Luděk Kučera (Ed.)

LNCS 2573

Graph-Theoretic Concepts in Computer Science

28th International Workshop, WG 2002
Český Krumlov, Czech Republic, June 2002
Revised Papers

Springer

Lecture Notes in Computer Science 2573

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

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Luděk Kučera (Ed.)

Graph-Theoretic Concepts in Computer Science

28th International Workshop, WG 2002
Český Krumlov, Czech Republic, June 13-15, 2002
Revised Papers

Springer

Volume Editors

Luděk Kučera
Charles University Department of Applied Mathematics
Malostranské nám. 25
118 00 Prague, Czech Republic
E-mail: ludek@kam.mff.cuni.cz

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.1.2, G.1.6, G.2, G.3, E.1, I.3.5

ISSN 0302-9743
ISBN 3-540-00331-2 Springer-Verlag Berlin Heidelberg New York

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

http://www.springer.de

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 10872302 06/3142 5 4 3 2 1 0

Preface

The 28th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2002) was held in Český Krumlov, a beautiful small town in the southern part of the Czech Republic on the river Vltava (Moldau), June 13–15, 2002. The workshop was organized by the Department of Applied Mathematics of the Faculty of Mathematics and Physics of Charles University in Prague.

Since 1975, WG has taken place in Germany 20 times, twice in Austria and The Netherlands, and once in Italy, Slovakia, and Switzerland. As in previous years, the workshop aimed 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.The workshop was devoted to the theoretical and practical aspects of graph concepts in computer science, and its contributed talks 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.

Altogether 61 research papers were submitted and reviewed by the program committee. The program committee represented the wide scientific spectrum, and in a careful reviewing process with four reports per submission it selected 36 papers for presentation at the workshop. The referees' comments as well as the numerous fruitful discussions during the workshop have been taken into account by the authors of these conference proceedings.

The participants of WG 2002 came from universities and research institutes in various countries such as Australia, Belgium, Brazil, Canada (2), Czech Republic (6), France (7), Germany (13), Great Britain, Greece (2), Hong Kong, Ireland, Israel (2), Italy (2), Japan (2), New Zeeland, Norway (6), Poland (3), Russia, Slovakia, Spain, The Netherlands (2), U.S.A. The unusually small number of participants from the U.S.A. and Canada was most likely due to the short space of time between 11th of September, 2001 and the workshop deadline.

It is our pleasure to thank all those who contributed to the scientific success of WG 2001: all the authors of submitted and of presented papers, and in particular the speakers, the referees, and the subreferees. The organizational work of Lucie Štěpánová, Viliam Holub, and Štěpán Kučera during the workshop and the historical ambience, technical facilities, and helpful personnel of the Hotel Růže in Český Krumlov greatly contributed to the succeess of the workshop. A concert given by Olga Štěpánová (mezzosoprano), Lucie Štěpánová (violloncello), and Alexandr Zlobin (piano) in Castle Kratochvíle was a beautiful counterpart to the exhausting scientific program of the workshop.

November 2002 Luděk Kučera

Program Committee

Hans Bodlaender, Universiteit Utrecht (NL)
Andreas Brandstädt, Universität Rostock (DE)
Michel Habib, LIRMM Montpellier (F)
Juraj Hromkovič, RWTH Aachen (DE)
Michael Kaufmann, Universität Tübingen (DE)
Luděk Kučera, Karlova Universita, Prague (CZ), chair
Alberto Marchetti-Spaccamela, Università di Roma "La Sapienza" (I)
Ernst Mayr, TU München (DE)
Manfred Nagl, RWTH Aachen (DE)
Hartmut Noltemeier, Universität Würzburg (DE)
Andrzej Proskurowski, University of Oregon, Eugene (US)
Ondrej Sýkora, Loughborough University (UK)
Gottfried Tinhofer, TU München (DE)
Dorothea Wagner, Universität Konstanz (D)
Peter Widmayer, ETH Zürich (CH)
Christos Zaroliagis, CTI Patras (GR)

Additional Reviewers

Jochen Alber, Luzi Anderegg, Maria I. Andreou, Hans-Joachim Böckenhauer, Thomas Bayer, Dirk Bongartz, Franz Brandenburg, Ulrik Brandes, Kathie Cameron, Serafino Cicerone, Mark Cieliebak, Andrea Clementi, Olivier Cogis, Sabine Cornelsen, Camil Demetrescu, Ingo Demgensky, Joerg Derungs, Hristo Djidjev, Stefan Dobrev, Feodor F. Dragan, Stephan Eidenbenz, Frank van den Eijkhof, Jens Ernst, Dimitris Fotakis, Jean-Luc Fouquet, Cyril Gavoille, H.-D.O.F. Gronau, Yubao Guo, Jens Gustedt, Volker Heun, Klaus Holzapfel, Miriam DiIanni, Ton Kloks, Spyros Kontogiannis, Sven Kosub, Dieter Kratsch, Van Bang Le, Zsuzsanna Lipták, Matthias Müller, Moritz Maaß, Maria Flavia Mammana, Sebastian Maneth, Fabien de Montgolfier, Matthew Newton, Sotiris Nikoletseas, Alexander Offtermatt-Souza, Gabriel Okša, Aris Pagourtzis, Jérôme Palaysi, Vicky Papadopoulou, Francesco Parisi-Presicce, Christophe Paul, Leon Peeters, Paolo Penna, Guido Proietti, Martin Raab, Udi Rotics, Thomas Schickinger, Konrad Schlude, Frank Schulz, Sebastian Seibert, Riccardo Silvestri, Jeremy Spinrad, Ladislav Stacho, Ludwig Staiger, Yannis Stamatiou, Gabriele Di Stefano, Gabor Szabo, Hanjo Täubig, Dimitrios M. Thilikos, Peter Ullrich, Walter Unger, Jean-Marie Vanherpe, Imrich Vrťo, Birgitta Weber, Thomas Willhalm, Hans-Christoph Wirth, Mark Withall.

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 Bernried, 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 Fischbachau, 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 Jakob, Konstanz
2001 A. Brandstädt – Boltenhagen, near Rostock
2002 L. Kučera – Český Krumlov

Table of Contents

Maximum Cardinality Search for Computing Minimal Triangulations

Anne Berry[1], Jean R. S. Blair[2], and Pinar Heggernes[3]

[1] LIMOS, Universite Clermont-Ferrand II, F-63177 Aubiere, France, berry@isima.fr
[2] US Military Academy, West Point, NY, USA, Jean-Blair@usma.edu
[3] Informatics, University of Bergen, N-5020 Bergen, Norway, pinar@ii.uib.no

Abstract. We present a new algorithm, called MCS-M, for computing minimal triangulations of graphs. Lex-BFS, a seminal algorithm for recognizing chordal graphs, was the genesis for two other classical algorithms: Lex-M and MCS. Lex-M extends the fundamental concept used in Lex-BFS, resulting in an algorithm that also computes a minimal triangulation of an arbitrary graph. MCS simplified the fundamental concept used in Lex-BFS, resulting in a simpler algorithm for recognizing chordal graphs. The new simpler algorithm MCS-M combines the extension of Lex-M with the simplification of MCS, achieving all the results of Lex-M in the same time complexity.

1 Introduction

Many important problems in graph theory rely on the computation of a chordal completion or, equivalently, a triangulation of a graph. Typically the goal is to compute a *minimum* triangulation, that is, a triangulation with the fewest number of edges. Computing a minimum triangulation is NP-hard [11]. In this extended abstract, we study the problem of finding a minimal triangulation. A *minimal* triangulation H of a given graph G is a triangulation such that no subgraph of H is a triangulation of G.

Several practical algorithms exist for finding minimal triangulations [1], [2], [3], [5], [8], [9]. One such classical algorithm, called Lex-M [9], is derived from the Lex-BFS (lexicographic breadth first search) algorithm [9] for recognizing chordal graphs. Both Lex-BFS and Lex-M use lexicographic labels of the unprocessed vertices. As processing continues, the remaining labels grow, each potentially reaching a length proportional to the number of vertices in the graph. Lex-BFS adds to the labels of the neighbors of the vertex being processed, while Lex-M adds to the labels of vertices that can be reached along special kinds of paths. Interestingly, the simple extension of adding to labels based on reachability along special kinds of paths, rather than only along single edges, results in an algorithm that produces minimal triangulations.

The adjacency-labeling concepts developed for the Lex-BFS algorithm have proved to be central in the understanding of chordal graphs and triangulations. Tarjan and Yannakakis later came up with the surprising result that for the

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 1–12, 2002.

case of recognizing chordality, knowing the specific processed neighbors (i.e., labels) is not necessary; one need only maintain and compare the cardinality of processed neighbors [10]. This was a major breakthrough, resulting in a significantly simplified implementation of Lex-BFS which has come to be known as the MCS (maximum cardinality search) algorithm. A natural question that arises is whether or not cardinality comparisons are also sufficient for the case of minimal triangulations. That is, is there a significantly simplified implementation of Lex-M that uses only the cardinality of processed vertices that can be reached along special kinds of paths? Or, equivalently, can MCS be extended from neighbors to paths in order to yield a minimal triangulation algorithm, imaging the extension from Lex-BFS to Lex-M? In this paper, we introduce an algorithm called MCS-M to fill exactly this gap.

The relationships between the four algorithms discussed thus far are summarized in Figure 1. In the figure, the algorithms on the left recognize chordal graphs while those on the right produce provably minimal triangulations of arbitrary graphs, as well as recognizing chordality. Both algorithms on the left have time complexity $O(n + m)$; both algorithms on the right have time complexity $O(nm)$.

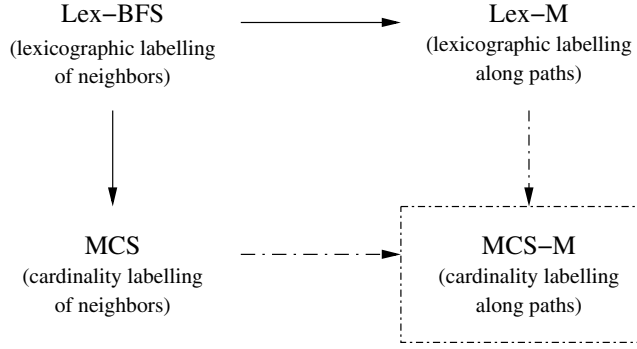


Fig. 1. Relationships between algorithms. Solid arrows represent previous evolution. Dashed arrows represent the natural evolution to a new MCS-M algorithm.

This paper is organized as follows. In the next section we assume that the reader is familiar with standard graph terminology, and briefly review only a few key definitions before presenting background material. Included in that section is a classical characterization of minimal triangulations that forms the basis for our proofs of correctness. The three algorithms that lead to the results in this paper are presented in Section 3. In Section 4 we present the new minimal triangulation algorithm MCS-M, and prove its correctness.

2 Background

All graphs in this work are undirected and finite. A graph is denoted by $G = (V, E)$, with $n \equiv |V|$, and $m \equiv |E|$. The *neighborhood* of a vertex x in G is $N_G(x) = \{y \neq x \mid xy \in E\}$. The neighborhood of a set of vertices A is $N_G(A) =$

$\cup_{x \in A} N_G(x) - A$, and we define $N_G[A] = N_G(A) \cup A$. When the graph G is clear from the context, we will omit the subscript G.

A *clique* is a set of pairwise adjacent vertices. A vertex x is *simplicial* if $N(x)$ is a clique. A *chord* of a cycle is an edge connecting two non-consecutive vertices of the cycle. A graph is *chordal*, or equivalently *triangulated*, if it contains no chordless cycle of length ≥ 4. A *triangulation* of a graph G is a chordal graph $G^+ = (V, E \cup F)$ that results from the addition of a set F of fill edges.

Given any graph $G = (V, E)$, an *elimination ordering* α on G is simply a numbering of the vertices of G with integers from 1 to n. The algorithm shown

Algorithm EliminationGame
Input: A general graph G, and an elimination ordering α of the vertices in G.
Output: The filled graph G^+_α.
begin
  $G^0 = G$;
  for $i = 1$ **to** n **do**
    Let v be the vertex for which $\alpha(v) = i$;
    Add edges to G^{i-1} so that $N_{G^{i-1}}(v)$ becomes a clique;
    $G^i = G^{i-1} - v$;
  $G^+_\alpha = \cup_{i=0}^{n-1} G^i$;
end

Fig. 2. The elimination game.

in Figure 2, called the *elimination game*, was first introduced by Parter [7]. For any graph G and any ordering α of G, we will denote by G^k_α the transitory graph after step k of the elimination game on G. The resulting filled graph G^+_α is a triangulation of G [4]. The ordering α is a *perfect elimination ordering* if no fill edges are added during the elimination game i.e. $G^+_\alpha = G$. Note that this is equivalent to choosing a simplicial vertex at each step of the elimination game. Fulkerson and Gross [4] showed that the class of chordal graphs is exactly the class of graphs having perfect elimination orderings.

The following theorem characterizes the edges of the filled graph.

Theorem 1. (Rose, Tarjan, and Lueker [9]) *Given a graph $G = (V, E)$ and an elimination ordering α of G, yz is an edge in G^+_α if and only if $yz \in E$ or there exists a path $y, x_1, x_2, ..., x_k, z$ in G where $\alpha(x_i) < \min\{\alpha(y), \alpha(z)\}$, for $1 \leq i \leq k$.*

Ohtsuki, Cheung, and Fujisawa [6] define α to be a *minimal elimination ordering* if G^+_α is a minimal triangulation of G and further characterize a sufficient condition for a vertex to be numbered one in a minimal elimination ordering. Below we define an OCF-vertex (OCF representing the initials of the authors of [6]) as a vertex that satisfies their condition and summarize in a theorem their results that are key in proving the correctness of our algorithm.

Definition 1. *A vertex x in $G = (V, E)$ is an OCF-vertex if, for each pair of non-adjacent vertices $y, z \in N(x)$, there is a path $y, x_1, x_2, ..., x_k, z$ in G where $x_i \in G - N[x]$, for $1 \leq i \leq k$.*

Theorem 2. (Ohtsuki, Cheung, and Fujisawa [6]) *A minimal elimination ordering α is computed by choosing an OCF-vertex x in G^{i-1} for elimination so that $\alpha(x) = i$, at each step i of the elimination game.*

3 Lex-BFS, Lex-M, and MCS Algorithms

The MCS algorithm, which is shown in Figure 3, is a simple linear time algorithm that processes first the vertex x for which $\alpha(x) = n$ and continues generating an elimination ordering in reverse. The MCS algorithm maintains, for each vertex v, an integer weight $w(v)$ that is the cardinality of the already processed neighbors of v. When given a chordal graph as input, MCS produces a perfect elimination ordering.

Algorithm MaximumCardinalitySearch - MCS
Input: A graph G.
Output: An elimination ordering α of G.
begin
  for all vertices v in G **do** $w(v) = 0$;
  for $i = n$ **downto** 1 **do**
    Choose an unnumbered vertex z of maximum weight; $\alpha(z) = i$;
    for all unnumbered vertices $y \in N(z)$ **do** $w(y) = w(y) + 1$;
end

Fig. 3. Maximum Cardinality Search.

Lex-BFS has the exact same description as MCS, but uses labels that are lists of the names of the already processed neighbors instead of using weights. In the beginning $l(v) = \emptyset$ for all vertices. At step $n - i + 1$, an unnumbered vertex v of lexicographically highest label is chosen to receive number i, and i is added to the end of the label lists of all unnumbered neighbors of v.

Lex-M is an extension of Lex-BFS that computes a minimal triangulation in the following way. When v receives number i at step $n - i + 1$, it adds i to the end of the label lists of all unnumbered vertices x for which there exists a path between v and x consisting only of unnumbered vertices with lexicographically lower labels than those of v and x.

The fact that using weights rather than the labels of Lex-BFS is sufficient for computing a perfect elimination ordering was a major breakthrough, resulting in the substantially simpler implementation of MCS. In the next section we show that using weights rather than the labels of Lex-M is also sufficient for computing a minimal triangulation. This results in a substantially simpler implementation of Lex-M which we call MCS-M.

Throughout the remainder of this paper, while speaking about MCS or MCS-M, the following phrases are considered to be equivalent: *u is numbered higher than v* and *u is processed earlier than v*. The symbols $v-$ and $v+$ are used as time stamps, denoting the time right before and right after v receives its number. For any two vertices u and v, where v is numbered higher than u during an execution of MCS or MCS-M, $w_{v-}(u)$ is the weight of u at time $v-$, and $w_{v+}(u)$ is the weight of u at time $v+$. Analogously, $h_{v-}(A)$ and $h_{v+}(A)$ denote the highest

weight of a vertex among the unnumbered vertices of $A \subseteq V$, at times $v-$ and $v+$, respectively.

4 The New MCS-M Algorithm

The new algorithm MCS-M is an extension of MCS in the same way that Lex-M is an extension of Lex-BFS. That is, in MCS-M when v receives number i at step $n - i + 1$, it increments the weight of all unnumbered vertices x for which there exists a path between v and x consisting only of unnumbered vertices with weight strictly less than $w_{v-}(v)$ and $w_{v-}(x)$. The details of this $O(nm)$ time algorithm are given in Figure 4. An example of an MCS-M ordering on a given graph is shown in Figure 5(a).

Algorithm MCS-M
Input: A general graph $G = (V, E)$.
Output: A minimal elimination ordering α of G and the corresponding filled graph H.
begin
  $F = \emptyset$; **for** all vertices v in G **do** $w(v) = 0$;
  for $i = n$ **downto** 1 **do**
    Choose an unnumbered vertex z of maximum weight; $\alpha(z) = i$;
    for all unnumbered vertices $y \in G$ **do**
      if there is a path $y, x_1, x_2, ..., x_k, z$ in G through unnumbered vertices
      such that $w_{z-}(x_i) < w_{z-}(y)$ for $1 \leq i \leq k$ **then**
        $w(y) = w(y) + 1$;
        $F = F \cup \{yz\}$;
  $H = (V, E \cup F)$;
end

Fig. 4. The MCS-M algorithm.

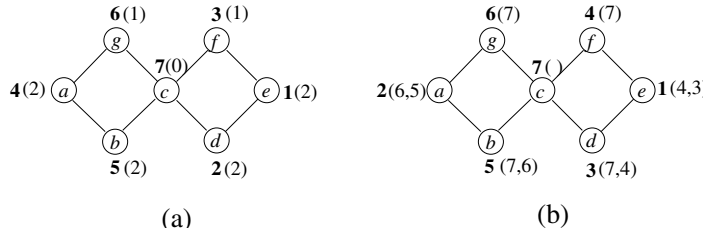


Fig. 5. (a) An MCS-M numbering. (b) A Lex-M numbering. Numbers in bold represent the produced ordering α. The weight/label of each vertex at the time it receives its number is given in parentheses.

We will show that MCS-M simulates a process of choosing an OCF vertex at each step of the elimination game, thereby producing a minimal triangulation. We begin by proving a property about paths with lower weight intermediary vertices, after which we prove that MCS-M produces exactly the same graph as the one that would be produced by the elimination game using the ordering α produced by MCS-M.

Lemma 1. *Let α be an elimination ordering produced by an execution of MCS-M on G. For any step of MCS-M, let v be the vertex chosen to receive its number. Among the unnumbered vertices, if*

$$w_{v-}(x_i) < w_{v-}(y) \leq w_{v-}(z)$$

for all x_i on a path $y, x_1, x_2, ..., x_r, z$ in G, then

$$\alpha(x_i) < \min\{\alpha(y), \alpha(z)\}.$$

Proof. Suppose there is a path for which $w_{v-}(x_i) < w_{v-}(y) \leq w_{v-}(z)$ as in the premise of the lemma. Note that for any u such that $\alpha(v) > \alpha(u) > \min\{\alpha(x_i), \alpha(y), \alpha(z)\}$, $w_{u-}(u) \geq \max\{w_{u-}(y), w_{u-}(z)\}$. Thus, if $w_{u-}(x_i) < \min\{w_{u-}(y), w_{u-}(z)\}$ then any lower weight path from u to some x_i that causes $w_{u+}(x_i) = w_{u-}(x_i) + 1$, can be extended as a lower weight path through x_i to y and z causing $w_{u+}(y) = w_{u-}(y) + 1$ and $w_{u+}(z) = w_{u-}(z) + 1$. Since MCS-M always chooses next a vertex with highest weight to receive the highest remaining number, the result follows by induction. ∎

Theorem 3. *Let H and α be the graph and ordering produced by an execution of MCS-M on G. Then $H = G^+_\alpha$.*

Proof. Given an input graph G, let α be the elimination ordering and H be the supergraph computed by an execution of MCS-M. In order to prove that $H = G^+_\alpha$, we will prove that a fill edge yz with $\alpha(y) < \alpha(z)$ is added by MCS-M if and only if there is a path $y, x_1, x_2, ..., x_r, z$ in G with $\alpha(x_i) < \alpha(y)$ for $1 \leq i \leq r$. The result will then follow from Theorem 1. **(⇒)** Since yz is added, there is a path $y, x_1, x_2, ..., x_r, z$ in G where x_i is unnumbered with $w_{z-}(x_i) < w_{z-}(y) \leq w_{z-}(z)$ for $1 \leq i \leq r$. Then by Lemma 1 $\alpha(x_i) < \alpha(y)$, for $1 \leq i \leq r$. **(⇐)** Let $X = \{x_1, x_2, ..., x_r\}$. Since z is the first to receive its number among all mentioned vertices, $w_{z-}(z) \geq w_{z-}(y)$ and $w_{z-}(z) \geq h_{z-}(X)$. We want to prove that $h_{z-}(X) < w_{z-}(y)$, which means that $w(y)$ is incremented and yz is added when z receives its number. Assume on the contrary that $h_{z-}(X) \geq w_{z-}(y)$ and that yz is not added. Then $h_{z+}(X) > w_{z+}(y)$. Let j be the index such that $x_j \in X$ is the closest to y among vertices of X with $w_{z+}(x_j) > w_{z+}(y)$. When a vertex q receives its number and increments $w(y)$ for the first time after the numbering of z, it will also increment $w(x_j)$ since y is on the path between x_j and q and has lower weight. Thus we cannot increment $w(y)$ without incrementing $w(x_j)$, which contradicts that $\alpha(y) > \alpha(x_j)$. ∎

We have shown that the filled graph produced by MCS-M is equivalent to the graph produced by the elimination game using the same ordering. In proving our main lemma (Lemma 4), we will use this to infer the existence of fill edges added during MCS-M, which in turn implies the existence of paths in G through lower numbered vertices. First we prove two other necessary results.

Lemma 2. *Let α be an elimination ordering produced by an execution of MCS-M on G. For any step of MCS-M, let v be the vertex chosen to receive its number. Among the unnumbered vertices, if*

$$w_{v-}(x_i) < w_{v-}(y) \leq w_{v-}(z)$$

for all x_i on a path $y, x_1, x_2, ..., x_r, z$ in G, then for all u with $\alpha(u) > \alpha(v)$,

$$w_{u-}(x_i) \leq \min\{w_{u-}(y), w_{u-}(z)\}.$$

Proof. Let v and the path $y, x_1, x_2, ..., x_r, z$ in G be as stated in the premise and suppose to the contrary that for some vertex u with $\alpha(u) > \alpha(v)$, there exists a vertex x_i on the path for which $w_{u-}(x_i) > \min\{w_{u-}(y), w_{u-}(z)\}$. Without loss of generality assume $w_{u-}(y) \leq w_{u-}(z)$ and let u and x_i be such that x_i is the the closest vertex to y on the path that has $w_{u-}(x_i) > w_{u-}(y)$ at some time before v is numbered. Let p_x be the portion of the path between y and x_i. Thus, we have $w_{u-}(x_i) > w_{u-}(y) \geq w_{u-}(x_j)$ for all x_j on p_x. Since $w_{v-}(y) > w_{v-}(x_i)$ for the later (lower) numbered vertex v, there must be a vertex q, $\alpha(u) > \alpha(q) > \alpha(v)$, such that $w_{q-}(x_j) \leq w_{q-}(y) < w_{q-}(x_i)$ and $w_{q+}(y) = w_{q+}(x_i)$. But this cannot happen for the following reasons. The fact that q is the next to be numbered vertex means that $w_{q-}(q) \geq w_{q-}(x_i)$. The increase of y when q is numbered means there is a path (possibly a single edge), say p_1, between q and y that allowed the weight of y to be increased. The path $q - p_1 - y - p_x - x_i$ then is a lower weight path between q and x_i that would result in the weight of x_i being incremented as well, contradicting the assumption that the weight of y and not x_i is increased when q is numbered. ∎

Lemma 3. *Let α be an elimination ordering produced by an execution of MCS-M and consider the vertices u and v_1, $\alpha(u) < \alpha(v_1)$, such that MCS-M increments $w(u)$ through a path (or single edge) $p_v = v_1, v_2, \cdots, v_r, u$ of lower weight intermediate vertices when processing v_1. Let x be any vertex with $\alpha(x) < \alpha(u)$ and define $k = \alpha(x)$. If $w_{v_1-}(x) = w_{v_1-}(u)$ and $v_i x$ is an edge in G_α^{k-1} for some $1 \leq i \leq r$ then MCS-M also increments $w(x)$ when processing v_1.*

Proof. Assume $w_{v_1-}(x) = w_{v_1-}(u)$ and $v_i x$ is an edge in G_α^{k-1}. Either xv_i is an edge in G or it is a fill edge introduced when v_i is numbered by MCS-M. In either case, there is a path p_{small} (or single edge) connecting x and v_i in G such that $h_{v_i-}(p_{small}) < w_{v_i-}(x) \leq w_{v_i-}(v_i)$. Applying Lemma 2 we see that

$$\begin{aligned} h_{v_1-}(p_{small}) &\leq \min\{w_{v_1-}(x), w_{v_1-}(v_1)\} \\ &\leq w_{v_1-}(v_1) \\ &< w_{v_1-}(u) \ \text{ (by the definition of } p_v) \\ &= w_{v_1-}(x) \end{aligned}$$

It follows that $v_2, \cdots, v_i - p_{small}$ is a lower weight path through unnumbered vertices connecting v_1 and x just before v_1 is processed by MCS-M. Thus, $w_{u+}(x) = w_{u-}(x) + 1$. ∎

Lemma 4. *Let α be an elimination ordering produced by an execution of MCS-M. For $1 \leq k \leq n$, if $\alpha(y) = k$ then y is an OCF vertex in G_α^{k-1}.*

Proof. Let α be an elimination ordering produced by an execution of MCS-M, and consider a vertex y_0 with $\alpha(y_0) = k$. Let y_1 and y_2 be any two vertices in $N_{G_\alpha^{k-1}}(y_0)$ with $y_1 y_2 \notin E(G_\alpha^{k-1})$. We will show that there exists a path p_h between y_1 and y_2 in G_α^{k-1} with all intermediate vertices belonging to $G_\alpha^{k-1} - N_{G_\alpha^{k-1}}[y_0]$, thereby proving that y_0 is an OCF vertex in G_α^{k-1}.

Without loss of generality, assume $\alpha(y_1) < \alpha(y_2)$ and hence that $\alpha(y_0) < \alpha(y_1) < \alpha(y_2)$. Since $y_0 y_1$ is an edge in G_α^{k-1}, either $y_0 y_1$ is in G or it is introduced by MCS-M when y_1 is processed. In either case, at time y_1- there is a

path (or possibly an edge) $p_{y_0y_1}$ in G through unnumbered vertices such that $h_{y_1-}(p_{y_0y_1}) < w_{y_1-}(y_0) \leq w_{y_1-}(y_1)$. Likewise at time y_2- there is a path $p_{y_0y_2}$ through unnumbered vertices such that $h_{y_2-}(p_{y_0y_2}) < w_{y_2-}(y_0) \leq w_{y_2-}(y_2)$.

The fill edge $y_1y_2 \in G^+_\alpha$ because it is introduced during the elimination game by y_0. It follows then from Theorem 3 that the edge y_1y_2 is introduced by MCS-M when y_2 is numbered. Hence there is a path $y_1, v_1, v_2, \cdots, v_r, y_2$, $r \geq 1$, such that $w_{y_2-}(v_i) < w_{y_2-}(y_1) \leq w_{y_2-}(y_2)$, for all $1 \leq i \leq r$. If $w_{y_2-}(y_0) < w_{y_2-}(y_1)$, then the path $p_{y_0y_1} - y_0 - p_{y_0y_2}$ provides such a path. We consider first, however, the case where $w_{y_2-}(y_0) \geq w_{y_2-}(y_1)$.

Observe that since $y_1y_2 \notin G^{k-1}_\alpha$, there is at least one vertex on $p_{alt} = v_1, v_2, \cdots v_r$ that is higher numbered then y_0. We will show that the vertices on p_{alt} that are higher numbered than y_0 form the desired path p_h in G^{k-1}_α. By Theorem 1 the vertices on p_{alt} that are higher numbered than y_0 induce a path in G^{k-1}_α between y_1 and y_2. Thus, we need only show that no vertex on p_{alt} is adjacent to y_0 in G^{k-1}_α.

Assume to the contrary that there is a vertex v_i on p_{alt} that is adjacent to y_0 in G^{k-1}_α. Either y_0v_i is an edge in G or it is a fill edge introduced when v_i is numbered by MCS-M. In either case, there is a path (or edge) p_{small} connecting y_0 and v_i in G such that $h_{v_i-}(p_{small}) < w_{v_i-}(y_0) \leq w_{v_i-}(v_i)$. Applying Lemma 2 we see that $h_{y_2-}(p_{small}) \leq \min\{w_{y_2-}(y_0), w_{y_2-}(v_i)\} \leq w_{y_2-}(v_i)$. We further know that $w_{y_2-}(v_i) < w_{y_2-}(y_1) \leq w_{y_2-}(y_0)$, since p_{alt} is the path through which MCS-M added the edge y_1y_2. Therefore $h_{y_2-}(p_{small}) < w_{y_2-}(y_0)$. This gives us two paths in G:

$$y_0 - p_{small} - v_i, v_{i-1}, ..., v_1, y_1$$

and

$$y_0 - p_{small} - v_i, v_{i+1}, ..., v_r, y_2$$

that, just before y_2 is numbered by MCS-M, satisfy the premise to Lemma 1. Combined the two paths contain all of the vertices of p_{alt} as internal vertices. Thus, by Lemma 1 we can conclude that every vertex v_i on p_{alt} is such that $\alpha(v_i) < \alpha(y_0)$, contradicting the fact that at least one vertex on p_{alt} is numbered higher than y_0. It follows that our assumption that v_i is adjacent to y_0 in G^{k-1}_α was wrong, and therefore that the path p_h of vertices on p_{alt} that are higher numbered than y_0 is a path in G^{k-1}_α from y_1 to y_2 through vertices that are not adjacent to y_0.

We are left then with the case where the path $p_{y_0y_1} - y_0 - p_{y_0y_2}$ is a path of lower weight vertices between y_2 and y_1 just before y_2 is numbered, and hence $w_{y_2-}(y_0) < w_{y_2-}(y_1)$. In this case we know that there is some vertex, say y_3 that first (earliest in MCS-M) increases the weight of y_1 to a value greater than the weight of y_0.

The path p_h will be constructed iteratively from its two endpoints, y_1 and y_2, towards its center, through y_3 and vertices like it. That is, we will be growing two subpaths, p_{odd} from y_1 and p_{even} from y_2, that will eventually meet to form p_h. Simultaneously we will show by induction that the vertices on p_{odd} and p_{even} are not adjacent to y_0 in G^{k-1}_α. In order to prove this, we will utilize properties of another path that goes through y_0 and overlaps with portions of p_{odd} and p_{even}.

At odd steps of the induction the subpath p_{odd} is extended from its endpoint that is furtherest away from y_1; at even steps of the induction the subpath p_{even} is extended from its endpoint that is furtherest away from y_2. If y_{i-2} is the current to-be-extended endpoint, then the corresponding subpath is extended through a particular path to a vertex y_i. These extensions are defined as follows. Let y_i be the first vertex for which $w_{y_i+}(y_{i-2}) > w_{y_i+}(y_{i-3})$. Note then that $w_{y_i-}(y_{i-2}) = w_{y_i-}(y_{i-3})$, and $w_{y_i+}(y_{i-2}) = w_{y_i-}(y_{i-2}) + 1$. Define $p_{y_i y_{i-2}}$ to be the path of lower weight unnumbered vertices at time y_i- through which MCS-M increments the weight of y_{i-2}. The partial path of p_h is then defined recursively as follows.

$$p_h^i = \begin{cases} \emptyset & \text{if } i = 2 \\ p_h^{i-1} \text{ extended to include } p_{y_i y_{i-2}} \text{ and } y_i & \text{if } i > 2 \end{cases}$$

The overlapping path that goes through y_0 is defined recursively as follows.

$$p_0^i = \begin{cases} p_{y_0 y_1} - y_0 - p_{y_0 y_2} & \text{if } i = 2 \\ p_0^{i-1} \text{ extended to include } p_{y_i y_{i-2}} \text{ and } y_{i-2} & \text{if } i > 2 \end{cases}$$

The path p_0^5 and the corresponding p_h^5 are shown in Figure 6. Note that for $i > 2$ the path p_0^i contains all of the partial path p_h^i except for its two internal endpoints.

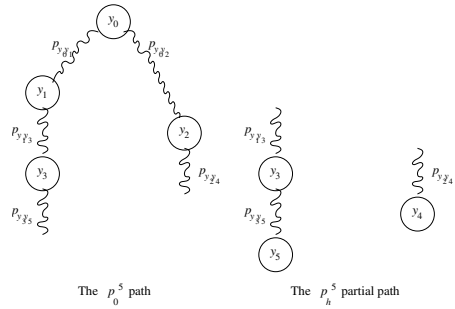


Fig. 6. The path p_0^5 and the partial path p_h^5.

There are four properties, shown in the induction hypothesis below, that we will maintain throughout the induction. The second and third are properties of the p_0 path and the last is the desired property of the p_h subpaths.

Induction hypotheses: For all l, $2 \leq l < i$, the following properties hold:

α-ORDER: $\alpha(y_{l-1}) < \alpha(y_l) < \alpha(y_{l+1})$.

SAME-WEIGHT: $w_t(y_0) = w_t(y_j)$ for $1 \leq j \leq l-1$ at times $t = y_{l+1}-$ and earlier.

p_0-WEIGHT: $h_t(p_0^l) = w_t(y_0) \leq \min\{w_t(y_l), w_t(y_{l-1})\}$ at times $t = y_l-$ and earlier.

NO-y_0-ADJ: No vertex on p_h^l is adjacent to y_0 in G_α^{k-1}.

Base case ($i = 2$): Here we begin with the fact that $\alpha(y_0) < \alpha(y_1) < \alpha(y_2)$ and observe that $\alpha(y_3) > \alpha(y_2)$, since at the time that y_2 is processed by MCS-M the weight of y_1 is already higher than the weight of y_0. Thus, the α-ORDER property holds for the base case.

The SAME-WEIGHT property trivially holds since, by definition of y_3, $w_t(y_0) = w_t(y_1)$ at all times $t = y_3-$ and earlier.

For the p_0-WEIGHT property recall that when $p_{y_0y_1}$ and $p_{y_0y_2}$ were defined above we saw that $h_{y_1-}(p_{y_0y_1}) < w_{y_1-}(y_0) \leq w_{y_1-}(y_1)$ and $h_{y_2-}(p_{y_0y_2}) < w_{y_2-}(y_0) \leq w_{y_2-}(y_2)$. Combining these two facts with the fact that $w_{y_2-}(y_0) < w_{y_2-}(y_1) \leq w_{y_2-}(y_2)$ and applying Lemma 2, we see that $h_t(p_{y_0y_1} - y_0 - p_{y_0y_2}) \leq \min\{w_t(y_1), w_t(y_2)\}$ at all times $t = y_2-$ and earlier, giving the base case p_0-WEIGHT property.

