $p_{\min} \geqslant t$*, then* PCOLOR *is approximable within ratio* $O(n \log^2 \log n/(t \log^3 n))$.

Consider the graph G and two vertex-probabilities p_0 and p', $p_0 < p'$. The main idea of the algorithm for PCOLOR is to partition the vertices of G into three subsets: the first, $V_{\rm sp}$ including the vertices with "small" probabilities, i.e., at most p_0; the second, $V_{\rm ip}$, including the ones with "intermediate" probabilities, i.e., greater than p_0 and at most p'; and the third, $V_{\rm lp}$, including the vertices with "large" probabilities, i.e., greater than p'. Then, it separately colors the vertices of $G[V_{\rm sp}]$, $G[V_{\rm ip}]$ and $G[V_{\rm lp}]$ by using a proper set of colors for any subgraph, and it finally takes the union of the colors used as solution for G. For $G[V_{\rm sp}]$, and $G[V_{\rm ip}]$, we can use, as we will see just below, any polynomial feasible-coloring algorithm; for $G[V_{\rm lp}]$ we will use the algorithm of [20]. For simplicity in the analysis of the algorithm that follows, we fix $p_0 = 1/n$. This has no important impact in the approximation ratio concluded; p' will be fixed later. The following lemma deals with the approximation ratios of the algorithm just sketched in $G[V_{\rm sp}]$, $G[V_{\rm ip}]$.

Lemma 2. ***(The ratios in $G[V_{\rm sp}]$, $G[V_{\rm ip}]$ and $G[V_{\rm lp}]$)***

1. *Any feasible polynomial time approximation algorithm for* PCOLOR *achieves in $G[V_{\rm sp}]$ approximation ratio bounded above by 2.*
2. *Any feasible polynomial time approximation algorithm for* PCOLOR *achieves in $G[V_{\rm ip}]$ approximation ratio $O(np')$.*
3. *By lemma 1, the algorithm of [20], used for* PCOLOR*, achieves in $G[V_{\rm lp}]$ approximation ratio $O(n_{\rm lp} \log^2 \log n_{\rm lp}/p' \log^3 n_{\rm lp}) \leqslant O(n \log^2 \log n/p' \log^3 n)$.*

In what follows, denote by C^* an optimal a priori coloring for PCOLOR in G, by $C^*[V_{\rm sp}]$, $C^*[V_{\rm ip}]$ and $C^*[V_{\rm lp}]$ the solutions induced by C^* in $G[V_{\rm sp}]$, $G[V_{\rm ip}]$ and $G[V_{\rm lp}]$, respectively, and by $C^*_{\rm sp}$, $C^*_{\rm ip}$ and $C^*_{\rm lp}$, $\hat{C}_{\rm sp}$, $\hat{C}_{\rm ip}$ and $\hat{C}_{\rm lp}$ the optimal and approximated a priori solutions in $G[V_{\rm sp}]$, $G[V_{\rm ip}]$ and $G[V_{\rm lp}]$, respectively.

Theorem 1. PCOLOR *is approximable within ratio $O(n \log\log n/\log^{3/2} n)$ in polynomial time.*

Proof (Sketch). We prove that, $\forall x \in \{{\rm sp}, {\rm ip}, {\rm lp}\}$, $E(G, C^*) \geqslant E(G[V_x], C^*[V_x]) \geqslant E(G[V_x], C^*_x)$. Remark first that $C^*[V_x]$ is a particular feasible solution for $G[V_x]$; hence, $E(G[V_x], C^*[V_x]) \geqslant E(G[V_x], C^*_x)$. In order to prove the first inequality, fix an x, consider a color, say S^*_j of C^* and set $|S^*_j| = \ell$. Then, the contribution of S^*_j in $C^*[V_x]$ is $1 - \prod_{v_i \in S^*_j \cap V_x}(1 - p_i) \leqslant 1 - \prod_{v_i \in S^*_j}(1 - p_i)$, which is its contribution in C^*. Iterating this argument for all the colors in $C^*[V_x]$, the claim follows. Recall finally, that the algorithm sketched before theorem's statement, colors the vertices of any $G[V_x]$, $x \in \{{\rm sp}, {\rm ip}, {\rm lp}\}$ with a distinct set of colors and the a priori solution $\hat{C}$ finally provided is the union of these sets. Consequently, $E(G, \hat{C}) = E(G[V_{\rm sp}], \hat{C}_{\rm sp}) + E(G[V_{\rm ip}], \hat{C}_{\rm ip}) + E(G[V_{\rm ip}], \hat{C}_{\rm lp})$. So, using the fact that $E(G, C^*)$ is at least as large as any of $E(G[V_x], C^*_x)$, $x \in \{{\rm sp}, {\rm ip}, {\rm lp}\}$, shown above, one immediately deduces that the overall ratio of the algorithm in G is at most the sum of the ratios provided by items 1, 2 and 3 of lemma 2, i.e., at most

$O(2 + np' + (n \log^2 \log n / p' \log^3 n))$. Remark that the ratio claimed in item 2 is increasing with p', while the one in item 3 is decreasing with p'. Equality of expressions np' and $n \log^2 \log n / p' \log^3 n$ holds for $p' = \log \log n / \log^{3/2} n$. In this case the value of the ratio obtained is $O(n \log \log n / \log^{3/2} n)$.

3 The Restriction of Probabilistic Coloring in Bipartite Graphs

3.1 The Complexity of PCOLOR

In what follows we denote by $B(V, U, E)$ a bipartite graph with bipartition V and U and edge-set E. We first do the following emphasized preliminary observation: *in any bipartite graph, the bipartition (bicoloring) of its vertices is unique.* Another observation is that the unique 2-coloring is not always the best a priori solution PCOLOR in bipartite graphs ([14]).

We prove that PCOLOR is **NP**-hard even in bipartite graphs. For doing this, we first need to introduce a variant of PCOLOR and to prove an initial completeness result that will serve as a basis. We consider the following problem denoting by PCOLOR$(B, \mathbf{Pr}, 3)$. It is a variant of PCOLOR where, given a probabilistic bipartite graph B, we are looking for the best 3-coloring, i.e., the three coloring C^* for which the functional $E(B, C^*)$ associated is the best over the ones of any other 3-coloring of B. The decision version of PCOLOR$(B, \mathbf{Pr}, 3)$, denoted by PCOLOR$(B, \mathbf{Pr}, 3, K)$, is the one where we look for a 3-coloring of functional's value at most K.

Proposition 2. PCOLOR$(B, \mathbf{Pr}, 3, K)$ *is* ***NP****-complete.*

Proof (Sketch). The completeness is proved by reduction from the following problem, called 1-PrExt: *"given a bipartite graph $B(V, U, E)$ with $|V \cup U| \geqslant 3$ and three vertices v_1, v_2, v_3, does there exist a 3-coloring (S_1, S_2, S_3) of B such that $v_i \in S_i$ for $i = 1, 2, 3$?"*, which is **NP**-complete ([21]). Consider an instance $B'(V, U', E', v_1, v_2, v_3)$ of 1-PrExt and remark that we can assume that v_1, v_2, v_3 belong all either to V, or to U'; in the opposite case 1-PrExt is polynomial. Suppose $\{v_1, v_2, v_3\} \in V$. We transform $B'(V, U', E', v_1, v_2, v_3)$ into an instance of PCOLOR$(B, \mathbf{Pr}, 3, K)$ in the following way: add in U' three new vertices u_1, u_2, u_3 and set $U = U' \cup \{u_1, u_2, u_3\}$; add in E' the edge-set $E'' = \{v_i u_j : i, j = 1, 2, 3, i \neq j\}$ and take $E = E' \cup E''$; set $B = B(V, U, E)$; the probability vector $\mathbf{Pr}$ is as follows: $p(u_1) = p(v_1) = \epsilon$, $p(u_2) = p(v_2) = \epsilon^2$, $p(u_3) = p(v_3) = \epsilon^3$, for $\epsilon \leqslant 1/10$, $p(v_i) = 0$, $v_i \in (V \cup U) \setminus \{v_i, u_i : i = 1, 2, 3\}$; finally, set $K = 2\epsilon + \epsilon^2 + 2\epsilon^3 - \epsilon^4 - \epsilon^6$. We claim that $(B, \mathbf{Pr}, 3, K)$ has a 3-coloring with functional at most $2\epsilon + \epsilon^2 + 2\epsilon^3 - \epsilon^4 - \epsilon^6$ iff we can 3-color $B'(V, U', E', v_1, v_2, v_3)$ by assigning any of v_1, v_2, v_3 with a distinct color.

It is easy to see that the contribution of any vertex in $(V \cup U) \setminus \{v_i, u_i : i = 1, 2, 3\}$ in any coloring of B is null. Denote by $\bar{M}_{3,3}$ the graph $B[\{v_i, u_i : i = 1, 2, 3\}]$ (this graph is a kind of bipartite complement of a perfect matching

on 3 edges, in our case on edges v_iu_i, $i = 1, 2, 3$), and observe that the (non-zero) value of any coloring of B is value of some coloring of $\bar{M}_{3,3}$. Observe also that the value of K, introduced in the third item just above, corresponds to a 3-coloring C^* of B taking $\{v_i, u_i\}$, $i = 1, 2, 3$ in the same color, say S_i; this coloring has functional equal to $\sum_{i=1}^{3}(1-(1-\epsilon^i)^2) = 2\epsilon + \epsilon^2 + 2\epsilon^3 - \epsilon^4 - \epsilon^6$. By a simple inspection of any other 3-coloring C of B (there exist 7 distinct functional-value such colorings), one can easily see that the functional of C is, for $\epsilon \leqslant 1/10$, greater than the functional K of C^*.

So, if a 3-coloring C^* of B is polynomially computed with functional $K = 2\epsilon+\epsilon^2+2\epsilon^3-\epsilon^4-\epsilon^6$, then C^* restricted to $\bar{M}_{3,3}$ is of the form $\{v_i, u_i\}$, $i = 1, 2, 3$ (recall that the contribution of the vertices of $(V \cup U) \setminus \{v_i, u_i : i = 1, 2, 3\}$ in any coloring of B is 0). Consequently, C^* 3-colors the vertices of B' by assigning a distinct color to each of v_1, v_2, v_3. Conversely, if a 3-coloring assigning a distinct color, say S_1, S_2 and S_3 to each of v_1, v_2, v_3, respectively, is computed for B', then $(S_1 \cup \{u_1\}, S_2 \cup \{u_2\}, S_3 \cup \{u_3\})$ is a coloring for B with functional $K = 2\epsilon + \epsilon^2 + 2\epsilon^3 - \epsilon^4 - \epsilon^6$.

Remark that the functional of the (unique) 2-coloring of $B[\{v_i, u_i : i = 1, 2, 3\}]$ has value $2(\epsilon + \epsilon^2 - \epsilon^4 - \epsilon^5 + \epsilon^6) > 2\epsilon + \epsilon^2 + 2\epsilon^3 - \epsilon^4 - \epsilon^6$ for $\epsilon \leqslant 1/10$.

We now consider problem PCOLOR$(B, \mathbf{Pr}, k)$, where we look for the best k-coloring for any $k \in \{4, \ldots, n\}$ and its decision version PCOLOR$(B, \mathbf{Pr}, k, K)$. We will establish that PCOLOR$(B, \mathbf{Pr}, k, K)$ is **NP**-complete for any such k.

Consider the bipartite complement $\bar{M}_{k,k}$ of a perfect matching with k edges (i.e., a bipartite graph $B(V, U, E)$ with $|V| = |U| = k$ and with $E = E(B_{k,k}) \setminus \{v_iu_i, v_i \in V, u_i \in U, i = 1, \ldots, k\}$), where by $B_{k,k}$ we denote the complete bipartite graph with $|V| = |U| = k$. Call a color *horizontal* if it is a *proper subset* either of V, or of U; a coloring will be called horizontal if it is composed only of horizontal colors. On the other hand, call a color *vertical* if it contains vertices from both V and U; a coloring of $\bar{M}_{k,k}$ will be called vertical if *all* its colors are vertical; otherwise, it will be called non-vertical. The following properties hold for the colorings of $\bar{M}_{k,k}$:

1. any vertical color of $\bar{M}_{k,k}$ is exclusively of the form $\{v_i, u_i\}$, $i = 1, \ldots, k$; this is easily deduced from the particular form of $\bar{M}_{k,k}$ implying that independent sets $\{v_i, u_i\}$, $i = 1, \ldots, k$ are all maximal for the inclusion;
2. the non-vertical colors of any non-vertical coloring of $\bar{M}_{k,k}$ are horizontal, i.e., there is no coloring of $\bar{M}_{k,k}$ with other than horizontal or vertical colors;
3. for any $i = 1, \ldots, k$, if v_i and u_i belong to two different colors, these colors are horizontal; this is concluded by the fact that v_i excludes any vertex of U (other than u_i), while u_i excludes any vertex of V (other than v_i).

Proposition 3. *Consider $\bar{M}_{k,k}$ and assume that there exists a vertex-probability system* $\mathbf{Pr}$ *with $p(v_i) = p(u_i) = p_i$, $i = 1, \ldots, k$, such that: (i) for any i, $3 \leqslant i < k$, the functional of a vertical coloring of any subgraph $\bar{M}_{i,i}$ of $\bar{M}_{k,k}$ is smaller than the functional of the 2-coloring of $\bar{M}_{i,i}$, and (ii) for any induced subgraph B' of $\bar{M}_{k,k}$, the functional-value of any horizontal coloring is greater*

than the one of the 2-coloring of B'. Consider a k-coloring C of $\bar{M}_{k,k}$ with value equal to $\sum_{i=1}^{k}(1-(1-p_i)^2)$. Then this coloring is vertical, i.e., of the form $\{\{v_i,u_i\}: i=1,\ldots,k\}$, and the functional associated with it is the smallest over the functional of any feasible coloring of $\bar{M}_{k,k}$.

Proof (Sketch). Set $C=(S_1,\ldots,S_k)$, and remove vertices v_k and u_k from $\bar{M}_{k,k}$ together with their incident edges, in order to obtain graph $\bar{M}_{k-1,k-1}$. Then, the following three cases can appear: **(a)** $\bar{M}_{k-1,k-1}$ remains colored with k colors; **(b)** $\bar{M}_{k-1,k-1}$ is colored with $k-1$ colors, i.e., one color is removed from C; **(c)** $\bar{M}_{k-1,k-1}$ is colored with $k-2$ colors, i.e., two colors are removed from C. We can prove that only a subcase of case **(b)**, namely the one where both v_k and u_k belong to S_k can hold with respect to assumptions (i) and (ii). An easy backwards induction shows finally that the claims of the proposition remain valid for any $k \geqslant 3$.

The following lemma shows that a vertex-probability system satisfying assumptions (i) and (ii) in the statement of proposition 3 really exists.

Lemma 3. ***(The feasibility of assumptions (i) and (ii) of proposition 3)***

1. *Consider a bipartite graph $\bar{M}_{n,n}$, set $V=\{v_1,\ldots,v_n\}$ and $U=\{u_1,\ldots,u_n\}$, both sets ranged in decreasing vertex probability. Set $p(v_i)=p(u_i)=\epsilon^i$, for $\epsilon \leqslant 1/3$. Then, this vertex-probability system verifies assumption (i) of proposition 3.*
2. *Let $B(V,U,E)$ be a bipartite graph and $C=(S_1,\ldots,S_k)$ a horizontal k-coloring of B. Then, $E(B,C) \geqslant E(B,(V,U))$. This holds for any vertex-probability system and for any bipartite graph $B(V,U,E)$.*

We are well prepared now to introduce the main complexity result of this section, namely that PCOLOR$(B,\mathbf{Pr},k)$ is **NP**-hard, for any $k>3$.

Theorem 2. PCOLOR$(B,\mathbf{Pr},k,K)$ *is **NP**-complete.*

Proof (Sketch). In fact, PCOLOR$(B,\mathbf{Pr},3,K)$ is **NP**-complete even for bipartite graphs B' having the following four additional characteristics: (a) only six vertices v_i,u_i, $i=1,2,3$ of B' have non-zero probabilities; (b) the subgraph of B' induced by these six vertices is a $\bar{M}_{3,3}$; (c) $p_i=p(v_i)=p(u_i)=\epsilon^i$, $i=1,2,3$, for a suitable ϵ, for example $\epsilon<1/10$. We reduce PCOLOR$(B',\mathbf{Pr},3,K')$, where B' fits characteristics (a) to (c) and $K'=\sum_{i=1}^{3}(1-(1-p_i)^2)$, to PCOLOR$(B,\mathbf{Pr},k,K)$. We consider an instance of PCOLOR$(B',\mathbf{Pr},3,K')$ and construct B as follows: we put B' together with a $\bar{M}_{k-3,k-3}$; we link the vertices of $V(B')$ with the ones of $U(\bar{M}_{k-3,k-3})$ in such a way that the graph induced by $V(B')\cup U(\bar{M}_{k-3,k-3})$ is a complete bipartite graph; we do so with $U(B')$ and $V(\bar{M}_{k-3,k-3})$. Setting $V(\bar{M}_{k-3,k-3})=\{v_4,\ldots v_k\}$ and $U(\bar{M}_{k-3,k-3})=\{u_4,\ldots u_k\}$, we set $p_i=p(v_i)=p(u_i)=\epsilon^i$, $i=4,\ldots,k$. Finally, we set $K=\sum_{i=1}^{k}(1-(1-p_i)^2)$. By what has been discussed previously, around proposition 3 and property 1 (a vertical color on v_i and u_i cannot have vertices other than these two ones) and by the fact

that the contribution of any other vertex of B' is the functional of any coloring is 0, one can immediately deduce that B has a k-coloring with functional at most $K = \sum_{i=1}^{k}(1-(1-p_i)^2)$, if and only if B' has a 3-coloring with functional at most $K' = \sum_{i=1}^{3}(1-(1-p_i)^2)$, q.e.d.

3.2 On the Approximation of PCOLOR in Bipartite Graphs

Consider a bipartite graph $B(V,U,E)$ of order n and the 2-coloring $C=(V,U)$. Fix an optimal-functional coloring $C^* = (S_1^*, \ldots, S_{k^*}^*)$ of B and an $\epsilon \in [0,2]$. Assume that the first $\hat{k}^*$ colors are such that, for any $S_j^* \in (S_1^*, \ldots S_{\hat{k}^*}^*)$, $\sum_{v_i \in S_j^*} p_i \leqslant \epsilon$, while for the $k^* - \hat{k}^*$ remaining ones, the sum of the probabilities of the vertices in any color is greater than ϵ. Set $\hat{C}^* = (S_1^*, \ldots S_{\hat{k}^*}^*)$ and denote by $\hat{n}$ the order of the subgraph of B induced by $\cup_{S_j^* \in \hat{C}^*} S_j^*$.

Theorem 3. *In any bipartite graph, its unique 2-coloring achieves approximation ratio bounded above by 2.773 for* PCOLOR.

Proof (Sketch). We study the ratio $E(B,C^*)/E(B,C)$, i.e., the inverse of the approximation ratio of the solution C. Set $|S_j^*| = \ell_j^*$; using (1) and (3), we can obtain

$$\frac{E(B,C^*)}{E(B,C)} \geqslant \min \left\{ \frac{\sum_{i=1}^{\hat{n}} p_i - \sum_{S_j^* \in \hat{C}^*} \sum_{v_i \in S_j^*} \sum_{k=i+1}^{\ell_j^*} p_i p_k}{\sum_{i=1}^{\hat{n}} p_i}, \right.$$
$$\left. \frac{\sum_{S_j^* \in C^* \setminus \hat{C}^*} \left(1 - \exp\left\{ - \sum_{v_i \in S_j^*} p_i \right\}\right)}{2} \right\}$$
$$\geqslant \min \left\{ \frac{2-\epsilon}{2}, \frac{1-\exp\{-\epsilon\}}{2} \right\}$$

In other words, the approximation ratio of the 2-coloring (V,U) for PCOLOR in any bipartite graph $B(V,U,E)$ is bounded above by $\max\{2/(2-\epsilon), 2/(1-\exp\{-\epsilon\})\}$. Equality for these terms implies $\epsilon \approx 1.278$ and then, the value of both of them is smaller than 2.773.

3.3 The Graphs $\bar{M}_{n,n}$ under Identical Vertex-Probabilities

Let us focus ourselves on the case of identical vertex-probabilities and consider a $\bar{M}_{n,n}$. Using the notations of section 3.1, any coloring here is either a vertical one, or an horizontal one, or finally, a mixed one with some vertical and some

horizontal independent sets. Recall that item 2 of lemma 3 holds for any vertex-probability system. Consequently, a 2-coloring of $\bar{M}_{n,n}$ always dominates any other horizontal coloring. On the other hand, since probabilities are all identical, the values of any mixed coloring with a precise number of vertical colors and a 2-coloring for the graph induced by the rest of the vertices are all identical too (depending on the number of the vertical colors). Consequently, an algorithm consisting of: (a) evaluating the functional associated with the 2-coloring of $\bar{M}_{n,n}$; (b) for $i = 1$ to n evaluating the functional associated with a mixed coloring consisting of i vertical colors and a 2-coloring on the subgraph of $\bar{M}_{n,n}$ induced by its non-colored vertices; (c) evaluating the value of a vertical coloring; (d) retaining the solution associated with the best of the functionals computed in steps (a) to (c); constitutes an optimal polynomial algorithm for PCOLOR in $\bar{M}_{n,n}$, for the case where vertex-probabilities are all identical.

Proposition 4. PCOLOR *is polynomial in* $\bar{M}_{n,n}$*, in the case where vertex-probabilities are all equal.*

3.4 The Complements of Bipartite Graphs

Given a bipartite graph $B(V, U, E)$, its complement $\bar{B}(V, U, \bar{E})$ is a loop-less graph consisting of two cliques, one on V and one on U, plus the set of edges $\bar{E}' = \{v_i v_j \notin E : v_i \in V, u_j \in U\}$; in other words the edges of $\bar{E}$ are the edges of the cliques $K_{|V|}$ and $K_{|U|}$, and the edges between V and U missing from E. These graphs have the property that any independent set is of cardinality at most 2, in other words, any coloring is a collection of independent sets of size 2 and of singletons. The following lemma characterizes the functional's value of such a coloring.

Lemma 4. *Let* $\bar{B}$ *be the complement of bipartite graph* B*, let* n *be the order of* $\bar{B}$ *and* B*, let* C *be a coloring of* $\bar{B}$*, and let* $S = \{\{v_{i_k}, v_{j_k}\} : k = 1, \ldots, |S|\}$ *be the collection of independent sets of size 2 in* C*. Then,* $E(\bar{B}, C) = \sum_{i=1}^{n} p_i - \sum_{k=1}^{|S|} p_{i_k} p_{j_k}$.

It is easy to see from lemma 4 that the first term of the functional is constant; so, $E(\bar{B}, C)$ is minimized when its second term is maximized. Consider the bipartite graph $B'(V, U, E(B'))$ with $E(B') = (V \times U) \setminus \bar{E}'$, and assign to any edge $v_i v_j \in E(B')$ weight $p_i p_j$. Then, collection S becomes a matching of B' and the term $\sum_{k=1}^{|S|} p_{i_k} p_{j_k}$ the total weight of this matching. Recall finally that a maximum weight matching can be polynomially computed in any graph ([22]). Then consider the following algorithm: given $\bar{B}$, transform it into B' and weight any of its edges with the product of the probabilities of its endpoints; compute a maximum weight matching M in B'; color endpoints of any edge of M with an unused color (the same for both endpoints); color the remaining vertices of B' with an unused color by such vertex. From what has been discussed, the coloring so produced is optimal and the following result concludes the section.

Theorem 4. PCOLOR *is polynomial in complements of bipartite graphs.*

4 Concluding Remarks

There exists a lot of interesting open problems dealing with the results of this paper. For example, the complexity of PCOLOR remains open, notably for natural classes of bipartite graphs as chains, or trees, or even for classes of graphs "close" to bipartite ones, for example the split graphs. In another order of ideas, an interesting approximation strategy for solving hard minimization problem, called "master-slave" approximation, consists of solving a minimization problem, the master problem, by repeatedly solving a maximization problem, the slave one (for more details on this technique, see [23]). This kind of technique has a very natural application in the case of minimum coloring where the slave problem is the maximum independent set. It consists of iteratively computing an independent set in the graph, of coloring its vertices with the same unused color, of removing it from the graph, and of repeating these stages in the subsequent surviving subgraphs until all vertices are colored. The slave independent set problem for PCOLOR is the one of determining the independent set S^* maximizing quantity $|S|/(1-\prod_{v_i \in S}(1-p_i))$ over any independent set of the input graph. Obviously, this problem is **NP**-hard in general graphs, since for $p_i = 1$ for any vertex of the input graph, we recover the classical maximum independent set problem. However, approximation of it in general graphs and complexity, and eventually, approximation results in graph-families as the ones dealt in this paper seem to us interesting to be studied.

References

1. Karp, R.M.: Reducibility among combinatorial problems. In Miller, R.E., Thatcher, J.W., eds.: Complexity of computer computations. Plenum Press, New York (1972) 85–103
2. Garey, M.R., Johnson, D.S.: Computers and intractability. A guide to the theory of NP-completeness. W. H. Freeman, San Francisco (1979)
3. Bertsimas, D.J., Jaillet, P., Odoni, A.: A priori optimization. Oper. Res. **38** (1990) 1019–1033
4. Jaillet, P.: Probabilistic traveling salesman problem. Technical Report 185, Operations Research Center, MIT, Cambridge Mass., USA (1985)
5. Averbakh, I., Berman, O., Simchi-Levi, D.: Probabilistic a priori routing-location problems. Naval Res. Logistics **41** (1994) 973–989
6. Bertsimas, D.J.: On probabilistic traveling salesman facility location problems. Transportation Sci. **3** (1989) 184–191
7. Bertsimas, D.J.: The probabilistic minimum spanning tree problem. Networks **20** (1990) 245–275
8. Jaillet, P.: A priori solution of a traveling salesman problem in which a random subset of the customers are visited. Oper. Res. **36** (1988) 929–936
9. Jaillet, P.: Shortest path problems with node failures. Networks **22** (1992) 589–605
10. Jaillet, P., Odoni, A.: The probabilistic vehicle routing problem. In Golden, B.L., Assad, A.A., eds.: Vehicle routing: methods and studies. North Holland, Amsterdam (1988)
11. Murat, C., Paschos, V.T.: The probabilistic minimum vertex-covering problem. Int. Trans. Opl Res. **9** (2002) 19–32

12. Murat, C., Paschos, V.T.: The probabilistic longest path problem. Networks **33** (1999) 207–219
13. Murat, C., Paschos, V.T.: A priori optimization for the probabilistic maximum independent set problem. Theoret. Comput. Sci. **270** (2002) 561–590
14. Murat, C., Paschos, V.T.: The probabilistic minimum coloring problem. Annales du LAMSADE 1, LAMSADE, Université Paris-Dauphine (2003) Available on `http://www.lamsade.dauphine.fr/cahdoc.html#cahiers`.
15. Demange, M., de Werra, D., Monnot, J., Paschos, V.T.: Time slot scheduling of compatible jobs. Cahier du LAMSADE 182, LAMSADE, Université Paris-Dauphine (2001) Available on `http://www.lamsade.dauphine.fr/cahdoc.html#cahiers`.
16. Demange, M., de Werra, D., Monnot, J., Paschos, V.T.: Weighted node coloring: when stable sets are expensive. In Kučera, L., ed.: Proc. 28th International Workshop on Graph Theoretical Concepts in Computer Science, WG'02. Volume 2573 of Lecture Notes in Computer Science., Springer-Verlag (2002) 114–125
17. Bar-Noy, A., Bellare, M., Halldórsson, M.M., Shachnai, H., Tamir, T.: On chromatic sums and distributed resource allocation. Inform. and Comput. **140** (1988) 183–202
18. Jansen, K.: Approximation results for the optimum cost chromatic partition problem. In Pardalos, P.M., Du, D., eds.: Theoretical Computer Science, Network Design: Connectivity and Facilities Location. (1997) 143–168
19. Nicoloso, S., Sarrafzadeh, M., Song, X.: On the sum coloring problem on interval graphs. Algorithmica **23** (1999) 109–126
20. Halldórsson, M.M.: A still better performance guarantee for approximate graph coloring. Inform. Process. Lett. **45** (1993) 19–23
21. Bodlaender, H.L., Jansen, K., Woeginger, G.J.: Scheduling with incompatible jobs. Discrete Appl. Math. **55** (1994) 219–232
22. Papadimitriou, C.H., Steiglitz, K.: Combinatorial optimization: algorithms and complexity. Prentice Hall, New Jersey (1981)
23. Johnson, D.S.: Approximation algorithms for combinatorial problems. J. Comput. System Sci. **9** (1974) 256–278

Recognizing Bipolarizable and P_4-Simplicial Graphs*

Stavros D. Nikolopoulos and Leonidas Palios

Department of Computer Science, University of Ioannina GR-45110 Ioannina, Greece
{stavros,palios}@cs.uoi.gr

Abstract. Hoàng and Reed defined the classes of Raspail (also known as Bipolarizable) and P_4-simplicial graphs, both of which are perfectly orderable, and proved that they admit polynomial-time recognition algorithms [16]. In this paper, we consider the recognition problem on these classes of graphs and present algorithms that solve it in $O(nm)$ time, where n and m are the numbers of vertices and of edges of the input graph. In particular, we prove properties and show that we can produce bipolarizable and P_4-simplicial orderings on the vertices of a graph G, if such orderings exist, working only on P_3s that participate in P_4s of G. The proposed recognition algorithms are simple, use simple data structures and require $O(n+m)$ space. Moreover, we present a diagram on class inclusions and the currently best recognition time complexities for a number of perfectly orderable classes of graphs and some preliminary results on forbidden subgraphs for the class of P_4-simplicial graphs.