The NO-y_0-ADJ property holds since at this first step in the iteration the path p_h^2 is empty, and hence has no adjacency to y_0 in G_α^{k-1}.

Induction step ($i > 2$): Assume the induction hypotheses hold. We must either close the path p_h at this step, or prove the four properties hold for the i^{th} iteration. We begin by establishing that $w_{y_i-}(y_{i-2}) \leq w_{y_i-}(y_{i-1})$. By definition of y_i,

$$\begin{aligned} w_{y_i-}(y_{i-2}) &= w_{y_i-}(y_{i-3}) \\ &\leq h_{y_i-}(p_0^{i-1}) \quad \text{(because } y_{i-3} \in p_0^i\text{)} \\ &\leq \min\{w_{y_i-}(y_{i-1}), w_{y_i-}(y_{i-2})\} \quad \text{(by the induction hypothesis)} \end{aligned}$$

Thus, $w_{y_i-}(y_{i-2}) \leq w_{y_i-}(y_{i-1})$, and we have two cases.

Case 1 ($w_{y_i-}(y_{i-2}) = w_{y_i-}(y_{i-1})$): In this case y_i must also increment the weight of y_{i-1} through a path p_{alt} not containing y_0. To see this, observe first that it cannot increment the weight of y_{i-1} using a path containing y_0 since, by the SAME-WEIGHT induction hypothesis, $w_{y_i-}(y_0) = w_{y_i-}(y_{i-2}) = w_{y_i-}(y_{i-1})$ and thus y_0 cannot be on a lower weight path between y_i and y_{i-1}. Furthermore, if the weight of y_{i-1} were not incremented when y_i was processed, then $w_{y_i+}(y_{i-2}) > w_{y_i+}(y_{i-1})$. Since the MCS-M processing time y_i+ is no later than $y_{i-1}-$, we know also from the p_0-WEIGHT induction hypothesis that

$$\begin{aligned} h_{y_i+}(p_0^{i-1}) &\leq \min\{w_{y_i+}(y_{i-2}), w_{y_i+}(y_{i-1})\} \\ &\leq w_{y_i+}(y_{i-1}) \\ &< w_{y_i+}(y_{i-2}) \end{aligned}$$

Therefore, at any time after y_i+ that the weight of y_{i-1} is incremented, the lower weight path (or edge) that was used to increment the weight of y_{i-1} can be extended through p_0^{i-1} as a lower weight path to increment the weight of y_{i-2}. But then the weight of y_{i-2} will always exceed the weight of y_{i-1}, contradicting the α-ORDER induction hypothesis.

Now consider this path p_{alt} of lower weight vertices connecting y_i and y_{i-1} at the time that y_i is processed. By Lemma 3 ($x = y_0$, $u = y_{i-1}$, $v_1 = y_i$ and $v_2, \cdots v_r = p_{alt}$) the vertices on p_{alt} are not adjacent to y_0 in G_α^{k-1}. Combining this with the NO-y_0-ADJ induction hypothesis, we see that the vertices on the p_h^{i-1} connected together through $p_{y_iy_{i-2}} - y_i - p_{alt}$ that are higher numbered than y_0 form the desired path p_h in G_α^{k-1} that is not adjacent to y_0 in G_α^{k-1}.

Case 2 ($w_{y_i-}(y_{i-2}) < w_{y_i-}(y_{i-1})$): For this case we prove that the four properties hold for the next iteration in constructing p_h. We begin with the p_0-WEIGHT property and observe that by definition, $h_{y_i-}(p_{y_iy_{i-2}}) < w_{y_i-}(y_{i-2})$. Also, by the p_0-WEIGHT induction hypothesis, $h_{y_i-}(p_0^{i-1}) \leq w_{y_i-}(y_{i-2})$. Thus, since $w_{y_i-}(y_{i-2}) < w_{y_i-}(y_{i-1})$, we have $h_{y_i-}(p_{y_iy_{i-2}} - y_{i-2} - p_0^{i-1}) < w_{y_i-}(y_{i-1}) \leq$

$w_{y_i-}(y_i)$. And then by Lemma 2 $h_t(p_{y_iy_{i-2}}-y_{i-2}-p_0^{i-1}) \leq \min\{w_t(y_i), w_t(y_{i-1})\}$ at all times $t = y_i-$ and earlier, proving the p_0-WEIGHT property.

Note that since $w_{y_i-}(y_{i-2}) < w_{y_i-}(y_{i-1})$ there exists a vertex y_{i+1} that increases the weight of y_{i-1} beyond the weight of y_{i-2} for the first time. Furthermore, $\alpha(y_i) < \alpha(y_{i+1})$ since at time y_i- the vertex y_{i+1} had already increased the weight of y_{i-1} past the weight of y_{i-2}. This proves the next α-ORDER property.

The next NO-y_0-ADJ property comes from Lemma 3 where $x = y_0$, $u = y_{i-2}$, $v_1 = y_i$ and $v_2, \cdots v_r = p_{y_iy_{i-2}}$.

By the definition and existence of y_{i+1} we know that $w_t(y_{i-1}) = w_t(y_{i-2})$ at times $t = y_{i+1}-$ and earlier. This, together with the SAME-WEIGHT induction hypothesis, gives us the SAME-WEIGHT property for the next iteration.

We have proven by induction that at each step in the iterative process either the path p_h is completed or there exists an extension to one of the subpaths of p_h that is being constructed. Since there are a finite number of vertices in the graph G, the iteration process must eventually not be able to extend a subpath of p_h and hence, the path p_h must be completed. It follows then, that the vertex y_0 is an OCF-vertex in G_α^{k-1}. ∎

Theorem 4. *MCS-M computes a minimal triangulation.*

Proof. Follows from Lemma 4 and Theorem 2. ∎

5 Conclusion

We have described a new algorithm MCS-M that computes a minimal elimination ordering and a minimal triangulation of a graph. MCS-M can be viewed as a simplification of the Lex-M algorithm for computing a minimal triangulation. In fact, in [9] a clever implementation of Lex-M is described that uses label numbers, rather than lists of vertices as labels. The storage used and comparisons made in that implementation are similar to those required with the use of weights in MCS-M. However, in order for the label numbers to properly implement the relative lexicographic labels in Lex-M, their implementation must sort and normalize all unprocessed label numbers after each vertex is processed. This effectively adds a (lower-order) term to their time complexity, requiring $O(nm + n^2) = O(nm)$ time. Our MCS-M implementation does not require this extra sorting step, thereby avoiding the extra term in the time complexity.

As can be seen in the example of Figure 5, Lex-M and MCS-M are not equivalent; the MCS-M ordering shown in Figure 5(a) cannot be produced by Lex-M, and the Lex-M ordering shown in Figure 5(b) cannot be produced by MCS-M.

The impetus for generating minimal triangulations is the desire to approximate minimum triangulations, since in general finding minimum triangulations is NP-hard. It is well know that minimal triangulations can have substantially more fill edges than minimum triangulations. In many cases, MCS-M can produce triangulations with less than half the fill of other minimal triangulations.

But like Lex-M, MCS-M cannot always do so well. For example, for the graph representing an $n \times n$ square grid, MCS-M produces exactly the same fill as Lex-M does, and as pointed out in [9], the minimum fill for such graphs is $O(n^2 \log n)$ whereas the fill produced by Lex-M (and hence MCS-M) is $O(n^3)$.

References

1. A. Berry, *A wide-range efficient algorithm for minimal triangulation*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, 1999.
2. J. R. S. Blair, P. Heggernes, and J. A. Telle, *A practical algorithm for making filled graphs minimal*, Theoretical Computer Science, 250 (2001), pp. 125–141.
3. E. Dahlhaus, *Minimal elimination ordering inside a given chordal graph*, in Graph Theoretical Concepts in Computer Science - WG '97, R. H. Möhring, ed., Springer Verlag, 1997, pp. 132–143. Lecture Notes in Computer Science 1335.
4. D. R. Fulkerson and O. A. Gross, *Incidence matrices and interval graphs*, Pacific J. Math., 15 (1965), pp. 835–855.
5. T. Ohtsuki, *A fast algorithm for finding an optimal ordering in the vertex elimination on a graph*, SIAM J. Comput., 5 (1976), pp. 133–145.
6. T. Ohtsuki, L. K. Cheung, and T. Fujisawa, *Minimal triangulation of a graph and optimal pivoting ordering in a sparse matrix*, J. Math. Anal. Appl., 54 (1976), pp. 622–633.
7. S. Parter, *The use of linear graphs in Gauss elimination*, SIAM Review, 3 (1961), pp. 119–130.
8. B. Peyton, *Minimal orderings revisited*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 271–294.
9. D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic aspects of vertex elimination on graphs*, SIAM J. Comput., 5 (1976), pp. 266–283.
10. R. E. Tarjan and M. Yannakakis, *Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs*, SIAM J. Comput., 13 (1984), pp. 566–579.
11. M. Yannakakis, *Computing the minimum fill-in is NP-complete*, SIAM J. Alg. Disc. Meth., 2 (1981), pp. 77-79.

DNA Sequencing, Eulerian Graphs, and the Exact Perfect Matching Problem

Jacek Błażewicz[1], Piotr Formanowicz[1], Marta Kasprzak[1], Petra Schuurman[2], and Gerhard J. Woeginger[3]

[1] Poznań University of Technology, Poland
[2] CWI, The Netherlands
[3] University of Twente, The Netherlands

Abstract. We investigate the computational complexity of a combinatorial problem that arises in DNA sequencing by hybridization: The input consists of an integer ℓ together with a set S of words of length k over the four symbols A, C, G, T. The problem is to decide whether there exists a word of length ℓ that contains every word in S at least once as a subword, and does not contain any other subword of length k.
The computational complexity of this problem has been open for some time, and it remains open. What we prove is that this problem is polynomial time equivalent to the exact perfect matching problem in bipartite graphs, which is another infamous combinatorial optimization problem of unknown computational complexity.

Keywords: graph theory, computational complexity, computational biology, DNA computing, DNA sequencing.

1 Introduction

This paper is centered around two algorithmic problems. The first problem is the *Exact Perfect Matching* problem that asks whether a given edge weighted graph possesses a perfect matching with weight exactly equal to a given bound. This problem is of great practical importance, and it has applications in bus-driver scheduling, in biomedical image analysis, and in the Ising model in theoretical physics; see Leclerc [10]. In 1982, Papadimitriou & Yannakakis [14] observed that this problem is NP-complete when the weights are encoded in binary, and they asked about its complexity when the weights are encoded in unary. Barahona & Pulleyblank [2] and Leclerc [11] show that some special cases (like the case of planar graphs) are polynomially solvable for unary encoded weights. Mulmuley, Vazirani & Vazirani [13] show that the Exact Perfect Matching problem with unary encoded weightd lies in the complexity class RP, and consequently has a *randomized* polynomial time solution algorithm. However, determining the exact deterministic complexity of this problem still remains unsettled after twenty years, and by now is recognized as an outstanding open problem. A slightly simpler looking, but equally open variant is the following restriction of the exact perfect matching problem to bipartite graphs:

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 13–24, 2002.

Problem: Exact bipartite matching (Ex-Match)

Input: A bipartite multigraph $(X \cup Y, E)$ where E is a multi-subset of $X \times Y$. Non-negative integer weights $w(e)$ on the edges $e \in E$ that are encoded in unary. An integer α.

Question: Is there a perfect matching of weight exactly α?

This bipartite variant is also discussed by Barahona & Pulleyblank [2]. Karzanov [9] gives a polynomial time algorithm for the special case where the input graph is a complete bipartite graph, and where the edge weights are restricted to 0 and 1.

Now let us turn to the second main problem that will be investigated in this paper. Deoxyribonucleic acid (DNA, for short) exists in the form of a double helix that consists of two twisted strings of *nucleotides*. These nucleotides only differ in their nitrogenous bases adenine (A), cytosine (C), guanine (G), and thymine (T). Their order codes genetic information that is symbolically written as a sequence over the four letters A, C, G, and T. The first stage of discovering genetic information is to analyze and to determine this sequence of bases for a given DNA sequence (see for instance Bains & Smith [1]). The length of the sequence can be determined by so-called gel electrophoresis. One method of sequence structure analysis is based on *hybridization* experiments that compare a DNA chain against a library of all possible single-stranded DNA fragments of length k. The outcome of the hybridization is the *k-spectrum* of the sequence, i.e., the set of all fragments of length k of this sequence.

In the ideal case no errors occur in the hybridization experiment, and the k-spectrum of a DNA sequence of length ℓ consists of exactly $\ell - k + 1$ pairwise distinct words of length k. This ideal case with a complete k-spectrum is well-understood (see Theorem 1), but never occurs in the real world. In the real world, the hybridization experiments suffer from *negative* errors (if the measured spectrum is incomplete) and from *positive* errors (if the measured spectrum contains additional elements that do not show up in the sequence). Especially, a fragment of length k may appear *many* times in the sequence, whereas the experiments only detect that it occurs at least *once*. DNA sequencing under negative and positive errors is discussed by Błażewicz, Formanowicz, Kasprzak, Markiewicz & Węglarz [4]. In a follow-up paper, Błażewicz & Kasprzak [6] investigate the computational complexity of several variants of DNA sequencing under negative and positive errors; all these variants turn out to be unary NP-hard in their search versions. However, the complexity of the following fairly innocent looking variant remained open.

Problem: DNA sequencing with unknown multiplicities (DNA-SEQ)

Input: Integers ℓ and k. A set S of words of length k over the alphabet $\{A, C, G, T\}$.

Question: Is there a word of length ℓ with k-spectrum equal S?

In this paper we will prove that the two problems Ex-Match and DNA-SEQ are polynomially reducible to each other and hence are polynomial time

equivalent. If one of them is polynomially solvable, then the other one is polynomially solvable, too. If one of them is NP-complete, then the other one is NP-complete, too. More generally, if one of these two problems is complete for some complexity class under polynomial time reductions, then so is the other problem.

Organization of this paper. The paper is a long (cyclic!) chain of polynomial time reductions that traverses a number of intermediate problems. In Section 2 we reduce problem DNA-SEQ to the problem of deciding the existence of certain Eulerian multigraphs. Then in Section 3, this Eulerian problem Ex-EulerCycle is reduced to problem Ex-Match. Next, in Section 4 problem Ex-Match is reduced to a highly restricted special case of Ex-Match that we call Ex-Match$^-$. Finally, in Section 5 this highly restricted special case Ex-Match$^-$ will be reduced to problem DNA-SEQ. By doing this, we return to our starting point, and the chain of reductions is closed to a cycle.

Notation and definitions. Throughout the paper, we stick to the standard graph terminology as given in the book of Berge [3]. However, we want to clarify the usage of the following terms: A graph $G = (V, E)$ is *cubic*, if each vertex in V is incident to exactly three edges. For a multigraph $G = (V, E)$, its *underlying simple graph* $G' = (V, E')$ is the (unique) simple graph that results from G by keeping exactly one edge from every bundle of multiple edges. A multigraph G is a *super-multigraph* of a simple graph G', if G' is the underlying simple graph of G. In other words, a super-multigraph of the simple graph G' contains every edge in G' at least once, and does not contain any other edges.

A *path* is an alternating sequence $v_0, e_1, v_1, \ldots, e_n, v_n$ of vertices and edges such that edge e_i always connects vertex v_{i-1} to v_i. A path is called *simple* if its vertices are pairwise distinct (except possibly the pair v_0 and v_n). A *cycle* is a path with $v_0 = v_n$. The *length* of a path (cycle) is the number of edges in this path (cycle). A *Eulerian path* (Eulerian cycle) in a graph $G = (V, E)$ is a path (cycle) in G that uses every edge in E exactly once. It is known that a directed multigraph possesses a Eulerian cycle if and only if all vertices have in-degree equal to their out-degree and if its underlying undirected graph is connected.

2 From DNA Sequencing to Exact Eulerian Cycles

In this section we recapitulate the relationship between DNA sequencing and the existence of Eulerian paths in certain directed graphs. This relationship has been observed by Pevzner [15], and lateron has been discussed in detail by Błażewicz, Hertz, Kobler & de Werra [5]. The underlying combinatorial ideas go (at least) back to De Bruijn [7].

For a word u of length k over the alphabet $\{A, C, G, T\}$, we denote by $\text{prefix}_{k-1}(u)$ the word that consists of the first $k-1$ letters in u, and by $\text{suffix}_{k-1}(u)$ the word that consists of the last $k-1$ letters in u. For a set S of words of length k over the alphabet $\{A, C, G, T\}$, we introduce the following directed graph $G_S = (V_S, A_S)$: The vertex set is defined by

$$V_S \;=\; \{\text{prefix}_{k-1}(u),\ \text{suffix}_{k-1}(u) :\quad u \in S\}.$$

The arc set A_S contains $|S|$ arcs that are defined as follows: For every word $u \in S$, there is a corresponding arc $a(u)$ in A_S that goes from the vertex $\text{prefix}_{k-1}(u)$ to the vertex $\text{suffix}_{k-1}(u)$.

Theorem 1 *(Pevzner [15])*
Let S be a set of words of length k over the alphabet $\{A, C, G, T\}$, and let $G_S = (V_S, A_S)$ be the corresponding directed graph. Then there exists a word w of length $|S| + k - 1$ with k-spectrum equal to S, if and only if G_S possesses a Eulerian path. ■

The statement of this theorem is actually quite easy to see: For $i = 1, \ldots, |S|$ let w_i denote the subword of w of length k that starts with the ith letter and ends with the $(i + k - 1)$th letter. Then $S = \{w_1, \ldots, w_{|S|}\}$ and the arcs $a(w_1), \ldots, a(w_{|S|})$ in this ordering form a Eulerian path for G_S. Vice versa, every Eulerian path in G_S gives a sequence $w_1, \ldots, w_{|S|}$ of words of length k that can be combined into a word of length $|S| + k - 1$. To summarize, Theorem 1 yields a polynomial time algorithm for the special case of problem DNA-SEQ where every word in S occurs *exactly once* as a subword in the word w. By considering the exact number of occurrences of every subword from S, the statement in Theorem 1 can be generalized to our sequencing problem DNA-SEQ.

Theorem 2 *Let S be a set of words of length k over the alphabet $\{A, C, G, T\}$, let $G_S = (V_S, A_S)$ be the corresponding directed graph, and let $\ell \geq |S| + k - 1$ be an integer.*

Then there exists a word w of length ℓ with k-spectrum equal to S, if and only if there exists a super-multigraph of G_S that contains exactly $\ell - k + 1$ arcs and that possesses a Eulerian path. ■

As an immediate consequence of Theorem 2, problem DNA-SEQ is polynomial time reducible to the problem Ex-EulerPath defined below. Indeed, by trying all possibilities for a start vertex s and an end vertex t in the graph G_S and by setting $\beta = \ell - k + 1$, one may solve an instance of DNA-SEQ by solving $O(|V_S|^2) = O(|S|^2)$ instances of Ex-EulerPath.

Problem: Exact Eulerian path (Ex-EulerPath)

Input: A simple directed graph $G = (V, A)$. Two vertices $s, t \in V$. An integer β.

Question: Does there exist a super-multigraph of G that contains exactly β arcs and that possesses a Eulerian path that starts in s and ends in t?

In the special case of problem Ex-EulerPath where $s = t$ holds, we are looking for an exact Eulerian cycle in some super-multigraph of G (Ex-EulerCycle). The general problem Ex-EulerPath is polynomial time reducible to Ex-EulerCycle: Take an instance G of Ex-EulerPath, and create a new directed path with β arcs that goes from t to s. Consider this new graph G^+ together with the value $\gamma = 2\beta$ as an instance of Ex-EulerCycle.

It can be easily verified that the original graph G has a super-multigraph with a Eulerian path of length β going from s to t, if and only if the new graph G^+ has a super-multigraph with a Eulerian cycle of length $\gamma = 2\beta$.

Problem: Exact Eulerian cycle (Ex-EulerCycle)

Input: A simple directed graph $G = (V, A)$. An integer γ.

Question: Does there exist a super-multigraph of G that contains exactly γ arcs and that possesses a Eulerian cycle?

To summarize, in this section we have shown that DNA-SEQ is polynomial time reducible to Ex-EulerPath, and that Ex-EulerPath in turn is polynomial time reducible to Ex-EulerCycle.

3 From Exact Eulerian Cycles to Exact Bipartite Matchings

In this section we will show that problem Ex-EulerCycle is polynomial time reducible to the exact bipartite matching problem Ex-Match. Hence, consider an arbitrary instance $(G; \gamma)$ of problem Ex-EulerCycle. We denote by $\deg^+(v)$ and $\deg^-(v)$ the in-degree and the out-degree of the vertex $v \in V$. A vertex v is called *positive* if $\deg^+(v) > \deg^-(v)$, *neutral* if $\deg^+(v) = \deg^-(v)$, and *negative* if $\deg^+(v) < \deg^-(v)$. Furthermore, we denote by Δ the sum of $\deg^+(v) - \deg^-(v)$ taken over all positive vertices v. Clearly, this value Δ also equals the sum of $\deg^-(v) - \deg^+(v)$ taken over all negative vertices v. Note furthermore that $|\Delta| \leq |V|^2$.

For vertices $x, y \in V$ and for integers k with $1 \leq k \leq |V|$, we introduce the Boolean predicate $P[x, y; k]$ that is true if and only if in the graph G there exists a (not necessarily simple) directed path of length k that goes from x to y. Moreover, for integers k with $1 \leq k \leq |V|$ we introduce the Boolean predicate $C[k]$ that is true if and only if in the graph G there exists a (not necessarily simple) directed cycle of length k. The truth values of all these predicates can be determined in polynomial time by standard dynamic programming approaches.

Now assume that the instance $(G; \gamma)$ of Ex-EulerCycle has answer YES, and let $G^* = (V, A^*)$ be a super-multigraph of G with γ arcs that certifies this answer. That means that the graph G^* is strongly connected (or equivalently: that the graph G is strongly connected), and that in G^* every vertex $v \in V$ has its in-degree equal to its out-degree.

Lemma 3 *The arc set A^* of the graph G^* can be partitioned as follows:*

(A1) The set A of arcs in the underlying simple graph G.

(A2) A set of Δ (not necessarily simple) directed paths of length less or equal to $|V|$. Each such path starts in a positive vertex and ends in a negative vertex. In every positive vertex v, there start exactly $\deg^+(v) - \deg^-(v)$ of these paths. In every positive vertex v, there end exactly $\deg^-(v) - \deg^+(v)$ of these paths.

(A3) A set of (not necessarily simple) directed cycles of length less or equal to $|V|$.

We denote by $m(A1)$, $m(A2)$, $m(A3)$ the number of arcs of type (A1), (A2), (A3) in such a partition. Now if we want to solve Ex-EulerCycle, we must determine whether there exist arc sets of type (A1), (A2), (A3) with $m(A1) + m(A2) + m(A3) = \gamma$. Trivially, $m(A1) = |A|$. Moreover, since $|\Delta| \leq |V|^2$ we get $m(A2) \leq |V|^3$. But what are the exact candidate values that $m(A2)$ and $m(A3)$ can take?

The candidate values for $m(A2)$ can be determined via the exact bipartite matching problem (see the introductory section for an exact definition of this problem). This is done as follows. For every positive vertex x in G, we introduce $\deg^+(x) - \deg^-(x)$ independent copies in the set X of the bipartition. For every negative vertex y in G, we introduce $\deg^-(y) - \deg^+(y)$ independent copies in the set Y of the bipartition. Whenever the predicate $P[x, y; k]$ is true for some positive vertex x, some negative vertex y, and some integer k with $1 \leq k \leq |V|$, we introduce an edge of weight k in E from every copy of x to every copy of y. Since the edge weights are polynomially bounded by $|V|$, we may as well encode them in unary. It is easily verified that α_2 is a possible value for $m(A2)$ if and only if the resulting bipartite graph has a perfect matching of weight exactly α_2. Hence, by solving a polynomial number of exact bipartite matching instances we can determine for every α_2 with $1 \leq \alpha_2 \leq |V|^3$ whether it is a candidate value for $m(A2)$.

The candidate values for $m(A3)$ can be expressed in terms of the predicates $C[k]$ with $1 \leq k \leq |V|$. We define $C = \{k : \ C[k], \ 1 \leq k \leq |V|\}$ as the set of possible cycle lengths. In determining whether an integer α_3 is a candidate for $m(A3)$ we distinguish the two cases $\alpha_3 \leq |V|^2$ and $\alpha_3 > |V|^2$. The first case $\alpha_3 \leq |V|^2$ boils down to an unbounded subset sum problem in which the item sizes in C and the goal value α_3 all are polynomially bounded in $|V|$. It is well-known that such a polynomially bounded special case of the subset sum problem is polynomially solvable by dynamic programming (see for instance the book by Martello & Toth [12]). For the second case $\alpha_3 > |V|^2$ we apply a famous result on the Frobenius problem.

Theorem 4 *(Erdős & Graham [8])*
Let $0 < z_1 < z_2 < \cdots < z_n \leq Z$ *be integers with* $\gcd(z_1, z_2, \ldots, z_n) = 1$. *Then every integer above* $2Z^2/n$ *can be expressed in the form* $\sum_{i=1}^{n} x_i z_i$ *with non-negative integers* $x_1, \ldots, x_n$. ■

In our case, all the values in the set C are bounded by $|V|$; therefore we have $Z = |V|$ and $n = |C|$. If $|C| \geq 2$ and the greatest common divisor of the numbers in C is one, then every $\alpha_3 > |V|^2$ is a candidate for $m(A_3)$. If $|C| \geq 2$ and the greatest common divisor of the numbers in C equals d, then $\alpha_3 > |V|^2$ is a candidate for $m(A_3)$ if and only if it is divisible by d. The case $|C| = 0$ is straightforward. In the case $|C| = 1$, a number $\alpha_3 > |V|^2$ is a candidate value for $m(A_3)$ if and only if it is divisible by the unique element d in C.

Now let us put everything together. In order to solve an instance of Ex-EulerCycle, we first determine the candidate values for $m(A2)$ (by solving a polynomial number of exact bipartite matching instances) and for $m(A3)$ (by solving a polynomially bounded subset sum instance). The instance has a solu-

tion if and only if G is strongly connected and $\gamma = m(A1) + m(A2) + m(A3)$ has a solution over the candidate values for $m(A1)$, $m(A2)$, $m(A3)$. We know that $m(A1) = |A|$ and that all candidate values for $m(A2)$ are between 1 and $|V|^3$. Therefore, we can simply search through all $O(|V|^3)$ possibilities for $m(A2)$ and check whether $\gamma - |A| - m(A2)$ is a feasible candidate for $m(A3)$. Summarizing, this yields that problem Ex-EulerCycle indeed is polynomial time reducible to problem Ex-Match.

4 From Exact Matching to Restricted Exact Matching

In this section we will prove that the exact bipartite matching problem Ex-Match is polynomial time reducible to the following highly restricted special case of it.

Problem: Restricted exact bipartite matching (Ex-Match$^-$)

Input: A bipartite graph $B = (X \cup Y, E)$ with $E \subseteq X \times Y$ that is connected and cubic. For every edge $e \in E$ a weight $w(e) \in \{0, 1, \alpha + 1\}$ where α is a given integer.

Question: Does this bipartite graph have a perfect matching of weight exactly α?

The reduction to Ex-Match$^-$ is done in four steps: The first step is a simple preprocessing step that makes the graph connected, and that also gets rid of vertices of degree one. In the second step, we get rid of the vertices of degree four and more; simultaneously, the graph becomes simple. In the third step, we bring the edge weights down to 0-1. Finally in the fourth step, we make the graph cubic. All details of these four steps are omitted in this extended abstract.

Corollary 5 *Problem* Ex-Match *is polynomial time equivalent to its special case where the bipartite input graph is simple and connected, and has only vertices of degree two and three, and has only edge weights zero and one.*

5 From Restricted Exact Matching Back to DNA Sequencing

In this section we will prove that the restricted exact bipartite matching problem Ex-Match$^-$ is polynomial time reducible to problem DNA-SEQ. This will be done by moving through an auxiliary instance of problem Ex-EulerCycle (see Section 2 for the definition of this problem).

Indeed, consider a connected, cubic, bipartite graph $B = (X \cup Y, E)$ as instance for Ex-Match$^-$. If $|X| \neq |Y|$ holds, then the graph B does not possess any perfect matching, and the answer to Ex-Match$^-$ is trivially NO. Therefore, we will assume that $|X| = |Y| = q$ holds. Without loss of generality, we assume furthermore that q is sufficiently large to satisfy $2^q \geq 100q^3$. Note that any perfect matching in B has weight at most q (in case it does not use edges of weight

$\alpha + 1$) or weight at least $\alpha + 1$ (in any other case). Therefore, we will assume from now on that $\alpha \leq q$. We denote by m_0, m_1, and $m_{\alpha+1}$ the number of edges of weight 0, 1, and $\alpha + 1$, respectively. Note that $m_0 + m_1 + m_{\alpha+1} = |E| = 3q$. Finally, let f be an arbitrary bijection between the two sets X and Y, and let $Q = 2q + 8$.

From the bipartite graph B we will now construct a directed graph $G = (V, A)$ as an instance of Ex-EulerCycle. This directed graph G consists of $2q$ *primary* vertices and of many *secondary* vertices. The primary vertices are grouped into the two sets $X' = \{x' | x \in X\}$ and $Y' = \{y' | y \in Y\}$. The arcs in G are defined as follows.

(P1) For every edge $e = [x, y]$ in B with $x \in X$ and $y \in Y$, there is a corresponding directed path $P(e)$ in G that goes from x' to y'. If the edge e has weight $w(e)$, then this path has length $(w(e) + 1)Q$.
(P2) For every $y \in Y$ and $x = f(y)$ in B, there are four directed paths of length $8(q^2 + q)Q$ in G that all connect y' to x'.

All these introduced directed paths are internally pairwise vertex-disjoint. Their internal vertices (that all have in-degree and out-degree one) form the secondary vertices of the graph G. Every vertex in X' has in-degree four and out-degree three. Every vertex in Y' has in-degree three and out-degree four. Since the bipartite graph B is connected, also the constructed graph G is connected. A crude estimation shows that G has less than $50q^3Q$ vertices.

Lemma 6 *The instance B of* Ex-Match$^-$ *has a perfect matching of weight exactly α, if and only if the constructed instance G of* Ex-EulerCycle *has a super-multigraph with a Eulerian cycle of length exactly*

$$\gamma \;=\; (m_0\, Q + m_1\, 2Q + m_{\alpha+1}\, (\alpha + 2)Q) \;+\; 32q\, (q^2 + q)Q \;+\; (\alpha + q)Q.$$

Proof. (Only if). Assume that B has a perfect matching M of weight α. We construct a super-multigraph G^* of G, in which all vertices in $X' \cup Y'$ have in-degree and out-degree four. We take all arcs from G into G^*. Moreover, for each of the q edges $e \in M$, we take one additional copy of the directed path $P(e)$ into G^*. This completes the description of G^*. It is easily verified that in G^* every vertex has in-degree equal to out-degree. Since the graph G^* is also connected, it has Eulerian cycle.

Now let us determine the number of arcs in G^*. First, G^* contains all paths of type (P1). There are m_0 of those paths that have length Q, m_1 of length $2Q$, and $m_{\alpha+1}$ of length $(\alpha + 2)Q$. Secondly, G^* contains all paths of type (P2). There are $4q$ of them, and each has length $8(q^2 + q)Q$. Thirdly, there are the arcs in the additional q paths $P(e)$ with $e \in M$. These paths contribute

$$\sum_{e \in M} (w(e) + 1)Q \;=\; Q \sum_{e \in M} w(e) \;+\; Q\, |M| \;=\; (\alpha + q)Q.$$

arcs. Altogether, this yields exactly γ arcs in G^*.

(If). Assume that G has a super-multigraph G^* with a Eulerian cycle of length γ. It can be seen that this multigraph G^* must consist of one or more copies of every path introduced in (P1) and (P2). We claim that G^* cannot contain two copies of any path of type (P2). Otherwise, the total number of arcs in the copies of paths of type (P2) is at least

$$(4q+1)\cdot 8(q^2+q)Q \;=\; 6(q^2+q)Q + 32q\,(q^2+q)Q + 2(q^2+q)Q \;>\; \gamma.$$

Since G^* has only γ arcs, we have arrived at the desired contradiction. We conclude that every path of type (P2) shows up exactly once in G^*, and hence these paths altogether contribute $32q\,(q^2+q)Q$ arcs. As a consequence, in G^* every vertex in X' has in-degree four, and every vertex in Y' has out-degree four.

Since G^* has a Eulerian cycle, every vertex in X' must have out-degree four, and every vertex in Y' must have in-degree four. These degrees can only result from copies of paths of type (P1). Each path $P(e)$ with $e \in E$ must show up at least once in G^*. This yields $(m_0\,Q + m_1\,2Q + m_{\alpha+1}\,(\alpha+2)Q)$ arcs, out-degrees three for vertices in X', and in-degrees three for vertices in Y'. The remaining $(\alpha+q)Q$ arcs in G^* must come from a system of q paths of type (P1) that connect every vertex in X' to exactly one vertex in Y'. In the bipartite graph B, the edge set M that contains those edges e for which $P(e)$ is in this system of paths forms a perfect matching of weight α. ■

Our next goal is to establish that for an appropriate value k, there exists a k-spectrum S of words over the alphabet $\{A, C, G, T\}$ such that the above constructed directed graph $G = (V, A)$ equals $G_S = (V_S, A_S)$; see Section 2 for definitions. The technical main difficulty is to ensure that distinct vertices in V are associated with distinct DNA words of length $k-1$.

We partition the vertices in V into a set C of so-called *crucial* and a set $V - C$ of *non-crucial* vertices. Every vertex in the set $X' \cup Y'$ is crucial. Moreover, every vertex on the paths of type (P1) and (P2) whose distance to $X' \cup Y'$ is divisible by Q is a crucial vertex. Note that the length of every path of type (P1) and (P2) is a multiple of Q, and that the crucial vertices divide these paths into connected chunks of Q arcs and $Q-1$ non-crucial vertices. If there is a directed path of length Q from a crucial vertex u to another crucial vertex v in G, then we say that u is a *predecessor* of v, and that v is a *successor* of u. It can be seen that every vertex in X' has four predecessors and three successors, that every vertex in Y' has three predecessors and four successor, and that all remaining crucial vertices have one predecessor and one successor. Moreover, it can be checked that $|C| \leq 50q^3$ holds.

For every crucial vertex $v \in C$, we fix two labels $\textsc{LLabel}(v)$ and $\textsc{RLabel}(v)$. These labels are $2|C|$ pairwise distinct words of length q over the alphabet $\{A, G\}$. We need $2|C| \leq 100q^3$ distinct labels that we can choose from 2^q words. Since we assumed $100q^3 \leq 2^q$, such pairwise distinct labels indeed exist and can be found easily. With the help of these labels, we will now define for every vertex $z \in V$ a corresponding word $\textsc{Word}(z)$ of length $Q = 2q + 8$.