Keywords: Bipolarizable (Raspail) graph, P_4-simplicial graph, perfectly orderable graph, recognition, algorithm, complexity, forbidden subgraph.

1 Introduction

A linear order $\prec$ on the vertices of a graph G is *perfect* if the ordered graph $(G, \prec)$ contains no induced P_4 *abcd* with $a \prec b$ and $d \prec c$ (such a P_4 is called an *obstruction*). In the early 1980s, Chvátal [4] defined the class of graphs that admit a perfect order and called them *perfectly orderable* graphs. Chvátal proved that if a graph G admits a perfect order $\prec$, then the greedy coloring algorithm applied to $(G, \prec)$ produces an optimal coloring using only $\omega(G)$ colors, where $\omega(G)$ is the clique number of G. This implies that the perfectly orderable graphs are perfect; a graph G is *perfect* if for each induced subgraph H of G, the chromatic number $\chi(H)$ equals the clique number $\omega(H)$ of the subgraph H. The class of perfect graphs was introduced and studied by Berge [1], who also conjectured that a graph is perfect if and only if it has no induced subgraph isomorphic to an odd cycle of length at least five, or to the complement of such an odd cycle. This conjecture, known as the *strong perfect graph conjecture*, has been recently established due to the work of Chudnovsky *et al.* [3].

* Research partially funded by the European Commission and the Hellenic Ministry of Education through EPEAEK II.

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 358–369, 2003.

It is well-known that many interesting problems in graph theory, which are NP-complete in general graphs, have polynomial solutions in graphs that admit a perfect order [2,8]; unfortunately, it is NP-complete to decide whether a graph admits a perfect order [22]. Since the recognition of perfectly orderable graphs is NP-complete, we are interested in characterizing graphs which form polynomially recognizable subclasses of perfectly orderable graphs. Many such classes of graphs, with very interesting structural and algorithmic properties, have been defined so far and shown to admit polynomial-time recognitions (see [2,8]); note however that not all subclasses of perfectly orderable graphs admit polynomial-time recognitions [13].

Hoàng and Reed [16] introduced four subclasses of perfectly orderable graphs, namely, the Raspail (also known as Bipolarizable), P_4-simplicial, P_4-indifference, and P_4-comparability graphs, and provided polynomial-time recognition algorithms for these four classes of graphs. A graph G is *bipolarizable* if it admits a linear order $\prec$ on its vertices such that every P_4 *abcd* has either ($b \prec a$, $b \prec c$, $c \prec d$) or ($b \prec a$, $c \prec b$, $c \prec d$). A graph G is *P_4-simplicial* if it admits a linear order $\prec$ such that every P_4 has either a P_4-indifference ordering (i.e., every P_4 *abcd* has either ($a \prec b$, $b \prec c$, $c \prec d$) or ($d \prec c$, $c \prec b$, $b \prec a$)) or a bipolarizable ordering. Note that every linear order $\prec$ on the vertices of a graph G yields an acyclic orientation of the edges, where each edge ab is oriented from a to b if and only if $a \prec b$. On the other hand, every acyclic orientation gives at least one linear order (for example, the order taken by a topological sorting). Hence, bipolarizable and P_4-simplicial graphs can also be defined in terms of orientations.

As mentioned in the previous paragraph, the recognition problem on both bipolarizable and P_4-simplicial graphs has been addressed by Hoàng and Reed [16]; for a graph on n vertices, their algorithms run in $O(n^4)$ and $O(n^5)$ time respectively. Recently, Eschen *et al.* [7] described recognition algorithms for several classes of perfectly orderable graphs, among which $O(n^{3.376})$-time algorithms for both bipolarizable and P_4-simplicial graphs. We note that Hoàng and Reed also presented algorithms which solve the recognition problem for P_4-indifference and P_4-comparability graphs which run in $O(n^6)$ and $O(n^4)$ time [16,17]; recent results on these problems include $O(n + m)$-time and $O(nm)$-time algorithms respectively [10,23], where m is the number of edges of the input graph.

In this paper, we consider the recognition problems for bipolarizable and P_4-simplicial graphs and present $O(nm)$-time algorithms for their solution. Our algorithms rely on properties that we establish and which allow us to work only with P_3s which participate in P_4s of the input graph G; such P_3s can be computed in $O(nm)$ time by means of the BFS-trees of the *complement* of G rooted at each of its vertices [23]. The proposed recognition algorithms are simple, use simple data structures and require $O(n+m)$ space. Furthermore, we give class inclusion results for a number of perfectly orderable classes of graphs and show the currently best time complexities to recognize members of these classes, and finally we also present results on forbidden subgraphs for the class of P_4-simplicial graphs.

2 Preliminaries

We consider finite undirected graphs with no loops or multiple edges. Let G be such a graph; then, $V(G)$ and $E(G)$ denote the set of vertices and of edges of G respectively. The subgraph of G induced by a subset S of G's vertices is denoted by $G[S]$. The *neighborhood* $N(x)$ of a vertex $x \in V(G)$ is the set of all the vertices of G which are adjacent to x. The *closed neighborhood* of x is defined as $N[x] := \{x\} \cup N(x)$.

A *path* in a graph G is a sequence of vertices $v_0v_1 \dots v_k$ such that $v_{i-1}v_i \in E(G)$ for $i = 1, 2, \dots, k$; we say that this is a path from v_0 to v_k and that its *length* is k. A path is called *simple* if none of its vertices occurs more than once; it is called *trivial* if its length is equal to 0. A path (simple path) $v_0v_1 \dots v_k$ is called a *cycle* (*simple cycle*) of length $k+1$ if $v_0v_k \in E(G)$. A simple path (cycle) $v_0v_1 \dots v_k$ is *chordless* if $v_iv_j \notin E(G)$ for any two non-consecutive vertices v_i, v_j in the path (cycle). The chordless path (chordless cycle, respectively) on n vertices is commonly denoted by P_n (C_n, respectively). In particular, a chordless path on 4 vertices is denoted by P_4.

Let *abcd* be a P_4 of a graph. The vertices b and c are called *midpoints* and the vertices a and d *endpoints* of the P_4 *abcd*. The edge connecting the midpoints of a P_4 is called the *rib*; the other two edges (which are incident on the endpoints) are called the *wings*. For the P_4 *abcd*, the edge *bc* is its rib and the edges *ab* and *cd* are its wings.

Computing all the P_3s participating in P_4s of a graph G: In [23], it has been shown that all the P_3s participating in P_4s of a graph G on n vertices and m edges can be computed in $O(nm)$ time and $O(n+m)$ space as follows: for each vertex v, the BFS-tree $T_{\overline{G}}(v)$ of the *complement* of G rooted at v is constructed and the vertices in the 2nd level of the tree are partitioned into sets $S_1, \dots, S_{k_v}$, where two vertices belong to the same S_i iff they have the same neighbors in the 1st level of $T_{\overline{G}}(v)$; the root v of $T_{\overline{G}}(v)$ is assumed to be located in the 0th level. Then, *avb* is a P_3 participating in a P_4 of G iff $ab \notin E(G)$ and either exactly one of a, b belongs to the 2nd level and the other to the 3rd level of $T_{\overline{G}}(v)$, or both a and b belong to the 2nd level but they are in different sets of the partition $S_1, \dots, S_{k_v}$.

Since the vertices in the 2nd and 3rd level of $T_{\overline{G}}(v)$ form a subset of the neighborhood of v, we can give a more unified criterion for deciding whether a P_3 *avb* participates in a P_4 of G by defining the following partition of $N(v)$:

Definition 2.1. *For each vertex v of a graph G, we consider the following partition of the neighborhood $N(v)$ of v:*

- ▷ *the partition of the vertices in the 2nd level of $T_{\overline{G}}(v)$ into $S_1, \dots, S_{k_v}$ as described above;*
- ▷ *all the vertices in the 3rd level of $T_{\overline{G}}(v)$ are placed in a set S_{k_v+1};*
- ▷ *all remaining vertices in $N(v)$ are placed in a set S_0 (no such vertex a forms a P_3 avb participating in P_4s of G for any vertex b of G).*

Then, for any $a, b \in N(v)$, avb is a P_3 participating in a P_4 of G iff $ab \notin E(G)$, and if $a \in S_i$ and $b \in S_j$ then $i \neq 0$, $j \neq 0$, and $i \neq j$.

Convention: Throughout the paper, we assume that the input graph G has n vertices and m edges and is given in adjacency list representation.

3 Recognition of Bipolarizable Graphs

The definition of bipolarizable graphs implies that they can be efficiently recognized as soon as the wings of all the P_4s have been computed. The method described in [23] for computing all the P_3s participating in P_4s of a given graph does not seem to extend to produce within the same time complexity which edge of the P_3 is the rib and which is the wing of the P_4. However, in the case of bipolarizable graphs, we establish a property that can be used for their efficient recognition. First, we need the following lemma:

Lemma 3.1. *Let G be a graph that contains no induced subgraph isomorphic to a house graph or the graphs F_1 and F_2 of Figure 2. Then, G contains a C_4 $abcd$ such that abc and bcd are P_3s participating in P_4s of G.*

Proof: Suppose for contradiction that G contains a C_4 $abcd$ meeting the conditions in the statement of the lemma. We distinguish cases. Suppose first that the P_3 abc participates in the P_4 $abcx$ and that the P_3 bcd participates in the P_4 $bcdy$. Then, $xd \notin E(G)$, otherwise the vertices a, b, c, d, x would induce a house in G. In a similar fashion, $ya \notin E(G)$ either. But then, if $xy \notin E(G)$, then the subgraph induced by a, b, c, d, x, y is isomorphic to F_1 whereas if $xy \in E(G)$, it is isomorphic to F_2; a contradiction in either case. The remaining three cases (depending on whether abc participates in a P_4 $xabc$ or $abcx$ and on whether bcd participates in a P_4 $ybcd$ or $bcdy$) are handled similarly. ∎

Since the bipolarizable graphs do not contain the house graph, F_1, or F_2 (and also some other subgraphs [12,16]), Lemma 3.1 implies the following corollary.

Corollary 3.1. *Let G be a bipolarizable graph and let abc be a P_3 participating in a P_4 of G. If bcd is another such P_3, then G contains the P_4 $abcd$.*

Proof: If the path $abcd$ is not a P_4 then G must contain the edge ad. But this creates a C_4 meeting the conditions of Lemma 3.1; a contradiction. ∎

(We note that Corollary 3.1 in fact holds for the class of weak bipolarizable graphs [25], a superclass of the bipolarizable graphs.) Corollary 3.1 implies the following result.

Corollary 3.2. *Let G be a bipolarizable graph and let F be the orientation of G that results from the bipolarizable ordering of the vertices of G (i.e., the wings of each P_4 are oriented towards the P_4's endpoints). Then, for each edge bc of G for which there exist P_3s abc and bcd participating in P_4s of G, the edges ab and cd (for all such a and d) get oriented towards a and d respectively.*

The algorithm for the recognition of bipolarizable graphs applies Corollary 3.2. The algorithm uses two arrays, an array $M[\,]$ and an array $S[\,]$, of size $2m$ each. The array $M[\,]$ has entries $M[xy]$ and $M[yx]$, for each edge xy of G; the entry $M[xy]$ is equal to 1 if there exist P_3s xyz participating in P_4s of G, and is equal to 0 otherwise. As a result, for an edge xy, both $M[xy]$ and $M[yx]$ are equal to 1 iff there exist P_3s xyz and txy participating in P_4s of G. The array $S[\,]$ too has entries $S[xy]$ and $S[yx]$, for each edge xy of G; the entry $S[xy]$ is equal to the index number of the partition set of $N(y)$ to which x belongs (see Definition 2.1). As a result, a path xyz is a P_3 participating in P_4s of G iff $S[xy] \neq 0$, $S[zy] \neq 0$, and $S[xy] \neq S[zy]$. In more detail, the algorithm works as follows.

Bipolarizable Graph Recognition Algorithm

1. Initialize the entries of the arrays $M[\,]$ and $S[\,]$ to 0; for each vertex v, sort the records of the neighbors of v in v's adjacency list in increasing vertex index number;
2. Find all the P_3s participating in P_4s of G; for each such P_3 abc, set the entries $M[ab]$ and $M[cb]$ equal to 1, and update appropriately the entries $S[ab]$ and $S[cb]$;
3. For each edge uv of G such that $M[uv] = 1$ and $M[vu] = 1$ do
   3.1 traverse the adjacency lists of u and v in lockstep fashion in order to locate the non-common neighbors of u and v;
   3.2 for each neighbor w of v which is not adjacent to u do
   if $S[uv] \neq 0$ and $S[wv] \neq 0$ and $S[uv] \neq S[wv]$
   then $\{uvw$ *is a* P_3 *in a* P_4 *of* $G\}$
   if the edge vw has not received an orientation
   then orient it towards w;
   else if it is oriented towards v
   then print that G is not a bipolarizable graph; exit.
   3.3 work similarly as in case 3.2 for each neighbor w of u which is not adjacent to v;
4. Check if the directed subgraph induced by the oriented edges contains a directed cycle; if it does not, print that G is a bipolarizable graph; otherwise, print that it is not.

The correctness of the algorithm follows directly from Corollary 3.2. Observe that for any P_4 $abcd$ of G, the edge bc will be considered in Step 3 of the algorithm, and then the edges ab and cd will be oriented correctly.

Time and Space Complexity. Step 1 takes $O(n+m)$ time since the sorted adjacency lists can be obtained through radix sorting an array of all the ordered pairs of adjacent vertices, while Step 2 takes $O(nm)$ time [23]. Steps 3.2 and 3.3 take constant time per such vertex w; it is assumed that the orientation of an edge is stored in an array of size m for constant-time access and update. For an edge uv, Steps 3.2 and 3.3 is executed $O(deg(u) + deg(v))$ times, where $deg(u)$

denotes the degree of vertex u. Since Step 3.1 also takes $O(deg(u) + deg(v))$ time, Step 3 takes $O\big(\sum_{uv \in E(G)}(deg(u) + deg(v))\big) = O(nm)$ time. Step 4 can be executed by constructing the resulting directed graph and then applying topological sorting on it; if the topological sorting succeeds then no directed cycle exists, otherwise there exists a directed cycle. From this description, it is clear that Step 4 can be completed in $O(n + m)$ time and space. Since the computation of the P_3s participating in P_4s takes linear space, the total space needed by the recognition algorithm is clearly linear in the size of the input graph G.

Summarizing, we obtain the following theorem.

Theorem 3.1. *Let G be an undirected graph on n vertices and m edges. Then, it can be determined whether G is a bipolarizable graph in $O(nm)$ time and $O(n + m)$ space.*

The recognition algorithm can be used to produce a bipolarizable ordering of the vertices of a bipolarizable graph G. The bipolarizable ordering coincides with the topological ordering of the vertices of the directed graph in Step 4, possibly extended by an arbitrary ordering of any vertices of G which do not participate in the directed graph.

4 Recognition of P_4-Simplicial Graphs

Our P_4-simplicial graph recognition algorithm relies on the corresponding algorithm of Hoàng and Reed [16]; our contribution is that we restate the main condition on which their algorithm is based in terms of P_3s participating in P_4s of the input graph, and we show how to efficiently take advantage of it in order to achieve an $O(nm)$ time complexity. In particular, their algorithm works as follows: it initially sets $H = V(G)$ and then it iteratively identifies a vertex x in H such that G does not contain a P_4 of the form $abxc$ with $b, c \in H$, and removes it from H; the graph G is P_4-simplicial iff the above process continues until H becomes the empty set.

It is not difficult to see that the property a vertex x has to have in order to be removed from H can be equivalently stated as follows:

Property 4.1. *Let H be the current set of vertices of a given graph G. Then, a vertex x can be removed from H if and only if there does not exist any P_3 bxc participating in a P_4 of G with $b, c \in H$.*

In light of Property 4.1, we can obtain an algorithm for deciding whether a given graph G is P_4-simplicial by keeping count, for each vertex $v \in H$, of the number of P_3s bvc with $b, c \in H$ which participate in P_4s of G, and by removing a vertex from H whenever the number of such P_3s associated with that vertex is 0. The proposed algorithm implements precisely this strategy; it takes advantage of the computation of the P_3s in P_4s of G in $O(nm)$ time, and maintains an array $NumP3[\,]$ of size n, which stores for each vertex v in H the number of

P_3s *bvc* which participate in P_4s of G and have $b, c \in H$. In more detail, the algorithm works as follows.

P_4-simplicial Graph Recognition Algorithm

1. Collect all the vertices of G into a set H;
   make a copy $A[v]$ of the adjacency list of each vertex v of G while attaching at each record of the list an additional field *set*;
2. For each vertex v of G do
   2.1 compute the partition of the vertices in $N(v)$ into sets $S_0, \ldots, S_{k_v}, S_{k_v+1}$ as described in Definition 2.1, and update appropriately the fields *set* of the records in the adjacency list $A[v]$ of v;
   2.2 compute the number of P_3s *avb* participating in P_4s of G and assign this number to $NumP3[v]$;
3. Collect in a list L the vertices v for which $NumP3[v] = 0$;
4. While the list L is not empty do
   4.1 remove a vertex, say, x, from L;
   4.2 for each vertex u adjacent to x in G do
       if u belongs to H
           traverse the adjacency list $A[u]$ of u and let s_x be the value of the field *set* for the vertex x;
           if $s_x \neq 0$
           then {*there may exist P_3s xuw participating in P_4s of G*}
               for each vertex w in the adjacency list $A[u]$ of u do
                   $s_w \leftarrow$ value of the field *set* for the vertex w;
                   if $w \in H$ and $s_w \neq 0$ and $s_w \neq s_x$
                   then {*xuw is such a P_3 with $x, u, w \in H$*}
                       $NumP3[u] \leftarrow NumP3[u] - 1$;
               if $NumP3[u] = 0$
               then insert u in the list L;
   4.3 remove x from the set H;
5. if the set H is empty, then print that G is a P_4-simplicial graph; otherwise, print that it is not.

To ensure correct execution, the algorithm maintains the following invariant throughout the execution of Step 4 (the proof can be found in [24]).

Invariant 4.1. *At the beginning of every iteration of the while loop in Step 4 of the algorithm, for each vertex v in H, $NumP3[v]$ is equal to the number of P_3s bvc participating in P_4s of G with $b, c \in H$.*

Sketch of the Proof: The proof relies on the fact that $NumP3[v]$ will be decremented precisely once for each P_3 *avb* participating in a P_4 of G: if a is removed from H before b, then $NumP3[v]$ will be decremented during the removal of a; when b is removed, the P_3 *avb* will not be considered, even if $v \in H$, because $a \notin H$. ∎

Then, the correctness of the algorithm follows from the correctness of the algorithm of Hoàng and Reed, Property 4.1, and the fact that at any given time the list L contains precisely those vertices that can be removed from H (a vertex x is inserted in L if and only if $NumP3[x] = 0$, i.e., there does not exist any P_3 bxc participating in a P_4 of G with $b, c \in H$).

Time and Space Complexity. The set H can be implemented by means of an array $M[\,]$ of size n, where $M[v] = 1$ if $v \in H$ and 0 otherwise; in this way, insertion, deletion, and membership queries for any vertex of G can be answered in constant time, while the emptiness of H can be checked in $O(n)$ time. Then, Step 1 takes $O(n + m)$ time, Step 4.3 takes $O(1)$ time per vertex removed, and Step 5 $O(n)$ time. Step 2 takes $O(nm)$ time [23], while Step 3 takes $O(n)$ time. As a vertex is inserted at most once in the list L, the time complexity of Step 4 is $O\left(\sum_x \left(1 + \sum_{u \in N(x)} deg(u)\right)\right)$, where $deg(u)$ denotes the degree of u in G. Since $\sum_{u \in N(x)} deg(u) = O(m)$, the time complexity of Step 4 is $O(nm)$. Since the computation of the P_3s participating in P_4s takes linear space, the total space needed by the recognition algorithm is clearly linear in the size of the input graph G.

Summarizing, we obtain the following theorem.

Theorem 4.1. *Let G be an undirected graph on n vertices and m edges. Then, it can be determined whether G is a P_4-simplicial graph in $O(nm)$ time and $O(n + m)$ space.*

5 Class Inclusions and Recognition Time Complexities

Figure 1 shows a diagram of class inclusions for a number of perfectly orderable classes of graphs and the currently best time complexities to recognize members of these classes. For definitions of the classes shown, see [2,8]; note that the P_4-free and the chordal graphs are also known as co-graphs and triangulated graphs respectively. In the diagram, there exists an arc from a class $\mathcal{A}$ to a class $\mathcal{B}$ if and only if $\mathcal{B}$ is a proper subset of $\mathcal{A}$. Hence, if any two classes are not connected by an arc, then each of these classes contains graphs not belonging to the other class (there are such sample graphs for each pair of non-linked classes).

Most of these class inclusions can be found in [2] where a similar diagram with many more graph classes appears; Figure 1 comes from a portion of the diagram in [2] augmented with the introduction of the inclusion relations for the classes of P_4-simplicial, bipolarizable, and P_4-indifference graphs, as described in the following lemmata (the complete proofs have been omitted due to lack of space but can be found in [24]):

Lemma 5.1. *The class of P_4-simplicial graphs is a proper subset of the class of brittle graphs and a proper superset of the class of weak bipolarizable*[1] *graphs.*

[1] A graph is weak bipolarizable if it has no induced subgraph isomorphic to C_k ($k \geq 5$), the house graph, or to any of the graphs F_1 and F_2 of Figure 2 [25].

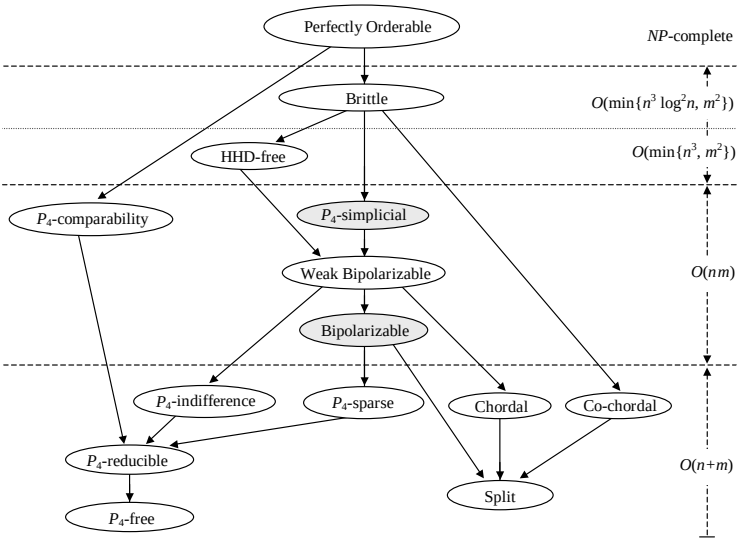


Fig. 1. Class inclusions and recognition time complexities.

Sketch of the Proof: The fact that P_4-*simplicial* $\subseteq$ *Brittle* has been shown in [16]; the subset relation is proper since the graph F_1 of Figure 2 is brittle but not P_4-simplicial. To show that *Weak Bipolarizable* $\subseteq$ P_4-*simplicial*, we apply induction on the size of the graph by taking advantage of Theorem 1 of [25] which states that a graph G is weak bipolarizable if and only if every induced subgraph of G is chordal or contains a homogeneous set; the proper inclusion follows from the fact that the house graph is P_4-simplicial but not weak bipolarizable. ∎

Lemma 5.2. *The class of bipolarizable graphs is a proper subset of the class of weak bipolarizable graphs and a proper superset of the classes of P_4-sparse and split graphs.*

Lemma 5.3. *The class of P_4-indifference graphs is a proper subset of the class of weak bipolarizable graphs and a proper superset of the class of P_4-reducible graphs.*

Regarding the relation of P_4-simplicial and the HHD-free and co-chordal graphs, we note that the graph F_1 of Figure 2 is both HHD-free and co-chordal but is not P_4-simplicial whereas the house graph and P_5 are P_4-simplicial but not HHD-free and not co-chordal respectively. The non-inclusion relation between bipolarizable and co-chordal graphs follows from the counterexamples for the non-inclusion relation of the P_4-simplicial and co-chordal graphs. A non-inclusion relation also holds for the bipolarizable and the chordal graphs (consider a C_4 and the forbidden subgraph D of [12]) and for the bipolarizable and the P_4-indifference graphs (consider the forbidden subgraphs F_5 of [15] and D of [12]).

Figure 1 also shows the depicted classes of graphs partitioned based on the time complexities of the currently best recognition algorithms: see [7,27] for the $O(\min\{n^3 \log^2 n, m^2\})$-time complexity range, [14,18] for the $O(\min\{n^3, m^2\})$-time complexity range, [23,25] for the $O(nm)$-time complexity range, and [10, 19,20,5,26,9,11] for the $O(n+m)$-time range. We note that the algorithm of [14] for the recognition of HHD-free graphs has a stated time complexity of $O(n^4)$; this can be easily seen to be $O(m^2)$ if the number m of edges of the graph is taken into account. Similarly, the algorithm of [25] for the recognition of weak bipolarizable graphs has a stated time complexity of $O(n^3)$; since $O(n+m)$ time suffices to determine whether a graph is chordal and to compute a homogeneous set (by means of modular decomposition [21,6]), if one exists, the stated time complexity can be seen to be $O(nm)$.

6 On Forbidden Subgraphs for P_4-Simplicial Graphs

The minimal set of forbidden subgraphs for the class of bipolarizable graphs has been established in [12,16]. For the class of P_4-simplicial graphs, however, no work on forbidden subgraphs is available in the literature to the best of our knowledge; in this section, we give a number of forbidden subgraphs for this class, and attempt to give a first characterization of them.

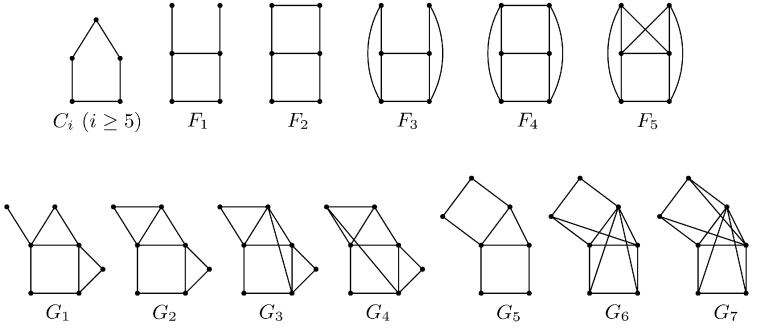


Fig. 2. Some forbidden subgraphs for the class of P_4-simplicial graphs

Clearly, a hole, the graph F_1 (sometimes also called "A"), and the graph F_2 (also known as domino graph or D_6) in Figure 2 are all forbidden subgraphs for P_4-simplicial graphs. On the other hand, the house graph (i.e., $\overline{P}_5$) is P_4-simplicial. Figure 2 shows all forbidden subgraphs on up to 7 vertices; note that F_3 is $\overline{F}_2$, F_4 is $\overline{C}_6$, and F_5 is $\overline{P}_6$. Additionally, even if the holes are excluded, one can easily generate a number of arbitrarily large forbidden subgraphs. Figure 3 gives two such examples.