- If z is a crucial vertex, then $\text{WORD}(z)$ starts with one of the letters A, C, G, T, followed by the label $\text{LLABEL}(z)$, followed by a string of six T's, then followed by the label $\text{RLABEL}(z)$, and ending with one of the letters A, C, G, T. For crucial vertices, these words fulfill the following conditions: (W1) If v_1 and v_2 are successors of the same vertex u, then $\text{WORD}(v_1)$ and $\text{WORD}(v_2)$ end with different letters. (W2) If u_1 and u_2 are predecessors of the same vertex v, then $\text{WORD}(u_1)$ and $\text{WORD}(u_2)$ start with different letters. Since every crucial vertex has at most four successors and at most four predecessors, these conditions can indeed be met with the alphabet $\{A, C, G, T\}$ of size four.
- If z is a non-crucial vertex, then it lies on a uniquely defined directed path of length Q that connects some crucial vertex u to another crucial vertex v. Let $u = z_0, z_1, \ldots, z_Q = v$ denote the sequence of vertices along such a path. Then $\text{WORD}(z_i)$ consists of the last $Q - i$ letters of $\text{WORD}(u)$ followed by the first i letters of $\text{WORD}(v)$.

Lemma 7 *For $u, v \in V$ with $u \neq v$, we always have $\text{WORD}(u) \neq \text{WORD}(v)$.*

Proof. Consider a word $\text{WORD}(z)$ of length $2q + 8$ over $\{A, C, G, T\}$. We show that from $\text{WORD}(z)$, we can uniquely localize the corresponding vertex z in G.

By our construction, $\text{WORD}(z)$ either contains a subword of six consecutive T's, or it starts with i consecutive T's and ends with $6 - i$ consecutive T's for some $1 \leq i \leq 5$. In either case, the q letters preceding or succeeding these T's in $\text{WORD}(z)$ will form one of the labels $\text{LLABEL}(v)$ or $\text{RLABEL}(v)$ for some crucial vertex v. If vertex v is not contained in $X' \cup Y'$, then the relative position of this label in $\text{WORD}(z)$ uniquely determines the vertex z. If v is contained in $X' \cup Y'$, then either (i) $\text{WORD}(z)$ contains the first letter of the word of a successor of v, or (ii) it contains the last letter of the word of a predecessor of v, or (iii) it contains none of these letters. In the cases (i) and (ii), by conditions (W1) and (W2) these letters uniquely identify the corresponding successor or predecessor. Then we can again use the relative position of the label $\text{LLABEL}(v)$ or $\text{RLABEL}(v)$ in $\text{WORD}(z)$ to uniquely determine the vertex z. Finally, in case (iii) the vertex z must coincide with v. ■

Now we are almost done: We choose $k = Q + 1 = 2q + 9$ as the word length for the k-spectrum S. Every vertex $v \in V$ is labeled by the word $\text{WORD}(v)$ of length $k - 1$, and thus becomes a member of V_S. By Lemma 7, different vertices are labeled by different words. For every arc $a = (u, v) \in A$, the last $k - 2$ letters in $\text{WORD}(u)$ agree with the first $k - 2$ letters in $\text{WORD}(v)$. Hence, this arc a can be correctly encoded by the word $\text{WORD}(a)$ of length k that starts with the first letter of $\text{WORD}(u)$, followed by these $k - 2$ agreeing letters, and ending with the last letter of $\text{WORD}(v)$. We define $S = \{\text{WORD}(a) | a \in A\}$ as our k-spectrum, and we thus derive $G_S = G$. By the above discussion and by Lemma 6, the graph G_S has a super-multigraph with a Eulerian cycle of length γ, if and only if the bipartite graph B has a perfect matching of weight α.

In the final step, we move from problem Ex-EulerCycle to problem Ex-EulerPath: Let $a = (u, v)$ be an arc on one of the paths of type (P2) such that u and v both are non-crucial vertices. We remove this arc a from the graph G, and we simultaneously define $S_{final} = S - \{\text{Word}(a)\}$. This leads to a vertex u of out-degree 0 and to a vertex v of in-degree 0. It can be verified that the resulting graph has a super-multigraph with a Eulerian path of length $\gamma - 1$ (that of course starts in vertex v and ends in vertex u), if and only if the bipartite graph B has a perfect matching of weight α. Finally, we get from Theorem 2 that there exists a word of length $\gamma - 1$ with k-spectrum S_{final}, if and only if the bipartite graph B has a perfect matching of weight α. This means that problem Ex-Match$^-$ is polynomially reducible to problem DNA-SEQ.

By combining the reductions from Sections 2 through 5, we arrive at the main result of this paper.

Theorem 8 *The two problems* Ex-Match *and* DNA-SEQ *are polynomial time reducible to each other.* ■

Acknowledgement

Jacek Błażewicz, Piotr Formanowicz, and Marta Kasprzak acknowledge support by KBN grant 7T11F02621. Gerhard J. Woeginger acknowledges support by the START program Y43-MAT of the Austrian Ministry of Science.

References

1. W. Bains and G.C. Smith [1988]. A novel method for nucleic acid sequence determination. *Journal of Theoretical Biology 135*, 303–307.
2. F. Barahona and W.R. Pulleyblank [1987]. Exact arborescences, matchings, and cycles. *Discrete Applied Mathematics 16*, 91–99.
3. C. Berge [1973]. *Graphs and Hypergraphs*. North Holland.
4. J. Błażewicz, P. Formanowicz, M. Kasprzak, W.T. Markiewicz, and J. Węglarz [1999]. DNA sequencing with positive and negative errors. *Journal of Computational Biology 6*, 113–123.
5. J. Błażewicz, A. Hertz, D. Kobler, and D. de Werra [1999]. On some properties of DNA graphs. *Discrete Applied Mathematics 98*, 1–19.
6. J. Błażewicz and M. Kasprzak [2001]. Complexity of DNA sequencing by hybridization. To appear in *Theoretical Computer Science*.
7. N.G. de Bruijn [1946]. A combinatorial problem. *Koninklijke Nederlandse Akademie van Wetenschappen te Amsterdam. Proceedings 49*, 758–764.
8. P. Erdős and R.L. Graham [1972]. On a linear diophantine problem of Frobenius. *Acta Arithmetica 21*, 399–408.
9. A.V. Karzanov [1987]. Maximum matching of given weight in complete and complete bipartite graphs. *Cybernetics 23*, 8–13; translation from *Kibernetika 1*, 1987, 7–11.
10. M. Leclerc [1986]. Polynomial time algorithms for exact matching problems. Master's thesis, University of Waterloo, Waterloo.

11. M. Leclerc [1988/89]. Optimizing over a slice of the bipartite matching polytope. *Discrete Mathematics 73*, 159–162.
12. S. Martello and P. Toth [1990]. Knapsack problems: Algorithms and computer implementations. John Wiley & Sons.
13. K. Mulmuley, U. Vazirani, and V.V. Vazirani [1987]. Matching is as easy as matrix inversion. *Combinatorica 7*, 105–113.
14. C.H. Papadimitriou and M. Yannakakis [1982]. The complexity of restricted spanning tree problems. *Journal of the ACM 29*, 285–309.
15. P.A. Pevzner [1989]. ℓ-tuple DNA sequencing: Computer analysis. *Journal of Biomolecular Structure and Dynamics 7*, 63–73.

On the Minimum Size of a Contraction-Universal Tree

Olivier Bodini

LIP, École Normale Supérieure de Lyon,
46 Allée d'Italie, 69364 Lyon Cedex 05, France

Abstract. A tree T_{uni} is *m-universal* for the class of trees if for every tree T of size m, T can be obtained from T_{uni} by successive contractions of edges. We prove that a m-universal tree for the class of trees has at least $m\ln(m)+(\gamma-1)m+O(1)$ edges where γ is the Euler's constant and we build such a tree with less than m^c edges for a fixed constant $c = 1.984\ldots$

1 Introduction

What is the minimum size of an object in which every object of size m embeds? Issued from the category theory, questions of this kind appeared in graph theory. For instance, R. Rado [1] proved the existence of an "initial countable graph". Recently, Z. Füredi and P. Komjàth [2] studied a connected question.

We use here the following definition: given a sub-class C of graphs (trees, planar graphs, etc.), a graph G_{uni} is *m-universal* for C if for every graph G of size m in C, G is a minor of G_{uni}, i.e. it can be obtained from G_{uni} by successive contractions or deletions of edges.

Inspired by the Robertson and Seymour work [3] on graph minors, P. Duchet asked whether a polynomial bound in m could be found for the size of a m-universal tree for the class of trees. We give here a positive sub-quadratic answer.

From an applied point of view, such an object would possibly allows us to define a tree from the representation of its contraction.

The main results of this paper are the following theorems which give bounds for the minimum size of a m-universal tree for the class of trees:

Theorem 1. *A m-universal tree for the class of trees has at least $m\ln(m) + (\gamma-1)m + O(1)$ edges where γ is the Euler's constant.*

Theorem 2. *There exists a m-universal tree T_{uni} for the class of trees with less than m^c edges for a fixed constant $c = 1.984...$*

Our proof follows a recursive construction where large trees are obtained by some amalgamation process involving simpler trees. With this method, the constant c could be reduced to 1.88... but it seems difficult to improve this value.

We conclude the paper with related open questions.

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 25–34, 2002.

2 Terminology

Our graphs are undirected and simple (with neither loops nor multiple edges). We denote by $G(V, E)$ a graph (its vertex set is $V(G)$ and its edge set is $E(G)$ (a subset of the family of all the $V(G)$-subsets of cardinality 2)). Referring to C. Thomassen [4], we recall some basic definitions that are useful for our purpose:

We denote by P_n the path of size n.

If x is a vertex then $d(x)$, the *degree* of x, is the number of edges incident to x.

Let e be an edge of $E(G)$, the graph denoted by $G - e$ is the graph on the vertex set of G, whose edge set is the edge set of G without e. We call classically this operation *deletion*.

Let $e = \{a, b\}$ be an edge of $G(V, E)$, we name *contraction of G along e*, the graph denoted by $G/e = H(V', E')$, with $V' = (V/\{a, b\}) \cup \{c\}$ where c is a new vertex and E' the edge set which contains all the edges of the sub-graph G_1 on V/e and all the edges of the form $\{c, x\}$ for $\{a, x\}$ or $\{b, x\}$ belonging to E.

We say that H is a *minor* of G if and only if we can obtain it from G by successively deleting and /or contracting edges, in an other way, we can define the set $M(G)$ of minors of G by the recursive formula:

$$M(G) = G \cup \left(\bigcup_{e \in E(G)} M(G/e) \right) \cup \left(\bigcup_{e \in E(G)} M(G - e) \right)$$

The notion of minor induces a partial order on graphs. We write $A \preceq B$ to mean "A is a minor of B".

For technical reasons, we prefer to use the size of a tree (edge number) rather than its order (vertex number).

Finally, let us recall that, a graph G_{uni} is *m-universal for a sub-class C* of graphs if for every element G of C with m edges, G is a minor of G_{uni}.

3 A Lower Bound

In this section, we prove that a m-universal tree T_{uni} for the trees has asymptotically at least $m \ln(m)$ edges. We use the fact that T_{uni} has to contain all spiders of size m as minors. A *spider S on a vertex w* is a tree such that $\forall v \in V(S) \setminus \{w\}, d(v) \leq 2$. We denote the spider constituted by paths of lengths $1 \leq m_1 \leq ... \leq m_k$ by $Sp(m_1, ..., m_k)$ (Fig. 1).

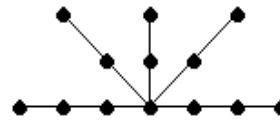


Fig. 1. $Sp(2, 2, 2, 3, 3)$

Definition 1. *Let T be a tree, we denote by ∂T the subtree of T with $V(\partial T) = V(T) \backslash A$, where A is the set of the leaves of T. Also, we denote by ∂^k the k-th iteration of ∂.*

Lemma 1. *$Sp(m_1, ..., m_k) \preceq T$ involves that $\partial Sp(m_1, ..., m_k) \preceq \partial T$. Moreover, if for all i, $m_i = 1$ then $\partial Sp(m_1, ..., m_k)$ is a vertex. Otherwise, put a the first value such that $m_a > 1$, we have $\partial Sp(m_1, ..., m_k) = Sp(m_a - 1, ..., m_k - 1)$ excepted for $k = 1$, in this last case we have $\partial Sp(m_1) = Sp(m_1 - 2)$.*

Proof. This just follows from an observation. □

Lemma 2. *For every tree T, $Sp(m_1, ..., m_k) \preceq T \Rightarrow T$ has at least k leaves.*

Proof. Trivial. □

Theorem 3. *A m-universal tree T_{uni} for the class of trees has at least $\sum_{i=1, i\neq 2}^{m} \left\lfloor \frac{m}{i} \right\rfloor$ edges.*

Proof. A m-universal tree T_{uni} for the class of trees has to contain as minors all spiders of size m. So, for all p it contains as minors the spiders $Sp(p, ..., p)$ where we have $\left\lfloor \frac{m}{p} \right\rfloor$ times the letter p. By the lemma 1, for all $p \leq \frac{m}{2}$, $Sp(1, ..., 1) \preceq \partial^{p-1} T_{uni}$ and if m is odd, $Sp(1) \preceq \partial^{\left\lfloor \frac{m}{2} \right\rfloor - 1} T_{uni}$. Moreover, it is clear that the terminal edges of the $\partial^p T_{uni}$ constitute a partition of T_{uni}. By the lemma 2, this involves that T_{uni} has at least $\sum_{i=1}^{\left\lfloor \frac{m}{2} \right\rfloor} \left\lfloor \frac{m}{i} \right\rfloor$ edges if m is even and $1 + \sum_{i=1}^{\left\lfloor \frac{m}{2} \right\rfloor} \left\lfloor \frac{m}{i} \right\rfloor$ edges if m is odd. An easy calculation proves that these values are always equal to $\sum_{i=1, i\neq 2}^{m} \left\lfloor \frac{m}{i} \right\rfloor$. □

Proof. (of the theorem 1) it follows from the usual estimate $\sum_{i=1}^{n} \frac{1}{i} \sim \ln(n) + \gamma + O\left(\frac{1}{n}\right)$ and the inequality $\sum_{i=1, i\neq 2}^{m} \left\lfloor \frac{m}{i} \right\rfloor \geq 1 + \sum_{i=1, i\neq 2}^{m-1} \left(\frac{m}{i} - 1\right)$. □

What the above proof shows, in fact, is the following:

Corollary 1. *A minimum m-universal spider for the class of spiders has $\sum_{i=1, i\neq 2}^{m} \left\lfloor \frac{m}{i} \right\rfloor$ edges.*

Proof. The spider $Sp\left(\left\lfloor \frac{m}{m} \right\rfloor, \left\lfloor \frac{m}{m-1} \right\rfloor, ..., \left\lfloor \frac{m}{2} \right\rfloor, \left\lceil \frac{m}{2} \right\rceil\right)$ is clearly a m-universal spider of size $\sum_{i=1, i\neq 2}^{m} \left\lfloor \frac{m}{i} \right\rfloor$ for the class of spiders, and by theorem 3 it is a minimum value. □

4 The Main Stem

In the sequel, we deal with *rooted graph*, i.e. graph G where we can distinguish a special vertex denoted by $r(G)$, called the *root*. Conventionally, any contracted graph G' of same rooted graph G will be rooted at the unique vertex which is the image of the root under the contraction mapping, we say in this case that the rooted graph G' is a *rooted contraction* of G. Note that, the contraction operator suffices to obtain all minor trees of a tree. So, we can now define the following new notion for sub-classes of rooted trees: a rooted tree T_{uni} is *strongly m-universal for a sub-classes C of rooted trees* if for every rooted tree T *in* C of size m, T is a rooted contraction of T_{uni}. The concept of root is introduced to avoid problems with graph isomorphisms that, otherwise would greatly impede our inductive proof.

For every edge e of a tree T, the forest $T\backslash e$ has two connected components. We call *e-branch*, denoted by B_e, the connected component of T' which does not contain $r(T)$, we define the root of B_e as $e \cap V(B_e)$.

A *main stem* of a rooted tree of size m is defined as a path P which is issued from the root and such that for all e-branches B_e with $e \notin E(C)$, we have $|E(B_e)| < \left\lfloor \frac{m}{2} \right\rfloor$ (Fig. 2).

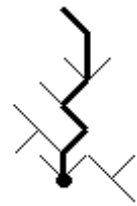


Fig. 2. A main stem in bold

The following lemma suggests the procedure which will be used to find a sub-quadratic upper bound for universal trees. Roughly speaking, it endows every tree with some recursive structure constructed with the help of main stems.

Lemma 3. *Every rooted tree has a main stem.*

Proof. By induction on the size of the rooted tree. Let T be a rooted tree, if T has one or two edges, it is trivial. Otherwise let us consider the sub-graph $T\backslash r(T)$, which is a forest. We choose a connected component T_1 with maximum size and we denote by b_1 the unique vertex of T_1 which is adjacent to $r(T)$. Tree T_1, rooted in b_1, has, by the induction hypothesis, a main stem B. Then the path $(V(B) \cup \{r(T)\}, E(B) \cup \{\{r(T), b_1\}\})$ is a main stem of T. □

Remark 1. A tree may possess in general several main stems. Let us notice also that a main stem is not necessarily one of the longest paths which contain the root.

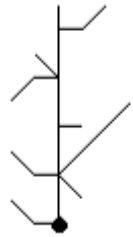


Fig. 3. A rooted brush

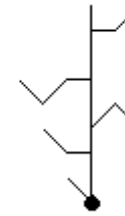


Fig. 4. A rooted comb

5 The Upper Bound

We need some new definitions. A *rooted brush* (Fig. 3) is a rooted tree such that the vertices of degree greater than 2 are on a same path P issued from the root.

A *rooted comb* X (Fig.4) is a rooted brush with $d(r(X)) \leq 2$ and $\forall v \in V(X)$, $d(v) \leq 3$.

The *length of a rooted comb* corresponds to the length of the longest path P issued from the root which contains all vertices of degree greater than 2.

To obtain an upper bound, we consider two building processes: the first one, a brushing M_B, maps rooted trees with a main stem into rooted brushes, the second one, a ramifying M_T, consists in obtaining a sequence of rooted trees, assuming that we have an increasing sequence of rooted combs. We note M_T^k the k-th element of the sequence. These building processes will possess the following fundamental property:

Property 1. Let (T, σ) a rooted tree with a main stem σ and $(X_n)_{n\in\mathbb{N}}$ a sequence of rooted combs:

$$\left(\forall T' \preceq T, M_B(T', \sigma) \preceq X_{|E(T')|}\right) \Rightarrow T \preceq M_T^{|E(T)|}\left((X_n)_{n\in\mathbb{N}}\right).$$

Lemma 4. *If building processes verify the property 1 and if for all i, the rooted comb X_i is strongly i-universal for the class of rooted brushes then the rooted tree $M_T^m\left((X_n)_{n\in\mathbb{N}}\right)$ is strongly m-universal for the class of rooted trees.*

Proof. It is just an interpretation of the property. □

We now establish the existence of building processes which satisfy property 1.

Brushing M_B (Fig. 5). Let T be a rooted tree with a main stem σ. We are going to associate a rooted brush B with it, denoted $M_B(T, \sigma)$ of the same size built from the same main stem σ with the following process: every e-branch B_e connected to the main stem by edge e is replaced by a path of length $|E(B_e)|$ connected by the same edge.

Ramifying M_T^k**.** For the second building process we work in two steps:

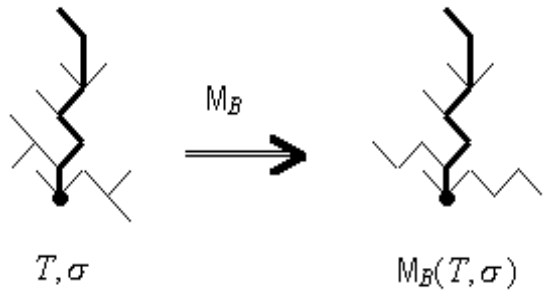


Fig. 5.

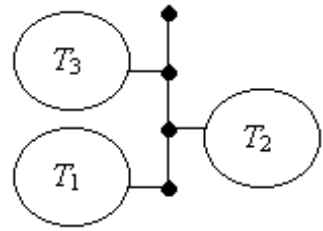


Fig. 6. A rooted comb $[T_1, T_2, T_3]$

First step. Given rooted trees $T_1, ..., T_k$ with disjoint vertex sets, we build another rooted tree T, denoted $[T_1, ..., T_k]$, in the following way:

$$V(T) = \bigcup_{i=1}^{k} V(T_i) \cup \{v_1, ..., v_{k+1}\},$$

$$E(T) = \bigcup_{i=1}^{k} E(T_i) \cup \{\{v_1, r(T_1)\}, ..., \{v_k, r(T_k)\}\} \cup \{\{v_1, v_2\}, ..., \{v_k, v_{k+1}\}\},$$

and $r(T) = v_1$.

If $T_i = \emptyset$, conventionally $\{v_i, r(T_i)\} = \emptyset$.

Prosaically, from a path $P_k = [v_1, ..., v_{k+1}]$ of size k and from k rooted trees $T_1, ..., T_k$, we build a rooted tree joining a branch T_i to the vertex v_i of P (Fig. 6).

Second step. By convention, $P_{-1} = \emptyset$.

We are going to construct rooted trees T_k in the following way:

$T_{-1} = \emptyset$, $T_0 = X_0$, and $\forall i,\ 1 \leq i \leq k, T_i = \left[T_{\min(u_1, i-1)}, ..., T_{\min\left(u_{n_i}, i-1\right)}\right]$ if $X_i = \left[P_{u_1}, ..., P_{u_{n_i}}\right]$.

We can now define M_T^k:

$$M_T^k\left((X_n)_{n\in\mathbb{N}}\right) = T_k.$$

Lemma 5. *The building processes described above verify the property 1.*

Proof. First, note that $M_T\left((X_n)_{n\in\mathbb{N}}\right)$ is an increasing sequence. We prove the lemma by recurrence on the size m of T. When $m = 0$ or $m = 1$, this is

trivial. We suppose the property is verified for T with size $m < m_0$. Let T be a rooted tree of size m_0 with a stem σ, we note $e_1, ..., e_k$ the edges of T issued from σ which do not belong to σ. To each e-branch of T with $e \in \{e_1, ..., e_k\}$ corresponds by M_B a e-branch (it is a path of same size) in $M_B(T, \sigma)$. So there exists k distinct e-branches $R_1, ..., R_k$ in X_{m_0} that we can respectively contract to obtain each e-branch with $e = e_1, ..., e_k$ in $M_B(T, \sigma)$. By recurrence hypothesis, we have for $1 \leq i \leq k, B_{e_i} \preceq M_T^{|E(B_{e_i})|}\left((X_n)_{n\in\mathbb{N}}\right)$ and we have also $M_T^{|E(B_{e_i})|}\left((X_n)_{n\in\mathbb{N}}\right) \preceq M_T^{|E(R_i)|}\left((X_n)_{n\in\mathbb{N}}\right)$. So each e-branch of T is a minor contraction of $M_T^{|E(R_i)|}\left((X_n)_{n\in\mathbb{N}}\right)$. By associativity of contraction map, we have $T \preceq M_T^{|E(T)|}\left((X_n)_{n\in\mathbb{N}}\right)$. □

In this phase, we determine a sequence of rooted combs $(X_i)_{i\in\mathbb{N}}$ such that the rooted combs X_i are strongly i-universal for the rooted brushes.

In order to achieve this result, we define F_p as the set of functions $f : \{1, ..., p\} \to \left\{1, ..., \left\lfloor \frac{p}{2} \right\rfloor\right\}$ satisfying the following property:

$$(\forall n \in \{1, ..., p\})\left(\forall i \leq \left\lfloor \frac{n}{2} \right\rfloor\right)(\exists k \in \mathbb{N})\,(n - i + 1 \leq k \leq n \text{ and } f(k) \geq i)$$

Lemma 6. *F_p is not empty, it contains the following function φ_p, defined for $1 \leq i \leq p$ by:*

$$\varphi_p(i) = \min\left(2^{v_2(i)+1} - 1, \left\lfloor \frac{p}{2} \right\rfloor, i - 1\right)$$

where $v_2(k)$ is the 2-valuation of k (i.e. the greatest power of 2 dividing k).

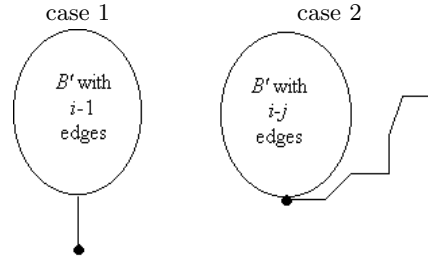
Proof. The verification is obvious. □

Lemma 7. *For every sequence $F = (f_1, f_2, ...)$ of functions such that $f_i \in F_i$ for $i \geq 1$ and $f_i(k) \leq f_{i+1}(k)$ for all $i \geq 1$ and $1 \leq k \leq i$, the rooted comb defined by $Comb_m^F = [Pf_1^m, ..., Pf_m^m]$ where Pf_i^m designs the path of size $f_m(m+1-i)-1$, for $1 \leq i \leq m$ is strongly m-universal for the rooted brushes.*

Proof. By induction on m : $Comb_1^F$ is strongly 1-universal for the rooted brushes.

Suppose that $Comb_i^F$ has all rooted brushes with $i - 1$ edges as rooted contractions.

We consider two cases depending on the shape of a rooted brush B of size i:



Brushes of case 1 are clearly rooted contractions of the rooted comb $Comb_i^F$ ($B' \preceq Comb_{i-1}^F$, so $B \preceq \left[P_0, Pf_1^{i-1}, ..., Pf_{i-1}^{i-1}\right] \preceq Comb_i^F$). Let us study case 2: B' is by induction hypothesis a rooted contraction of the rooted comb $Comb_{i-j}^F$, moreover $Comb_{i-j}^F \preceq \left[Pf_{j+1}^i, ..., Pf_i^i\right]$. Finally, by the property of f_i, there exists $1 \leq \alpha \leq j$, such that Pf_α^i has more than j edges. Linking these two points, we can conclude that the rooted brush B is always a rooted contraction of the rooted comb $Comb_i^F$. □

The rooted comb built as in lemma 7 will be said to be *associated to the sequence* F and denoted by $Comb_m^F$.

Theorem 4. *A minimum strongly m-universal rooted brush for the rooted brushes has $O(m\ln(m))$ edges.*

Proof. Proceeding as for theorem 1, we obtain, mutatis mutandis, that a m-universal brush for the brushes has at least $m\ln(m)+O(m)$ edges. This order of magnitude is precisely the size of the strongly m-universal rooted comb $Comb_m^F$ for the class of rooted brushes. □

We have this immediate corollary:

Corollary 2. *A minimum m-universal brush for the brushes has $O(m\ln(m))$ edges.*

By convention, we put $Comb_0^F = P_0$ (tree reduced in a vertex)

We define $Tree_m^F = M_T^m\left(\left(Comb_n^F\right)_{n\in\mathbb{N}}\right)$.

As before, we will say that the tree built in such a way is *recursively associated to the sequence* F and denoted by $Tree_m^F$.

Thus, we have:

Theorem 5. *The rooted tree $Tree_m^F$ is strongly m-universal for the class of rooted trees.*

We now analyze the size of $Tree_m^F$.

Proposition 1. *Let $F = (f_1, f_2, ...)$ be a sequence of functions such that $f_i \in F_i$ for $i \geq 1$. The size of a m-universal tree constructed from the sequence is given by the following recursive formula:*

$$u_{-1} = -1, u_0 = 0 \text{ and } u_k = 2k - 1 + \sum_{i=1}^{k} u_{f_k(i)-1}$$

Proof. It derives from the following observation:
m edges constitute the main stem, we have to add $m-1$ edges to link branches to the main stem and $\sum_{i=1}^{k} u_{f_k(i)-1}$ edges for the branches. □

Theorem 6. *There is a sequence of functions $G = (g_1, g_2, ...)$ such that $g_i \in F_i$ and $\left|E\left(Tree_m^G\right)\right| < (2m)^c$ where $c = 1.984...$ is the unique positive solution of the equation $\frac{1}{2^c} + \frac{1}{2^{2c}} + \frac{1}{2^{(c-1)}-1} - \frac{1}{2^c-1} = 1$.*

Proof. We take the following sequence of functions:
$g_m(i) = \min\left(2^{\upsilon_2(i)+1}, i\right)$ if $i < m$ and i even, $g_m(i) = 1$ if i odd and $g_m(m) = \left\lfloor \frac{m}{4} \right\rfloor$. It is clear that, if m is a power of 2, the comb $Comb_m^G$ is strongly m-universal for the brushes.

In fact, the function g_m takes the value $2^{\upsilon_2(i)+1}$ when i is not a power of 2, otherwise it is equal to i. Thanks to this remark and with $u_m < m + \sum_{i=1}^{m} u_{f_m(i)}$, (the sequence of sizes is increasing), we obtain $u_{2^n} < 2^n + 2^{n-1} + \sum_{i=2}^{n-1} 2^{n-i} u_{2^i} - \sum_{i=2}^{n-1} u_{2^i} + u_{2^{n-1}} + u_{2^{n-2}}$. Thus, in evaluating the sums and reorganizing the terms, we obtain:

$$u_{2^n} < \alpha_n + 2^{nc}\beta$$

with

$$\alpha_n = 2^{n-1} + 1 + 2^c + \frac{1}{2^c - 1} - \left(\frac{2^n}{2^{(c-1)} - 1} + 2^{n(c-1)} \right)$$

$$\beta = \frac{1}{2^c} + \frac{1}{2^{2c}} + \frac{1}{2^{(c-1)} - 1} - \frac{1}{2^c - 1}$$

Now $\alpha_n < 0$ when $m > 1$ and $\beta \leq 1$ by definition of c.
So $u_{2^n} < 2^{nc}$, hence $u_m < (2m)^c$. □

Remark 2. We observe that $c = \frac{\ln(x)}{\ln(2)}$, where x is the positive root of $X^4 - 5X^3 + 4X^2 + X - 2 = 0$.

Theorem 2 then follows since any rooted tree which is strongly m-universal for the rooted trees is also clearly m-universal for the class of trees.

6 Conclusion and Related Questions

When using the sequence $\Phi = (\varphi_1, \varphi_2, ...)$ of lemma 7, the induction step leads to involved expressions that do not allow us to find the asymptotic behavior of the corresponding term u_m. A computer simulation gives that such a m-universal tree for the trees has less than $m^{1.88}$ edges. In any case, the constructive approach we proposed here, seems to be hopeless to reach the asymptotic best size of a m-universal tree for the trees.

Conjecture 1. The minimal size of a m-universal tree for the trees is $m^{1+o(1)}$.

As a possible way to prove such a conjecture, it would be interesting to obtain an explicit effective coding of a tree of size m using a list of contracted edges taken in a m-universal tree for the trees.

A variant of our problem consists in determining a minimum tree which contains as a subtree every tree of size m. This is closely related to a well known still open conjecture due to Erdös and Sös (see [5]).

References

1. R. Rado, Universal graphs and universal functions, *Acta Arith.,* **9** (1964), 331-340.
2. Z. Füredi and P. Komjàth, Nonexistence of universal graphs without some trees, *Combinatorica,* **17**, (2) (1997), 163-171.
3. N. Robertson and P.D. Seymour, series of papers on Graph minors, *Journal of combinarotics, serie B,* (1983-...).
4. C. Thomassen, Embeddings and Minors, chapter 5 in *Handbook of Combinatorics,* (R. Graham, M. Grötschel and L. Lovàsz, eds.), Elsevier Science B.V., 1995, 301-349.
5. P. Erdös and T. Gallai, On maximal paths and circuits of graphs, *Acta Math. Sci. Hungar.* **10** (1959), 337-356.

Optimal Area Algorithm for Planar Polyline Drawings*

Nicolas Bonichon, Bertrand Le Saëc, and Mohamed Mosbah

LaBRI-Université Bordeaux 1, 351 Cours de la Libération,
33405 Talence, France
{bonichon,lesaec,mosbah}@labri.fr

Abstract. We present a linear time algorithm based on Schnyder trees that produces planar polyline drawings. These drawings have the optimal area ($\frac{4(n-1)^2}{9}$) and width ($\lfloor\frac{2(n-1)}{3}\rfloor$), and have at most $n-2$ bends, where n is the number of vertices of the graph. Moreover, at most one bend per edge is needed.

1 Introduction

The subject of graph drawing has received intense attention due to a large variety of applications. We focus on planar graph drawings. Such graphs can be drawn without any edge crossing. So, several classes of drawings [1, 2, 9, 11, 17, 19] are needed. Common objectives include small area, few bends and good angular resolution. In this paper, we deal with polyline drawings [3, 4, 7, 14–16]. A polyline drawing is a drawing of a graph in which each edge is represented by a polygonal chain. In this paper, we focus on the grid size with a small number of bends.

In general, three kinds of criteria are used for polyline drawings: the grid size, the angular resolution and the number of bends. Clearly, it is difficult to optimize simultaneously all of them. In [14], a tradeoff is proposed. Precisely, a linear-time algorithm has been given to construct a polyline drawing of any plane graph with n vertices and maximum degree d on a $(2n-5)\times(\frac{3n}{2}-\frac{7}{2})$ grid with at most $5n-15$ bends and minimum angle $>\frac{2}{d}$ where each edge has at most three bends. In [7], polyline drawings with one bend per edge and an angular resolution of $\Theta(1/d(v))$ on a grid of size $30n\times 15n$ are given where $d(v)$ is the degree of a vertex v and n the number of vertices of the graph.

Our goal is to have a tradeoff between the grid size and the number of bends. It is known from [20] that any plane graph can be drawn in a $(n-2)\times(n-2)$ grid without bends; i.e. straight-line segments between vertices (for $n\geq 3$). However, no grid smaller than $(\lfloor\frac{2(n-1)}{3}\rfloor)\times(\lfloor\frac{2(n-1)}{3}\rfloor)$ can be used for such a drawing [12, 18].

We present an algorithm that produces polyline drawings with a grid $(n-\lfloor\frac{p}{2}\rfloor-1)\times(p+1)$ where $p\leq\frac{2n-5}{3}$, the number of bends is at most $n-2$ and each edge has at most one bend. If we consider optimal width polyline drawings, we

* This work has been supported by the TMR Research Network GETGRATS.

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 35–46, 2002.

can obtain drawings of height at most $(n-2)$. The width and the area of the grid are optimal. Since these drawings are obtained in linear-time, our contribution has both a theoretical interest and a practical application to graph drawing.

With a maximal plane graph G, is associated a *realizer* R [20], which is a partition of the set of internal edges into three particular trees. This partition can be computed in linear-time. Realizers are useful for many graph algorithms [6, 10, 13, 16], and particularly for graph drawing [8, 21]. We use the realizer of a plane graph to obtain a polyline drawing. The algorithm consists of mainly two steps. The first step is to compute a function, called *weak-stratification*, which associates to each vertex a horizontal layer corresponding to its ordinate. The second step is to compute the vertex abscissas and the bend coordinates. The weak-stratification verifies a set of conditions which guarantees that it is possible to compute such vertex abscissas and the bend coordinates so that the resulting polyline drawing is planar.

The paper is organized as follows. In Section 2, we recall a few definitions and we present realizers. In Section 3, the weak-stratification is defined and is used to obtain a polyline drawing algorithm of a plane graph. We give an algorithm that builds a weak-stratification associated with a realizer in Section 4.

2 Realizers of Plane Graphs

We assume that the reader is familiar with graph theory, and we use definitions from [2]. The graphs, we deal with, are simple and undirected. A *plane graph* is a planar graph with a given planar embedding, represented combinatorially by cyclic orderings of edges incident to all vertices, and by the choice of the external face. The vertices of the external face are also called *external* and the other vertices are called *inner* vertices.