In any case, Lemma 5.1 implies the following property for all forbidden subgraphs other than a hole, and the graphs F_1 and F_2 of Figure 2:

Lemma 6.1. *Any forbidden subgraph for the class of P_4-simplicial graphs, other than a hole, F_1, and F_2, contains at least one house graph as induced subgraph.*
Following up on Lemma 6.1, we believe that any minimal forbidden subgraph for the class of P_4-simplicial graphs, other than a hole, F_1, and F_2, has at least two houses as induced subgraphs. In fact, we conjecture that the set of such forbidden subgraphs includes a number of graphs containing at least two vertex-sharing houses (see Figure 2) and a small number of graphs that have exactly two vertex-disjoint houses as induced subgraphs (as in Figure 3).

Fig. 3.

7 Concluding Remarks

We have presented recognition algorithms for the classes of bipolarizable (also known as Raspail) and P_4-simplicial graphs running in $O(nm)$ time. Our proposed algorithms are simple, use simple data structures and require $O(n + m)$ space. We have also presented results on class inclusions and recognition time complexities for a number of perfectly orderable classes of graphs, and also some results on forbidden subgraphs for the class of P_4-simplicial graphs.

We leave as an open problem the designing of $o(nm)$-time algorithms for recognizing bipolarizable and/or P_4-simplicial graphs. In light of the linear-time recognition of P_4-indifference graphs [10], it would be worth investigating whether the recognition of P_4-comparability, P_4-simplicial, and bipolarizable graphs is inherently more difficult; it must be noted that the approach used in [10] is different from those used for the recognition of the remaining classes as it reduces in part the problem to the linear-time recognition of interval graphs. Finally, another interesting open problem is that of completing the characterization of the P_4-simplicial graphs by forbidden subgraphs.

References

1. C. Berge, Färbung von Graphen deren sämtliche bzw. deren ungerade Kreise starr (Zusammenfassung), *Wissenschaftliche Zeitschrift*, Martin Luther Universität Halle-Witterberg, Mathematisch-Naturwissenschaftliche Reihe, 114–115, 1961.
2. A. Brandstädt, V.B. Le, and J.P. Spinrad, *Graph classes: A survey*, SIAM Monographs on Discrete Mathematics and Applications, 1999.
3. M. Chudnovsky, N. Robertson, P.D. Seymour, and R. Thomas, The strong perfect graph theorem, *submitted*.

4. V. Chvátal, Perfectly ordered graphs, *Annals of Discrete Math.* **21**, 63–65, 1984.
5. D.G. Corneil, Y. Perl, and L.K. Stewart, A linear recognition algorithm for cographs, *SIAM J. Comput.* **14**, 926–934, 1985.
6. E. Dahlhaus, J. Gustedt, and R.M. McConnell, Efficient and practical algorithms for sequential modular decomposition, *J. Algorithms* **41**, 360–387, 2001.
7. E.M. Eschen, J.L. Johnson, J.P. Spinrad, and R. Sritharan, Recognition of some perfectly orderable graph classes, *Discrete Appl. Math.* **128**, 355–373, 2003.
8. M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, Inc., 1980.
9. M. Habib, R.M. McConnell, C. Paul, and L. Viennot, Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing, *Theoret. Comput. Sci.* **234**, 59–84, 2000.
10. M. Habib, C. Paul, and L. Viennot, Linear time recognition of P_4-indifference graphs, *Discrete Math. and Theor. Comput. Sci.* **4**, 173–178, 2001.
11. P.L. Hammer and B. Simeone, The splittance of a graph, *Combinatorica* **1**, 275–284, 1981.
12. A. Hertz, Bipolarizable graphs, *Discrete Math.* **81**, 25–32, 1990.
13. C.T. Hoàng, On the complexity of recognizing a class of perfectly orderable graphs, *Discrete Appl. Math.* **66**, 219–226, 1996.
14. C.T. Hoàng and N. Khouzam, On brittle graphs, *J. Graph Theory* **12**, 391–404, 1988.
15. C.T. Hoàng, F. Maffray, and M. Noy, A characterization of P_4-indifference graphs, *J. Graph Theory* **31**, 155–162, 1999.
16. C.T. Hoàng and B.A. Reed, Some classes of perfectly orderable graphs, *J. Graph Theory* **13**, 445–463, 1989.
17. C.T. Hoàng and B.A. Reed, P_4-comparability graphs, *Discrete Math.* **74**, 173–200, 1989.
18. C.T. Hoàng and R. Sritharan, Finding houses and holes in graphs, *Theoret. Comput. Sci.* **259**, 233–244, 2001.
19. B. Jamison and S. Olariu, A linear-time recognition algorithm for P_4-sparse graphs, *SIAM J. Comput.* **21**, 381–407, 1992.
20. B. Jamison and S. Olariu, A linear-time algorithm to recognize P_4-reducible graphs, *Theoret. Comput. Sci.* **145**, 329–344, 1995.
21. R.M. McConnell and J. Spinrad, Linear-time modular decomposition and efficient transitive orientation, *Proc. 5th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'94)*, 536–545, 1994.
22. M. Middendorf and F. Pfeiffer, On the complexity of recognizing perfectly orderable graphs, *Discrete Math.* **80**, 327–333, 1990.
23. S.D. Nikolopoulos and L. Palios, On the recognition of P_4-comparability graphs, *Proc. 28th Workshop on Graph Theoretic Concepts in Computer Science (WG'02)*, LNCS 2573, 355–366, 2002.
24. S.D. Nikolopoulos and L. Palios, Recognizing bipolarizable and P_4-simplicial graphs, *Technical Report 7-03*, Dept of Computer Science, Univ. of Ioannina, 2003.
25. S. Olariu, Weak bipolarizable graphs, *Discrete Math.* **74**, 159–171, 1989.
26. D.J. Rose, R.E. Tarjan, and G.S. Lueker, Algorithmic aspects of vertex elimination on graphs, *SIAM J. Comput.* **5**, 266–283, 1976.
27. A.A. Schäffer, Recognizing brittle graphs: remarks on a paper of Hoàng and Khouzam, *Discrete Appl. Math.* **31**, 29–35, 1991.

Coloring Powers of Graphs of Bounded Clique-Width

Ioan Todinca

LIFO - Université d'Orléans, Orléans 45067, Cedex 2, France
Ioan.Todinca@lifo.univ-orleans.fr

Abstract. Given a graph G, the graph G^l has the same vertex set and two vertices are adjacent in G^l if and only if they are at distance at most l in G. The l-coloring problem consists in finding an optimal vertex coloring of the graph G^l, where G the input graph. We show that, for any fixed value of l, the l-coloring problem is polynomial when restricted to graphs of bounded clique-width, if an expression of the graph is also part of the input.

1 Introduction

Many generalizations of the coloring problem of graphs were proposed as models for real-life applications. One of them is the radio-coloring problem (also called λ-coloring or $L(2,1)$-coloring), in which the vertices of the graph represent transmitters, and two vertices are adjacent if the two transmitters are very close. Each vertex must receive an integer frequency, such that adjacent vertices (very close transmitters) get frequencies differing by at least two units, and vertices at distance two in the graph (close transmitters) get different frequencies. The aim is to minimize the range of the used frequencies, i.e. we search for the minimum λ such that G admits a radio-coloring with frequencies between 0 and λ. Let us call $\lambda_{2,1}(G)$ this minimum. Another generalization of the classical coloring is the l-coloring problem, i. e. the coloring of G^l, where G is the input graph. Here $G^l = (V, E^l)$ denotes the l^{th} power of G: two vertices are adjacent in G^l if and only if they are at distance at most l in G. The radio-coloring and the l-coloring problems are related. The number $\lambda_{1,1}$ of colors necessary for a 2-coloring of G is at least $\lambda_{2,1}(G) + 1$, and conversely from an optimal 2-coloring of G we can easily obtain a radio-coloring of G with frequencies between 0 and $2\lambda_{1,1} - 1$. Hence, an algorithm solving the 2-coloring problem for a class of graphs also provides a 2-approximation for the radio-coloring problem.

Both problems are NP-hard for general graphs [4,5]. The l-coloring problem is polynomial for partial k-trees, for any fixed k and l [8].

In this paper we study the l-coloring problem on graphs of bounded clique-width. The clique-width of a graph G, denoted by $\mathrm{cwd}(G)$, is the minimum number of labels needed to construct the graph G, using four operators: $\bullet$, $\oplus$, ρ and η. The operation $\bullet_i$ creates a graph with a single vertex labeled i. The binary operator $\oplus$ constructs the union of two disjoint graphs. The operation

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, pp. 370–382, 2003.

$\rho_{i\to j}$ renames all vertices labeled i with label j. The operation $\eta_{i,j}$, with $i \neq j$, adds all the edges between every vertex of label i and every vertex of label j. Many NP-hard problems become polynomially solvable for graphs of bounded clique-width, if an expression of the graph is part of the input. The classical coloring problem is one of them [3,6]. Let us note that the partial k-trees are a particular case of graphs of bounded clique-width, more precisely if G is a partial k-tree than $\text{cwd}(G) \leq 2^{k+1} + 1$ [2]. Several other classes of graphs (distance-hereditary graphs, P_4-sparse and P_4-tidy graphs, ...) are known to have bounded clique-width.

The initial motivation for this work was the radio-coloring problem for graphs of bounded clique-width, but it turns out that the problem is NP-complete even for graphs of clique-width at most 3 [1]. We show in this article that the l-coloring problem is polynomial when restricted to graphs with bounded clique-width, if an expression of the graph is part of the input. As mentioned above, this provides a 2-approximation for the radio-coloring problem on this class of graphs.

After introducing the basic notions in Section 2, we show in Section 3 that, for any graph G of clique-width k, the clique-width of G^2 is at most $k2^{k+1}$. We also state that the clique-width of G^l is at most $2kl^k$. Using the coloring algorithm for graphs of bounded cliquewidth proposed in [6], this result directly implies that, for fixed k and l, the l-coloring problem is solvable in $\mathcal{O}(n^{2^{2kl^k+1}})$ time, when restricted to graphs of clique-width at most k if an expression of G is part of the input. In Section 4 we give an algorithm for the l-coloring problem with a $\mathcal{O}(n^{2^{2kl}+2^{kl+1}+1})$ time bound. Although the complexity remains extremely huge it is significantly better than the previous one. Moreover, this time bound is to be compared with the existing time bound for classical coloring problem on graphs with clique-width at most k ($\mathcal{O}(n^{2^{2k+1}})$, see [6]) or with the time-bound of the l-coloring problem on partial k-trees ($\mathcal{O}(n^{2^{2(k+1)(l+2)+1}})$, see [8]).

2 Basic Definitions

Throughout this paper we consider simple, finite, undirected graphs. Given a graph $G = (V, E)$, we denote by n the number of vertices of G. An edge $\{x, y\}$ will be simply denoted by xy. A path $\mu = [x_1, \ldots, x_l]$ is a sequence of vertices such that $x_i x_{i+1} \in E$, $\forall i, 1 \leq i < l$. The length $|\mu|$ of this path is $l - 1$. The distance between two vertices x and y of G, denoted by $d_G(x, y)$, is the length of the shortest path from x to y. We define by $G^l = (V, E^l)$ the l^{th} power of G, i.e. $E^l = \{xy \mid d_G(x, y) \leq l\}$.

Let $[k] = \{1, \ldots, k\}$ be the set of integers from 1 to k. A k-labeled graph $G = (V, E, \text{lab})$ is a graph (V, E) whose vertices are labeled by some mapping $\text{lab} : V \to [k]$. Given a labeled graph $G = (V, E, \text{lab})$, we denote by $\text{unlab}(G)$ the unlabeled graph (V, E).

Definition 1 (Clique-width [2]). *Let k be a positive integer. The class CW_k of k-labeled graphs is defined as follows.*

1. *For any $i \in [k]$, the graph $\bullet_i$ with a single vertex labeled i is in CW_k.*
2. *Given two disjoint graphs $G = (V_G, E_G, \text{lab}_G)$ and $H = (V_H, E_H, \text{lab}_H)$ in CW_k, their disjoint union $G \oplus H = (V', E', \text{lab}')$ defined by $V' = V_G \cup V_H$, $E' = E_G \cup E_H$, and*

$$\text{lab}'(x) = \begin{cases} \text{lab}_G(x) & \text{if } x \in V_G \\ \text{lab}_H(x) & \text{if } x \in V_H \end{cases}, \forall x \in V'$$

   is in CW_k.
3. *Given a labeled graph $G = (V, E, \text{lab}) \in CW_k$ and two distinct integers $i, j \in [k]$, the graphs $\rho_{i\to j}(G)$ and $\eta_{i,j}(G)$ are in CW_k, where*
   a) $\rho_{i\to j}(G) = (V, E, lab')$, where $lab'(x) = \begin{cases} lab(x) & \text{if } lab(x) \neq i \\ j & \text{if } lab(x) = i \end{cases}$
   b) $\eta_{i,j}(G) = (V, E', lab)$, where $E' = E \cup \{xy \mid x, y \in V,\ lab(x)=i,\ lab(y)=j\}$

An unlabeled graph $G = (V, E)$ is in CW_k if there is a labeling lab *of G such that $(V, E, \text{lab}) \in CW_k$. The* clique-width *of a graph $G = (V, E)$, denoted* $\text{cwd}(G)$*, is the smallest positive integer k such that $G \in CW_k$.*

To any graph $G \in CW_k$ we can associate an expression tree T using the operators $\bullet_i, \oplus, \rho_{i\to j}$ and $\eta_{i,j}$, with distinct integers $i, j \in [k]$. We say that T is a CW_k-expression tree and that G is the value $\text{val}(T)$ of T.

3 Clique-Width and Powers of Graphs

We show in this section that, for any a graph $G \in CW_k$, the clique-width of G^2 is at most $k2^{k+1}$. At the end of the section we discuss a similar bound for $\text{cwd}(G^l)$.