Definition 1 *(Schnyder [20]) A realizer of a maximal plane graph G is a partition of the interior edges of G in three sets T_0, T_1, T_2 of directed edges such that for each interior vertex u there hold:*

1. *u has out-degree exactly one in each of T_0, T_1, T_2.*
2. *The counter-clockwise order of the edges incident on u is: leaving in T_0, entering in T_2, leaving in T_1, entering in T_0, leaving in T_2 and entering in T_1 (see Fig. 1).*

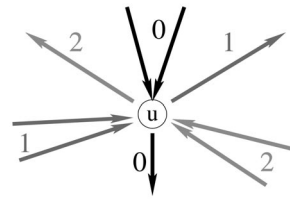


Fig. 1. Edge coloration and orientation around a vertex.

Schnyder showed that, T_0, T_1 and T_2 are three *ordered rooted trees* where their edges are oriented toward their roots, which are the external vertices v_0, v_1, v_2. An example of a maximal plane graph, and one of its realizer is given in Fig. 2.

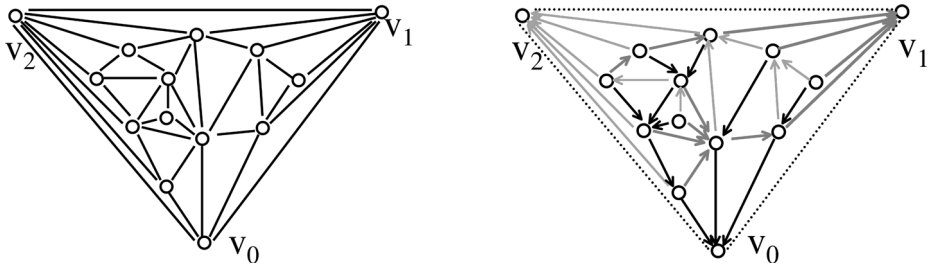


Fig. 2. An example of a realizer (a graph on the left side, and one of its realizers on the right side).

In the sequel, the edges of the tree T_i are colored with color i, where $i \in \{0, 1, 2\}$. $P_i(u)$ denotes the *parent* of u in the tree T_i. $Ch_i(u)$ denotes the *set of children* of u in the tree T_i. u is a *descendant* of v in T if u is a child of v or u is a child of a descendant of v. If u is a descendant of v, v is an *ancestor* of u. If v is an ancestor of u, $u \xrightarrow{i} v$ denotes the path colored i from u to v. Moreover, $v \xleftarrow{i} u$ denotes also $u \xrightarrow{i} v$. If $Ch_i(u) = \emptyset$ u is a *leaf* of T_i otherwise, u is an *inner vertex* of T_i. We write $u_1 >^i_{ccw} u_2$ (resp. $u_1 >^i_{cw} u_2$) if u_1 is after u_2 in the *counterclockwise preordering* (resp. *clockwise preordering*) of the tree T_i. Counterclockwise (resp. clockwise) preordering of a tree means visiting the root, then recursively the subtrees in the counterclockwise (resp. clockwise) order. Similarly, *counterclockwise postordering* (resp. *clockwise postordering*) of a tree T_i means recursively visiting the subtrees in the counterclockwise (resp. clockwise) order and then visiting the root.

Theorem 1 *[20] Let G be a maximal plane graph. A realizer R of G can be computed in linear-time.*

3 Polyline Drawing

In this section, we show how to compute a planar polyline drawing of a graph such that a weak-stratification is given. We recall first a few definitions and results related to polyline drawings.

A *polyline drawing* of a graph G is a drawing of G where the vertices are represented by points having integer coordinates and where edges are represented by polylines whose bends have integer coordinates. A *planar polyline drawing* is a polyline drawing without edge crossings. The *width* of a given polyline drawing is defined as the difference between the x-coordinates of the leftmost vertex or bend and the rightmost vertex or bend. Similarly, the *height* of a polyline drawing is

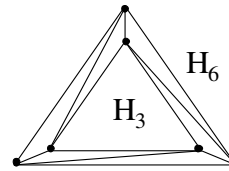


Fig. 3. Construction of the graph H_6.

given by the difference between the y-coordinates of the highest vertex or bend and the lowest vertex or bend.

Property 1 *For each $n \geq 3$, there is an n-vertex plane graph H_n such that the width and the height of each polyline drawing of H_n is at least $\lfloor \frac{2(n-1)}{3} \rfloor$ and the grid area is at least $\lfloor \frac{2(n-1)}{3} \rfloor \times \lfloor \frac{2(n-1)}{3} \rfloor$*

Proof. The result has been proved for straight-line drawings [12], using nested triangles (see Fig. 3). Using the same construction, the result can be directly generalized to polyline drawings: a drawing of H_n needs at least two more columns and at least two more rows than a drawing of H_{n-3}

3.1 Outline of the Polyline Drawing of a Realizer

Given a plane graph G and one of its realizers $R = (T_0, T_1, T_2)$ we compute a planar polyline drawing of G.

After choosing a tree of R, say T_0, a column is allocated to each leaf u of T_0 in clockwise prefix order. We denote by $x(u)$ this column. Each inner vertex u of T_0 is placed on a column of a particular leaf of its subtree.

It remains to compute the ordinate $y(u)$ of each vertex u of G. In order to compute these ordinates, we first set some rules for the edge bend positions:

- If a bend is needed for an edge $(u, P_0(u))$, it will be located at $(x(u), y(P_0(u)) + 1)$.
- If a bend is needed for an edge $(u,P_1(u))$, it will be located at $(x(last\_leaf(u)), y(u))$, where $last\_leaf(u)$ denotes the last leaf of the subtree of u.
- Similarly, if a bend is needed for an edge $(u, P_2(u))$, it will be located at $(x(first\_leaf(u)), y(u))$, where $first\_leaf(u)$ denotes the first leaf of the subtree of u.

Fig. 4 illustrates the way the different kinds of edges are drawn.

For a planar polyline drawing, partial overlapping of edges are also forbidden. Therefore, other rules are needed. Fig. 5 illustrates an overlapping configuration. We propose a new set of rules on vertex abscissas: if $v = P_2(u)$ and y$(v) = y(u)$ then $x(v) = x(last\_leaf(v))$ else $x(v) = x(first\_leaf(v))$.

Finally, we will consider the external edges (v_1, v_0) and (v_2, v_0) as if they were in T_0 and the edge (v_1, v_2) as is it was in T_2.

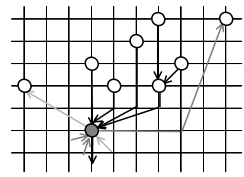


Fig. 4. Edge Drawings.

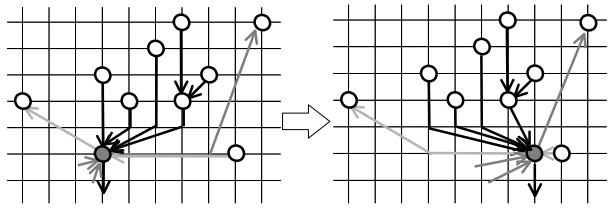


Fig. 5. Edge overlapping configuration and its corrected configuration.

Having defined the drawing design of the realizer (and so the graph), we propose a set of constraints on vertex ordinates which ensures that such a drawing is possible. In the sequel, ordinates satisfying these constraints will be computed in linear-time for any realizer. Moreover, we will show that the obtained drawing grid size is optimal.

3.2 Weak-Stratification of a Realizer

A *Layering* of a graph G is an application from the set of vertices of G to the set of non-negative integers. As we will show, a layering will define the layers of the plane on which the vertices will be positioned. In order to obtain a planar polyline drawing, a particular layering, defined as follows, will be considered.

Definition 2 *Let G be a maximal plane graph, $R = (T_0, T_1, T_2)$ be a realizer of G, and L be a Layering of G. Assume that $L_{max_1}(u)$ and $L_{max_2}(u)$ are defined as follows: $L_{max_1}(u) = max(L(u'), u' \in Ch_1(u))$ and $L_{max_2}(u) = max(L(u'), u' \in Ch_2(u))$. L is a weak-stratification of R if for every inner vertex u of G, we have:*

1. $L(v_0) = 0$
2. $L(P_0(u)) < L(u)$
3. $L(u) \leq L(P_1(u))$
4. $L(u) \leq L(P_2(u))$
5. $L_{max_2}(u) < L(P_1(u))$
6. $L_{max_1}(u) < L(P_2(u))$
7. $min(L_{max_1}(u), L_{max_2}(u)) < L(u)$
8. $L(v_1) \geq L_{max_2}(v_2)$

$L(v)$ is called the *layer* of the vertex v. The *height* of a layering is $max(L(v), v \in V(G))$.

Conditions 2, 3 and 4 set the ordinate of a vertex relatively of the ordinates of its parents in T_0, T_1 and T_2.

Condition 5 ensures that all the children of u in T_2 are placed on a lower layer than $L(P_1(u)) - 1$. Since they are also children of u in T_2 they are on a lower layer than $L(u)$ (see condition 4).

Moreover, all the descendant of u in T_2 are also on lower layers than $L(u)$ and $L(P_1(u)) - 1$). So the edge $(u, P_1(u))$ can be drawn without crossing. Similarly, condition 6 ensures that the edge $(u, P_2(u))$ can also be drawn without crossing.

Condition 7 ensures that any vertex u cannot have on its layer one of its children in T_1 and one of its children in T_2.

The last condition guarantees that we can draw the external edges.

In [5] a more restrictive definition of layering, called *stratification*, was defined. This definition has been used to compute orthogonal drawing of plane graph. The weak-stratification is less restrictive and then allows more compact drawing.

3.3 Polyline Drawing Algorithm of a Weak-Stratification

For each vertex v, we denote by $x_L(v)$ (resp. $x_R(v)$) the abscissa of the leftmost (resp. rightmost) leaf of the subtree of v.

The following algorithm computes the abscissas of inner vertices and the coordinates of edge bends of a realizer's weak-stratification. The resulting vertex and bend coordinates give a planar polyline drawing.

Algorithm 1 Polyline drawing.

for each vertex u of G, $y(u) \leftarrow L(u)$
Associate with each leaf of T_0 a column from left to right
for each inner vertex u of T_0 **do**
  if $L(u) = L_{max_2}(u)$ **then**
    $x(u) \leftarrow x_R(u)$
  else
    $x(u) \leftarrow x_L(u)$
  end if
  for each child v of u in T_0 **do**
    add the bend $(x(v), y(u) + 1)$ on the edge (u, v) if needed.
  end for
  add the bend $(x_L(u), y(u))$ on the edge $(v, P_2(u))$ if needed
  add the bend $(x_R(u), y(u))$ on the edge $(v, P_1(u))$ if needed
end for
add the bend $x_L(v_0), y(v_0) + 1$ on the edge (v_0, v_1)

Lemma 1 *Let G be a maximal plane graph with n vertices. Let L be a weak-stratification of a realizer $R = (T_0, T_1, T_2)$ of G. Let p be the number of leaves*

of T_0 and k the height of L. Algorithm 1 computes a planar polyline drawing of G in linear-time, on a grid $(p+1) \times (k)$. Moreover, the obtained drawing has at most $n-2$ bends and at most one bend per edge.

Proof. Conditions 5 and 6 of the definition 2 ensure that for each vertex u we can draw the edges toward $P_1(u)$ and $P_2(u)$.

Condition 7 ensures that we cannot have the 2 children of u, one in T_1 and one in T_2, on its layer. So, if a vertex u has a child in T_2 on its layer, the abscissa of u is set to $x_R(u)$. This avoids any overlapping with the edge $(u, P_1(u))$ (see Fig. 5). Otherwise, the abscissa of u is set to $x_L(u)$ (even if u has no child in T_1). This avoids any overlapping of with the edge $(u, P_2(u))$ (see Fig. 4).

Condition 8 ensures that the edge (v_1, v_2) can be drawn with a straight-line. Since v_0 has no child in T_2, $x(v_0) = x_L(v_0)$. So the edge (v_0, v_2) can be drawn with a straight-line.

Let us see when the edges are drawn with straight lines.

- If u is a leaf of T_0, then the edges $(u, P_1(u))$ and $(u, P_2(u))$ are drawn with straight-lines.
- If $x(u) = x_L(u)$ (resp. $x(u) = x_R(u)$) then the edge $(u, P_2(u))$ (resp. the edge $(u, P_1(u))$) and the edge from u toward its first child (resp. last child) in T_0 are drawn with straight-lines.

All the other edges are bended once. As shown in the previous subsection, the chosen coordinates for the bends ensure that the polyline drawing contains neither overlapping nor edge crossing.

The number of bended edges from a vertex u toward its children in T_0 or toward $P_1(u)$ and $P_2(u)$ is bounded by the number of its children in T_0 (at least two of these edges are straight-lines). Hence the number of bends on internal edges is bounded by number of edges in T_0: $n-3$. Moreover, the edges (v_0, v_1) and (v_1, v_2) can be drawn with a straight-line. So, the number of bends is at most $n-2$.

4 An Algorithm for Computing a Weak-Stratification of a Realizer

We present in this section an algorithm that builds a weak-stratification of a realizer. First, all the vertices of G are placed on layer 0. The children of v_0 in T_0 are placed on layer 1. Then, the algorithm treats each inner vertex of G with respect to the counter-clockwise postordering of T_2. The treatment of a vertex v consists of applying several *rules* on $L(v)$ (1, 2, 3.a, 3.b, 4.a and 4.b.). These rules can only increase the layers of vertices. Let us recall, as stated in Section 2, that v_0, v_1 and v_2 are respectively the roots of T_0, T_1 and T_2. When rules 1 and 2 are applied on a vertex v, the value $L(v)$ will no longer change. This vertex becomes *fixed*.

Property 2 *Let R be a realizer. If u is a descendant of v in T_i, then u cannot be a descendant nor an ancestor of v in T_j where $i \neq j$.*

Algorithm 2 Weak-stratification construction.

for each vertex u of G **do**
  $L(u) \leftarrow 0$
end for
for each inner vertex u of G in the counter-clockwise postordering of T_2 **do**
  1. $L(u) \leftarrow max(L(u), L(P_0(u)) + 1)$
  2. $L(u) \leftarrow max(L(u), min(L_{max_1}(u), L_{max_2}(u)) + 1)$
  3.a $L(P_2(u)) \leftarrow max(L(P_2(u)), L(u))$
  3.b $L(P_1(u)) \leftarrow max(L(P_1(u)), L(u))$
  4.a $L(P_2(u)) \leftarrow max(L(P_2(u)), L_{max_1}(u) + 1)$
  4.b $L(P_1(u)) \leftarrow max(L(P_1(u)), L_{max_2}(u) + 1)$
end for
5. $L(v_1) \leftarrow max(L(v_1), L_{max_2}(v_2) + 1)$

Property 3 *Let $R = (T_0, T_1, T_2)$ be a realizer of a maximal plane graph G. Let (u, v) be an edge of R. If $u = P_1(v)$ or $u = P_2(v)$ or $u \in Ch_0(v)$ then u is after v in the counter-clockwise postordering of T_2.*

Lemma 2 *Let G be a maximal plane graph. Let $R = (T_0, T_1, T_2)$ be a realizer of G. Algorithm 2 computes a weak-stratification of R in linear-time.*

Proof. v_0 is fixed on layer 0 and condition 1 of definition 2 is verified. The vertices are fixed in the counter-clockwise postordering of T_2, so a vertex u is treated, its children in T_1 and T_2 and its parent in T_0 have already been fixed (see property 3). Since the first vertex u treated in the main loop is a child of v_0, and a leaf of T_1 and a leaf of T_2, conditions $2-7$ of the definition 2 are verified for u.

Assume now that these conditions are verified for the m first fixed vertices and let us show that it is also true after the treatment of the next vertex u.

Rule 1 ensures that vertex u is on a higher layer than its parent in T_0. So after applying the rule 1, condition 2 is verified for u.

Rule 2 says that if u has 2 children, one in T_1 and the other in T_2 on its layer, then $L(u)$ is incremented. So after applying the rule 2, condition 7 is true for u.

Rules 3.a (resp. 3.b) ensures that vertex $P_2(u)$ (resp. $P_1(u)$) is on a higher layer than u. This corresponds to condition 3 (resp. 4) of definition 2.

Similarly, rules 4.a and 4.b ensure that conditions 5 and 6 are satisfied for vertex u.

During the treatment of u, $L(u)$, $L(P_1(u))$ and $L(P_2(u))$ are not decreased, so conditions $2-7$ remain true for the m^{th} first vertices.

So at the end of the main loop, conditions $2-7$ are verified for all inner vertices of G. Moreover, the last step of the algorithm ensures that condition 8 is true. Hence, at the end of the execution, L is a weak-stratification of the realizer R. Obviously the algorithm is linear-time.

Fact 1 *At each step of Algorithm 2, for all $i < max\{L(u), u \in V(G)\}$, there exists a fixed vertex v such that $L(v) = i$.*

Lemma 3 *Let L be a weak-stratification of R generated by Algorithm 2. If v is a leaf of T_0, then there exists $u \neq v$ such that $L(v) = L(u)$. Moreover, it is also true for v_2.*

Proof. Since v_2 is neither in T_0 nor in T_1, the only rules that can change its layer are 3.a and 4.a. When applying rule 5, either $L(v_2) = L_{max_2}(v_2)$ and then v_2 is on the same layer than one of its children, or $L(v_2) = L_{max_2}(v_2) + 1$ then v_2 and v_1 are on the same layer.

Let v be a leaf of T_0. When v becomes fixed, two configurations could have occur:

- Case 1: there was already a vertex u such that $L(u) > L(v)$. Since at each step of the algorithm, each layer lower than $L(u)$ contains at least one fixed vertex, it is also the case for the layer $L(v)$ before v was fixed (see fact 1).
- Case 2: vertex v was the highest vertex of L. Consider u the first vertex set on $L(v)+1$. This vertex was set on this layer after that v became fixed. Such a vertex always exists since at the end of the algorithm, v_1 is the highest vertex of L.

  Let us consider the rule that has set u on $L(v) + 1$.
  - Case 2.1: if it is rule 1, then $L(P_0(u)) = L(v)$. Since v is a leaf of T_0, $P_0(u) \neq v$.
  - Case 2.2: if it is rule 2. This means that there are at least 2 fixed vertices on layer $L(v)$, one child of u in T_1 and one child of u in T_2.
  - Case 2.3: if it is rule 3.a (resp. 3.b). This implies that u has a fixed child w in T_2 (resp. in T_1) such that $L(w) = L(v) + 1$. This is in contradiction with the fact that u was the first one to appear in $L(v) + 1$.
  - Case 2.4: if it is rule 4.a (resp. 4.b). Then if $u = P_1(v)$ (resp. $u = P_2(v)$), we have $L(v) = L_{max_2}(v)$ (resp. $L(v) = L_{max_1}(v)$). So, v has a fixed child w in T_2 (resp. T_1) such that $L(v) = L(w)$.
  - Case 2.5: if it is rule 5, then $u = v_1$ and $L(v_2) = L_{max_2}(v_2)$. So $L(v) = L(v_2)$.

Lemma 4 *Algorithm 2 computes, in $O(n)$ time, a weak-stratification of a realizer $R = (T_0, T_1, T_2)$, with height at most $n - \lfloor \frac{p}{2} \rfloor - 1$ where p is the number of leaves of T_0.*

Proof. We know that there is at least one vertex per layer; moreover a leaf of T_0 is never alone on its layer. So in the worst case, the highest layer contains only v_1, and each other layer contains either an inner vertex of T_0, two leaves of T_0 or one leaf of T_0 and v_2.

So we have $n - 2 - p$ layers with one inner vertex of T_0, $\lfloor \frac{p+1}{2} \rfloor$ layers for the leaves and v_2 and one other layer for v_1. Hence, the height of the computed weak-stratification by the previous algorithm is at most $n - \lfloor \frac{p}{2} \rfloor - 1$.

Theorem 2 *[8] Let $R = (T_0, T_1, T_2)$ be a realizer. At least one of the trees of R has at most $\lfloor \frac{2n-5}{3} \rfloor$ leaves.*

Theorem 3 *Let G be a plane graph with n vertices. G has a planar polyline drawing with at most $\frac{4(n-1)^2}{9}$ as area and at most $\lfloor\frac{2(n-1)}{3}\rfloor$ as width. Such a drawing can be obtained in linear-time. Moreover, each edge has at most one bend and the drawing has at most $n-2$ bends.*

Proof. Let G' be a triangulation of G. G' can be obtained in linear-time. Let $R = (T_0, T_1, T_2)$ be a realizer of the maximal plane graph G'.

Let T_i be a tree with at most $\lfloor\frac{2n-5}{3}\rfloor$ leaves. Let $R' = (T'_0 = T_i, T'_1 = T_{i+1}, T'_2 = T_{i+2})$ be the realizer of G' obtained by circular permutation on the trees of R. Let p be the number of leaves of T'_0.

Algorithm 2 computes in linear-time a weak-stratification with p leaves and height $n-\lfloor\frac{p}{2}\rfloor-1$. Using this weak-stratification, Algorithm 1 computes a planar polyline drawing of G on a grid of $(p+1)\times(n-\lfloor\frac{p}{2}\rfloor-1)$ with at most $n-2$ bends. Since $p \leq \lfloor\frac{2n-5}{3}\rfloor$, the width of the drawing is at most $\lfloor\frac{2(n-1)}{3}\rfloor$. Since the area is an increasing function in p, its value is at most $\frac{4(n-1)^2}{9}$. Property 1 states that the width and the area of the drawing obtained by Algorithm 1 are optimal.

Here is an example of a drawing of a graph with 50 vertices on a grid of 22×20:

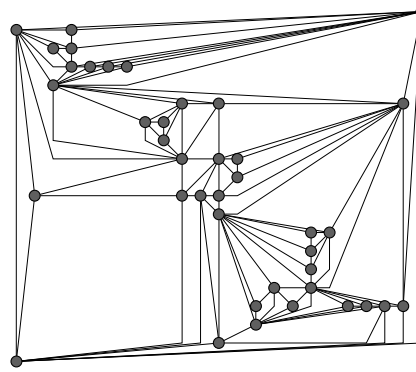


Fig. 6. Example of a polyline drawing.

References

1. C. Batini, , G. di Battista, and R. Tamassia. Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):61–79, 1988.
2. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the visualisation of graphs.* Prentice Hall, 1999.
3. G. Di Battista and R. Tamassia. Algorithms for plane representations of acyclic digraphs. In *Theoret. Comput. Sci*, volume 61, pages 175–198, 1988.

4. G. Di Battista, R. Tamassia, and I.G. Tollis. Constrained visibility representations of graphs. In *Inform. Process. Lett.*, volume 41, pages 1–7, 1992.
5. N. Bonichon, B. Le Saëc, and M. Mosbah. Orthogonal drawings based on the stratification of planar graphs. Technical Report RR-1246-00, LaBRI, 2000.
6. N. Bonichon, B. Le Saëc, and M. Mosbah. Wagner's theorem on realizers. In *International Colloquium on Automata, Languages and Programming 2002 (ICALP'02)*, LNCS, to appear.
7. C. C. Cheng, C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Drawing planar graphs with circular arcs. In *Graph Drawing (Proc. GD '99)*, volume 1731 of *Lecture Notes in Computer Science*, pages 117–126. Springer-Verlag, 1999.
8. Yi-Ting Chiang, Ching-Chi Lin, and Hsueh-I Lu. Orderly spanning trees with applications to graph encoding and graph drawing. In *Proc. 12th Symp. Discrete Algorithms*, pages 506–515. ACM and SIAM, 2001.
9. N. Chiba, T. Yamanouchi, and T. Nishizeki. Linear algorithms for convex drawings of planar graphs. *Progress in Graph Theory*, pages 153–173, 1984.
10. Richie Chih-Nan Chuang, Ashim Garg, Xin He, Ming-Yang Kao, and Hsueh-I Lu. Compact encodings of planar graphs via canonical ordering and multiple parentheses. In *Proc. 25th International Colloquium on Automata, Languages, and Programming (ICALP'98)*, volume 1443, pages 118–129, 1998.
11. U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In F. J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes in Computer Science*, pages 254–266. Springer-Verlag, 1996.
12. H. De Fraysseix, J. Pach, and J. Pollack. Small sets supporting fary embeddings of planar graphs. In *20th Annual ACM Symp. on Theory of Computing*, pages 426–433, 1988.
13. H. De Fraysseix, J. Pach, and J. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.
14. C. Gutwenger and P. Mutzel. Planar polyline drawings with good angular resolution. In S. Whitesides, editor, *Graph Drawing (Proc. GD '98)*, volume 1547 of *Lecture Notes in Computer Science*, pages 167–182. Springer-Verlag, 1998.
15. G. Kant. Hexagonal grid drawings. In *In Proc 18th Internat. Workshop Graph-Theoret. Concepts Comput. Sci*, 1992.
16. G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16:4–32, 1996.
17. Gunnar W. Klau and Petra Mutzel. Quasi-orthogonal drawing of planar graphs. Research Report MPI-I-98-1-013, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany, May 1998.
18. M.Chrobak and S. Nakano. Minimum-width grid drawings of plane graphs. In *Graph Drawing (Proc. GD '94)*, pages 104–110, 1995.
19. P. Rosenstiehl and R.E. Tarjan. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete Comput. Geom.*, 1:343–353, 1986.
20. W. Schnyder. Planar graphs and poset dimension. *Order*, 5:323–343, 1989.
21. W. Schnyder. Embedding planar graphs on the grid. *Proc. 1st ACM-SIAM Symp. Discrete Algorithms*, pages 138–148, 1990.

Appendix

Proof. (Property 2)
Assume that u is a descendant of v in T_0 and u is an ancestor of v in T_1. Since vertex v satisfies condition 2 of the realizer's definition, $P_0(u)$ is in the region delimited by the cycle $C = u \underset{\rightarrow}{0} v, v \underset{\leftarrow}{1} u$ (see Fig. 7). Let t be the common vertex of C and the path $u \underset{\rightarrow}{1} v_1$. Assume that t is on the path $u \underset{\rightarrow}{0} v$. This is impossible because this contradicts condition 2 on vertex t. So t can only be on the path $u \underset{\rightarrow}{1} v$. But in this case we have a cycle colored $1{:}t \underset{\leftarrow}{1} u, u : \underset{\leftarrow}{1} t$. This is also impossible since T_1 is a tree. So, if u is a descendant of v in T_0, then u cannot be an ancestor of v in T_1. A similar reasonning can be made to show that if u is a descendant of v in T_0, then u cannot be an descendant of v in T_1.

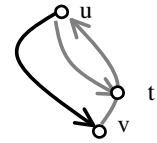


Fig. 7. Impossible configuration where u is a descendant of v in T_0 and u is an ancestor of v in T_1.

Proof. (Property 3) Let us consider the three following cases:

- $u = P_2(v)$: obvious.
- $u = P_1(v)$: Assume that u is before v in the counter-clockwise postordering of T_2. Let w be the nearest common ancestor of u and v in T_2. The cycle $C = ((v, u), u \underset{\rightarrow}{2} w, w \underset{\leftarrow}{2} v)$ determines a region of the plan (see Fig. 8 a.). In order to respect condition 2 of the realizer's definition for vertex u, $P_0(u)$ must be in this region. Let us consider the vertex t that belongs to $u \underset{\rightarrow}{0} v_0$ and to the cycle C. Condition 2 implies that t is on the path $u \underset{\rightarrow}{2} w$. So t is an ancestor of u in T_2 and t is an ancestor of u in T_1. This is a contradiction with Property 2.
- $u \in Ch_0(v)$: Let w be the nearest common ancestor of $P_1(v)$ and v in T_2. Vertex u must be in the region delimited by the cycle $C = ((v, P_1(v)), P_1(v) \underset{\rightarrow}{2} w, w \underset{\leftarrow}{2} v)$ (see Fig. 8 b.). In this region, all the vertices are after v in the counter-clockwise postordering of T_2. So u is after v in this order.

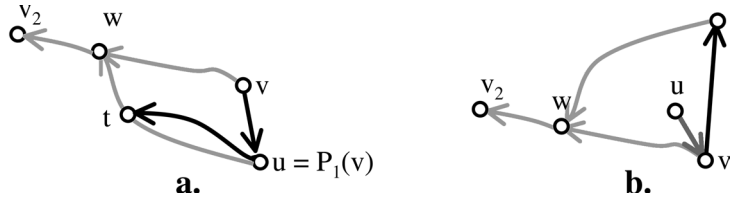


Fig. 8. a. An impossible configuration where $P_1(v)$ is before v in the counter-clockwise postordering of T_2. **b.** A region delimited by $C = ((v, P_1(v)), P_1(v) \rightarrow^2 w, w \leftarrow^2 v)$.

Cycles in Generalized Networks

Franz J. Brandenburg

University of Passau,
94030 Passau, Germany
brandenb@informatik.uni-passau.de

Abstract. Generalized networks specify two parameters for each arc, a cost and a gain. If x units enter an arc a, then $x \cdot g(a)$ exit. Arcs may generate or consume flow, i.e., they are gainy or lossy. The objective is a cheapest path of a unit flow from the source (SGSP) and the single-pair cheapest path (SPGSP).
There are several types of negative cycles. A lossy cycles decreases the gain. Then even a negative cost cycle has only bounded cost. A gainy cycle increases the flow. Then even a positive cost cycle may induce a total cost of minus infinity.
We solve SGSP by an extension of the Bellman-Ford algorithm. At the heart of the algorithm is a new and effective cycle detection strategy. The algorithm solves SGSP in $O(nm \log n)$, which improves to $O(nm)$ in lossless networks and to $O(n \log n + m)$ in a monotone setting. Our algorithm is simpler and at least a factor of $O(n)$ faster than the previous algorithms using linear programming or complex parametric search and scaling techniques. This improvement is a big step for such a well-investigated problem.
To the contrary, the single-pair generalized shortest path problem SPGSP is NP-hard, even with nonnegative costs and uniformly lossy arcs.

1 Introduction

What is the cheapest way to send data through a communication network when links are charged per unit and each link expands or compresses the data, e.g., by attaching or erasing routing information? How much does it cost transporting a unit commodity in a generalized network? In generalized networks each arc has a gain $g(a)$ which might expand or consume the flow. If x units enter an arc, then $x \cdot g(a)$ units exit. An arc may be gainy or lossy, and accordingly there are gainy and lossy cycles. The cost may be positive or negative and is charged per unit. The objective is a cheapest path of a unit flow from the source.

Generalized network flow problems have been studied intensively in the literature from the early days of production planning and combinatorial optimization [8, 10, 14, 16, 18] to recent research [2, 9, 12, 21, 22, 25]. They arise in several application contexts and can be used to model many situations which are impossible to express as standard network flow problems [2, 8, 10, 14].

Examples of generalized network flow problems usually deal with the loss of a commodity. Examples are the conversion of currencies, the evaporation of liquids

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 47–56, 2002.

and gas or the loss due to damage, theft, or toll. The use of an arc is charged per unit. It is paid by a fraction of the transported commodity. As a particular example, suppose there is radioactive waste, waste oil, or rubbish and there is a toll for the transportation and a fee at a waste disposal site. What are the least costs to dispose the waste, if you do not hazard the consequences. Radioactive material dissociates over time, waste oil disposes by an ocean dumping, and you can put your rubbish into your neighbor's trash bin or litter it on highways. It may be cheaper to wait or drive around and get rid of your waste than paying for the ordinary disposal! The general scenario has losses and gains, such as stocks, data communication or behavioral sciences, with an increase due to better quotations, the attachment of routing information or junk to each data package, or breeding.

The generalized shortest path problem is a special case of the min-cost generalized network flow problem, where some flow is transported on a single path without capacities. The amount of flow is the product of the gains of the arcs, and all nodes except the ends are flow conserving. Most approaches towards a solution use linear programming techniques (LPs) and exploit the specific properties of the dual problem as a two-variables-per-inequality problem, which can be solved by specialized techniques in $O(n^2 m \log n)$ [1, 6, 13].

Our focus is on combinatorial algorithms. Goldberg et al. [12] stated in 1991 that no polynomial-time bounds were known for combinatorial algorithms of generalized maximal flow problems. Ahuja et al. [2] could not bound the number of iterations of their generalized network simplex algorithm for the generalized min-cost flow problem. Recently, Oldham [21, 22] investigated the generalized shortest path problem and presented fully polynomial-time approximation schemes. Oldham's algorithm combines a comparison subroutine with a good guess of the cost of a cheapest path with a scaling technique and has a running time of $O(n^2 m \log n)$. Restricted instances with nonnegative costs and flow multipliers at most one can be solved more efficiently by a variant of Dijkstra's algorithm, as discovered by Charnes and Raike [5] and stated as a combinatorial algorithm by Wayne and Fleischer [9, 25], if the objective is a unit at the target. All these approaches to the generalized shortest path problem attempt to use the classical shortest path algorithms by of Dijkstra and Bellman-Ford from the source.

Our approach is *"backwards"*. We solve the from-the-source generalized shortest path problem SGSP from the target. Proceeding backwards has been proposed by Vladimir Batagelj and Patrice Ossona de Mendez at the Dagstuhl Seminar 98301 *"Graph Algorithms and Applications"*, 1998, where the generalized shortest path problem has been discussed in an open problem session. By the backwards approach, an extension of the classical Bellman-Ford algorithm applies and solves SGSP in $O(nm \log n)$. Our algorithm is even more powerful and solves SGSP from all nodes.

At the heart of our algorithm is an advanced cycle detection strategy, which combines Tarjan's subtree disassembly [3, 19, 24] with a direct subtree test. The direct subtree test costs a factor of $O(\log n)$, which is an improvement over the $O(n)$ factor for the *"walk-to-the-root"* and *"subtree-traversal"* strategies of [3].

In standard networks the cycle detections can be done in $O(1)$ amortized time, since the detected cycles are deleted.

There are well-founded specializations of generalized networks. It can be checked in $O(nm)$ whether or not it has lossy cycles; otherwise it is lossless. In lossless networks SGSP can be solved in $O(nm)$. If the costs are nonnegative and all arcs are gainy then SGSP can be solved by a variant of Dijkstra's algorithm in $O(n \log n + n)$. For the symmetric to-the-target problem this has been discovered by Charnes and Raike [5] and has been stated explicitly by Fleischer and Wayne [9, 25]. Hence, generalized shortest path problems are not much harder than standard shortest path problems.

To the contrary, if cycles are excluded, then the generalized shortest path problem is NP-hard by a reduction of the Hamilton path problem. Hence, the single-pair shortest path problem SPGSP is NP-hard. The NP-hardness result also holds for SPGSP instances with nonnegative costs and lossy arcs with uniform gains $g < 1$ and with negative costs and purely gainy arcs or purely lossy arcs.

Our NP results increase the list of known infeasible instances of generalizations of shortest paths and maximum flow problems, such as the longest simple path [11], the exact path length problem [20], the constraint shortest path problem [2] and the integer generalized flow problem [2].

2 Generalized Shortest Paths

Let $G = (V, A)$ be a directed graph with n nodes and m arcs. There are two weighting functions for the arcs, a cost function $c : A \to R$ into the reals and a gain function $g : A \to R^+$ into the positive reals. The cost of a path is the sum of the cost of its arcs. The cost of each arc is $c(a)$ per unit of flow. The gain $g(a)$ of each arc increments or decrements the flow by a certain percentage. If x units of flow enter an arc a, then $x \cdot g(a)$ units exit. An arc is *lossy*, if $g(a) < 1$, and *gainy*, if $g(a) > 1$.

The cost of a path τ with the arcs $(a_1, \ldots, a_r)$ is $c(\tau) = \sum_{i=1}^{r} c(a_i)(\Pi_{j<i} g(a_j))$, and an initial flow of k units costs $k \cdot c(\tau)$. The objective is a cheapest path with minimal cost. Various applications and solutions of generalized network problems are described in Chapter 15 in [2].

The generalized shortest paths problem is commonly known as the restricted generalized uncapacitated transshipment problem. A solution partitions the set of nodes into feasible and infeasible nodes. A node is *infeasible*, if it is unreachable or if it has minus infinite cost. The *feasible* nodes have bounded cost, and the cheapest path from a feasible node either reaches the target or ends in a lossy cycle. These four cases are detected by our algorithm.

3 Lossy and Negative Cycles

At the heart of our algorithm is a cycle detection strategy for cost decreasing cycles. For efficiency reasons we use an extension of Tarjan's subtree disassem-

bly together with a direct subtree test [19, 24]. Other negative cycle detection strategies are discussed in [3]. With unrestricted costs and gains there are cycles with minus infinite cost. These negative cycles are detected and, as usual, the nodes of such cycles and beyond are eliminated from further shortest path computations at $O(1)$ amortized time. As a particularity, there are cheap lossy cycles. These are the Type III paths of Theorem 2.6 in [12]. If a lossy cycle is traversed infinitely often it absorbes the flow and operates as a sink.

The cycle detection needs a test whether or not a node is in a particular subtree of the cheapest path tree. In Tarjan's subtree disassembly this costs $O(1)$ amortized time, since the subtree can be destroyed while being traversed. Here we must preserve the subtree for efficient lossy cycle detections, which costs a factor of $O(\log n)$. This is an improvement over the $O(n)$ factor of the walk-to-the-root and subtree-traversal strategies of [3].

Before we come to the main algorithm we develop some formulas for the computation of generalized shortest paths and characterize lossy and negative cycles.

Let G be a generalized network with the cost and gain functions c and g. Suppose that a path $\tau = \tau_1 \circ \tau_2$ is the concatenation of two subpaths τ_1 and τ_2. Then $g(\tau) = g(\tau_1) \cdot g(\tau_2)$ and $c(\tau) = c(\tau_1) + g(\tau_1)c(\tau_2)$.

These equations are used for the analysis of negative cycles and for the solution of the generalized path problem by our Reverse-Generalized-Bellman-Ford algorithm. They define a left-distributive closed semi-ring, which is not right-distributive, as stated by Oldham [21, 22]. Since the gains are positive, we can conclude that the suffix of a cheapest path is a cheapest path. This does not necessarily hold for prefixes.

Consider *bad* cycles which decrease the cost or the gain. A cycle is *lossy*, if its gain is less than one. It is a *negative* cycle, if the cost is negative.

Let γ be a cycle at some node v with cost $c(\gamma)$ and suppose that the target t is reachable from v by a path τ with cost $c(\tau)$ and gain $g(\tau)$. γ decreases the cost of a unit flow from v, if $c(\gamma \circ \tau) < c(\tau)$. Since $c(\gamma \circ \tau) = c(\gamma) + g(\gamma)c(\tau)$, this is equivalent to $c(\gamma) < c(\tau)(1 - g(\gamma))$. This formula gives raise to cycles with bounded and unbounded costs.

First consider the case with bounded cost. A lossy cycle γ at v with $0 < g(\gamma) < 1$ operates as a flow absorbing sink. Its cost is $\sum_{i=1}^{\infty} c(\gamma)g(\gamma)^i = c(\gamma)/(1 - g(\gamma))$. Running through γ decreases the cost of transporting one unit from v to the target t if and only if $c(\gamma) < c(\tau)(1 - g(\gamma))$. Then the path τ from v to t is replaced by the infinite cycle γ at v and the cost $c(\tau)$ of v is replaced by the limit $c(\gamma)/(1 - g(\gamma))$. Hence, it may be cheaper to absorb the flow than transporting it to the destination. The algorithm detects this situation and settles it by a new arc (v, t) with the label *lossy* and the cost $c(\gamma)/(1 - g(\gamma))$. A cheapest path is a path of Type I or Type III according to [12].

Lemma 1. *If τ is the cheapest path with a minimal set of arcs from a feasible node v, then τ is a simple path from v to the target t or $\tau = \sigma \circ \gamma$, where σ is a simple path from v to some node x, γ is a lossy cycle at x and x is the only common node of $\sigma \circ \gamma$.*

Proof. Assume the contrary, and let x be the first node that repeats on τ and has two distinct successors. Then $\tau = (v_1, \ldots, v_p, \ldots, v_q, \ldots)$, where $v = v_1$, $x = v_p = v_q$ and $v_{p+1} \neq v_{q+1}$. The subpath from v_p to v_q is a cycle γ at x with cost $c(\gamma)$ and gain $g(\gamma)$. For each node v_i of τ let $c(v_i)$ be the cost of the suffix of τ from v_i. $c(v_i)$ is minimal, since the suffix of a cheapest path is a cheapest path. Then $c(v_q) = c(\gamma)/(1 - g(\gamma))$; otherwise the deletion of γ or using γ twice yields a cheaper path. Then the deletion of γ yields a shorter path $(v_1, \ldots, v_{p-1}, v_q, \ldots)$ with the same cost and a subset of the set of arcs of τ, and the number of nodes with two distinct successors is decreased by one. By induction all nodes with two distinct successors are deleted from τ. Then τ is a simple path or τ is a simple path and a lossy cycle.

Next, we investigate cycles with minus infinite cost. Such nodes define an infeasible instance and are deleted immediately.

Lemma 2. *Let γ be a cycle at some node v and let τ be a path from v. γ induces minus infinite cost at v if and only if*

(i) $g(\gamma) \geq 1, c(\gamma) < 0$ and $c(\tau)(g(\gamma) - 1) < |c(\gamma)|$
(ii) or $g(\gamma) > 1, c(\tau) < 0$ and $c(\gamma) < |c(\tau)|(g(\gamma) - 1)$.

Proof. After i rounds of γ the cost at v is $c(\gamma^i \circ \tau) = \sum_{j=0}^{i} c(\gamma)g(\gamma)^j + c(\tau)g(\gamma)^{i+1}$. Then $c(\gamma^i \circ \tau) \to -\infty$ if and only if (i) or (ii) hold true.

In the first case, the cycle itself decreases the cost and this decrease is not consumed by the path τ transporting $g(\gamma)^{i+1}$ units. In the second case, the path τ has negative cost and the cycle increases the gain faster than its cost. Observe, that a positive cost cycle may give raise to minus infinite total cost.

4 The Reverse-Generalized-Bellman-Ford Algorithm

Our algorithm for SGSP is the Reverse-Generalized-Bellman-Ford algorithm. The algorithm runs twice. In the first run it computes the cheapest paths tree of simple paths to the target and detects the cheapest lossy cycles. This detection costs an extra factor of $O(\log n)$ for a direct subtree test. The output are the unreachable nodes and cost estimates for the other nodes, which are translated into new arcs directly to the target. In $O(nm)$ the second run completes the computation of a cheapest path from each node and its cost.

For efficiency reasons we use an extension of Tarjan's subtree disassembly [24] as an immediate cycle detection strategy. Tarjan's variant has been described in detail in [19], Section 7.5.9. It performs very well in experimental evaluations [3, 4, 19]. However, the subtree disassembly destroys subtrees at $O(1)$ amortized cost. Subtree disassembly is inadaequate for lossy cycles, since the nodes of such cycles must be reused, if they belong to a cheaper lossy cycle. A deletion and a reconstruction would cause an $O(n)$ delay. This situation is illustrated by a graph with a node v such that the cheapest path tree T looks like a palm tree. T has a path of length $n/4$ from v to some node x, and x branches to $n/4$ nodes

$w_1, \ldots, w_{n/4}$ such that for $1 \leq i \leq n/4$, w_i has a back-arc to v and the cycle from v through w_i is a lossy cycle with lower cost than the lossy cycle through w_{i-1}. Our algorithm performs an explicit cycle test in $O(\log n)$ time and tests whether or not a node w is in a subtree rooted at another node v.

The input of the Reverse-Generalized-Bellman-Ford algorithm is a directed graph $G = (V, A)$, a target t, and for each arc a cost $c(a)$ and a gain $g(a)$.

The nodes of G are stored in a node array and are accessed directly in $O(1)$. Each node has its list of incoming arcs of G, which is scanned sequentially, and a pointer to its copy in T, and reverse. The computed costs and gains are recorded by $c(v)$, $g(v)$, and $c_{\text{cycle}}(v)$.

The main data structure is a cheapest path tree T consisting of the nodes v of G. The arcs of T are defined by the successor relation while relaxing arcs of G. T is an in-tree rooted at the target t. Moreover, T is traversed in preorder and in postorder, and the pre- and postorder numbers $pre(v)$ and $post(v)$ of the nodes v of T are stored in balanced trees with $O(\log n)$ for the operations search, insert, and delete-subtree. Delete-subtree deletes the preorder (postorder) numbers of all nodes of a subtree of T. These are consecutive. A 2-3-4 tree guarantees $O(\log n)$ time for all operations using split and join operations for delete-subtree, see [7].

The pre- and postorder numbers are used for subtree tests $w \in T(v)$, where $T(v)$ is the subtree of T rooted at v. For nodes $v, w \in T$, $w \in T(v)$ if and only if $pre(v) < pre(w)$ and $post(v) > post(w)$. All other operations can be done in $O(1)$ time. Recall that in Tarjan's subtree disassembly $w \in T(v)$ is checked in a straight-forward manner in $O(|T(v)|)$ while traversing and deleting $T(v)$. This amortizes to $O(1)$, since $T(v)$ is destroyed.

The Reverse-Generalized-Bellman-Ford algorithm operates in phases. In the k-th phase it visits the nodes of G at distance k to the target. The distance is the number of arcs of a simple path and is used in the correctness proof. The nodes in the k-th phase are the current leaves of T. They are stored in a FIFO-queue Q, as described in [19]. The algorithm inspects the incoming arcs of these nodes for a relaxation and an improvement of the cost of the other endnode. It starts with the target t in the 0-th phase.

Reverse-Generalized-Bellman-Ford(G, t, c, g)
Input: a generalized graph $G = (V, A, c, g)$ and a target t
Output: the cheapest path tree T, and parameters for the nodes.

```
1  for each v ∈ V do
2      v = nil; c_cycle(v) ← ∞
3  T ← t; Q ← t
4  c(t) ← 0; g(t) ← 1
5  while Q ≠ ∅ do
6      w ← extract-first(Q)
7      for each v ∈ V with (v, w) ∈ A do     // Relax(v, w, c, g)
8          if c(v) > c(v, w) + g(v, w)c(w) then
9              if w ∉ T(v) then              // no cycle at v, update v
10                 delete the nodes of T(v) from T and from Q
```

```
11                  c(v) ← c(v, w) + g(v, w)c(w)
12                  g(v) ← g(v, w)g(w)
13                  T ← v; Q ← v                    // re-insert(v)
14              else                                // w ∈ T(v) and a cycle γ
15                  c(γ) = c(v, w) + g(v, w)(c(w) − c(v)g(w)/g(v))
16                  g(γ) = g(v, w)g(w)/g(v)
17                  if g(γ) < 1 then                // lossy or negative cycles
18                      c_cycle(v) ← min{c_cycle(v), c(γ)/(1 − g(γ))}
19                  else
20                      delete v and all nodes which reach v in G
```

After the first run all nodes of G with $v = nil$ are deleted. These nodes are unreachable; there is no path from v to the target t. The nodes with a computed cost of minus infinity are deleted too; more such nodes may be detected in the second run. For each remaining node v add an arc (v, t) with cost $\min\{c_{\text{cycle}}(v), c(v)\}$ and gain one. Moreover, if $c_{\text{cycle}}(v) < c(v)$ then label the arc *lossy*. Hence, the nodes are initialized with the computed cost of the first run. Then the Reverse-Generalized-Bellman-Ford algorithm is run on the so modified graph. It can be simplified to run in $O(nm)$, since there are no lossy cycles and the tests $g(\gamma) < 1$ in line 17 are false.

Theorem 1. *The Reverse-Generalized-Bellman-Ford algorithm solves SGSP from all nodes in* $O(nm \log n)$.

Proof. The correctness of the Reverse-Generalized-Bellman-Ford algorithm is proved as for Tarjan's variant of the Bellman-Ford algorithm [19, 24]. The number of arcs of a simple path from a node v to the target is used as the inductive parameter, and corresponds to the phases of the algorithm.

For each node v there is a path from v to the target t if and only if $v \neq nil$. Secondly, the nodes with minus infinite cost are detected in line 19 in either run of the algorithm. Finally, let v be a feasible node of G with $-\infty < c(v) < +\infty$, and let $\tau = (v_1, \ldots, v_r)$ be a cheapest path from v with minimal length.

If τ is a simple path to the target t of G, then the path is computed in the first run. Otherwise, $\tau = \sigma \circ \gamma$ consists of a simple path and a lossy cycle at a node x, as shown in Lemma 1. Then there is a node u of γ such that in the first run the lossy cycle is detected at u and $c_{\text{cycle}}(u) = c(\gamma)/(1 - g(\gamma))$ is the cost of the cycle γ at u. Using this value the second run skips lossy cycles and computes the costs of the cheapest paths from each feasible node. Again this is an inductive argument on the length.

Each run of the Reverse-Generalized-Bellman-Ford algorithm takes at most $n - 1$ phases. In each phase at most $O(m)$ arcs are visited and relaxed, and all but the subtree tests in line 9 take $O(1)$ time. In the first run, these tests are done in $O(\log(|T|))$, storing the pre- and postorder numbers of the nodes in 2-3-4 trees. In the second run the costs are amortized $O(1)$ as in the standard Bellman-Ford algorithm in [19, 24].

Theorem 2. *It can be checked in $O(nm)$ time whether or not a generalized network has lossy cycles, and in lossless networks SGSP can be solved in $O(nm)$.*

For the single-source problem SGSP care must be taken that the Reverse-Generalized-Bellman-Ford algorithm visits all lossy cycles and all negative cycles that are reachable from the source. This can be achieved by connecting all nodes or representatives of the cycles to the target by new arcs with appropriately high costs. This pre-processing takes at most $O(nm)$ using the standard Bellman-Ford algorithm from the source and the distance functions $-\log g(a)$ and $\log g(a)$ for the detection of lossy and negative cycles, respectively.

Theorem 3. *The single-source generalized shortest path problem SGSP can be solved in $O(nm \log n)$ and in $O(nm)$ in lossless networks.*

Standard shortest path problems with nonnegative arc lenghts can be solved by Dijkstra's algorithm in $O(n \log n + m)$, see [7]. This can be extended to monotone instances of the generalized shortest path problem with nonnegative costs and gainy arcs and running Dijkstra's algorithm backwards from the target with the distance function $c(v, w) + g(v, w)c(w)$.

Theorem 4. *In generalized networks with nonnegative costs and gainy arcs SGSP can be solved in $O(n \log n + m)$.*

To summarize, the solution of the generalized shortest path problem SGSP is only a $O(\log n)$ factor slower than the solution of the standard shortest paths problem by the Bellman-Ford algorithm. Our algorithm improves the previously known algorithms at least by a factor of $O(n)$. This improvement is a major (and probably the final) step in the long history of this problem.

5 Single-Pair Generalized Shortest Paths

The cheapest path from a node ends at the target or at a lossy cycle, where the flow is consumed. The single-pair generalized shortest path problem SPGSP excludes cycles and all flow reaches the target. Recall that the amount of flow may increase or decrease due to the gains of the used arcs. The exclusion of cycles completely changes the character and the complexity of generalized shortest path problems and relates them to the Hamilton path problem.

Theorem 5. *The single-pair generalized shortest path problem from the source is NP-hard. It remains NP-hard, even (1) if the costs are nonnegative and all arcs are uniformly lossy, e.g., $g(a) = 0.5$, or (2) if the costs are negative and all arcs are purely lossy (purely gainy).*

Proof. We reduce the directed Hamilton path problem. Let G be an instance of the Hamilton path problem with distinguished nodes s and t for the source and the target. For (1) let all arcs have unit cost and gain 0.5. Add a new target $\widehat{t}$ and an arc $(t, \widehat{t})$ with cost 2^{n+1} and gain 0.5. Then G has a Hamilton path from

s to t if and only if $c(s) = 2 - 1/2^{n-1} + 4$ is the cost of the cheapest path from s to $\widehat{t}$ excluding lossy cycles. Notice that each lossy cycle has cost two. For (2) let $c(a) = -1$ and $g(a) = g$ for some fixed $g > 0$ for each arc a of G. Then a Hamilton path induces minimal cost.

6 Conclusion

We have considered generalized shortest path problems with two parameters on the arcs. Surprisingly, if the gains multiply and accumulate progressively, the generalized version can be solved almost efficiently as the standard shortest path problems, if lossy cyles are allowed. However, the single-pair problem is NP-hard.

There is a related version of generalized shortest path and network flow problems, which has been discarded so far: additive gains. Additive gains describe a fixed servic charge for using an arc. This has lots of applications. Nevertheless we could not find any reference to algorithmic solutions of this problem. The additive generalized shortest path problem is not continuous, and thus not directly solvable by LP techniques. Its complexity is open. However, if the cost and gain of the arcs are non-negative and the gains are integral then there is a pseudo-polynomial time solution by a dynamic programming approach.

Acknowledgement

I wish to thank Arunabha Sen for bringing the problem of generalized paths to my attention and pointing out the application in communication networks, and to Vladimir Batagelj and Patrice Ossona de Mendez for the "backwards" strategy, and to Christian Bachmaier, Michael Forster, Andreas Pick, Marcus Raitner and Katja Strecker for intensive discussions.

References

1. I. Adler and S. Cosares, *A strongly polynomial algorithm for a special class of linear programs,* Oper. Res. **39** (1991), 955–960.
2. R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Network Flows,* Prentice Hall, Englewood Cliffs, 1993.
3. B.V. Cherkassky and A.V. Goldberg, *Negative-cycle detection algorithms,* Math. Program. **85** (1999), 277 – 311.
4. B.V. Cherkassky, A.V. Goldberg and T. Radzik, *Shortest paths algorithms: theory and experimental evaluation,* Math. Program. **73** (1996), 129 – 174.
5. A. Charnes and W.M. Raike, *One-pass algorithms for some generalized network problems,* Oper. Res. **14** (1962), 914–924.
6. E. Cohen and N. Megiddo, *Improved algorithms for linear inequalities with two variables per inequality,* SIAM J. Comput. **23** (1994), 1313 – 1347.
7. T.H. Cormen, C. E. Leiserson and R.L. Rivest, *Introduction to Algorithms,* MIT Press, Cambridge, 1990.

8. G.B. Dantzig, *Linear Programming and Extensions,* Princeton Univ. Press, Princeton, NJ (1963).
9. L.K. Fleischer and K.D. Wayne, *Fast and simple approximation schemes for generalized flow* Math. Program. **91** (2002), 215 – 238.
10. L.R. Ford Jr. and D.R. Fulkerson, *Flows in Networks,* Princeton Univ. Press, Princeton, NJ (1962).
11. M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W.H. Freeman, San Francisco, 1979.
12. A. V. Goldberg, S. A. Plotkin and E. Tardos, *Combinatorial Algorithms for the generalized circulation problem,* Math. Oper. Res., **16** (1991), 351 – 381.
13. D.S. Hochbaum and J. Naor, *Simple and fast algorithms for linear and integer programs with two variables per inequality,* SIAM J. Comput. **23** (1994), 1179 – 1192.
14. L.V. Kantorovich, *Mathematical methods of organizing and planning production,* Publication House of the Leningrad State University, (1939), 68. Translated in Management Science, **6** (1960), 366 – 422.
15. N. Karmarkar, *A New Polynomial-Time Algorithm for Linear Programming,* Combinatorica **4** (1984), 373 – 395.
16. W.S. Jewell, *Optimal flow trough networks with gains,* Oper. Res. **10** (1962), 476 – 499.
17. L. G. Khachian, *Polynomial Algorithms in Linear Programming,* Zhurnal Vychislitelnoi Matematiki i Matematicheskoi Fiziki **20** (1980), 53 – 72.
18. E.L. Lawler, *Combinatorial Optimization: Networks and Matroids* Holt, Rinehard, and Winston, New York, (1976)
19. K. Mehlhorn and S. Näher, *LEDA A Platform for Combinatorial and Geometric Computing* Cambridge University Press, Cambridge, (1999).
20. M. Nykänen and E. Ukkonen, *The exact path length problem,* J. Algorithms, **42** (2002), 41 – 53.
21. J.D. Oldham, *Combinatorial approximation algorithms for generalized flow problems,* Proc. Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, (1999), 704 – 714.
22. J.D. Oldham, *Combinatorial approximation algorithms for generalized flow problems,* J. Algorithms, **38** (2001), 135 – 168.
23. A. S. Tanenbaum, *Computer Networks,* Prentice Hall, Englewood Cliffs, 1996.
24. R.E. Tarjan, *Data Structures and Network Algorithms* Society for Industrial and Applied Mathematics, Philadelphia, 1983.
25. K.D. Wayne and L. Fleischer, *Faster approximation algorithms for generalized flow,* Proc. of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, (1999), 981 – 982.

New Graph Classes of Bounded Clique-Width*

Andreas Brandstädt[1], Feodor F. Dragan[2],
Hoàng-Oanh Le[1], and Raffaele Mosca[3]

[1] Institut für Theoretische Informatik, Fachbereich Informatik,
Universität Rostock, 18051 Rostock, Germany
{ab,hoang-oanh.le}@informatik.uni-rostock.de
[2] Department of Computer Science, Kent State University, Kent, Ohio 44242, USA
dragan@cs.kent.edu
[3] Via Latina 7, Pescara 65121 Italy
ciufolini@tiscalinet.it

Abstract. Clique-width of graphs is a major new concept with respect to efficiency of graph algorithms; it is known that every algorithmic problem expressible in a certain kind of Monadic Second Order Logic called *LinEMSOL*($\tau_{1,L}$) by Courcelle, Makowsky and Rotics, is solvable in linear time on any graph class with bounded clique-width for which a k-expression for the input graph can be constructed in linear time. The concept of clique-width extends the one of treewidth since bounded treewidth implies bounded clique-width.
We give a complete classification of all graph classes defined by forbidden one-vertex extensions of the P_4 with respect to their clique-width. Our results extend and improve recently published structural and complexity results in a systematic way.

1 Introduction

Recently, in connection with graph grammars, Courcelle, Engelfriet and Rozenberg in [15] introduced the concept of clique-width of a graph which has attracted much attention due to the fact that, in [16], Courcelle, Makowsky and Rotics have shown that every graph problem definable in LinEMSOL($\tau_{1,L}$) (a variant of Monadic Second Order Logic) is linear-time solvable on graphs with bounded clique-width if a k-expression describing the input graph is given. The problems Vertex Cover, Maximum Weight Stable Set (MWS), Maximum Weight Clique, Steiner Tree and Domination are examples of LinEMSOL($\tau_{1,L}$) definable problems. Note that every class of bounded treewidth has bounded clique-width as well (see [17].

It is known that the class of P_4-free graphs (also called *cographs*) is exactly the class of graphs having clique-width at most 2, and a 2-expression can be found in linear time along the cotree of a cograph. Due to the basic importance of cographs, it is of interest to consider graph classes defined by forbidden one-vertex extensions of a P_4 - see Figure 1 - which are natural generalizations of

* Research of the first author partially supported by Kent State University, Kent, Ohio, research of the third and fourth author partially supported by German Research Community DFG Br 1446-4/1

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 57–67, 2002.

cographs. The aim of this paper is to investigate the structure and to classify the clique-width of all these graph classes in a systematic way. This is also motivated by known examples such as the (P_5,co-P_5,bull)-free graphs studied by Fouquet in [20] (see Theorem 16) and the (P_5,co-P_5,chair)-free graphs studied by Fouquet and Giakoumakis in [21] (see Theorem 13). Moreover, there are papers such as [29, 34] dealing with (chair,co-P,gem)-free graphs and [24] dealing with (P_5,P,gem)-free graphs where it is shown that the MWS problem can be solved in polynomial time on these classes. Our results imply bounded clique-width and linear time for the MWS problem and any other LinEMSOL($\tau_{1,L}$) definable problem for these classes as well as for many other examples. This also continues research done in [2–5, 8–11].

Due to space limitations, all proofs in this extended abstract are omitted; they are contained in the full version of the paper.

Throughout this paper, let $G = (V, E)$ be a finite undirected graph without self-loops and multiple edges and let $|V| = n$, $|E| = m$. The edges between two disjoint vertex sets X, Y form a *join*, denoted by ① (*co-join*, denoted by ⓪) if for all pairs $x \in X$, $y \in Y$, $xy \in E$ ($xy \notin E$) holds. A vertex $z \in V$ *distinguishes* vertices $x, y \in V$ if $zx \in E$ and $zy \notin E$. A vertex set $M \subseteq V$ is a *module* if no vertex from $V \setminus M$ distinguishes two vertices from M, i.e., every vertex $v \in V \setminus M$ has either a join or a co-join to M. A module is *trivial* if it is either the empty set, a one-vertex set or the entire vertex set V. Nontrivial modules are called *homogeneous sets*. A graph is *prime* if it contains only trivial modules. The notion of modules plays a crucial role in the *modular* (or *substitution*) *decomposition* of graphs (and other discrete structures) which is of basic importance for the design of efficient algorithms - see e.g. [32] for modular decomposition of discrete structures and its algorithmic use.

For $U \subseteq V$ let $G(U)$ denote the subgraph of G induced by U. Throughout this paper, all subgraphs are understood to be induced subgraphs. A vertex set $U \subseteq V$ is *stable* (or *independent*) in G if the vertices in U are pairwise nonadjacent. Let co-$G = \overline{G} = (V, \overline{E})$ denote the complement graph of G. A vertex set $U \subseteq V$ is a *clique* in G if U is a stable set in $\overline{G}$.

For $k \geq 1$, let P_k denote a chordless path with k vertices and $k - 1$ edges, and for $k \geq 3$, let C_k denote a chordless cycle with k vertices and k edges. A *hole* is a C_k, $k \geq 5$. Note that the P_4 is the smallest nontrivial prime graph and the complement of a P_4 is a P_4 itself.

See Figure 1 for the definition of the chair, P, bull, gem and their complements. Note that the complement of a bull is a bull itself. The *diamond* is the $K_4 - e$, i.e., a four vertex clique minus one edge.

Let $\mathcal{F}$ denote a set of graphs. A graph G is *$\mathcal{F}$-free* if none of its induced subgraphs is in $\mathcal{F}$. There are many papers on the structure and algorithmic use of prime $\mathcal{F}$-free graphs for $\mathcal{F}$ being a set of P_4 extensions; see e.g. [20–22, 25, 26, 28, 2, 4, 3, 11]. A graph is a *split graph* if G is partitionable into a clique and a stable set. It is known [19] that G is a split graph if and only if it is a ($2K_2$,C_4,C_5)-free graph.

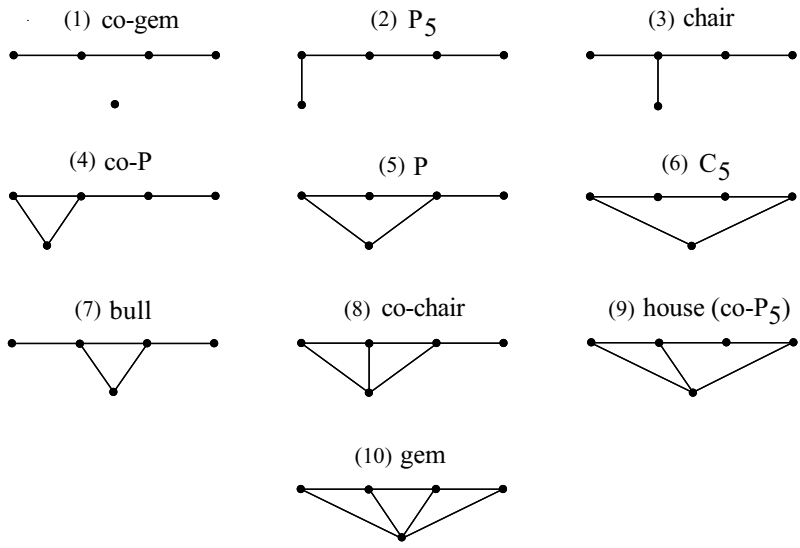


Fig. 1. All one-vertex extensions of a P_4

In what follows, we need the following classes of (prime) graphs:

- G is a *thin spider* if its vertex set is partitionable into a clique C and a stable set S with $|C| = |S|$ or $|C| = |S| + 1$ such that the edges between C and S are a matching and at most one vertex in C is not covered by the matching (an unmatched vertex is called the *head of the spider*).
- A graph is a *thick spider* if it is the complement of a thin spider.
- G is *matched co-bipartite* if its vertex set is partitionable into two cliques C_1, C_2 with $|C_1| = |C_2|$ or $|C_1| = |C_2| - 1$ such that the edges between C_1 and C_2 are a matching and at most one vertex in C_1 and C_2 is not covered by the matching.
- G is *co-matched bipartite* if G is the complement graph of a matched co-bipartite graph.
- A bipartite graph $B = (X, Y, E)$ is a *chain graph* [33] if for all vertices from X (Y), their neighborhoods in Y (X) are linearly ordered. If moreover, $|X| = |Y|$ and for all vertices from X (Y), their neighborhoods in Y (X) have size $1, 2, \ldots, |Y|$ $(1, 2, \ldots, |X|)$ then these graphs are prime.
- G is a *co-bipartite chain graph* if it is the complement of a bipartite chain graph.
- G is an *enhanced co-bipartite chain graph* if it is partitionable into a co-bipartite chain graph with cliques C_1, C_2 and three additional vertices a, b, c (a and c optional) such that $N(a) = C_1 \cup C_2$, $N(b) = C_1$, and $N(c) = C_2$, and there are no other edges in G.
- G is an *enhanced bipartite chain graph* if it is the complement of an enhanced co-bipartite chain graph.

2 Cographs, Clique-Width and Expressibility of Problems

The P_4-free graphs (also called *cographs*) play a fundamental role in graph decomposition; see [14] for linear time recognition of cographs, [12–14] for more information on P_4-free graphs and [7] for a survey on this graph class and related ones. For a cograph G, either G or its complement is disconnected, and the *cotree* of G expresses how the graph is recursively generated from single vertices by repeatedly applying join and co-join operations. The cotree representation allows to solve various NP-hard problems in linear time when restricted to cographs, among them the problems Maximum Weight Stable Set and Maximum Weight Clique. Note that the cographs are those graphs whose modular decomposition tree contains only join and co-join nodes as internal nodes.

Based on the following operations on vertex-labeled graphs, namely

- creation of a vertex labeled by integer l,
- disjoint union (i.e., co-join),
- join between all vertices with label i and all vertices with label j for $i \neq j$, and
- relabeling vertices of label i by label j,

the notion of *clique-width* $cwd(G)$ of a graph G is defined in [15] as the minimum number of labels which are necessary to generate G by using these operations. Cographs are exactly the graphs whose clique-width is at most two.

A *k-expression* for a graph G of clique-width k describes the recursive generation of G by repeatedly applying these operations using at most k different labels.

Proposition 1 ([16, 17]) *The clique-width of a graph G is the maximum of the clique-width of its prime subgraphs, and the clique-width of the complement graph $\overline{G}$ is at most twice the clique-width of G.*

Recently, the concept of clique-width of a graph attracted much attention since it gives a unified approach to the efficient solution of many algorithmic graph problems on graph classes of bounded clique-width via the expressibility of the problems in terms of logical expressions.

In [16], it is shown that every problem definable in a certain kind of Monadic Second Order Logic, called *LinEMSOL($\tau_{1,L}$)* in [16], is linear-time solvable on any graph class with bounded clique-width for which a k-expression can be constructed in linear time.

Hereby, in [16], it is mentioned that, roughly speaking, MSOL(τ_1) is Monadic Second Order Logic with quantification over subsets of vertices but not of edges; MSOL($\tau_{1,L}$) is the restriction of MSOL(τ_1) with the addition of labels added to the vertices, and LinEMSOL($\tau_{1,L}$) is the restriction of MSOL($\tau_{1,L}$) which allows to search for sets of vertices which are optimal with respect to some linear evaluation functions.

The problems Vertex Cover, Maximum Weight Stable Set, Maximum Weight Clique, Steiner Tree and Domination are examples of LinEMSOL($\tau_{1,L}$) definable problems.

Theorem 1 ([16]) *Let $\mathcal{C}$ be a class of graphs of clique-width at most k such that there is an $\mathcal{O}(f(|E|,|V|))$ algorithm, which for each graph G in $\mathcal{C}$, constructs a k-expression defining it. Then for every LinEMSOL($\tau_{1,L}$) problem on $\mathcal{C}$, there is an algorithm solving this problem in time $\mathcal{O}(f(|E|,|V|))$.*

As an application, it was shown in [16] that P_4-sparse graphs and some variants of them have bounded clique-width. Hereby, a graph is P_4-*sparse* if no set of five vertices in G induces at least two distinct P_4's [25, 26]. From the definition, it is obvious that a graph is P_4-sparse if and only if it contains no C_5, P_5, $\overline{P_5}$, P, $\overline{P}$, chair, co-chair (see Figure 1). See [11] for a systematic investigation of superclasses of P_4-sparse graphs.

In [25], it was shown that the prime P_4-sparse graphs are the spiders (which were called *turtles* in [25]), and according to Proposition 1 and the fact that the clique-width of thin spiders is at most 4 (which is easy to see), it follows that P_4-sparse graphs have bounded clique-width.

Recently, variants of P_4-sparse graphs attracted much attention because of their applications in areas such as scheduling, clustering and computational semantics. Moreover, all these classes are natural generalizations of cographs.

It is straightforward to see that the clique-width of matched co-bipartite (co-matched bipartite) graphs, bipartite chain (co-bipartite chain) graphs as well as the clique-width of induced paths and cycles is at most 4, and corresponding k-expressions can be determined in linear time. Distance-hereditary graphs are the (house,hole,domino,gem)-free graphs - see [1, 7]. In [23], Golumbic and Rotics have shown that their clique width is at most 3 and corresponding k-expressions can be determined in linear time.

3 Further Tools

Lemma 1 ([27]) *If a prime graph contains an induced C_4 (induced $2K_2$) then it contains an induced co-P_5 or A or domino (induced P_5 or co-A or co-domino).*

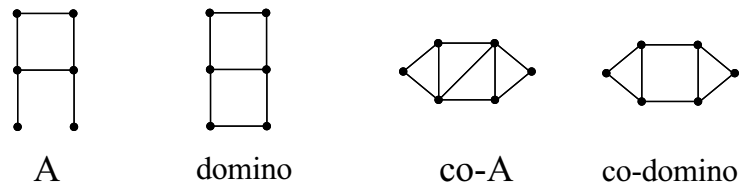


Fig. 2. The A and domino and their complements

The proof of Lemma 1 can be extended in a straightforward way to the case of a diamond instead of a C_4. For this purpose let us call *d-A* the graph resulting from an A graph by adding an additional diagonal edge in the C_4, and *d-domino*

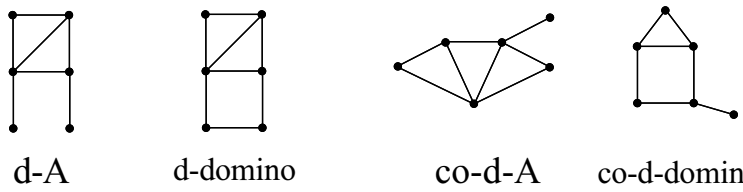


Fig. 3. The *d-A* and *d*-domino and their complements

the graph resulting from a domino graph by adding an additional diagonal edge in one of the C_4's - see Figure 3.