Notice that, in the definition of the class CW_k, the operator $\eta_{i,j}$ uses *distinct* integers i and j. For technical reasons, we want to allow the operator $\eta_{i,i}$. Let CW'_k the class of k-labeled graphs defined like CW_k, but allowing the operation $\eta_{i,i}$ for $i \in [k]$. Clearly $CW_k \subseteq CW'_k$, but these classes are not equal: a clique with $n \geq 2$ vertices is in CW'_1 and in CW_2, but not in CW_1. Nevertheless, we have the following proposition, whose proof is given in the full paper [7]:

Proposition 1. *For any positive integer k, $CW'_k \subseteq CW_{2k}$.*

Theorem 1. *Consider a labeled graph $G \in CW_k$ and let T be a CW_k-expression tree of G. We can construct a CW'-expression tree $f(T)$ with labels in $[k] \times \{0,1\}^k$ such that $\text{unlab}(\text{val}(f(T))) = (\text{unlab}(G))^2$ and, for each $x \in V$, the label $(i, b_1, b_2, \ldots, b_k)$ of x in $\text{val}(f(T))$ has the following properties:*

1. *i is the label of x in G;*
2. *for any $j \in [k]$, $b_j = 1$ if and only if x has a neighbor with label j in G.*

Proof. The tree $f(T)$ is obtained by parsing the tree T, from bottom to top. In the construction of $f(T)$, we distinguish four cases, depending whether the root of T is a $\bullet$ node, a ρ node, a η node or a $\oplus$ node.

If T has a unique node $\bullet_i$, then $f(T) = \bullet_{(i,0,0,\ldots,0)}$. If the root of T is a $\oplus$ node, let $T = T_1 \oplus T_2$, then $f(T) = f(T_1) \oplus f(T_2)$.

Suppose the root of T is a $\rho_{i\to j}$ node, let $T = \rho_{i\to j}(T_1)$. In order to obtain $f(T)$ from $f(T_1)$, we transform each label $L = (p, b_1, \ldots, b_k)$ of $f(T_1)$ into a label $L' = (p', b'_1, \ldots, b'_q)$ such that:

1. $p' = \begin{cases} j & \text{if } p = i \\ p & \text{otherwise} \end{cases}$
2. $b'_i = 0$, $b'_j = \max(b_i, b_j)$ and $b'_q = b_q$, $\forall q \in [k] - \{i, j\}$

The second condition expresses the fact that, for any vertex x of G, x has no neighbor with label i, and x has a neighbor with label j if and only if, before the $\rho_{i\to j}$ operation, x had a neighbor labeled i or a neighbor labeled j. Let $(L_1, L'_1), (L_2, L'_2), \ldots, (L_q, L'_q)$ be the couple of distinct labels of $f(T_1)$ satisfying the two conditions above. Then

$$f(T) = \rho_{L_1\to L'_1}(\rho_{L_2\to L'_2}(\ldots(\rho_{L_q\to L'_q}(f(T_1))))).$$

It remains to consider the case when the root of T is a $\eta_{i,j}$ node, so $T = \eta_{i,j}(T_1)$, and again we construct $f(T)$ from $f(T_1)$. Let $G_1 = \mathrm{val}(T_1)$. Let V_i be the set of vertices labeled i in G_1 and let N_i be the set of vertices of G_1 having at least one neighbor in V_i (the neighborhood is taken in the graph G_1). We define V_j and N_j in a similar way. If V_i or V_j are empty, then $G = G_1$, so $f(T) = f(T_1)$. Suppose now that V_i and V_j are not empty. Consider an edge xy of G^2 which is not an edge of G_1^2. If $\mathrm{dist}_G(x, y) = 1$, then (x, y) is contained in $V_i \times V_j$ or $V_j \times V_i$. If $\mathrm{dist}_G(x, y) = 2$, there is a path $[x, z, y]$ of length 2 in G which does not exist in G_1, so $z \in V_i \cup V_j$. Hence (x, y) or (y, x) is in one of the following sets: $V_i \times N_j$, $V_j \times N_i$, $V_i \times V_i$ or $V_j \times V_j$. Conversely, since V_i and V_j are not empty, for any couple (x, y) in one of these four sets, we observe that x and y are at distance at most 2 in G. We take all the couples of labels (L, L') of $f(T_1)$, $L = (p, b_1, \ldots, b_k)$, $L' = (p', b'_1, \ldots, b'_k)$ satisfying one of the following conditions: $(p, p') \in \{(i, j), (i, i), (j, j)\}$ or ($p = i$ and $b'_j = 1$) or ($p = j$ and $b'_i = 1$). Let $(L_1, L'_1), \ldots (L_q, L'_q)$ be these couples, we take

$$T'_1 = \eta_{L_1,L'_1}(\eta_{L_2,L'_2}(\ldots(\eta_{L_p,L'_p}(f(T_1))))).$$

Then the unlabeled graph corresponding to the expression tree T'_1 is exactly G^2. Observe that T'_1 may contain nodes of type $\eta_{L,L'}$ with $L = L'$ – and this is the reason why $f(T)$ is a CW'-expression tree, but not necessarily a CW-expression tree.

It remains to update the labels of T'_1, in order to satisfy the conditions 1 and 2 of our proposition. Each vertex labeled i (resp. j) in G has at least one neighbor labeled j (resp. i). Thus, each label $L = (i, b_1, b_2, \ldots, b_k)$ (resp. $L = (j, b_1, b_2, \ldots, b_k)$) is transformed into $L' = (i, b_1, \ldots, b_{j-1}, 1, b_{j+1}, \ldots, b_k)$

(resp. $L' = (j, b_1, \ldots, b_{i-1}, 1, b_{i+1}, \ldots, b_k)$) and all the other labels remain unchanged. We denote by $(L_1, L'_1), \ldots, (L_q, L'_q)$ the couples of distinct labels of $f(T_1)$ satisfying the conditions above. Then

$$f(T) = \rho_{L_1 \to L'_1}(\rho_{L_2 \to L'_2}(\ldots(\rho_{L_q \to L'_q}(T'_1)))).$$

□

From Theorem 1 and Proposition 1 we deduce:

Theorem 2. *For any graph G,* $\mathrm{cwd}(G^2) \leq cwd(G) \times 2^{\mathrm{cwd}(G)+1}$.

The result of Theorem 1 can be extended to arbitrary powers of G, For any graph G and any integer $l \geq 2$, $\mathrm{cwd}(G^l) \leq 2cwd(G) \times l^{\mathrm{cwd}(G)}$. We omit the proof of this statement. The technique is similar with the one used in the next section for the l-coloring problem.

4 The l-Coloring Problem for Graphs of Bounded Clique-Width

Given a graph $G \in CW_k$ and a k-expression of G, one can compute a coloring of G in polynomial time [3,6]. We can conclude that, for graphs of bounded clique-width and for bounded l, the chromatic number of G^l can be computed in $\mathcal{O}(n^{2^{2kl^k+1}+1})$ time, if the CW_k-expression tree of G is part of the input.

In this section, we give an algorithm for the l-coloring problem, with a $\mathcal{O}(n^{2^{2kl}+2^{kl+1}+1})$ time bound. The technique used here generalizes the coloring algorithm for graphs of bounded clique-width proposed by Kobler and Rotics. The data structures and the general framework of the algorithm for the l-coloring problem (Subsections 4.1 to 4.3) are very similar to [6], but the treatment of the $\oplus$ operation (Subsections 4.5 and 4.6) is completely different in our case.

4.1 A Sketch of the Algorithm

For computing a l-coloring of G, we parse the expression tree T of G from bottom to top. We may assume that T has at most k^2n nodes (see [6]). For each node u of T, let $T[u]$ denote the subtree of T rooted in u and let $G[u] = (V[u], E[u], \mathrm{lab}_u)$ be the graph $\mathrm{val}(T[u])$. We compute for each node u an information $F(u)$, representing some colorings of $G[u]$. A coloring c of $G[u]$ is simply a mapping $c : V[u] \to \mathbb{N}$. Let us say that a coloring c of $G[u]$ is *admissible* with respect to G if, for any $x, y \in V[u]$ such that $\mathrm{dist}_G(x, y) \leq l$, the vertices x and y have different colors. We point out the that the distance between x and y is taken in G and not in $G[u]$. The information $F(u)$ kept in the node u represents all the admissible colorings of $G[u]$. In particular, the root r of T will contain in $F(r)$ the information about the optimal l-colorings of G.

Subsection 4.2 gives a complete description of $F(u)$ and shows that $F(u)$ has polynomial size. In Subsection 4.3, we show how to compute this information in polynomial time.

4.2 Encoding of F(u)

We denote by $\langle l \rangle$ the set of integers $\{0, 1, \ldots, l-1\}$. Given a label $i \in [k]$ and a number $d \in \langle l \rangle$, let

$$V_{i,d}[u] = \{x \in V[u] \mid \exists y \in V[u] \text{ such that } lab_u(y) = i \text{ and } \text{dist}_G(x,y) \leq d\}$$

So $V_{i,0}[u] = V_i[u]$ denotes the set of vertices having label i in $G[u]$, and $V_{i,d}[u]$ is the set of vertices at distance at most d from $V_i[u]$, where the distance is taken in the graph G and not $G[u]$.

Consider a graph H and a coloring c of H. Let K be a finite set. For each $a \in K$ let X_a be a subset of $V(H)$ and let $\mathcal{X}$ be the family of not necessarily distinct sets $\{X_a \mid a \in K\}$. We suppose that the family covers $V(H)$, i.e. each vertex of $V(H)$ is in some set X_a of $\mathcal{X}$. Consider a subset $\mathcal{B}$ of K, we denote by $\mathcal{C}^c_{\mathcal{X}}(\mathcal{B})$ the set of colors appearing in $c(X_a)$ for each $a \in \mathcal{B}$ and not appearing in $c(X_b)$ for each $b \notin \mathcal{B}$. We associate to c an array $N^c_{\mathcal{X}}$ with $2^{|K|}$ elements, defined by $N^c_{\mathcal{X}}(\mathcal{B}) = |\mathcal{C}^c_{\mathcal{X}}(\mathcal{B})|$ for all subsets $\mathcal{B}$ of K. Observe that the sets $\mathcal{C}^c_{\mathcal{X}}(\mathcal{B})$ form a partition of the colors of c. Indeed, for any color *col* used by c, there is a unique subset $\mathcal{B}_{col}$ of K such that $col \in \mathcal{C}^c_{\mathcal{X}}(\mathcal{B}_{col})$. More precisely, $\mathcal{B}_{col} = \{a \in K \mid col \in c(X_a)\}$. Therefore, the number of colors used by c is $\sum_{\mathcal{B} \subseteq K} N^c_{\mathcal{X}}(\mathcal{B})$.

Given a node u of the tree T, we use the above definitions with $H = G[u]$ and $K = [k] \times \langle l \rangle$. For each $(i,d) \in [k] \times \langle l \rangle$, $X_{(i,d)} = V_{i,d}[u]$, so the family $\mathcal{X}[u]$ is the family of sets $\{V_{i,d} \mid (i,d) \in [k] \times \langle l \rangle\}$. Clearly, this family covers $V[u]$. To simplify the notation, for any coloring c of $G[u]$ and any $\mathcal{B} \subseteq [k] \times \langle l \rangle$ we denote by $\mathcal{C}^c_u(\mathcal{B})$ the set $\mathcal{C}^c_{\mathcal{X}[u]}[\mathcal{B}]$. Let $N^c_u(\mathcal{B})$ be the cardinality of $\mathcal{C}^c_u(\mathcal{B})$. A matrix N *represents a coloring* c *of* $G[u]$ *in the node* u if $N = N^c_u$. A matrix may represent several colorings, but all these colorings use the same number of colors. If c is an admissible coloring with respect to G, then we say that $N = N^c_u$ is admissible. We will see later that, given a matrix N representing some colorings in the node u, either all these colorings are admissible, or they are all non admissible w.r.t. G.

The information $F(u)$ related to the node u is precisely the set of all the admissible matrices w.r.t. u. Since each admissible matrix is an array with 2^{kl} elements in $\{0, \ldots, |V[u]|\}$, the size of $F(u)$ is $\mathcal{O}((|V[u]|+1)^{2^{kl}})$.

4.3 Computation of $F(u)$

First Case: u *Is a* $\bullet$ *Node.* When u is a leaf node $\bullet_i$, let $\mathcal{B}_0 = \{(i,d) \mid 0 \leq d \leq l-1\}$. Clearly, the only admissible matrix N satisfies $N(\mathcal{B}_0) = 1$ and $N(\mathcal{B}) = 0$ for all subsets $\mathcal{B} \neq \mathcal{B}_0$ of $[k] \times \langle l \rangle$.

Second Case: u *Is a* ρ *Node.* Let v be the son of u in T. The proof of the following lemma is straightforward.

Lemma 1. *Suppose that the node u is labeled $\rho_{i\to j}$ and let v be the son of u in T. Then, for each $d \in \langle l \rangle$, we have that $V_{i,d}[u] = \emptyset$, $V_{j,d}[u] = V_{j,d}[v] \cup V_{i,d}[v]$ and $V_{q,d}[u] = V_{q,d}[v]$, $\forall q \in [k] - \{i, j\}$.*

For computing the admissible matrices of $F(u)$ from the admissible matrices of $N(v)$ we use the following lemma – which is also stated, in different terms, in [6]. See [6,7] for a proof.

Lemma 2 (See also [6]). *Let H be a graph, c a coloring of H and $\mathcal{X} = \{X_q \mid q \in K\}$ a family of subsets of $V(H)$, covering $V(H)$. Consider two distinct elements $a, b \in K$ and a new family $\mathcal{Y} = \{Y_q \mid q \in K\}$ defined by $Y_a = \emptyset$, $Y_b = X_a \cup X_b$ and $Y_q = X_q$ for any $q \in K - \{a, b\}$.*

Then, for any $\mathcal{B} \subseteq K$,

1. *$N^c_{\mathcal{Y}}(\mathcal{B}) = N^c_{\mathcal{X}}(\mathcal{B})$ if a and b are not in $\mathcal{B}$.*
2. *$N^c_{\mathcal{Y}}(\mathcal{B}) = 0$ if $a \in \mathcal{B}$.*
3. *$N^c_{\mathcal{Y}}(\mathcal{B}) = N^c_{\mathcal{X}}(\mathcal{B}) + N^c_{\mathcal{X}}(\mathcal{B} \cup \{a\}) + N^c_{\mathcal{X}}(\mathcal{B} - \{b\} \cup \{a\})$ if $a \notin \mathcal{B}$ and $b \in \mathcal{B}$.*

Clearly, $\mathrm{unlab}(G[u]) = \mathrm{unlab}(G[v])$, so a coloring c of $G[u]$ is admissible with respect to u if and only if the same coloring is admissible with respect to v. Given an admissible coloring c of $G[v]$, we have to compute the matrix N^c_u from N^c_v. We consider the family $\mathcal{X} = \{V_{i',d'}[v] \mid (i', d') \in [k] \times \langle l \rangle\}$, so $N^c_{\mathcal{X}} = N^c_v$. For each $d \in \langle l \rangle$ we replace in $\mathcal{X}$ the set $V_{i,d}[v]$ by $V_{i,d}[u] = \emptyset$ and the set $V_{j,d}[v]$ by $V_{j,d}[v] = V_{i,d}[v] \cup V_{j,d}[v]$ and we compute the matrix $N^c_{\mathcal{X}}$ for this new family using Lemma 2. According to Lemma 1, at the end of this loop we obtain the matrix N^c_u.

Third Case: u Is a η Node. If v is the son of u in T, clearly $V[u] = V[v]$ and the labels of any vertex $x \in V[u]$ are the same in $G[u]$ and in $G[v]$. Since the admissible colorings of $G[u]$ are exactly the admissible colorings of $G[v]$, we have $F(u) = F(v)$.

Fourth Case: u Is a $\oplus$ Node. Let v and w be the two children of u in T. The algorithm for computing $F(u)$ proceeds by considering each pair of matrices $(N_1, N_2) \in F(v) \times F(w)$ and by computing all the matrices $N \in F(u)$ such that N represents an admissible coloring c of $G[u]$ whose restriction to $G[v]$ (resp. $G[w]$) is represented by N_1 (resp. N_2).

Consider a coloring c of $G[u]$ and let e (resp. f) be the restriction of c to $G[v]$ (resp $G[w]$). Let $N_1 = N^e_v$ (resp. $N_2 = N^f_w$) be the matrices representing e (resp. f) in the node v (resp. w). We associate to c an array S defined as follows. For any couple of non-empty sets $\mathcal{B}_1, \mathcal{B}_2 \in [k] \times \langle l \rangle$, let $S(\mathcal{B}_1, \mathcal{B}_2)$ be the number of colors occuring both in $\mathcal{C}^e_v(\mathcal{B}_1)$ and $\mathcal{C}^f_w(\mathcal{B}_2)$. Let $S(\mathcal{B}_1, \emptyset)$ (resp. $S(\emptyset, \mathcal{B}_2)$) be the number of colors appearing in $\mathcal{C}^e_v(\mathcal{B}_1)$ (resp. $\mathcal{C}^f_w(\mathcal{B}_2)$) and not appearing in $f(G[w])$ (resp. $e(G[v])$). We put $S(\emptyset, \emptyset) = 0$. We say that the matrices S, N_1 and N_2 *represent the coloring c of $G[u]$ in the nodes* (v, w). Indeed, in subsection 4.6 we will see that the matrix N^c_u only depends on N_1, N_2 and S and we give an explicit algorithm for computing this matrix.

The following proposition is given in a slightly different form in [6], for the proof see also the full version of the paper [7].

Proposition 2 (See also [6]). *Let c be a coloring of $G[u]$ and let S, N_1 and N_2 be the three matrices representing c in the nodes (v, w). Then:*

1. $S(\emptyset, \emptyset) = 0$.
2. *For each non-empty set $B_1 \subseteq [k] \times \langle l \rangle$, we have* $\sum_{\mathcal{B} \subseteq [k] \times \langle l \rangle} S(\mathcal{B}_1, \mathcal{B}) = N_1(\mathcal{B}_1)$.
3. *For each non-empty set $B_2 \subseteq [k] \times \langle l \rangle$, we have* $\sum_{\mathcal{B} \subseteq [k] \times \langle l \rangle} S(\mathcal{B}, \mathcal{B}_2) = N_2(\mathcal{B}_2)$.

Conversely, let N_1 be a matrix representing a coloring of $G[v]$ in the node v, N_2 be a matrix representing a coloring of $G[w]$ in the node w and S be a matrix of positive integers satisfying the above conditions. There is a coloring c of $G[u]$ such that c is represented in the nodes (v, w) by S, N_1 and N_2.

We want to compute all the matrices of $F(u)$, representing in the node u the admissible colorings of $G[u]$. If c is an admissible coloring of $G[u]$, then its restrictions to $G[v]$ and $G[w]$ are also admissible. Hence, if S, N_1 and N_2 are the matrices representing c in the nodes (v, w), we have $N_1 \in F(v)$ and $N_2 \in F(w)$.

For computing $F(u)$ we proceed as follows. For each couple of matrices $N_1 \in F(v)$ and $N_2 \in F(w)$, we generate all the $2^{kl} \times 2^{kl}$ matrices S with elements in $\{0, \ldots, n\}$. For each matrix S, we

1. Check in that S satisfies the conditions of Proposition 2.
2. Check if S, N_1 and N_2 represent an admissible coloring of $G[u]$, using Theorem 4 of Subsection 4.5. Indeed, for any coloring c of $G[u]$ represented in the nodes (v, w) by the three matrices, the admissibility of c only depends on the matrices.
3. Compute the matrix $N(S, N_1, N_2)$, representing in the node u the same coloring as S, N_1 and N_2 in the nodes (v, w), using Theorem 5 of Subsection 4.6. Indeed, for any coloring c of $G[u]$ represented in the nodes (v, w) by the three matrices, the matrix N_u^c is the same.

The second and the third point of the algorithm will be postponed to the next subsections.

4.4 Complexity of the Algorithm

We deduce that the information $F(u)$ can be computed for each node u of T, and in particular we know how to obtain, in the root r of T, the matrix N_r^c representing a l-coloring c of G with minimum number of colors. By keeping the matrices S used for the merging operations, we can also compute an optimal l-coloring of G.

Theorem 3. *The l-coloring problem can be solved in $\mathcal{O}(k^3 l^3 2^{3kl} n^{2^{2kl}+2^{kl+1}+1})$ time on the class of graphs of clique-width at most k, if a CW_k-expression of the input graph is also given.*

Proof. It remains to discuss the complexity of the algorithm, which is given by the cost of the $\oplus$ operation. Let u be a $\oplus$ node and v and w be its descendants. Then $|V[v]| \leq n-1$ and $|V[w]| \leq n-1$, so $F(v)$ and $F(u)$ have at most $n^{2^{kl}}$ matrices. We consider each couple of matrices $(N_1, N_2) \in F(v) \times F(w)$, we generate all the $n^{2^{2kl}}$ matrices S of size $2^{kl} \times 2^{kl}$ with elements in $\{0, \ldots, n-1\}$ and we use Proposition 2, Theorem 4 and Theorem 5 for computing (or rejecting) the matrix $N(S, N_1, N_2)$. Since there are at most n nodes of type $\oplus$ the theorem follows. □

4.5 Checking if S, N_1, and N_2 Represent an Admissible Coloring of $G[u]$

The aim of this part is to prove the following theorem.

Theorem 4. *Given three matrices $S, N_1 \in F(v)$ and $N_2 \in F(w)$ representing a coloring c of $G[u]$ in the nodes (v, w), we can verify if c is admissible in $\mathcal{O}(k^2 l^2 2^{2kl})$ time.*

We denote by $\text{dist}_G^{\text{ext}}(v, i; w, j)$ the length of the shortest path μ of G such that μ goes from a vertex labeled i in $G[v]$ to a vertex labeled j in $G[w]$ and μ does not intersect $V[v]$ or $V[w]$, except in its extremities. We may suppose that, for each $\oplus$ node u of the expression-tree T, the distances $\text{dist}_G^{\text{ext}}(v, i; w, j)$ are calculated, for all couples $(i, j) \in [k]^2$, in a preprocessing phase. This calculation is made, for each $\oplus$ node, by an all-pairs shortest paths algorithm. There are at most n nodes of type $\oplus$, so this preprocessing step costs $\mathcal{O}(n^4)$ time.

Let us start on an easy remark on $\text{dist}_G^{\text{ext}}(v, i; w, j)$.

Lemma 3. *Let $x \in V_i[v]$ and $y \in V_j[w]$. Then $\text{dist}_G(x, y) \leq \text{dist}_G^{\text{ext}}(v, i; w, j)$.*

Proof. Let $\mu = [x_1, x_2, \ldots, x_p]$ be a shortest path from $V_i[v]$ to $V_j[w]$ in G, such that the interior of μ does not meet $V_i[v]$ or $V_j[w]$. Then $x_1 \in V_i[v]$ and $x_p \in V_j[w]$. The second vertex x_2 of μ is not in $V[v]$, and since it is adjacent to x_1 in G it also adjacent to all vertices having the same label as x_1 in $G[v]$, by definition of the class CW_k. In particular, x_2 is adjacent to x. For similar reasons, x_{p-1} is adjacent to y in G. So $\mu' = [x'_1 = x, x_2, \ldots, x_{p-1}, x'_p = y]$ is a path of G. Hence, $\text{dist}_G(x, y) \leq |\mu'| = \text{dist}_G^{\text{ext}}(v, i; w, j)$. □

Lemma 4. *Let $x \in V[v]$ and $y \in V[w]$. Then $\text{dist}_G(x, y) \leq l$ if and only if there are $i, j \in [k]$, $d, d' \in \langle l \rangle$ such that $x \in V_{i,d}[v]$, $y \in V_{j,d'}[w]$ and $d + d' + \text{dist}_G^{\text{ext}}(v, i; w, j) \leq l$.*

Proof. Suppose there are i, j, d and d' such that $x \in V_{i,d}[v]$, $y \in V_{j,d'}[w]$ and $\text{dist}_G(v, i; w, j) + d + d' \leq l$, we show that $\text{dist}_G(x, y) \leq l$. Since $x \in V_{i,d}[v]$, there is a vertex $x' \in V_i[v]$ such that $\text{dist}_G(x, x') \leq d$. For similar reasons, there is a vertex $y' \in V_j[w]$ such that $\text{dist}_G(y, y') \leq d'$. By Lemma 3, $\text{dist}_G(x', y') \leq \text{dist}_G^{\text{ext}}(v, i; w, j)$. Hence, $\text{dist}_G(x, y) \leq \text{dist}_G(x, x') + \text{dist}_G(x', y') + \text{dist}_G(y', y) \leq d + d' + \text{dist}_G^{\text{ext}}(v, i; w, j) \leq l$.

Conversely, we show that if $\text{dist}_G(x,y) \leq l$, then there are i,j,d and d' satisfying the conditions of the lemma. Consider a shortest path $\mu = [x = x_1, \ldots, x_p = y]$ from x to y in G. Let $s, 1 < s \leq p$ be smallest index such that $x_s \in V[w]$ and $q, 1 \leq q < s$ be the highest index such that $x_q \in V[v]$. Let i (resp. j) be the label of x_q (resp. x_s) in $G[v]$ (resp. $G[w]$). The sub-path $\mu' = [x_q, x_{q+1}, \ldots, x_s]$ of μ intersects $V[v]$ and $V[w]$ only in its ends. Thus, by definition of $\text{dist}^{\text{ext}}_G(v,i;w,j)$, we have that $\text{dist}^{\text{ext}}_G(v,i;w,j) \leq |\mu'|$. Let d (resp. d') be the length of the sub-path from x to x_q (resp. from x_q to y) of μ. Clearly, $x \in V_{i,d}[v]$, $y \in V_{j,d'}[w]$ and the length of μ is $d + d' + |\mu'|$. We conclude that $d + d' + \text{dist}^{\text{ext}}_G(v,i;w,j) \leq l$. □

Proposition 3. *Let S, $N_1 \in F(v)$ and $N_2 \in F(w)$ be three matrices representing a coloring c of $G[u]$. Then c is admissible w.r.t. G if and only if, for all $(i,d),(j,d') \in [k] \times \langle l \rangle$ such that $d + d' + \text{dist}_G(v,i;w,j) \leq l$ and for all sets $\mathcal{B}_1, \mathcal{B}_2 \in [k] \times \langle l \rangle$ with $(i,d) \in \mathcal{B}_1$ and $(j,d') \in \mathcal{B}_2$ we have that $S(\mathcal{B}_1, \mathcal{B}_2) = 0$.*

Proof. Let e (resp. f) be the restrictions of c to $G[v]$ (resp. $G[w]$).

Suppose that the matrix S does not satisfy the condition of the proposition, so there are two non-empty sets $\mathcal{B}_1, \mathcal{B}_2 \subset [k] \times \langle l \rangle$ with $S(\mathcal{B}_1, \mathcal{B}_2) > 0$ and two couples $(i,d) \in \mathcal{B}_1$ and $(j,d') \in \mathcal{B}_2$ such that $d + d' + \text{dist}_G(v,i;w,j) \leq l$. Since $S(\mathcal{B}_1, \mathcal{B}_2) > 0$, there is at least one color of c appearing in $\mathcal{C}^e_v(\mathcal{B}_1)$ and in $\mathcal{C}^f_w(\mathcal{B}_2)$. In particular, this colors occurs in $e(V_{i,d}[v])$ and in $f(V_{j,d'}[w])$. Hence, there are two vertices $x \in V_{i,d}[v]$ and $y \in V_{j,d'}[w]$ such that $e(x) = f(y)$. By Lemma 4, the inequality $d + d' + \text{dist}_G(v,i;w,j) \leq l$ implies $\text{dist}_G(x,y) \leq l$, and therefore the coloring c is not an admissible w.r.t. G.

Conversely, suppose that c is not admissible w.r.t. G, we show that S does not satisfiy the condition of the proposition. The coloring e of $G[v]$ is represented in the node v by $N_1 \in F(v)$ so, by construction of $F(v)$, e is admissible w.r.t. G. For similar reasons, f is also admissible w.r.t. G. Since c is not admissible, the only possibility is the existence of two vertices $x \in V[v]$ and $y \in V[w]$ such that $\text{dist}_G(x,y) \leq l$ and $e(x) = f(y)$. By Lemma 4, there are two couples $(i,d),(j,d') \in [k] \times \langle l \rangle$ such that $x \in V_{i,d}[v]$, $y \in V_{j,d'}[w]$ and $d + d' + \text{dist}^{\text{ext}}_G(v,i;w,j) \leq l$. Let $\mathcal{B}_1 = \{(i'',d'') \in [k] \times \langle l \rangle \mid e(x) \in e(V_{i'',d''}[v])\}$ and $\mathcal{B}_2 = \{(i'',d'') \in [k] \times \langle l \rangle \mid f(x) \in f(V_{i'',d''}[w])\}$. In particular, $(i,d) \in \mathcal{B}_1$ and $(j,d') \in \mathcal{B}_2$. Hence $e(x) = f(y)$ occurs in $\mathcal{C}^e_v(\mathcal{B}_1)$ and in $\mathcal{C}^f_w(\mathcal{B}_2)$. Therefore $S(\mathcal{B}_1, \mathcal{B}_2) > 0$, so S does not satisfy the conditions of the proposition. □

Given three matrices S, $N_1 \in F(v)$ and $N_2 \in F(w)$ representing a coloring c of $F(v)$, we can check if c is admissible w.r.t. G in $\mathcal{O}(k^2 l^2 2^{2kl})$ time, if we assume that the distances $\text{dist}_G(v,i;w,j)$ were computed in a preprocessing step. Indeed, we take all the tuples $(i,j,d,d') \in [k] \times [k] \times \langle l \rangle \times \langle l \rangle$, and if $d + d' + \text{dist}_G(v,i;w,j) \leq l$, we can check if S satisfies the conditions of Proposition 3. There are $k^2 l^2$ tuples (i,j,d,d'), and for each of tuple we need $\mathcal{O}(2^{2kl})$ time for verifying the matrix S. This achieves the proof of Theorem 4.