Lemma 2 *If a prime graph contains an induced diamond (co-diamond) then it contains an induced gem or d-A or d-domino (co-gem or co-d-A or co-d-domino).*

Theorem 2 ([2]) *Prime (P_5,diamond)-free graphs are either matched co-bipartite or a thin spider or an enhanced bipartite chain graph or have at most 9 vertices.*

For a structure description of (P_5,gem)-free graphs see [4] where the following Lemma is shown:

Lemma 3 ([4]) *Prime (P_5,gem)-free graphs containing a co-domino are matched co-bipartite.*

Theorem 3 ([9]) *If G is a prime (diamond,co-diamond)-free graph then G or $\overline{G}$ is either a matched co-bipartite graph or G has at most 9 vertices.*

Lemma 4 ([3]) *Prime chair-free bipartite graphs are co-matched bipartite, a path, or a cycle.*

Lemma 5 ([10]) *Prime chair-free split graphs are spiders.*

Lemma 6 ([3, 18]) *Prime (bull, chair)-free graphs containing a co-diamond are either co-matched bipartite or a cycle or a path.*

Lemma 7 *Prime co-gem-free bipartite graphs are co-matched bipartite.*

Lemma 8 *Prime (co-diamond,gem)-free graphs containing a diamond have at most 11 vertices.*

4 Structure and Clique-Width Results

Figure 4 contains all combinations of three forbidden P_4 extensions (enumerated according to Figure 1). Each class together with its complement class occurs only once; we take the lexicographically smaller class; for example, the (P_5,co-P_5,gem)-free graphs are the (2,9,10)-free graphs, and its complement class is the class of (P_5,co-P_5,co-gem)-free graphs, i.e., the (1,2,9)-free graphs; in Figure 4, only the class (1,2,9) occurs.

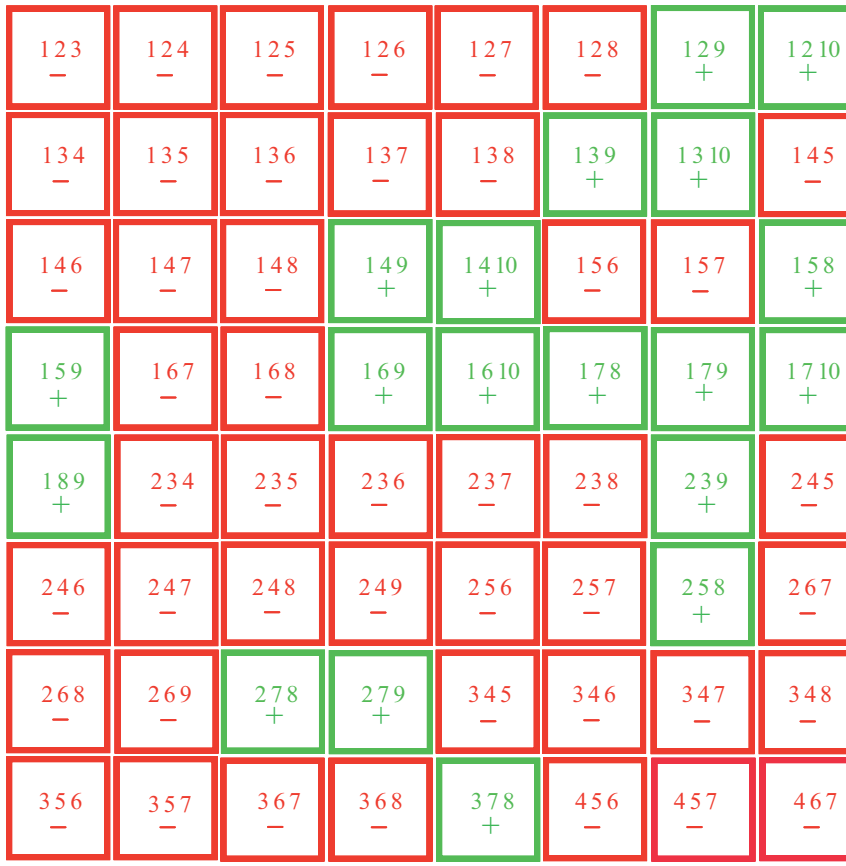


Fig. 4. All combinations of three forbidden P_4 extensions; + (−) denotes bounded (unbounded) clique-width

Theorem 4 ((1,2,9)) *If G is a prime (P_5,co-P_5,gem)-free graph then G is distance-hereditary or a C_5.*

The subsequent Theorems 5 and 6 are a simple consequence of Lemma 2.

Theorem 5 ((1,2,10),(1,4,10)) *If G is a prime (P_5,gem,co-gem)-free or (gem, co-P, co-gem)-free graph then G is (diamond,co-diamond)-free.*

Theorem 6 ((1,3,9),(1,7,9),(1,8,9)) *If G is a prime (P_5,gem,co-chair)-free or (P_5,gem,bull)-free or (P_5,gem,chair)-free graph then G is (P_5,diamond)-free.*

Thus, the structure of the classes considered in Theorems 5 and 6 is described in Theorems 2, 3 respectively.

Theorem 7 ((1,3,10)) *If G is a prime (co-gem,chair,gem)-free graph then G or $\overline{G}$ is a matched co-bipartite graph or G has at most 11 vertices.*

Theorem 8 ((1,4,9)) *If G is a prime (P_5,P,gem)-free graph then G is a matched co-bipartite graph or a distance-hereditary graph or a C_5 .*

Theorem 9 ((1,5,8), [8]) *If G is a prime (chair, co-P, gem)-free graph then G fulfills one of the following conditions:*

(i) *G is an induced path P_k, $k \geq 4$, or an induced cycle C_k, $k \geq 5$;*
(ii) *G is a thin spider;*
(iii) *G is a co-matched bipartite graph;*
(iv) *G has at most 11 vertices.*

The next theorem is partially based on the structure of (P_5, gem)-free graphs described in [4]:

Theorem 10 ([5]) *The clique width of (P_5, gem)-free graphs is at most 9.*

Thus, according to Theorem 10, the classes (1,5,9) and (1,6,9) have bounded clique-width as well; however, their structure is more complicated than the previous examples and we do not know any linear time algorithm for determining k-expressions for these graphs.

Theorem 11 ([6]) *The clique-width of (gem,co-gem)-free graphs is at most 24.*

The proof of Theorem 11 is technically very involved and does not give any simpler structure description for the subclasses (1,6,10), (1,7,10), (1,6,7,10); we do not know any linear time algorithm for determining k-expressions for these graphs.

Theorem 12 ((1,7,8)) *If G is a prime (bull,chair,gem)-free graph then G fulfills one of the following conditions:*

(i) *G or $\overline{G}$ is an induced path P_k, $k \geq 4$, or an induced cycle C_k, $k \geq 5$;*
(ii) *G or $\overline{G}$ is a co-matched bipartite graph;*
(iii) *G has at most 11 vertices.*

Theorem 13 ((2,3,9), [21]) *If G is a prime (P_5,$\overline{P_5}$,chair)-free graph then G is either a co-bipartite chain graph or a spider or C_5.*

Theorem 14 ((2,5,8), [11]) *If G is a prime (chair,co-P,house)-free graph then G fulfills one of the following conditions:*

(i) *G is an induced path P_k, $k \geq 4$, or an induced cycle C_k, $k \geq 5$;*
(ii) *G is a co-matched bipartite graph;*
(iii) *G is a spider.*

Theorem 15 ((2,7,8), [3]) *If G is a prime ($\overline{P_5}$,bull,chair)-free graph then G is either a co-matched bipartite graph or an induced path or cycle or $\overline{G}$ is (P_5,diamond)-free.*

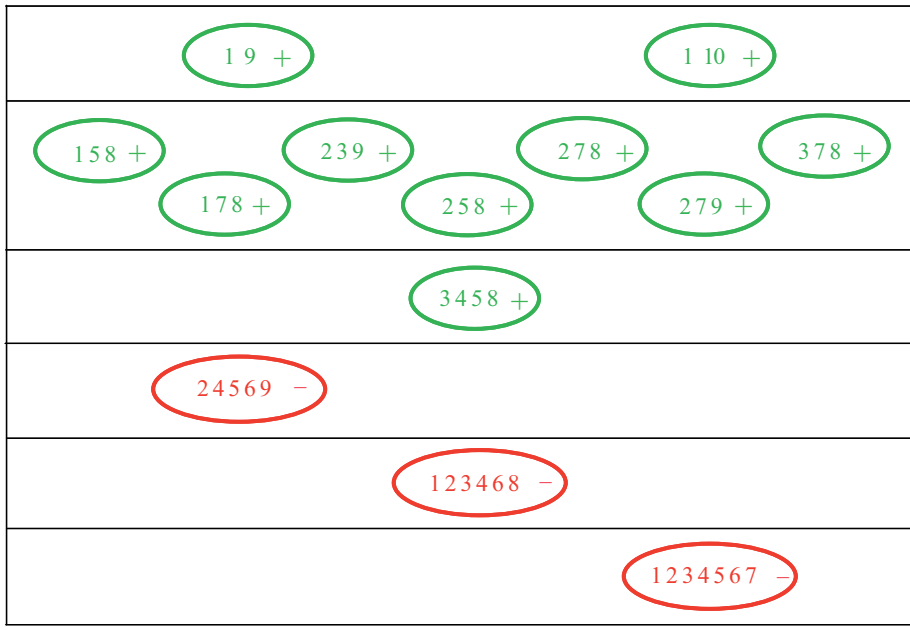


Fig. 5. Essential classes for all combinations of forbidden 1-vertex P_4 extensions; + (−) denotes bounded (unbounded) clique-width

Theorem 16 ((2,7,9), [20]) *If G is a prime (P_5,$\overline{P_5}$,bull)-free graph then G or $\overline{G}$ is a bipartite chain graph or a C_5.*

Theorem 17 ((3,7,8), [3]) *If G is a prime (bull,chair,co-chair)-free graph then G or $\overline{G}$ is either a co-matched bipartite graph or an induced path or cycle.*

Corollary 1 *Every LinEMSOL($\tau_{1,L}$) definable problem is solvable in linear time on all graph classes of bounded clique-width (i.e. not indicated with − in Figure 4), except the classes (1,5,9), (1,6,9), (1,6,10), (1,7,10), (1,6,7,10).*

Makowsky and Rotics have shown in [31] that the following grid types have unbounded clique-width:

- the F_n grid (whose complements are (1,2,3,4,6,8)-free);
- the $H_{n,q}$ grid (whose complements are (1,2,3,4,5,6,7)-free)

Moreover, they show that split graphs have unbounded clique width. This implies unbounded clique-width for all classes with − in Figure 4 and in Figure 5.

Let $\mathcal{F}$ denote the 10 one-vertex extensions of the P_4 (see Figure 1). For $\mathcal{F}' \subseteq \mathcal{F}$, there are 1024 classes of $\mathcal{F}'$-free graphs. Figure 5 shows all inclusion-minimal classes of unbounded clique-width and all inclusion-maximal classes of bounded clique-width. As before, we consider the class of $\mathcal{F}'$-free graphs together with its complement class, the co-$\mathcal{F}'$-free graphs, and mention only the lexicographically smaller class. Note that any subclass of bounded clique-width has

bounded clique-width as well, whereas any superclass of unbounded clique-width has unbounded clique-width as well. Obviously, a graph with at least 5 vertices is a cograph if and only if it contains none of the 10 possible one-vertex extensions of a P_4. For $|\mathcal{F}'| \in \{9, 8\}$, all these classes have bounded clique-width. For $|\mathcal{F}'| = 7$ there is exactly one inclusion-minimal class (together with its complement class) of unbounded clique-width namely (1,2,3,4,5,6,7) (enumeration with respect to Figure 1), and similarly for $|\mathcal{F}'| = 6$ and $|\mathcal{F}'| = 5$. For $|\mathcal{F}'| = 4$ there is exactly one inclusion-maximal class of bounded clique-width namely (3,4,5,8). For $|\mathcal{F}'| = 3$, the inclusion-maximal classes of bounded clique-width are (1,5,8), (1,7,8), (2,3,9), (2,5,8), (2,7,8), (2,7,9), (3,7,8). For $|\mathcal{F}'| = 2$, the only classes of bounded clique-width are (1,9) and (1,10), and for $|\mathcal{F}'| = 1$, all classes have unbounded clique-width.

Open problem.

1. Is there a linear time algorithm for determining a k-expression with constant k for the classes (1,9), (1,5,9), (1,6,9), (1,10), (1,6,10), (1,7,10), (1,6,7,10)?

Acknowledgement

The authors thank Van Bang Le for helpful discussions.

References

1. H.-J. Bandelt, H.M. Mulder, Distance-hereditary graphs, *J. Combin. Theory (B)* 41 (1986) 182-208
2. A. Brandstädt, (P_5,diamond)-Free Graphs Revisited: Structure, Bounded clique-width and Linear Time Optimization, *Manuscript* 2000; accepted for *Discrete Applied Math.*
3. A. Brandstädt, C.T. Hoàng, V.B. Le, Stability Number of Bull- and Chair-Free Graphs Revisited, *Manuscript* 2001; accepted for *Discrete Applied Math.*
4. A. Brandstädt, D. Kratsch, On the structure of (P_5,gem)-free graphs, *Manuscript* 2001
5. A. Brandstädt, H.-O. Le, R. Mosca, Chordal co-gem-free graphs have bounded clique-width, *Manuscript* 2002
6. A. Brandstädt, H.-O. Le, R. Mosca, (Gem,co-gem)-free graphs have bounded clique-width, *Manuscript* 2002
7. A. Brandstädt, V.B. Le, J. Spinrad, Graph Classes: A Survey, *SIAM Monographs on Discrete Math. Appl., Vol.* 3, SIAM, Philadelphia (1999)
8. A. Brandstädt, H.-O. Le, J.-M. Vanherpe, Structure and Stability Number of (Chair, Co-P, Gem)-Free Graphs, *Manuscript* 2001
9. A. Brandstädt, S. Mahfud, Linear time for Maximum Weight Stable Set on (claw,co-claw)-free graphs and similar graph classes, *Manuscript* 2001; to appear in *Information Processing Letters*
10. A. Brandstädt, R. Mosca, On the Structure and Stability Number of P_5- and Co-Chair-Free Graphs, *Manuscript* 2001; accepted for *Discrete Applied Math.*
11. A. Brandstädt, R. Mosca, On Variations of P_4-Sparse Graphs, *Manuscript* 2001

12. D.G. Corneil, H. Lerchs, L. Stewart-Burlingham, Complement reducible graphs, *Discrete Applied Math.* 3 (1981) 163-174
13. D.G. Corneil, Y. Perl, L.K. Stewart, Cographs: recognition, applications, and algorithms, *Congressus Numer.* 43 (1984) 249-258
14. D.G. Corneil, Y. Perl, L.K. Stewart, A linear recognition algorithm for cographs, *SIAM J. Computing* 14 (1985) 926-934
15. B. Courcelle, J. Engelfriet, G. Rozenberg, Handle-rewriting hypergraph grammars, *J. Comput. Syst. Sciences*, 46 (1993) 218-270
16. B. Courcelle, J.A. Makowsky, U. Rotics, Linear time solvable optimization problems on graphs of bounded clique width, extended abstract in: *Conf. Proc. WG'98*, LNCS 1517 (1998) 1-16; *Theory of Computing Systems* 33 (2000) 125-150
17. B. Courcelle, S. Olariu, Upper bounds to the clique-width of graphs, *Discrete Appl. Math.* 101 (2000) 77-114
18. C. De Simone, On the vertex packing problem, *Graphs and Combinatorics* 9 (1993) 19-30
19. S. Földes, P.L. Hammer, Split graphs, *Congress. Numer.* 19 (1977), 311-315
20. J.-L. Fouquet, A decomposition for a class of $(P_5, \overline{P_5})$-free graphs, *Discrete Math.* 121 (1993) 75-83
21. J.-L. Fouquet, V. Giakoumakis On semi-P_4-sparse graphs, *Discrete Math.* 165-166 (1997) 267-290
22. J.-L. Fouquet, V. Giakoumakis, H. Thuillier, F. Maire, On graphs without P_5 and $\overline{P_5}$, *Discrete Math.* 146 (1995) 33-44
23. M.C. Golumbic, U. Rotics, On the clique-width of some perfect graph classes, *Int. Journal of Foundations of Computer Science* 11 (2000) 423-443
24. A. Hertz, On a graph transformation which preserves the stability number, *Yugoslav Journal of Oper. Res.*, to appear,
25. C.T. Hoàng, *A Class of Perfect Graphs*, *Ms. Sc. Thesis*, School of Computer Science, McGill University, Montreal (1983)
26. C.T. Hoàng, *Perfect Graphs*, *Ph. D. Thesis*, School of Computer Science, McGill University, Montreal (1985)
27. C.T. Hoàng, B. Reed, Some classes of perfectly orderable graphs, *J. Graph Theory* 13 (1989) 445-463
28. B. Jamison, S. Olariu, A unique tree representation for P_4-sparse graphs, *Discrete Appl. Math.* 35 (1992), 115-129
29. V.V. Lozin, Conic reduction of graphs for the stable set problem, *Discrete Math.* 222 (2000) 199-211
30. N.V.R. Mahadev, U.N. Peled, Threshold Graphs and Related Topics, *Annals of Discrete Mathematics* 56 (1995)
31. J.A. Makowsky, U. Rotics, On the clique-width of graphs with few P_4's, *Int. J. of Foundations of Computer Science* 3 (1999) 329-348
32. R.H. Möhring, F.J. Radermacher, Substitution decomposition for discrete structures and connections with combinatorial optimization, *Annals of Discrete Math.* 19 (1984) 257-356
33. M. Yannakakis, The complexity of the partial order dimension problem, *SIAM J. Algebraic and Discrete Methods* 3 (1982) 351-358
34. I.E. Zverovich, I.I. Zverovich, Extended $(P_5,\overline{P_5})$-free graphs, *Rutcor Research Report* RRR 22-2001 (2001) `http://rutcor.rutgers.edu/~rrr`

More about Subcolorings
(Extended Abstract)*

Hajo Broersma[1], Fedor V. Fomin[2],
Jaroslav Nešetřil[3], and Gerhard J. Woeginger[1]

[1] Faculty of Mathematical Sciences, University of Twente,
7500 AE Enschede, The Netherlands
{broersma,g.j.woeginger}@math.utwente.nl
[2] Heinz Nixdorf Institute and Paderborn University,
Fürstenallee 11, D-33102 Paderborn, Germany
fomin@uni-paderborn.de
[3] Department of Applied Mathematics and
Institute of Theoretical Computer Science (ITI),
Faculty of Mathematics and Physics, Charles University,
118 00 Prague, Czech Republic
nesetril@kam.ms.mff.cuni.cz

Abstract. A subcoloring is a vertex coloring of a graph in which every color class induces a disjoint union of cliques. We derive a number of results on the combinatorics, the algorithmics, and the complexity of subcolorings.
On the negative side, we prove that 2-subcoloring is NP-hard for comparability graphs, and that 3-subcoloring is NP-hard for AT-free graphs and for complements of planar graphs. On the positive side, we derive polynomial time algorithms for 2-subcoloring of complements of planar graphs, and for r-subcoloring of interval and of permutation graphs. Moreover, we prove asymptotically best possible upper bounds on the subchromatic number of interval graphs, chordal graphs, and permutation graphs in terms of the number of vertices.

Keywords: graph coloring; subcoloring; special graph classes; polynomial time algorithm; computational complexity.

1 Introduction

We denote by $G = (V, E)$ a finite undirected and simple graph. The *complement* $\overline{G}$ of $G = (V, E)$ is the graph on V with edge set $\overline{E}$ such that $\{u, v\} \in \overline{E}$ if and

* The work of HJB and FVF is sponsored by NWO-grant 047.008.006. Part of the work was done while FVF was visiting the University of Twente, and while he was a visiting postdoc at DIMATIA-ITI (supported by GAČR 201/99/0242 and by the Ministry of Education of the Czech Republic as project LN00A056). FVF acknowledges support by EC contract IST-1999-14186: Project ALCOM-FT (Algorithms and Complexity - Future Technologies). JN acknowledges support of ITI - the Project LN00A056 of the Czech Ministery of Education. GJW acknowledges support by the START program Y43-MAT of the Austrian Ministry of Science.

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 68–79, 2002.

only if $\{u, v\} \notin E$. For a set of graphs $\mathcal{G}$, we denote by $\overline{\mathcal{G}}$ the set of complements of graphs from $\mathcal{G}$; hence $G \in \mathcal{G}$ if and only if $\overline{G} \in \overline{\mathcal{G}}$. Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be graphs with $V_G \cap V_H = \emptyset$. The *disjoint union* $G \dot\cup H$ is the graph with vertex set $V_G \cup V_H$ and edge set $E_G \cup E_H$. The *join* $G \vee H$ of G and H is the graph with vertex set $V_G \cup V_H$ and edge set $E_G \cup E_H \cup \{\{u, v\} \colon u \in V_G, v \in V_H\}\}$. Finally, $G \bowtie H$ denotes the graph that results by adding a new vertex v to the disjoint union of G and H, and by joining v to all the vertices in G and H.

For every non-empty $W \subseteq V$, the subgraph of $G = (V, E)$ induced by W is denoted by $G[W]$. A *clique* C of a graph $G = (V, E)$ is a non-empty subset of V such that all the vertices of C are pairwise adjacent, i.e., $G[C]$ is a complete graph. The maximum size of a clique in G is denoted by $\omega(G)$. A subset of vertices $I \subseteq V$ is *independent* if no two of its elements are adjacent. An *r-coloring* of the vertices of a graph $G = (V, E)$ is a partition $V_1, V_2, \ldots, V_r$ of V; the r sets V_j are called the *color classes* of the r-coloring. An r-coloring is *proper* if every color class is an independent set. The *chromatic number* $\chi(G)$ is the minimum value r for which a proper r-coloring exists.

Evidently, an r-coloring is proper if and only if for every color class V_j the induced subgraph $G[V_j]$ is the union of complete graphs of cardinality one. This awkward reformulation leads to several interesting generalizations of the classical chromatic number.

- An r-coloring $V_1, V_2, \ldots, V_r$ is an *r-subcoloring*, if for every color class the induced subgraph $G[V_i]$ is the disjoint union of complete graphs (there is no restriction on the sizes of these complete graphs).
- An r-coloring is a *cocoloring*, if for every color class the induced subgraph $G[V_i]$ either is a clique or an independent set.
- Let F be some fixed graph. An r-coloring is an *F-free coloring*, if for every color class the induced subgraph $G[V_i]$ does not contain F as an induced subgraph.

The *subchromatic number* $\chi_{sub}(G)$, the *cochromatic number* $\chi_{co}(G)$, and the *F-free chromatic number* $\chi(F, G)$ of a graph G, is the smallest number r for which G has an r-subcoloring, an r-cocoloring, and an F-free r-coloring, respectively. Note that a coloring is a subcoloring if and only if it is a P_3-free coloring (where P_k denotes the path on k vertices).

In this paper, we study the algorithmic and combinatorial behavior of the subchromatic number on various classes of specially structured graphs. See the books of Brandstädt et al. [3] and Golumbic [11] for definitions of these graph classes.

1.1 Known Results

Finding proper colorings for various classes of perfect graphs is a long studied and well understood problem. We refer to the book [11] of Golumbic for a classical source on algorithmic aspects of perfect graphs. By a celebrated result of Grötschel, Lovász & Schrijver [12], the chromatic number of a perfect graph

can be computed in polynomial time. Simple and fast algorithms are known for different subclasses of perfect graphs like chordal graphs, comparability graphs, permutation graphs etc. However, even small steps away from proper coloring towards more general concepts like subcoloring and cocoloring increase the computational complexity of coloring enormously.

For instance, the cochromatic number is NP-hard to compute even for permutation graphs (Wagner [17]). Gimbel, Kratsch & Stewart [9] proved the NP-hardness of computing the cochromatic number for circle graphs and line graphs of comparability graphs, and they derived a polynomial time algorithm for chordal graphs. Achlioptas [1] proved that for any graph F with at least three vertices and for any fixed integer $r \geq 2$, the problem of deciding whether a given input graph has an F-free r-coloring is NP-hard. By putting $F = P_3$, we get that r-subcoloring is NP-hard for any fixed integer $r \geq 2$. Fiala, Jansen, Le & Seidel [8] strengthened this hardness result to input graphs that are triangle-free, planar, and have maximum vertex degree four. On the positive side, [8] gave polynomial time algorithms for the subcoloring problem on cographs and on graphs of bounded treewidth.

The literature also contains a number of results on P_4-free colorings: Gimbel & Nešetřil [10] showed that P_4-free r-coloring in $r = 2$ or $r = 3$ colors is NP-hard even for planar input graphs. Since P_4 is isomorphic to its complement $\overline{P}_4$, we conclude that P_4-free coloring in 2 or 3 colors is NP-hard for complements of planar graphs. Hoàng & Le [13] proved that P_4-free 2-coloring is NP-hard for comparability and cocomparability graphs.

Now let us list a number of useful combinatorial results from the literature on subcolorings and cocolorings.

Proposition 1. *For any graph G, $\chi_{sub}(G) \leq \chi_{co}(G) \leq \min\{\chi(G), \chi(\overline{G})\}$.*

Proposition 2. *(Mynhardt & Broere [16])*
Let $K_{m,m,\ldots,m}$ be the complete m-partite graph containing m classes of m vertices. Then $\chi_{sub}(K_{m,m,\ldots,m}) = \chi_{co}(K_{m,m,\ldots,m}) = \chi(K_{m,m,\ldots,m}) = m$.

Proposition 3. *(Albertson, Jamison, Hedetniemi & Locke [2])*
Let G and H be graphs with $\chi_{sub}(G) \geq k$ and $\chi_{sub}(H) \geq k$. Then $\chi_{sub}(G \bowtie H) \geq k + 1$.

1.2 Our Results

We study combinatorial, algorithmic and complexity aspects of the subcoloring problems. In particular, we derive the following results.

- For general n-vertex graphs the subchromatic number may be $\Theta(n/\log n)$; for perfect graphs, permutation graphs, and cographs, it may be $\Theta(\sqrt{n})$; for chordal graphs and interval graphs, it may be $\Theta(\log n)$. All these bounds are best possible up to constant factors. These results are proved in Section 2.

- For complements of planar graphs 2-subcoloring is polynomially solvable (Section 4) whereas 3-subcoloring is NP-hard (Section 3).
- For AT-free graphs, r-subcoloring is NP-hard for any fixed $r \geq 3$ (Section 3). For comparability graphs, r-subcoloring is NP-hard for any fixed $r \geq 2$ (Section 5.1).
- For interval graphs (Section 5.2) and for permutation graphs (Section 5.3) r-subcoloring is polynomially solvable for any fixed $r \geq 2$.

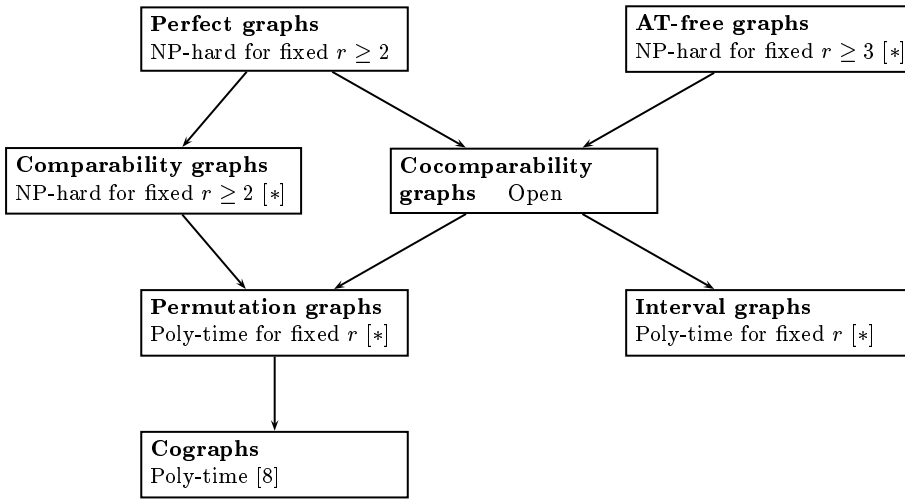


Fig. 1. Summary of some of our results on r-subcoloring for special graph classes, and the containment relations between these classes. [∗] denotes a contribution of this paper.

Figure 1 summarizes some of our results and illustrates the relations between some of the graph classes studied in this paper. The definitions of these graph classes can be found in books [3] and [11].

2 Upper and Lower Bound Results

In this section we derive several bounds on the subchromatic number of graphs in terms of their number of vertices. We first state two useful results from the literature.

Proposition 4. *(Albertson, Jamison, Hedetniemi & Locke [2])*
For any graph G on n vertices, $\chi_{sub}(G) \leq 2n/(\log_2 n - 2) + O(n/(\log_2 n)^2)$.

Proposition 5. *(Erdős, Gimbel & Kratsch [6])*
Every perfect *graph G on $\binom{k+2}{2} - 1$ vertices has cochromatic number at most k. Therefore, $\chi_{co}(G) \leq \lfloor \sqrt{2n + 1/4} - 1/2 \rfloor$.*

As our first result, we observe that up to constant factors the upper bound stated in Proposition 4 is best possible.

Lemma 1. *For every n, there exists a graph G on n vertices with $\chi_{sub}(G) \geq n/(2\log_2 n + 1)$.*

Proof. We slightly modify the famous argument of Erdős [5]. Consider the random graph on n vertices that contains every edge independently with probability $1/2$.

A subset X of $k = 2\log_2 n + 1$ vertices is called *good*, if it induces a disjoint union of cliques, and thus constitutes a feasible color class for a subcoloring. Let us estimate the probability that some fixed set X is good. Altogether, there are $2^{\binom{k}{2}}$ possibilities for the edges in X. Out of these exactly B_k are good, where B_k denotes the kth Bell number that is the number of ways a set of n elements can be partitioned into nonempty subsets. The crude upper bound $B_k \leq k!$ yields that the probability that X is good is at most $k!/2^{\binom{k}{2}}$.

Therefore, the expected total number of good subsets of cardinality k is at most $\binom{n}{k} \cdot k!/2^{\binom{k}{2}}$. Since $\binom{n}{k} \leq n^k/k!$ and since $k = 2\log_2 n + 1$, a straightforward calculation reveals that this expected number is strictly less than 1. Hence, there exists a graph G in the probability space that does not contain any good subset. In any subcoloring of G all color classes contain fewer than k vertices, and thus $\chi_{sub}(G) > n/k$. □

For perfect graphs, the subchromatic number is much smaller than $n/\log n$: Propositions 5 and 1 yield that for every perfect graph G on n vertices, $\chi_{sub}(G) \leq \chi_{co}(G) \leq \lfloor\sqrt{2n + 1/4} - 1/2\rfloor$. Erdős, Gimbel & Kratsch [6] observed that the disjoint union of cliques $H = K_1 \dot\cup K_2 \dot\cup \cdots \dot\cup K_k$ (this is a graph on $\binom{k+1}{2}$ vertices) has cochromatic number k. In every subcoloring of $\overline{H}$ every color class is either a clique, or an independent set. Thus $\chi_{sub}(\overline{H}) = \chi_{co}(\overline{H}) = \chi_{co}(H) = k$. Since $\overline{H}$ is a perfect graph (in fact, it is even a cograph and a permutation graph), we get that up to additive constant the bound $\sqrt{2n}$ is the best possible upper bound for the subchromatic number of perfect graphs.

In the rest of this section, we will discuss interval graphs. We will show that for interval graphs the subchromatic number is bounded by $O(\log n)$.

Lemma 2. *For every interval graph G on n vertices, $\chi_{sub}(G) \leq \lfloor\log_2(n+1)\rfloor$. This bound is best possible.*

Proof. For the upper bound we use induction on n. The statement is clearly true for $n = 1$. Consider an interval representation of an interval graph G on n vertices; without loss of generality we assume that the left endpoints of the intervals are the integers $1, 2, \ldots, n$. If n is odd, we take an arbitrary maximal clique C that contains the interval with left endpoint $(n+1)/2$. Then every component of $G - C$ contains at most $(n-1)/2$ vertices. We color C by one color, and we use $\lfloor\log_2((n+1)/2)\rfloor$ additional colors to color all these components inductively. If n is even, a similar analysis goes through.

For showing that the bound $\lfloor \log_2(n+1) \rfloor$ is best possible, we consider the following graphs G_i: For $k = 1$, the graph G_1 consists of one vertex. For $k > 1$, we set $G_k = G_{k-1} \bowtie G_{k-1}$ where the new vertex is called v. Note that G_k has $2^k - 1$ vertices, and that by Proposition 3 $\chi_{sub}(G_k) = k$. Moreover, G_k is an interval graph; its interval representation can be obtained by putting two disjoint interval representations of G_{k-1} next to each other, and by adding one long interval that corresponds to the vertex v. □

We mention without proof that a similar inductive argument yields $\chi_{sub}(G) = O(\log n)$ for any *chordal* graph G. Albertson, Jamison, Hedetniemi & Locke [2] observed that the interval graphs G_k in the proof of Lemma 2 form a class of interval graphs with unbounded subchromatic number. We now present a stronger result on the coloring of interval graphs with forbidden subgraphs.

Lemma 3. *For any m and r, there exists an interval graph $\mathcal{I}(m, r)$ that does not have a P_m-free r-coloring.*

Proof. Let $N = R(K_{m+1}; r)$ be the Ramsey number that specifies the smallest number of vertices in a complete graph such that every r-coloring of the edges of this graph induces a monochromatic clique K_{m+1} (that is, a clique in which all edges have the same color). Let K_N be the complete graph on vertex set $\{1, 2, \ldots, N\}$. Let $\mathcal{I}_N$ be the intersection graph of all closed intervals with integer endpoints from $\{1, 2, \ldots, N\}$. Every interval $[a, b]$ in $\mathcal{I}_N$ naturally corresponds to the edge $\{a, b\}$ in K_N.

Now consider an arbitrary r-coloring of the intervals in $\mathcal{I}_N$. This induces a corresponding r-coloring of the edges in K_N, and hence there exists a monochromatic clique K_{m+1} with vertex set X with $|X| = m + 1$. In $\mathcal{I}_N$, the intervals with both endpoints in X also form a monochromatic set $\mathcal{I}_X$. Since $\mathcal{I}_X$ contains an induced path P_m, every r-coloring of $\mathcal{I}_N$ contains an induced monochromatic path P_m. □

3 Negative Results: AT-Free Graphs

In this section we derive a generic NP-hardness result. As corollaries to this result, we derive the NP-hardness of 3-subcoloring for graphs with independence number two (and hence for AT-free graphs).

For an integer $p \geq 1$ and a graph G, we denote by pG the disjoint union of p copies of G.

We omit the proof of the following lemma in the extended abstract.

Lemma 4. *For any graph G and for any integer $p \geq \chi(G)$, the chromatic number $\chi(G)$ of G coincides with the subchromatic number $\chi_{sub}(\overline{pG})$ of the graph $\overline{pG}$.*

The following theorem is the main result of this section. It is an immediate consequence of Lemma 4.