4.6 Computing the Matrix N

We show in Lemma 5 that for any couple $(i,d) \in [k] \times \langle l \rangle$, the set $V_{i,d}[u]$ is the union of some sets $V_{i',d'}[v]$ and $V_{i'',d''}[w]$. We then use this observation in Theorem 5, for computing the matrix N from S, N_1 and N_2.

For any node t of T, we denote by $\text{dist}_G(t;i,j) = \text{dist}_G(V_i[t], V_j[t])$. Again we suppose that the distances $\text{dist}_G(t;i,j)$ are computed in a preprocessing step, which takes $\mathcal{O}(n^4)$ time.

Lemma 5. *Let $x \in V[v]$, $i \in [k]$ and $d \in \langle l \rangle$. Then $x \in V_{i,d}[u]$ if and only if one of the following holds:*

1. *$x \in V_{i,d}[v]$;*
2. *there exist $j, j' \in [k]$ and $d' \in \langle l \rangle$ such that $x \in V_{j,d'}[v]$ and*

$$d' + \text{dist}_G^{\text{ext}}(v, j; w, j') + \text{dist}_G(w; j', i) \leq d$$

Proof. We show that if one of the two conditions holds, then $x \in V_{i,d}[u]$. Clearly, if $x \in V_{i,d}[v]$ then $x \in V_{i,d}[u]$. Suppose now that x satisfies the second condition. Let $x' \in V_j[v]$ such that $\text{dist}_G(x,x') \leq d'$. Consider $y' \in V_{j'}[w]$ and $y \in V_i[w]$ such that $\text{dist}_G(y',y) = \text{dist}_G(w;j',i)$ (notice that if $d' = 0$ then $x = x'$, and if $j' = i$ then $y = y'$). According to Lemma 3, $\text{dist}_G(x',y') \leq \text{dist}_G^{\text{ext}}(v,j;w,j')$. We deduce that $\text{dist}_G(x,y) \leq \text{dist}_G(x,x') + \text{dist}_G(x',y') + \text{dist}_G(y',y) \leq d$. Since y has label i in $G[u]$, we conclude that $x \in V_{i,d}[u]$.

Conversely, suppose there is a vertex y labeled i in $G[u]$ s.t. $\text{dist}_G(x,y) \leq d$. We show that one of the two conditions holds. If $y \in V[v]$ then the first condition is true by definition of $V_{i,d}[v]$. Otherwise, take $y \in V_i[w]$ such that $\text{dist}_G(x,y)$ be minimum. Let $\mu = [x = x_1, \ldots, x_p = y]$ be a shortest path from x to y in G. Let $y' = x_s$ be the first vertex of $\mu \cap V[w]$. We denote by $x' = x_q$ the last vertex of $\mu \cap V[v]$, encountered before y (so $q < s$). Then x' and y' split μ into three paths: an x,x'-path μ_1, an x',y'-path μ_2 and an y',y-path μ_3. Let j be the label of x' in $G[v]$, j' be the label of y' in $G[w]$ and d' be the length of μ_1. Observe that μ_2 only intersects $V[u]$ in x' and y', so $\text{dist}_G^{\text{ext}}(v,j;w,j') \leq |\mu_2|$. Clearly, $\text{dist}_G(w;j',i) \leq |\mu_3|$. Then $d' + \text{dist}_G^{\text{ext}}(v,j;w,j') + \text{dist}_G(w;j',i) \leq |\mu_1| + |\mu_2| + |\mu_3| = |\mu| \leq d$. Therefore, d', j and j' satisfy the second condition of our theorem. □

The proof of the next lemma is omitted due to space restrictions. See the full paper [7].

Lemma 6. *Let H be a graph, c be a coloring of H and $\mathcal{X} = \{X_q \mid q \in K\}$ a family of subsets of $V(H)$, covering $V(H)$. Consider two distinct elements $a, b \in K$ and a new family $\mathcal{Y} = \{Y_q \mid q \in K\}$ defined by $Y_b = X_a \cup X_b$ and $Y_q = X_q$ for any $q \in K - \{b\}$.*

Then, for any $\mathcal{B} \subseteq K$,

1. *$N_{\mathcal{Y}}^c(\mathcal{B}) = N_{\mathcal{X}}^c(\mathcal{B})$ if a and b are not in $\mathcal{B}$.*
2. *$N_{\mathcal{Y}}^c(\mathcal{B}) = 0$ if $a \in \mathcal{B}$ and $b \notin B$.*
3. *$N_{\mathcal{Y}}^c(\mathcal{B}) = N_{\mathcal{X}}^c(\mathcal{B}) + N_{\mathcal{X}}^c(\mathcal{B} - \{b\})$ if $a, b \in \mathcal{B}$.*

Theorem 5. *Consider three matrices S, N_1, N_2 representing a coloring c in the nodes (v, w). The matrix N representing the coloring c in the node u is computable in $\mathcal{O}(k^3 l^3 2^{3kl})$ time.*

Proof. Let $K = [k] \times \langle l \rangle \times \{v, w, u\}$. We construct a family $\mathcal{X} = \{X_a \mid a \in K\}$ of subsets of $V[u]$, covering $V[u]$, as follows: for any element $a = (i, d, t)$ of K, we put $X_a = V_{i,d}[t]$ if $t \in \{v, w\}$, and $X_a = \emptyset$ if $t = u$. We compute first the matrix $N^c_{\mathcal{X}}$. Let $\mathcal{B}$ be a subset of K and, for each $t \in \{v, u, w\}$, let $\mathcal{B}_t = \{(i, d) \mid (i, d, t) \in K\}$. We show that $N^c_{\mathcal{X}}(\mathcal{B}) = \begin{cases} 0 & \text{if } \mathcal{B}_u \neq \emptyset \\ S(\mathcal{B}_v, \mathcal{B}_w) & \text{otherwise} \end{cases}$

If $\mathcal{B}_u$ contains a couple (i, d), since $X_{(i,d,u)}$ is empty and $(i, d, u) \in \mathcal{B}$, we have $N^c_{\mathcal{X}}(\mathcal{B}) = 0$. If $\mathcal{B}_u = \emptyset$, then by definition of $\mathcal{X}$ the set $\mathcal{C}^c_{\mathcal{X}}(\mathcal{B})$ is formed by the colors of c

- appearing in $c(V_{i,d}[v])$ for each $(i, d) \in \mathcal{B}_v$ and not appearing in $c(V_{i',d'}[v])$ for each $(i', d') \notin \mathcal{B}_v$;
- appearing in $c(V_{i,d}[w])$ for each $(i, d) \in \mathcal{B}_w$ and not appearing in $c(V_{i',d'}[v])$ for each $(i', d') \notin \mathcal{B}_w$;

Hence $N^c_{\mathcal{X}}(\mathcal{B}) = S(\mathcal{B}_v, \mathcal{B}_w)$. Thus, computing the matrix $N^c_{\mathcal{X}}$ takes $\mathcal{O}(kl2^{3kl})$ time (each of the 2^{3kl} elements of $N^c_{\mathcal{X}}$ is identified by the sets $\mathcal{B}_u, \mathcal{B}_v$ and $\mathcal{B}_w$ of size kl, and for a given couple $\mathcal{B}_v, \mathcal{B}_w$ we access $S(\mathcal{B}_v, B_w)$ in $\mathcal{O}(kl)$ time).

Let now $\mathcal{Y} = \{Y_a \mid a \in K\}$ be defined by $Y_{(i,d,t)} = V_{i,d}[t]$ for each $(i, d, t) \in [k] \times \langle l \rangle \times \{v, w, u\}$. We compute the matrix $N^c_{\mathcal{Y}}$ as follows. For each $(i, d) \in [k] \times \langle l \rangle$, replace in $\mathcal{X}$ the empty set $X_{(i,d,u)}$ with the set $Y_{(i,d,u)} = V_{i,d}[u]$. According to Lemma 5, $Y_{(i,d,u)}$ is the union of certain elements of $\mathcal{X}$. Indeed, $Y_{(i,d,u)} = V_{i,d}[u]$ is the union of some sets $V_{i',d'}[v] = X_{(i',d',v)}$ and $V_{i'',d''}[w] = X_{(i'',d'',w)}$. Therefore, after each replacement we can update the matrix $N^c_{\mathcal{X}}$ by applying Lemma 6 at most $2kl$ times. This costs $\mathcal{O}(k^2 l^2 2^{3kl})$ time by replacement. At the end of this loop, $\mathcal{X} = \mathcal{Y}$ and we have computed $N^c_{\mathcal{Y}}$. This computation takes $\mathcal{O}(k^3 l^3 2^{3kl})$ time.

It remains to calculate N^c_u. Consider three subsets $\mathcal{B}_v, \mathcal{B}_w$ and $\mathcal{B}_u$ of $[k] \times \langle l \rangle$, and let $\mathcal{B}_v \oplus \mathcal{B}_w \oplus \mathcal{B}_u = \{(i, d, t) \in K \mid (i, d) \in \mathcal{B}_t\}$. For any $\mathcal{B}_u \subseteq [k] \times \langle l \rangle$, $N^c_u(\mathcal{B}_u) = \sum_{\mathcal{B}_v, \mathcal{B}_w \subseteq [k] \times \langle l \rangle} N^c_{\mathcal{Y}}(\mathcal{B}_v \oplus \mathcal{B}_w \oplus \mathcal{B}_u)$. The last step costs $\mathcal{O}(kl2^{3kl})$ time. So the matrix N^c_u can be computed from $N^c_{\mathcal{Y}}$ in $\mathcal{O}(k^3 l^3 2^{3kl})$ time, which concludes the proof of Theorem 5. □

References

1. H. Bodlaender and D. Kratsch, 2002. Private communication.
2. B. Courcelle and S. Olariu. Upper bounds to the clique-width of graphs. *Discrete Applied Mathematics*, 101:77–114, 2000.
3. W. Espelage, F. Gurski, and E. Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In *Workshop on Graph-theoretic Concepts in Computer Science (WG 2001)*, volume 2204 of *Lecture Notes in Computer Science*, pages 117–128. Springer, 2001.

4. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completness*. Freeman, 1979.
5. J.R. Grigs and R.K. Yeh. Labelling graphs with a condition at distance 2. *SIAM J. on Discrete Math.*, 5:586–595, 1992.
6. D. Kobler and U. Rotics. Polynomial algorithms for partitioning problems on graphs with fixed clique-width (extended abstract). In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms (SODA 2001)*, pages 468–476, 2001.
7. I. Todinca. Coloring powers of graphs of bounded clique-width. Research Report RR-2002-10, LIFO – Université d'Orléans, 2002. http://www.univ-orleans.fr/SCIENCES/LIFO/prodsci/rapports/RR/RR2002/RR-2002-10.ps.gz.
8. X. Zhou, Y. Kanari, and T. Nishizeki. Generalized vertex-colorings for partial k-trees. *IEICE Trans. Fundamentals*, E83-A(4):671–677, april 2000.

Erratum: Cycles in Generalized Networks

Franz J. Brandenburg

University of Passau, 94030 Passau, Germany
brandenb@informatik.uni-passau.de

Abstract. The paper "Cycles in generalized networks" [2], which appeared in WG 2002, claims that the single source generalized shortest path problem SGSP can be solved in $O(nm \log n)$. This is incorrect. The fastest known algorithms take $O(n^2 m \log n)$ time. Below we briefly describe the problem and discuss which of our results hold.

Generalized networks specify two parameters for each arc, a cost and a gain. If x units enter an arc $x \cdot g(a)$ exit. The gains multiply along a path, and the cost is the sum of the cost of each arc. The objective is a cheapest path of a unit flow from the source, SGSP.

SGSP can be solved in $O(n^2 m \log n)$ time by specialized linear programming techniques or by Oldham's algorithm [3]. The difficulties of our approach arise from cheap lossy cycles. The cost and the gain of a cycle γ are defined by traversing γ once. γ is *lossy*, if its gain is less than one. If a lossy cycle is traversed infinitely often, it consumes all flow with the accumulated cost of $c^*(\gamma) = c(\gamma)/(1 - g(\gamma))$. γ is a *cheap* cycle, if the cost of entering γ at some node v and then consuming all flow is less than the cost of the cheapest path from v to the sink. The algorithm in [2] detects every cheap lossy cycle γ and finds a representative node v in the cycle. It then assigns a value $c_{\text{cycle}}(v)$ to v. However, $c_{\text{cycle}}(v) \geq c^*(\gamma)$, and equality is needed for the correctness of the algorithm. This can be obtained by further runs of the algorithm, but then there is no known polynomial time bound. This parallels a result of Ahuja et al. [1] who could not bound the number of iterations of their generalized network simplex algorithm for the generalized min-cost flow problem.

As stated in [2] SGSP can be solved in $O(nm)$ for lossless networks, and in $O(n \log n + m)$ in a monotonous setting, and the single-pair generalized shortest path problem is NP-hard.

References

1. R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Network Flows,* Prentice Hall, Englewood Cliffs, 1993.
2. F.J. Brandenburg, *Cycles in generalized networks,* Proc. 28th Workshop Graph-Theoretic Concepts in Computer Science, WG 2002, Lecture Notes in Computer Science Vol. 2573 (2003), 47–57.
3. J.D. Oldham, *Combinatorial approximation algorithms for generalized flow problems,* J. Algorithms, **38** (2001), 135–168.

H.L. Bodlaender (Ed.): WG 2003, LNCS 2880, p. 383, 2003.

Author Index