Theorem 1. *Let $\mathcal{G}$ be a graph class that is closed under taking disjoint unions (that is, $G, H \in \mathcal{G}$ implies $G \dot{\cup} H \in \mathcal{G}$). Let r be an integer.*

If the proper r-coloring problem is NP-hard for graphs from $\mathcal{G}$, then the r-subcoloring problem is NP-hard for graphs from $\overline{\mathcal{G}}$.

Corollary 1. *The 3-subcoloring problem is NP-hard even when restricted to*

(a) graphs with independence number at most two,
(b) AT-free graphs,
(c) complements of planar graphs.

Proof. Maffray & Preissmann [15] proved that proper 3-coloring is NP-hard even for triangle-free graphs. The class of triangle-free graphs is closed under taking disjoint unions, and a graph is triangle-free if and only if its complement has independence number at most two. With this, (a) follows from Theorem 1.

Since the graphs with independence number at most two form a subclass of the AT-free graphs, (b) is a consequence of (a). Finally, the class of planar graphs is closed under taking disjoint unions, and it is well-known that proper 3-coloring of planar graphs is an NP-hard problem. Thus, Theorem 1 implies (c). □

We conclude this section with some consequences of Lemma 4 on the hardness of approximation of the subcoloring problem for general graphs. We rely on the results of Feige & Kilian [7] on the hardness of approximating the chromatic number of a graph: For any $\varepsilon > 0$, the chromatic number of n-vertex graphs cannot be approximated within a factor of $n^{1-\varepsilon}$, unless NP $\subseteq$ ZPP.

Corollary 2. *For any $\varepsilon > 0$, the subchromatic number of n-vertex graphs cannot be approximated within a factor of $n^{1/2-\varepsilon}$, unless* NP $\subseteq$ ZPP.

Proof. Let G be an arbitrary graph on n vertices. Then the graph $\overline{nG}$ has n^2 vertices, and by Lemma 4 we have $\chi(G) = \chi_{sub}(\overline{nG})$. Now the result of Feige & Kilian [7] completes the argument. □

4 Positive Results: Complements of Planar Graphs

Gimbel & Nešetřil [10] showed that deciding P_4-free 2-colorability of a planar graph is NP-hard. Since the complement of P_4 is again P_4, this implies that P_4-free 2-subcolorability is NP-hard for complements of planar graphs, too. Fiala, Jansen, Le & Seidel [8] showed that deciding 2-subcolorability of a planar graph is NP-hard. Surprisingly, we will show in this section that 2-subcolorability is polynomially solvable for complements of planar graphs. This will follow as a corollary from the main theorem of this section. The proof of this theorem will appear in the full version of the paper.

Theorem 2. *Let $\ell \geq 2$, and let $\mathcal{G}$ be a class of graphs that do not contain $K_{\ell,\ell}$ as an (induced or non-induced) subgraph. Then 2-subcolorability of a graph $G = (V, E)$ in $\overline{\mathcal{G}}$ can be decided in polynomial time $O(|V|^{3\ell})$.*

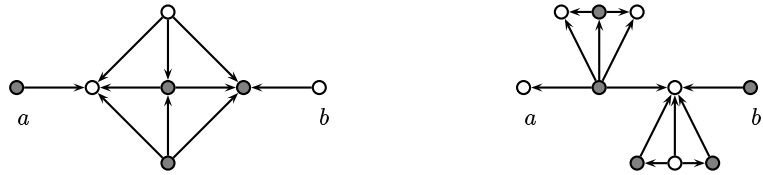


Fig. 2. The gadget graphs *Source-Source* and *Source-Sink*.

Since planar graphs do not contain $K_{3,3}$ as a subgraph, we have the following corollary to Theorem 2. Corollaries 1.(c) and 3 together provide a complete classification of the complexity of subcolorings of complements of planar graphs.

Corollary 3. *The 2-subcoloring problem on complements of planar graphs is polynomially solvable.*

5 Perfect Graphs

In this section, we discuss three classes of perfect graphs. In Section 5.1 we prove NP-hardness of r-subcolorability for every fixed $r \geq 2$ on comparability graphs. In Sections 5.2 and 5.3, we give polynomial time dynamic programming algorithms for r-subcolorability for every fixed $r \geq 2$, on interval graphs and on permutation graphs, respectively.

5.1 Comparability Graphs

In this section we prove the NP-hardness of r-subcolorings on comparability graphs for every fixed $r \geq 2$. Our NP-hardness reduction is based on the two graphs *Source-Source* (depicted to the left) and *Source-Sink* (depicted to the right) in Figure 2. Both graphs have two contact vertices a and b.

Lemma 5. *(a) The graphs* Source-Source *and* Source-Sink *are comparability graphs.*

(b) The graphs Source-Source *and* Source-Sink *possess a 2-subcoloring, in which no contact point receives the same color as its neighbor.*

(c) In every 2-subcoloring of Source-Source *and* Source-Sink*, the contact vertices a and b must receive different colors.*

Proof. Proof of (a). The orientations depicted in Figure 2 are transitive. Proof of (b). The 2-colorings depicted in Figure 2 are subcolorings. Proof of (c). By checking all possible cases. □

Statements (b) and (c) in Lemma 5 are extremely useful for our NP-hardness proofs: Consider a graph G, and let x, y be a pair of vertices in G. Let the graph G^+ result from G by adding an independent copy of a *Source-Source* or a *Source-Sink* gadget to G, and by identifying vertex x with contact point a, and vertex y with contact point b. Then the graph G^+ has a 2-subcoloring, if and only if G has a 2-subcoloring in which x and y receive different colors.

Theorem 3. *The 2-subcoloring problem is NP-hard for comparability graphs.*

Proof. The proof is by a reduction from the NP-complete SET SPLITTING problem [14]: Given a finite set S and a collection C of triples over S, decide whether there is a partition of S into two subsets S_1 and S_2 such that every triple in C has a non-empty intersection with S_1 and with S_2.

Now let $C = \{c_1, c_2, \ldots, c_m\}$ be a collection of triples over a finite set $S = \{s_1, s_2, \ldots, s_n\}$. We construct the following graph G_C from C: For every $s_j \in S$, there is a corresponding vertex $x_j \in X$. For every triple $c_i = (s_i^1, s_i^2, s_i^3) \in C$, there are three corresponding vertices y_i^1, y_i^2, y_i^3 that form a P_3; there is an edge between y_i^1 and y_i^2, and there is an edge between y_i^2 and y_i^3. Vertex y_i^2 is called the *middle vertex* of this path, and vertices y_i^1 and y_i^3 are called the *end vertices*. Moreover, we introduce the following copies of the *Source-Source* and *Source-Sink* gadgets:

- For every occurrence of s_j in the *first* or *third* position of some triple $c_i = (s_i^1, s_i^2, s_i^3)$, the graph G_C contains a copy of the *Source-Source* gadget; the contact points are identified with vertex x_j, and with vertex y_i^1 (first position) or y_i^3 (third position), respectively.
- For every occurrence of s_j in the *second* position of some triple $c_i = (s_i^1, s_i^2, s_i^3)$, the graph G_C contains a copy of the *Source-Sink* gadget; the contact point b is identified with vertex x_j, and the contact point a is identified with vertex y_i^2.

This completes the definition of the graph G_C. We argue that G_C is a comparability graph by considering the following orientation: All *Source-Sink* and *Source-Source* gadgets are oriented as shown in Figure 2. The edges of the paths y_i^1, y_i^2, y_i^3 are directed towards the middle vertices y_i^2. In the resulting orientation, all vertices $x_j \in X$ and all end vertices of paths are sources, and all middle vertices of paths are sinks. Hence, arcs incident to these vertices can not violate transitivity, and the remaining arcs are within the gadgets.

We claim that G_C has a 2-subcoloring if and only if the corresponding instance of SET SPLITTING has answer YES.

Assume that G_C has a 2-subcoloring. We construct the following set splitting: If x_j is colored 1, then $s_j \in S_1$. If x_j is colored 2, then $s_j \in S_2$. Consider a triple $c_i = (s_i^1, s_i^2, s_i^3)$ in C. If it is contained in S_1 or S_2, then the three vertices that correspond to s_i^1, s_i^2, s_i^3 must all have the same color, and the three vertices y_i^1, y_i^2, y_i^3 on the P_3 corresponding to C_i must all have the opposite color. But then this P_3 would be monochromatic, and the coloring would not be a subcoloring.

Next assume that C possesses a set splitting of S into S_1 and S_2. We construct the following coloring: If $s_j \in S_1$, then we color vertex x_j by 1. If $s_j \in S_2$, then we color vertex x_j by 2. Then we extend this coloring to the *Source-Sink* and *Source-Source* gadgets according to Figure 2. It is easily checked that the resulting coloring is a 2-subcoloring of G_C. This completes the proof of the theorem. □

The NP-hardness result on 2-subcoloring for comparability graphs can easily be generalized to r-subcolorings with $r \geq 3$.

Theorem 4. *For any $r \geq 2$, the r-subcoloring problem on comparability graphs is NP-hard.*

Proof. We proceed by induction on r. The starting case $r = 2$ has been settled in Theorem 3. So assume that we have proved the statement up to r, and that we want to prove it for $r + 1$. This will be done as follows: For every comparability graph G_r, we construct in polynomial time a comparability graph G_{r+1} such that G_r is r-subcolorable if and only if G_{r+1} is $(r + 1)$-subcolorable.

Let $K = K_{r+1,\ldots,r+1}$ be the complete $(r + 1)$-partite graph containing $r + 1$ classes of $r + 1$ vertices. Recall that by Proposition 2, $\chi_{sub}(K) = r + 1$. We put $G_{r+1} = G_r \bowtie K$ where the new vertex is called v. Observe that G_{r+1} is a comparability graph: G_r and K are comparability graphs; we take their transitive orientations, and we orient all edges that are incident with the new vertex v away from v.

Assume that G_{r+1} is subcolorable in $r+1$ colors. By Proposition 2 the vertices of the graph K must use all $r + 1$ colors; in particular, the color c of the new vertex v is used in K. But this implies that color c cannot be used for the vertices of G_r, since this would yield a monochromatic P_3 in color c. Hence, the graph G_r is subcolorable in r colors.

Now assume that G_r is subcolorable in r colors. Take this r-subcoloring, and color the new vertex v by a new color. Color K by $r+1$ colors in such a way that every independent class of $r+1$ vertices receives all $r+1$ colors; in other words, every color class induces a clique of size $r+1$ in K. The resulting $(r+1)$-coloring of G_{r+1} is a subcoloring. □

5.2 Interval Graphs

In this section we design for every fixed $r \geq 2$ a polynomial time algorithm for the r-subcoloring problem on interval graphs. Let $G = (V, E)$ be an interval graph with $|V| = n$. Without loss of generality we may assume that the left endpoints of the intervals $I_1, \ldots, I_n$ that represent G are the integers $1, 2, \ldots, n$. For $k = 1, \ldots, n$ we denote by G_k the subgraph that is induced by the first k intervals $I_1, \ldots, I_k$. For a clique Cl in G, we denote by $\text{inter}(Cl) \neq \emptyset$ the intersection of all intervals in Cl and by $\text{union}(Cl)$ the union of all these intervals. Note that $\text{inter}(Cl)$ and $\text{union}(Cl)$ are also intervals.

Consider an arbitrary color class C in an arbitrary r-subcoloring of G_k. This color class C is the union of a number q of disjoint cliques $Cl_1, \ldots, Cl_q$; without loss of generality we assume that $\text{union}(Cl_i)$ always lies completely to the left of $\text{union}(Cl_{i+1})$. Now assume that we would like to extend the subcoloring to the graph G_{k+1} by adding interval I_{k+1} (with left endpoint $k + 1$) to color class C. There are only two possibilities for doing this:

(a) If the point $k+1$ lies to the right of $\text{union}(Cl_q)$, then interval I_{k+1} may start a new clique in C.
(b) If the point $k + 1$ lies within $\text{inter}(Cl_q)$, then interval I_{k+1} may be added to the rightmost clique Cl_q in C.

Note furthermore that the *left* endpoints of inter(Cl_q) and union(Cl_q) do not exceed k. Hence, for deciding whether case (a) or case (b) holds, we only need to know the *right* endpoints of inter(Cl_q) and union(Cl_q). These observations suggest the following dynamic programming formulation.

Every state is specified by a $(2r+1)$-tuple $[k; i_1, i_2, \ldots, i_r; u_1, \ldots, u_r]$. Here $1 \leq k \leq n$, and the variables $i_1, \ldots, i_r$ and $u_1, \ldots, u_r$ either specify right endpoints of some of the intervals $I_1, \ldots, I_k$, or they take the dummy value '$*$'. Hence, altogether there are $O(n^{2r+1})$ states. For every state, we compute a Boolean value $B[k; i_1, \ldots, i_r; u_1, \ldots, u_r]$. This Boolean value is TRUE, if and only if there exists a subcoloring of G_k with color classes $C_1, \ldots, C_r$ with the following properties for $j = 1, \ldots, r$: If C_j is empty, then $i_j = u_j = *$. And if C_j is non-empty, then i_j is the right endpoint of inter(Cl) and u_j is the right endpoint of union(Cl) of the rightmost clique Cl in color class C_j.

The values $B[k; i_1, \ldots, i_r; u_1, \ldots, u_r]$ are computed first for level $k = 1$, then for level $k = 2$, and so on up to level $k = n$. Since in any subcoloring for G_k the interval I_{k+1} can be added in at most two possible ways (a) and (b) to at most r color classes, every TRUE value at level k generates at most $2r$ TRUE values at level $k+1$. The graph G is r-subcolorable, if and only if there exists a TRUE value at level n. Summarizing, we get the following theorem.

Theorem 5. *For any fixed r, the r-subcoloring problem for an interval graph with n vertices can be solved in $O(r \cdot n^{2r+1})$ time.*

5.3 Permutation Graphs

Theorem 6. *For any fixed r, the r-subcoloring problem for a permutation graph with n vertices can be solved in $O(r \cdot n^{3r+1})$ time.*

To prove the theorem we use a dynamic programming approach that is quite similar to the above algorithm for interval graphs. For details, see the full version of this paper.

6 Concluding Remarks and Questions

- What is the computational complexity of r-subcoloring for cocomparability graphs?
- What is the computational complexity of r-subcoloring for chordal graphs?
- What is the computational complexity of 2-subcoloring for AT-free graphs? (In Section 3, we have proved that 3-subcoloring of AT-free graphs is NP-hard).
- What is the computational complexity of r-subcoloring for interval graphs and permutation graphs, if r is part of the input? (In Section 5, we have proved that these problems are polynomially solvable if r is fixed and not part of the input).

Acknowledgments

We are grateful to Dieter Kratsch and Ochem Pascal for fruitful discussions on the topic of this paper.

References

1. D. Achlioptas, *The complexity of G-free colorability*, Discrete Math., 165/166 (1997), pp. 21–30.
2. M. O. Albertson, R. E. Jamison, S. T. Hedetniemi, and S. C. Locke, *The subchromatic number of a graph*, Discrete Math., 74 (1989), pp. 33–49.
3. A. Brandstädt, V. B. Le, and J. P. Spinrad, *Graph classes: a survey*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1999.
4. I. Broere and C. M. Mynhardt, *Generalized colorings of outerplanar and planar graphs*, in Graph theory with applications to algorithms and computer science (Kalamazoo, Mich., 1984), Wiley, New York, 1985, pp. 151–161.
5. P. Erdős, *Some remarks on the theory of graphs*, Bull. Amer. Math. Soc., 53 (1947), pp. 292–294.
6. P. Erdős, J. Gimbel, and D. Kratsch, *Some extremal results in cochromatic and dichromatic theory*, J. Graph Theory, 15 (1991), pp. 579–585.
7. U. Feige and J. Kilian, *Zero knowledge and the chromatic number*, J. Comput. System Sci., 57 (1998), pp. 187–199. Complexity 96—The Eleventh Annual IEEE Conference on Computational Complexity (Philadelphia, PA).
8. J. Fiala, K. Jansen, V. B. Le, and E. Seidel, *Graph subcoloring: Complexity and algorithms*, in Graph-theoretic concepts in computer science, WG 2001, Springer, Berlin, 2001, pp. 154–165.
9. J. Gimbel, D. Kratsch, and L. Stewart, *On cocolourings and cochromatic numbers of graphs*, Discrete Appl. Math., 48 (1994), pp. 111–127.
10. J. Gimbel and J. Nešetřil, *Partitions of graphs into cographs*, Technical Report 2000-470, KAM-DIMATIA, Charles University, Czech Republic, 2000.
11. M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
12. M. Grötschel, L. Lovász, and A. Schrijver, *Polynomial algorithms for perfect graphs*, in Topics on perfect graphs, North-Holland, Amsterdam, 1984, pp. 325–356.
13. C. T. Hoàng and V. B. Le, *P_4-free colorings and P_4-bipartite graphs*, Discrete Math. Theor. Comput. Sci., 4 (2001), pp. 109–122 (electronic).
14. L. Lovász, *Coverings and coloring of hypergraphs*, in Proceedings of the Fourth Southeastern Conference on Combinatorics, Graph Theory, and Computing (Florida Atlantic Univ., Boca Raton, Fla., 1973), Utilitas Math., Winnipeg, Man., 1973, pp. 3–12.
15. F. Maffray and M. Preissmann, *On the NP-completeness of the k-colorability problem for triangle-free graphs*, Discrete Math., 162 (1996), pp. 313–317.
16. C. M. Mynhardt and I. Broere, *Generalized colorings of graphs*, in Graph theory with applications to algorithms and computer science (Kalamazoo, Mich., 1984), Wiley, New York, 1985, pp. 583–594.
17. K. Wagner, *Monotonic coverings of finite sets*, Elektron. Informationsverarb. Kybernet., 20 (1984), pp. 633–639.

Search in Indecomposable Graphs

Alain Cournier

LaRIA 5, rue du moulin neuf, 80 000 Amiens
cournier@laria.u-picardie.fr

Abstract. In this paper we will present some properties of indecomposable graphs, furthermore we see how a search in an indecomposable graph can compute some of these properties. Endly we will see that using this search, when the vertices of an indecomposable graph G are visited in a given order O, the vertices of the complement of G (denoted $\overline{G}$) can also be visited in the same order O.

Keywords: Indecomposable Graphs, Prime Graphs, Search in a Graph.

1 Introduction

An undirected graph (or symmetric graph) G consists of a finite set V of vertices together with a prescribed collection E of unordered pairs of distinct vertices called the set of edges of G. Such a graph will be denoted $G = (V, E)$. With each subset X of V, is associated the *(induced) subgraph* $G(X) = (X, E \cap (X \times X))$ of G. Finally given a graph $G = (V, E)$ the *complement* of G is the graph $\overline{G} = (V, \overline{E})$ defined as follow: let $x \neq y \in V$, $(x, y) \in \overline{E}$ if and only if $(x, y) \notin E$.

A search in a graph $G = (V, E)$ will explore all vertices and edges of G. Some of these searches are very famous like Breadth-First Search [13], Deep-First Search [9], Maximum Cardinality Search [15] or Lexicographic Breadth-First Search [7].

In the following, we will use the following notations. Given a graph $G = (V, E)$, for a given vertex x of G, $N(x)$ is the set of $y \in V$ such that $(x, y) \in E$. Finally, given unordered pairs (x, y) and (x', y') of distinct vertices of G, (x, y) and (x', y') are equivalent denoted by $(x, y) \simeq (x', y')$ when either $(x, y) \in E \leftrightarrow (x', y') \in E$. Otherwise, we will denote this by $(x, y) \not\simeq (x', y')$.

Let $G = (V, E)$ be a graph, a subset X of V is an *interval* [6,10,16] (or an *autonomous subset* [12] or a *clan* [5] or an *homogeneous subset* [8] or a *module* [17] or a *partitive subset* [18]) whenever for all $a, b \in X$ and all $x \in V - X$, $(a, x) \simeq (b, x)$. For example, this notion is the classic notion of interval when G is a linear ordering. Given a graph $G = (V, E)$, $\emptyset$, V and $\{x\}$, where $x \in V$, are clearly intervals of G, called *trivial* intervals. A graph is then said to be *indecomposable* [10,16] (or prime [3] or primitive [5]) whenever $|V| \geq 3$ and all of its intervals are trivial. Otherwise, a graph $G = (V, E)$, which admits at least an interval X such that $2 \leq |X| < |V|$, is said to be *decomposable*[1]. An indecomposable graph $G = (V, E)$ is *minimal* for elements $x_1, \ldots, x_k$ of V when

[1] by convention, a graph $G = (V, E)$ such that $|V| \leq 2$ is said to be *decomposable*

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 80–91, 2002.

for each $Y \subset V$ $(Y \neq V)$ such that $\{x_1, \ldots, x_k\} \subseteq Y$ and $|Y| \geq 3$, $G(Y)$ is decomposable. In a similar type of problem, J. H. Schmerl and W. T. Trotter [16] examined *critically indecomposable* graphs which are indecomposable graphs $G = (V, E)$, with $|V| \geq 4$, such that for $x \in V$, $G(V - \{x\})$ is decomposable. Finally, we introduce the notion of quotient graph. Given a graph $G = (V, E)$, a partition P of V is an *interval partition* of G when all of the elements of P are intervals of G. For such a partition P, we may define the *quotient graph* $G/P = (P, E/P)$ of G by P as follows: let $X \neq Y \in P$,$(X, Y) \in E/P$ whenever for $x \in X$ and for $y \in Y$, $(x, y) \in E$.

The aim of this paper is to characterize a search in an indecomposable graph. As a consequences, we will see that such a search in an indecomposable graph can be useful to compute some properties.

2 The Indecomposable Graphs

2.1 General Properties

In this section, we will recall some of the properties of indecomposable graphs which will be used in what follows. We start with a review of the properties of the intervals of a graph as obtained in the papers concerning the decomposability of graphs.

Proposition 1 *Let $G = (V, E)$ be a graph, the graphs G and $\overline{G}$ have the same intervals. Moreover, these intervals satisfy the following assertions.*

1. V, $\emptyset$ and $\{x\}$, where $x \in V$, are intervals of G.

2. If X and Y are intervals of G, then $X \cap Y$ is an interval of G.

3. Let X, Y be intervals of G, if $X \cap Y \neq \emptyset$, $X \cup Y$ is an interval of G.

4. Let X and Y be intervals of G, if $X - Y \neq \emptyset$, where $X - Y = \{x \in X \mid x \notin Y\}$, then $Y - X$ is an interval of G.

5. Given a subset W of V, if X is an interval of G, then $X \cap W$ is an interval of $G(W)$.

The next propositions allow for the examination of the indecomposable subgraphs of an indecomposable graph.

Proposition 2 ([18]) *Given an indecomposable graph $G = (V, E)$, with $\mid V \mid \geq 3$, there is a subset X of V such that $\mid X \mid = 4$ and $G(X)$ is indecomposable.*

In order to construct indecomposable subgraphs of a larger size, we use the following partition.

Definition 1 *Given a graph $G = (V, E)$ and a subset X of V such that $\mid X \mid \geq 3$ and $G(X)$ is indecomposable. For $u \in X$, $Eq(u)$ is the set of $x \in V - X$ such that $\{u, x\}$ is an interval of $G(X \cup \{x\})$. The set of $x \in V - X$ such that X is an interval of $G(X \cup \{x\})$ is denoted by $[X]$ and the set of $x \in V - X$ such that $G(X \cup \{x\})$ is indecomposable is denoted by $Ext(X)$.*

Lemma 1 ([5]) *Given a graph $G = (V, E)$ and a subset X of V such that $| X | \geq 3$ and $G(X)$ is indecomposable.*

1. The family $p(X) = \{Ext(X), [X], Eq(u)(u \in X)\}$ is a partition of $V - X$.

2. For $x \neq y \in Ext(X)$, $G(X \cup \{x, y\})$ is decomposable if and only if $\{x, y\}$ is an interval of $G(X \cup \{x, y\})$.

3. For $x \in Eq(u)$ and for $y \in V - (X \cup Eq(u))$, where $u \in X$, $G(X \cup \{x, y\})$ is decomposable if and only if $\{x, u\}$ is an interval of $G(X \cup \{x, y\})$.

4. For $x \in [X]$ and for $y \in V - (X \cup [X])$, $G(X \cup \{x, y\})$ is decomposable if and only if $X \cup \{y\}$ is an interval of $G(X \cup \{x, y\})$.

The next result is a direct consequence of Lemma 1.

Proposition 3 ([5]) *Let $G = (V, E)$ be an indecomposable graph, if X is a subset of V such that $G(X)$ is indecomposable and $3 \leq | X | \leq | V | -2$, then there are $x \neq y \in V - X$ such that $G(X \cup \{x, y\})$ is indecomposable.*

Corollary 1 flows from Propositions 2 and 3.

Corollary 1 *If $G = (V, E)$ is an indecomposable graph, with $| V | \geq 5$, then there exits $X \subset V$ such that $G(X)$ is indecomposable and $| V - X | = 1$ or 2.*

Practically, in order to satisfy that a graph $G = (V, E)$ is indecomposable, we must first look for a subset X of V such that $G(X)$ is indecomposable and $|X| = 4$. We next calculate the partition $p(X)$ and, using Lemma 1, we try to find $x, y \in V - X$ such that $G(X \cup \{x, y\})$ is indecomposable. We continue this procedure by replacing X by $X \cup \{x, y\}$. For more details, refer to the recognition algorithm described in [2].

2.2 Minimal Indecomposable Graphs for Two Vertices

We first define on $\{1, \ldots, k\}$, where $k \geq 4$, the symmetric graphs P_k and Q_k (see Figures 1 and 2) in the following manner. For $i \neq j \in \{1, \ldots, k\}$, (i, j) is an edge of P_k when $| i - j | = 1$. For $i \neq j \in \{1, \ldots, k\}$, (i, j) is an edge of Q_k whenever either $i, j \in \{1, \ldots, k-2\}$ and $| i - j | = 1$ or $k - 1 \in \{i, j\}$ and there is $l \in \{1, \ldots, k-3\} \cup \{k\}$ such that $\{i, j\} = \{k - 1, l\}$.

Fig. 1. P_k

Proposition 4 ([4]) *Let $G = (V, E)$ be an indecomposable symmetric graph, with $|V| \geq 4$, and $x \neq y$ be elements of V, G is minimal for x and for y if and only if there is an isomorphism f from G or $\overline{G}$ onto P_k or Q_k, where $k \geq 4$, such that $f(\{x, y\}) = \{1, k\}$.*

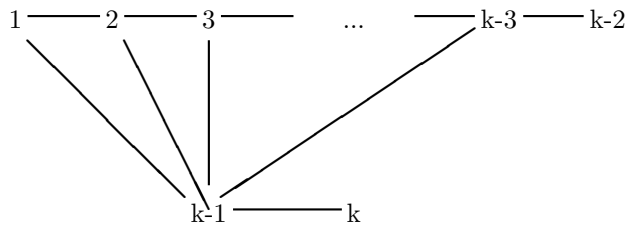


Fig. 2. Q_k

Proposition 4 is a direct consequence of the following result.

Proposition 5 ([4]) *Given an indecomposable symmetric graph $G = (V, E)$, with $\mid V \mid \geq 4$, for $x \neq y \in V$, there is a subset X of V satisfying: $x, y \in X$ and there is an isomorphism f from $G(X)$ or $\overline{G(X)}$ onto P_k or Q_k, where $k \geq 4$, such that $f(\{x, y\}) = \{1, k\}$.*

3 An Algorithm

The next algorithm will be used as a subroutine of our next search algorithm.

3.1 Principle

The algorithm takes an undirected graph G and two vertices x, y of G as inputs and computes the smallest interval I of G satisfying one of the two following conditions :

1. $\{x\} \cup (N(x) - N(y)) \subseteq I$ when $(x, y) \notin E$;
2. $\{x\} \cup (N(y) - N(x)) \subseteq I$ when $(x, y) \in E$.

In the same time we also compute in an array a function M. This function associates to each vertex of I an integer. We will use M later.

3.2 The Algorithm

In order to compute this set I, we will use the following sets: $Set1$, $Set2$, $Universal$, $Unseen$. At the end of the algorithm we want $Set1 = Set2 = \emptyset$, $u \in Universal \Leftrightarrow I \subseteq N(u)$ and $u \in Unseen \Leftrightarrow I \cap N(u) = \emptyset$. Of course we also need that I respects previous conditions.

At each iteration of the algorithm we can assume that I is the subset of the final result already computed. In $Set1$ we have the vertices of G we want to insert in I at this iteration of the algorithm. And let us suppose we maintain the two properties $u \in Universal \Leftrightarrow I \subseteq N(u)$ and $u \in Unseen \Leftrightarrow I \cap N(u) = \emptyset$. Then an iteration of the algorithm, put each vertices v of $Set1$ in I. At this moment some vertices of $Universal$ or $Unseen$ cannot stay anymore in their sets (with respect of the previous properties). They will be inserted in $Set2$. In Fact, $Set2$

contains the vertices previously in $Universal$ or $Unseen$ that violate interval property of $I \cup Set1$.

Algorithm: ComputeInterval
Data: $G = (V, E)$ an undirected graph; x, y two vertices of G;
Result: I a set of vertices of G;M: array[V] of integer;
Var: $Set1, Set2, Universal, Unseen$: sets of vertices; d:integer;
BeginAlgo
$I \leftarrow \{x\}$; $Set2 \leftarrow \emptyset$; $d \leftarrow 1$; $M[x] \leftarrow 0$;
If $xy \notin E$ **then**
$Unseen \leftarrow V - (N(x) \cup \{x\})$; $Universal \leftarrow N(x) \cap N(y)$;
$Set1 \leftarrow N(x) - (N(y) \cup \{y\})$
Else
$Unseen \leftarrow V - (N(y) \cup N(x))$; $Universal \leftarrow N(x)$;
$Set1 \leftarrow N(y) - (N(x) \cup \{x\})$;
Endif
While $Set1 \neq \emptyset$ **do**
While $Set1 \neq \emptyset$ **do**
Let z be a vertex of $Set1$; $M[z] \leftarrow d$;
Move z from $Set1$ to I;
$Set2 \leftarrow Set2 \cup (Unseen \cap N(z)) \cup (Universal - N(z))$;
$Universal \leftarrow Universal \cap N(z)$; $Unseen \leftarrow Unseen - N(z)$;
EndWhile
$d \leftarrow d + 1$;$Set1 \leftarrow Set2$; $Set2 \leftarrow \emptyset$;
EndWhile
EndAlago

3.3 Termination, Correctness and Complexity of the Algorithm

The first Invariant of the algorithm

Invariant 1 *At each step of the algorithm* $\{I, Set1, Set2, Universal, Unseen\}$ *is a partition of* V.

Proof. In fact one can notice that for any vertex v:
If the algorithm removes v from $Universal$, it inserts v in $Set2$;
If the algorithm removes v from $Unseen$, it inserts v in $Set2$;
If the algorithm removes v from $Set2$, it inserts v in $Set1$;
If the algorithm removes v from $Set1$, it inserts v in I.
Since we cannot lose any vertices and since at the beginning of the algorithm, $\{I, Set1, Set2, Universal, Unseen\}$ is a partition of V, the Invariant 1 is true.
The second invariant will give us the termination of the internal While loops.

Invariant 2 *Each time the algorithm executes the statment of the internal While loop, it increases the cardinality of* I *and decreases the cardinality of* $Set1$.

Corollary 2 *The internal while loops stops.*

Proof. Since the cardinality of $Set1$ strictly decreases during this loop, the internal while loops stops.

Invariant 3 *Each time the algorithm goes in the external While loop, it increases the cardinality of I.*

Proof. Since the test of the internal and external while loops are identical, if the algorithm comes in the external while loop, it comes in the internal while loop, so using Invariant 2, $|I|$ increases.

Proposition 6 *The algorithm ComputeInterval stops.*

Proof. We know that $1 \leq |I| \leq |V|$, using Invariant 3, we know that the algorithm cannot enter more than $|V|-1$ time in the external while loop. So the algorithm stops.

In order to verify that I is the smallest interval of G containing x and $N(x) - N(y)$ (respectively $N(y) - N(x)$) when $xy \notin E$ (respectively $xy \in E$). We will verify first: At the end of the algorithm I is an interval of G.

Invariant 4 *Let t, u, v be three vertices of G, such that $t, u \in Visited$ and $v \in Universal \cup Unseen$. We can say $(v,t) \simeq (v,u) \simeq (v,x)$*

Proof. This invariant is true at the beginning of the algorithm since $|I| = 1$. We will suppose that it stills true when $|I| = k$. Let us suppose ComputeInterval picks a new vertex z in $Set1$ and let us denote by $Universal'$ (respectively $Unseen'$, I') the following set of vertices $Universal' = Universal \cap N(z)$ (respectively $Unseen' = Unseen - N(z)$, $Visited' = Visited \cup \{z\}$).

Let t, u, v be three vertices of G, such that $t, u \in I'$ and $v \in Universal'$. Let us suppose, $t \neq z \neq u$, since $Universal' \subseteq Universal$, $(v,t) \simeq (v,u) \simeq (v,x)$. In the remaining case, $t = z \neq v$, since $Universal' \subseteq Universal$, $(v,u) \simeq (v,x)$, furthermore, since $v \in Universal'$, $(v,z), (v,x) \in E$ and $(v,z) \simeq (v,x)$.

Let t, u, v be three vertices of G, such that $t, u \in Visited'$ and $v \in Unseen'$. Let us suppose, $t \neq z \neq u$, since $Unseen' \subseteq Unseen$, $(v,t) \simeq (v,u) \simeq (v,x)$. In the remaining case, $t = z \neq v$, since $Unseen' \subseteq Unseen$, $(v,u) \simeq (v,x)$, furthermore, since $v \in Unseen'$, $(v,z), (v,x) \notin E$ and $(v,z) \simeq (v,x)$. The property still true for the new sets, $Visited', Unseen', Universal'$ with $|Visited| = k+1$.

So Invariant 4 is still true.

Corollary 3 *At the end of the algorithm I is an interval of G.*

Proof. Since the algorithm stops when $Set1 = Set2 = \emptyset$, $\{I, Universal, Unseen\}$ is a partition of V(Invariant 1), and I is an interval of G (Invariant 4).

The following Invariant of the internal While loop will be helpful to prove that I is the smallest interval. It gives also a characterization of $Set2$.

Invariant 5 *At the end of the internal while loop, we can claim*
$Set2 = \{u \in V - I | \exists v, w \in I \text{ and } (v,u) \not\simeq (w,u)\}$

Proof. When the algorithm inserts a vertex u in $Set2$ then either $u \in Universal - N(z)$ or $u \in Unseen \cap N(z)$ in both cases, $(u, z) \not\simeq (u, x)$.

Invariant 6 *At the end of the internal while loop, any interval W containing I must contain $Set2$.*

Proof. Let us suppose the contrary. Let W be an interval of G containing I, and s be a vertex of $Set2 - W$. Using Invariant 5, there exists $i_1, i_2 \in I$ such that $(i_1, s) \not\simeq (i_2, s)$. Since $i_1, i_2 \in W$ and $s \notin W$, W is not an interval of G. A contradiction.

Corollary 4 *At the end of the algorithm, I is the smallest interval of G containing x and $N(x) - N(y)$ (respectively $N(y) - N(x)$) when $xy \notin E$ (respectively $xy \in E$).*

Proof. This is a direct consequence of Invariant 6.

Proposition 7 *The algorithm ComputeInterval runs in $O(|V| + |E|)$ time complexity.*

Proof. In fact, we have to compute the 3 sets $Universal \cap N(z)$, $Unseen - N(z)$, and $Set2 \cup (Universal - N(z)) \cup (Unseen \cap N(z))$, in $O(|N(z)|)$ time complexity. So these sets can be represented as queues, and we can use some techniques like partition refinements developed in [14,2,1,3].

3.4 Properties of the Algorithm

The first property of this algorithm claims that the result is independent from running on a graph G or the complement of this graph.

Theorem 1 *Let G be a graph, x, y be two vertices of G, I, M the results of the algorithm $ComputeInterval(G, x, y, I, M)$ and $\overline{I}$, $\overline{M}$ the results of the algorithm $ComputeInterval(\overline{G}, x, y, \overline{I}, \overline{M})$. Wecan claim $I = \overline{I}$ and For each $i \in I$, $M[i] = \overline{M}[i]$.*

Proof. In this proof we will assume that $x \notin N_G(y)$.

Let us denote by $I, Set1, Set2, Universal, Unseen, M, d$, the variable sets of ComputeInterval(G, x, y, I, M) and $\overline{I}, \overline{Set1}, \overline{Set2}, \overline{Universal}, \overline{Unseen}, \overline{M}, \overline{d}$ the variable sets of ComputeInterval$(\overline{G}, x, y, \overline{I}, \overline{M})$.

One can notice that $C(G, x, y) \Longleftrightarrow C(\overline{G}, x, y)$.

At the beginning of the algorithm, we can say :
$Set1 = N(x) - (N(y) \cup \{y\}) = N(x) \cap (\overline{N(y)} \cup \{y\}) =$
$N(x) \cap N_{\overline{G}}(y) = N_{\overline{G}}(y) - (\overline{N(x)}) = N_{\overline{G}}(y) - (N_{\overline{G}}(x) \cup \{x\}) = \overline{Set1}$

$I = \{x\} = \overline{I}$

So using Corollary 4 and Proposition 1, we can claim $I = \overline{I}$ at the end of the algorithm.

Let us now suppose that the algorithm after k steps in the external while loop verify the following conditions :
$Set1_k = \overline{Set1}_k$, $Set2_k = \overline{Set2}_k$, $I_k = \overline{I}_k$, $d_k = \overline{d}_k = k$, $\forall x \in I_k, M[x] = \overline{M}[x]$

Then the algorithm execute a $k+1$th time the external while loop. At the end of this step : $I_{k+1} = I_k \cup Set1_k = \overline{I}_k \cup \overline{Set1}_k = \overline{I}_{k+1}$.
At the end of the internal While loop, using Invariant 5, $Set2_{k+1} = \overline{Set2}_{k+1}$. So at the end of the external While loop, $Set1_{k+1} = \overline{Set1}_{k+1}$ and $Set2_{k+1} = \overline{Set2}_{k+1} = \emptyset$. Of course $d_{k+1} = \overline{d}_{k+1} = k+1$.

The remaining question is: $\forall x \in I_{k+1}, M[x] = \overline{M}[x]$?

If $x \in I_{k+1}$ then either $x \in I_k$ and $M[x] = \overline{M}[x]$ or $x \in Set1_k$ then at the end of this step $M[x] = k+1$, since $Set1_k = \overline{Set1}_k$ $x \in \overline{Set1}_k$ then at the end of this step $\overline{M}[x] = k+1$. In both cases, $\forall x \in I_{k+1}, M[x] = \overline{M}[x]$.

Since the previous conditions are true for $k = 0$ Theorem 1 is true.

4 Search in an Indecomposable Graph

In a first time let us define the following condition:

Condition 1 *$G = (V, E)$ is an undirected graph and x, y are two vertices of G such that: $xy \notin E \Longrightarrow N(x) \not\subseteq N(y)$ and $xy \in E \Longrightarrow N(y) \not\subseteq N(x) \cup \{x\}$*

In the following we will denote by $C(G, x, y)$ the boolean value of Condition 1, where x, y are two vertices of a given graph G.

4.1 Principle

We will use the previous algorithm to do a search in an indecomposable graph. This algorithm takes an indecomposable graph G with more than 2 vertices and two distinct vertices of G as inputs and produces a set of vertices ($Visited$). In the same time, we also compute in an array a function M. This function associate to each $Visited$ vertices an integer. We will use M later.

4.2 The Algorithm

Algorithm: SearchInIndecomposableGraph
Data: $G = (V, E)$: an indecomposable graph;
x, y: two distinct vertices of G;
Result: $Visited$: a set of vertices;
Var: M: array[V] of integer;
BeginAlgo
  If C(G,x,y) **then** $ComputeInterval(G, x, y, Visited, M)$
  Else $ComputeInterval(G, y, x, Visited, M)$
  EndIf
EndAlgo

4.3 Correctness and Complexity

This algorithm is a search in a graph if and only if at the end of the algorithm, $Visited = V$. This is the aim of the following proposition.

Proposition 8 *Let $G = (V, E)$ be an indecomposable graph and x, y two distinct vertices of G. The algorithm SearchInIndecomposableGraph$(G, x, y, Visited, M)$ stops with $Visited = V$*

Proof. First one can notice that the Condition 1 in the particular cases of indecomposable graphs verifies $\neg C(G, x, y) \Longrightarrow C(G, y, x)$.
Otherwise $\{x, y\}$ is a non trivial interval of G.

Using Corollary 4, we know that $Visited$ is an interval of G.

Let us assume that $C(G, x, y)$ is true, then $x \in Visited$, and since $C(G, x, y)$ is true $Visited$ contains one of the two following nonempty sets:
$N(x) - N(y)$ when $xy \notin E$ or $N(y) - N(x)$ when $xy \in E$.

Since G is an indecomposable graph $Visited = V$.

If $C(G, x, y)$ is false, $C(G, y, x)$ is true and the Proposition still true.

From proposition 7 we can claim the following Theorem.

Theorem 2 *The algorithm SearchInIndecomposableGraph run in $O(|V| + |E|)$ time complexity.*

5 Consequences

The following proposition is a direct consequence of Theorem 1

Proposition 9 *For any indecomposable graph $G = (V, E)$ and for any $x, y \in V$ $(x \neq y)$, SearchInIndecomposableGraph$(G, x, y, Visited, M)$ and SearchInIndecomposableGraph$(\overline{G}, x, y, Visited, M)$ can visit the vertices in the same order. Furthermore the function M is independent from G and $\overline{G}$.*

6 Compute Minimal Indecomposable Graph for Two Vertices

In this section we will see how to compute a minimal indecomposable graph for two vertices. The algorithm will take an indecomposable graph G and two distinct vertices x, y as inputs and gives as an output, a set of vertices inducing a minimal indecomposable graph for the vertices x, y. This algorithm will use the algorithm SearchInIndecomposableGraph as a subroutine.

6.1 Outline of the Algorithm

Algorithm: FindMinimalIndecomposableGraph
Data: $G = (V, E)$ an indecomposable graph; x, y two vertices of G;
Result: S : A set of vertices inducing a minimal indecomposable graph;
Var: M : array [V] of integer; i: integer;s, t: vertices
$P = \{p_0, p_1, \ldots\}$: a partition of V;

BeginAlgo
SearchInIndecomposableGraph(G, x, y, S, M); $S \leftarrow \emptyset$;
Compute P such that $p_i = \{u \in V | M[u] = i\}$;
If M[x] $= 0$ **then** $s \leftarrow y$; $t \leftarrow x$
Else $s \leftarrow x$; $t \leftarrow y$;
EndIf
$i \leftarrow M[s]$; $S \leftarrow S \cup \{s\}$;
While $i \geq 2$ **do**
pick a vertex w in p_{i-1} such that $(w, s) \not\simeq (t, s)$;
$s \leftarrow w$; $S \leftarrow S \cup \{w\}$; $i \leftarrow i - 1$
EndWhile
$S \leftarrow S \cup \{t\}$;
EndAlgo

6.2 Proof of the Algorithm

First of all, we will see that the algorithm can pick a vertex w.

Proposition 10 *Let G be an indecomposable graph, x, y be two vertices of G and M the output of the algorithm SearchInIndecomposableGraph(G, x, y, S, M). For any vertex z of G, $M[z] \geq 2$ there exists a vertex v such that $M[v] = M[z]-1$ and either $M[x] = 0$ and $(x, z) \not\simeq (v, z)$ or $M[y] = 0$ and $(y, z) \not\simeq (v, z)$;*

Proof. This is a direct consequence of Invariant 5.

Theorem 3 *There exists an isomorphism f from the subgraph $G(S)$ onto*

$$P_{max(M[x],M[y])+1} \quad or \quad Q_{max(M[x],M[y])+1} \quad or$$

$$\overline{P_{max(M[x],M[y])+1}} \quad or \quad \overline{Q_{max(M[x],M[y])+1}},$$

such that $f(\{x, y\}) = \{1, max(M[x], M[y]) + 1\}$.

Proof. Using the Theorem 1, we will assume that $x \notin N(y)$. In the following we will denote by x_i the vertex of $S \cap p_i$ and by k the number $max(M[x], M[y]) + 1$. Let us first prove that for any j, $0 \leq j \leq k - 3$, $(x_j, x_{j+1}) \not\simeq (x, y)$.

Let us suppose the contrary, there exists j, $0 \leq j \leq k - 3$ with $(x_j, x_{j+1}) \simeq (x, y)$, so either $j = 0$ and by construction, $x_1 \in N(x_0) - N(x_k)$ or $j \geq 1$, in this case $x_{j+1} \in N(x_0)$, so $x_{j+1} \in N(x_0) \cap N(x_k)$, and $k = j + 2$. In these two cases we hold a contradiction.

Since $x \notin N(y)$ we know $(x_0, x_1) \simeq (x_{k-1}, x_k)$ then either $(x_0, x_1) \simeq (x_{k-2}, x_{k-1})$ and $G(S)$ is a P_{k+1}, or $(x_0, x_1) \not\simeq (x_{k-2}, x_{k-1})$ and $G(S)$ is a Q_{k+1}.

In the 2 cases the Theorem holds.

Proposition 11 *When G is an indecomposable graph and $x \neq y$, the algorithm FindMinimalIndecomposableGraph runs in linear time.*

Proof. The algorithm FindMinimalIndecomposableGraph needs to use techniques like partition refinements developed in [14,2,1,3].

7 Conclusion

In this paper, we saw another way to do a search in an indecomposable graph. But one can notice that the algorithm SearchInIndecomposableGraph(G, x, y) visits all the vertices of the graph when $G = (V, E)$ is connected, $\overline{G}$ is connected and V is the smallest interval containing x and y.

The search developed in this paper is close from the Breadth First Search in a graph. In the same way one can imagine a Deep First Search, a Lexicographic Breadth First search or a Maximal Cardinality Search in Indecomposable Graphs.

On the other hand, it would be interesting to continue the examination begun here by attempting to characterize the computable properties, using these searches.

References

1. A. Cournier. *Sur Quelques Algorithmes de Décomposition de Graphes*. PhD thesis, Université Montpellier II, 161 rue Ada, 34392 Montpellier Cedex 5, France, février 1993.
2. A. Cournier and M. Habib. An efficient algorithm to recognize prime undirected graphs. In E. W. Mayr, editor, *Lectures notes in Computer Science, 657.Graph-Theoretic Concepts in Computer Science. 18th International Workshop, WG'92. Wiesbaden-Naurod, Germany, June 1992. Proceding*, pages 212–224. Springer-Verlag, 1993.
3. A. Cournier and M. Habib. A new linear algorithm for modular decomposition. In S. Tison, editor, *Lectures notes in Computer Science, 787. Trees in Algebra and Programming-CAAP'94*, pages 68–84. Springer-Verlag, April 1994. 19th International Colloquium, Edinburgh, U.K., April 1994. Proceedings.
4. A. Cournier and P. Ille. Minimal indecomposable graphs. *Discrete Math*, (183):61–80, 1998.
5. A. Ehrenfeucht and G. Rozenberg. Primitivity is hereditary for 2-structures (fundamental study). *Theoretical Comp. Sci.*, 3(70):343–358, 1990.
6. R. Fraïssé. L'intervalle en thorie des relations, ses gnralisations, filtre intervallaire et clôture d'une relation. In M. Pouzet and D. Richard, editors, *Order, Description and Roles*, pages 313–342. North-Holland, 1984.
7. M. C. Golumbic. *Algorithmic graph theory and perfect graphs*. Academic Press, New-York, 1980.
8. M. Habib. *Substitution des structures combinatoires, théorie et algorithmes*. PhD thesis, Université Pierre et Marie Curie (Paris VI), 1981.
9. J. E. Hopcroft and R. E. Tarjan. Efficient algorithms for graph manipulation. *Communication of the ACM*, 16(6):372–378, 1973.
10. P. Ille. Indecomposable graphs. *Discrete Math*, (173):71–78, 1997.
11. B. Jamison and S. Olariu. P-components and the homogeneous decomposition of graphs. In *18th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'92)*, June 1992.
12. D. Kelly. Comparability graphs. In I. Rival, editor, *Graphs and Orders*, pages 3–40. D. Reidel Publishing Company, 1985.

13. C. Y. Lee. An algorithme for path connection and its applications. *IRE Transactions on Electronic Computers*, EC-10(3):346–365, 1961.
14. C. Paul. *Parcours en largeur lexicographique : un algorithme de partitionnement, applications aux graphes et généralisations*. PhD thesis, LIRMM, Université Montpellier II, 1998.
15. Donald J. Rose, R. Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal of computing*, 5(2):266–283, June 1976.
16. J. H. Schmerl and W. T. Trotter. Critically indecomposable partially ordered sets, graphs, tournaments and other binary relational structures. *Discrete Math*, (113):191–205, 1994.
17. J. Spinrad. P4-trees and substitution decomposition. *Discrete Applied Mathematics*, (39):263–291, 1992.
18. D. P. Sumner. Graphs indecomposable with respect to the X-join. *Discrete Math.*, (6):281–298, 1973.

On the Complexity of (k, l)-Graph Sandwich Problems*

Simone Dantas[1], Celina M.H. de Figueiredo[2], and Luerbio Faria[3]

[1] COPPE, Universidade Federal do Rio de Janeiro, Brazil
sdantas@cos.ufrj.br
[2] Instituto de Matemática and COPPE,
Universidade Federal do Rio de Janeiro, Brazil
celina@cos.ufrj.br
[3] Departamento de Matemática,
Faculdade de Formação de Professores, UERJ, Brazil
luerbio@cos.ufrj.br

Abstract. A graph G is (k, l) if its vertex set can be partitioned into at most k independent sets and l cliques. The (k, l)-Graph Sandwich Problem asks, given two graphs $G^1 = (V, E^1)$ and $G^2 = (V, E^2)$, whether there exists a graph $G = (V, E)$ such that $E^1 \subseteq E \subseteq E^2$ and G is (k, l). In this paper, we prove that the (k, l)-Graph Sandwich Problem is *NP*-complete for the cases $k = 1$ and $l = 2$; $k = 2$ and $l = 1$; or $k = l = 2$. This completely classifies the complexity of the (k, l)-Graph Sandwich Problem as follows: the problem is *NP*-complete, if $k+l > 2$; the problem is polynomial otherwise. In addition, we consider the degree Δ constraint subproblem and completely classifies the problem as follows: the problem is polynomial, for $k \leq 2$ or $\Delta \leq 3$; the problem is *NP*-complete otherwise.

1 Introduction

We say that a graph $G^1 = (V, E^1)$ is a *spanning* subgraph of $G^2 = (V, E^2)$ if $E^1 \subseteq E^2$; and that a graph $G = (V, E)$ is a *sandwich* graph for the pair G^1, G^2 if $E^1 \subseteq E \subseteq E^2$. For notational simplicity in the sequel, we let E^3 be the set of all edges in the complete graph with vertex set V which are not in E^2. Thus every sandwich graph for the pair G^1, G^2 satisfies $E^1 \subseteq E$ and $E \cap E^3 = \emptyset$. We call E^1 the *forced edge set*, $E^2 \setminus E^1$ the *optional edge set*, E^3 the *forbidden edge set*. The GRAPH SANDWICH PROBLEM FOR PROPERTY Π is defined as follows [10]:

GRAPH SANDWICH PROBLEM FOR PROPERTY Π
Instance: Vertex set V, forced edge set E^1, forbidden edge set E^3.
Question: Is there a graph $G = (V, E)$ such that $E^1 \subseteq E$ and $E \cap E^3 = \emptyset$ that satisfies property Π?

* This research was partially supported by CNPq, MCT/FINEP PRONEX Project 107/97, CAPES (Brazil)/COFECUB (France) Project number 213/97, FAPERJ and PROCIÊNCIA-UERJ Project.

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 92–101, 2002.

We shall use both forms (V, E^1, E^2) and (V, E^1, E^3) to refer to an instance of a graph sandwich problem.

Graph sandwich problems have attracted much attention lately arising from many applications and as a natural generalization of recognition problems [5,6,9,10,11,13,14]. The recognition problem for a class of graphs $\mathcal{C}$ is equivalent to the graph sandwich problem in which the forced edge set $E^1 = E$, the optional edge set $E^2 \setminus E^1 = \emptyset$, $G = (V, E)$ is the graph we want to recognize, and property Π is "to belong to class $\mathcal{C}$".

Golumbic et al. [10] have considered sandwich problems with respect to several subclasses of perfect graphs, and proved that the GRAPH SANDWICH PROBLEM FOR SPLIT GRAPHS remains in P. On the other hand, they proved that the GRAPH SANDWICH PROBLEM FOR PERMUTATION GRAPHS turns out to be *NP*-complete.

We are interested in graph sandwich problems for properties Π related to decompositions arising in perfect graph theory: homogeneous set [5], join composition [6]. In this paper, we consider the decomposition of a graph into independent sets and cliques.

Let G be an undirected, finite, simple graph. A (k,l) partition of a graph G is a partition of its vertex set into at most k independent sets and l cliques. A graph is (k,l) if it admits a (k,l) partition. The complexity of (k,l) graph recognition has been completely classified as follows: if $k = 3$ and $l = 0$ then the corresponding problem is 3-coloring, which implies [1,2] that the recognition of (k,l) graphs is *NP*-complete, whenever $k \geq 3$ or $l \geq 3$. For the remaining values of k and l, the problem is polynomial: $(1,1)$ graphs are split graphs; $(2,0)$ graphs are the bipartite graphs; the polynomial-time recognition of $(2,1)$ graphs and consequently of graphs $(1,2)$ was established in [1,2,3]; the polynomial time recognition of $(2,2)$ graphs was established in [1,2] and independently in [7].

The studies on sandwich problems focus on those problems which are interesting in terms of their complexity, i.e., neither trivially *NP*-complete nor trivially polynomial.

Fact 1 *If the recognition problem for a class of graphs $\mathcal{C}$ is NP-complete, then its corresponding sandwich problem is also NP-complete.*

Fact 2 *If the property Π is hereditary then there exists a sandwich graph for (V, E^1, E^2) with the property Π if and only if $G^1 = (V, E^1)$ has the property Π.*

Fact 3 *If the property Π is ancestral then there exists a sandwich graph for (V, E^1, E^2) with the property Π if and only if $G^2 = (V, E^2)$ has the property Π.*

Thus, Fact 1 says that the sandwich problem for (k,l) graphs is *NP*-complete, whenever $k \geq 3$ or $l \geq 3$. In addition, Fact 2 (respectively Fact 3) says that for each property which is hereditary (respectively ancestral), the graph sandwich problem reduces to the recognition problem for this property on the single graph G^1 (respectively G^2). Therefore, the hereditary properties defining $(1,0)$

and $(2,0)$ graphs, and the ancestral properties defining $(0,1)$ and $(0,2)$ graphs reduces these graph sandwich problems to recognition problems that are polynomial. Given a property Π, we define its *complementary property* $\overline{\Pi}$ as follows: for every graph G, G satisfies $\overline{\Pi}$ if and only if $\overline{G}$ satisfies Π.

Fact 4 *There is a sandwich graph with property Π for the instance (V, E^1, E^3) if and only if there is a sandwich graph with property $\overline{\Pi}$ for the instance (V, E^3, E^1).*

Thus, our proof of the *NP*-completeness of the sandwich problem for $(2,1)$ graphs implies the *NP*-completeness of the sandwich problem for $(1,2)$ graphs.

This paper is organized as follows: in Section 2 we prove that the $(2,1)$-Graph Sandwich Problem is *NP*-complete. Section 3 contains the proof that the $(2,2)$-Graph Sandwich Problem is *NP*-complete. This, together with the facts above completely classifies the complexity of the (k,l)-Graph Sandwich Problem as follows: the problem is *NP*-complete, if $k+l>2$; and polynomial otherwise. Finally, Section 4 defines and classifies the degree constraint subproblems obtained by bounding the maximum degree in G^2.

2 (2, 1)-Graph Sandwich Problem

In this section we prove that the $(2,1)$-GRAPH SANDWICH PROBLEM is *NP*-complete by reducing the *NP*-complete problem 3-SATISFIABILITY to $(2,1)$-GRAPH SANDWICH PROBLEM. These two decision problems are defined as follows.

3-SATISFIABILITY (3SAT)
Instance: Set $X = \{x_1, \ldots, x_n\}$ of variables, collection $C = \{c_1, \ldots, c_m\}$ of clauses over X such that each clause $c \in C$ has $|c| = 3$ literals.
Question: Is there a truth assignment for X such that each clause in C has at least one true literal?

$(2,1)$-GRAPH SANDWICH PROBLEM
Instance: Vertex set V, forced edge set E^1, forbidden edge set E^3.
Question: Is there a graph $G = (V, E)$, such that $E^1 \subseteq E$ and $E \cap E^3 = \emptyset$, and G is $(2,1)$?

Theorem 1. *The* $(2,1)$-GRAPH SANDWICH PROBLEM *is NP-complete.*

Proof. In order to reduce 3SAT to $(2,1)$-GRAPH SANDWICH PROBLEM we need to construct a particular instance (V, E^1, E^3) of $(2,1)$-GRAPH SANDWICH PROBLEM from a generic instance (X, C) of 3SAT, such that C is satisfiable if and only if (V, E^1, E^3) admits a sandwich graph $G = (V, E)$ which is $(2,1)$. First we describe the construction of a particular instance (V, E^1, E^3) of $(2,1)$-GRAPH SANDWICH PROBLEM; second we prove in Lemma 1 that every graph $G = (V, E)$ satisfying $E^1 \subseteq E$ and $E \cap E^3 = \emptyset$ and such that G is $(2,1)$, defines a truth assignment for (X, C); third we prove in Lemma 2 that every truth assignment for (X, C) defines a graph $G = (V, E)$ which is $(2,1)$ satisfying $E^1 \subseteq E$ and $E \cap E^3 = \emptyset$. These steps are explained in detail below. □

Construction of Particular Instance of $(2, 1)$-Graph Sandwich problem

The vertex set V contains: an auxiliary set of vertices $\{k_1, k_2, s_{11}, s_{12}, s_{21}, s_{22}\}$; for each variable x_i, $1 \leq i \leq n$, two vertices x_i, $\overline{x_i}$, corresponding to its literals and a vertex p_i; for each clause $c_j = (l_1^j \vee l_2^j \vee l_3^j)$, $1 \leq j \leq m$, three corresponding vertices t_1^j, t_2^j, t_3^j. In Figure 1, solid edges are forced E^1-edges and dashed edges are forbidden E^3-edges.

The Forced Edge Set E^1 contains: edges between auxiliary vertices $\{k_1k_2$, k_1s_{11}, k_1s_{12}, $s_{11}s_{12}$, k_2s_{21}, k_2s_{22}, $s_{21}s_{22}\}$; for each variable x_i, $1 \leq i \leq n$, the set $\{x_is_{11}$, $\overline{x_i}s_{12}$, x_ip_i, $\overline{x_i}p_i\}$; for each clause c_j, $1 \leq j \leq m$, the set $\{t_1^jt_2^j, t_1^jt_3^j, t_2^jt_3^j\}$.

The Forbidden Edge Set E^3 contains: edges between auxiliary vertices $\{k_1s_{21}$, k_1s_{22}, k_2s_{11}, k_2s_{12}, $s_{11}s_{21}$, $s_{11}s_{22}$, $s_{12}s_{21}$, $s_{12}s_{22}\}$; for each variable x_i, $1 \leq i \leq n$, the set $\{x_i\overline{x_i}, p_ik_2\}$; for each clause $c_j = (l_1^j \vee l_2^j \vee l_3^j)$, $1 \leq j \leq m$, $\{t_1^jl_1^j$, $t_2^jl_2^j$, $t_3^jl_3^j\}$.

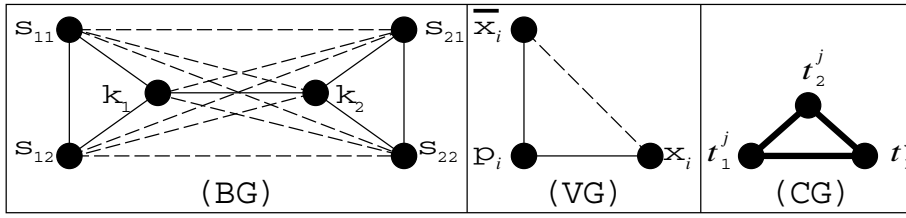


Fig. 1. Base Graph (BG), Variable Gadget (VG) and Clause Gadget (CG).

We call $(2, 1)$ *Base graph* the subgraph of $G^2 = (V, E^2)$ induced by $\{k_1, k_2, s_{11}, s_{12}, s_{21}, s_{22}\}$ (see Figure 1(BG)). For each $i \in \{1, \ldots, n\}$, we call *Variable gadget* the subgraph of $G^2 = (V, E^2)$ induced by $\{x_i, \overline{x_i}, p_i\}$ (see Figure 1(VG)). For each $j \in \{1, \ldots, m\}$, we call *Clause gadget* the subgraph of $G^2 = (V, E^2)$ induced by $\{t_1^j, t_2^j, t_3^j\}$ (see Figure 1(CG)). Lemmas 1 and 2 prove the required equivalence for establishing Theorem 1.

Lemma 1. *If the particular instance (V, E^1, E^3) of* $(2, 1)$-graph sandwich problem *constructed above admits a graph $G = (V, E)$ such that $E^1 \subseteq E$ and $E \cap E^3 = \emptyset$ and G is $(2, 1)$, then there exists a truth assignment that satisfies (X, C).*

Proof. Suppose there exists a $(2, 1)$ sandwich graph $G = (V, E)$ with $(2, 1)$ partition (S_1, S_2, K) where S_1, S_2 are independent sets and K is a clique.

Claim 1. $k_1, k_2 \in K$ and $s_{11}, s_{12}, s_{21}, s_{22} \in S_1 \cup S_2$.

Proof of Claim 1: Since $S_1 \cup S_2$ induce a bipartite subgraph in G, any triangle induced in G^1 must have at least one of its vertices in K. Hence, at least one vertex of the triangle induced by k_1, s_{11} and s_{12}; and at least one vertex

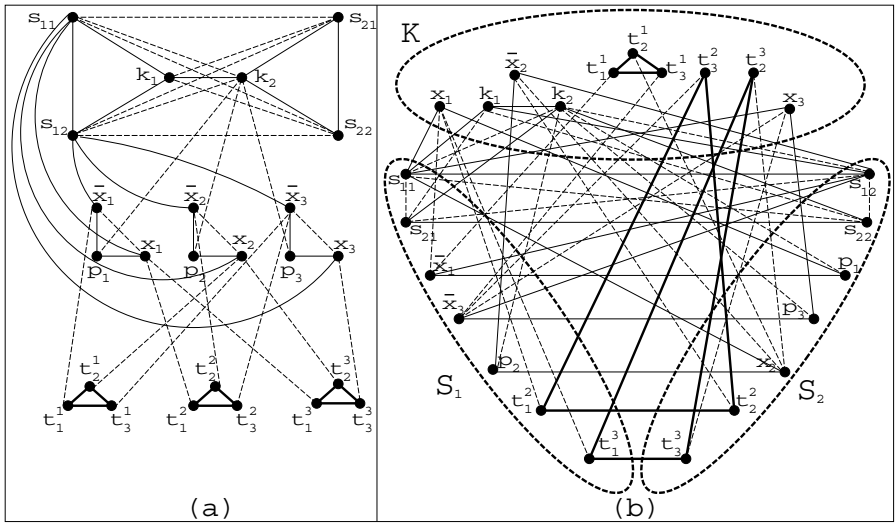


Fig. 2. (a) - Instance (V, E^1, E^3) obtained from the satisfiable instance of 3SAT: $I = (U, C) = (\{x_1, x_2, x_3\}, \{(\bar{x}_1 \vee x_2 \vee \bar{x}_3), (x_1 \vee \bar{x}_2 \vee \bar{x}_3), (x_1 \vee x_2 \vee x_3)\})$ and (b) - respective partition for the $(2,1)$ graph G defined from the satisfying truth assignment $x_1 = F$, $x_2 = T$, $x_3 = F$.

of the triangle induced by k_2, s_{21} and s_{22} belong to K. Now, each vertex in $\{s_{11}, s_{12}, s_{21}, s_{22}\}$ is joined by E^3-edges to three vertices that induce a triangle in G^1. If one of the vertices of $\{s_{11}, s_{12}, s_{21}, s_{22}\}$ belonged to K, then this would force at least one triangle to have no vertices in K, a contradiction. Thus, we must have $\{s_{11}, s_{12}, s_{21}, s_{22}\} \subseteq S_1 \cup S_2$, and $\{k_1, k_2\} \subseteq K$. □

Both $\{s_{11}, s_{12}\}$ and $\{s_{21}, s_{22}\}$ induce edges in G^1, which force $\{s_{11}, s_{12}\} \cap S_i \neq \emptyset$, $\{s_{21}, s_{22}\} \cap S_i \neq \emptyset$, $i = 1, 2$. We assume with no loss of generality that $s_{11}, s_{21} \in S_1$, which implies $s_{12}, s_{22} \in S_2$. In case the particular instance (V, E^1, E^3) admits a $(2,1)$ sandwich graph $G = (V, E)$ any $(2,1)$ partition (S_1, S_2, K) for G satisfies $S_1, S_2, K \neq \emptyset$.

Claim 2. For each $i \in \{1, \ldots, n\}$, $p_i \in S_1 \cup S_2$, $x_i \in K \cup S_2$ and $\overline{x_i} \in K \cup S_1$.

Proof of Claim 2: Since $p_i k_2 \in E^3$ and $k_2 \in K$, we have that p_i cannot be in K. In addition, $x_i s_{11}, \overline{x_i} s_{12} \in E^1$ and $s_{11} \in S_1$, $s_{12} \in S_2$, we have respectively $x_i \in K \cup S_2$ and $\overline{x_i} \in K \cup S_1$, $i \in \{1, ..., n\}$. □

Observe that since $x_i p_i \in E^1$ and $x_i \overline{x}_i \in E^3$, we have that if $x_i \in K$, then $\overline{x}_i \in S_1$, which implies $p_i \in S_2$; if $x_i \in S_2$, then $p_i \in S_1$, which implies $\overline{x}_i \in K$. Therefore, for each $i \in \{1, ..., n\}$, exactly one vertex of $\{x_i, \overline{x}_i\}$ belongs to K.

Claim 3. For each $j \in \{1, \ldots, m\}$, at least one of the vertices $\{t_1^j, t_2^j, t_3^j\}$ must be in K.

Proof of Claim 3: Since $S_1 \cup S_2$ induce a bipartite subgraph in G, for each $j \in \{1, \ldots, m\}$, at least one of the vertices of the triangle induced in G^1 by $\{t_1^j, t_2^j, t_3^j\}$ must be in K. □

We now define the truth assignment for (X,C): for $i \in \{1,...,n\}$, variable x_i is false if and only if the vertex $x_i \in K$. Suppose that for some $j \in \{1,...,m\}$, the clause $c_j = (l_1^j \vee l_2^j \vee l_3^j)$ is false. By the construction of (V, E^1, E^3), there is an edge of E^3 between the vertex assigned to the literal l_k^j and the vertex t_k^j, $k \in \{1,2,3\}$. Hence, if the literal l_k^j is false, then its corresponding vertex is in K which implies that t_k^j cannot be in K. So, all vertices of the triangle induced in G^1 by $\{t_1^j, t_2^j, t_3^j\}$ must be in $S_1 \cup S_2$. By Claim 3, this is a contradiction to the hypothesis that S_1, S_2 and K is a $(2,1)$ partition of the set of vertices of G. Hence, the above defined truth assignment satisfies (X,C). This ends the proof of Lemma 1. □

The converse of Lemma 1 is given next by Lemma 2.

Lemma 2. *If there exists a truth assignment that satisfies (X,C), then the particular instance (V, E^1, E^3) of* $(2,1)$-GRAPH SANDWICH PROBLEM *constructed above admits a graph $G = (V,E)$ such that $E^1 \subseteq E$ and $E \cap E^3 = \emptyset$ and G is $(2,1)$.*

Proof. Suppose there is a truth assignment that satisfies (X,C). We shall define a partition of V into sets S_1, S_2 and K that in turn defines a solution G for the particular instance (V, E^1, E^3) of $(2,1)$-GRAPH SANDWICH PROBLEM associated with the 3SAT instance (X,C).

Place vertices $k_1, k_2 \in K$ and $s_{11}, s_{21} \in S_1$ and $s_{12}, s_{22} \in S_2$. For $i \in \{1,...,n\}$ if variable x_i is false then place vertices x_i in K, $\overline{x_i}$ in S_1 and p_i in S_2. Otherwise, if variable x_i is true, then place vertices x_i in S_2, $\overline{x_i}$ in K and p_i in S_1.

For $j \in \{1,...,m\}$ and $c_j = (l_1^j \vee l_2^j \vee l_3^j)$, place the corresponding vertices t_1^j, t_2^j, t_3^j as follows. For $k \in \{1,2,3\}$, if the literal l_k^j is false then place t_k^j in $S_1 \cup S_2$; otherwise, place t_k^j in K. Since the truth assignment satisfies (X,C), for each j, we have at most two vertices t_k^j in $S_1 \cup S_2$. In addition, in case two vertices t_k^j and t_p^j are placed in $S_1 \cup S_2$, place one in S_1 and the other one in S_2.

To show that (S_1, S_2, K) is a (2,1) partition for a sandwich graph $G = (V,E)$ we need to prove that there is no E^1 edge with both endnodes in S_1, there is no E^1 edge with both endnodes in S_2 and there is no E^3 edge with both endnodes in K.

By the above placement, s_{11}, s_{21} are in S_1, and $\overline{x_i}$, t_k^j and p_i can be in S_1, $i \in \{1,...,n\}$, $j \in \{1,...,m\}$, $k \in \{1,2,3\}$. The only possible forced edges between these vertices are: the edge $\overline{x_i}p_i$ which does not have both endnodes in S_1, because $\overline{x_i} \in S_1$ if the variable x_i is false and $p_i \in S_1$ if x_i is true; and the edge $t_k^j t_q^j$ which does not have both endnodes in S_1, $k \neq q, k, q \in \{1,2,3\}$. Hence, there is no E^1 edge with both endnodes in S_1.

In the same way, s_{12}, s_{22} are in S_2, and the vertices x_i, t_k^j, p_i can be in S_2, $i \in \{1,...,n\}$, $j \in \{1,...,m\}$, $k \in \{1,2,3\}$. The only possible forced edges between these vertices are: the edge $x_i p_i$ which does not have both endnodes in S_2, because $x_i \in S_2$ if the variable x_i is true and $p_i \in S_1$ if x_i is false; and the edge $t_k^j t_q^j$ which does not have both endnodes in S_1, $k \neq q$, $k, q \in \{1,2,3\}$. Hence, there is no E^1 edge with both endnodes in S_2.

For the set K we have that k_1, k_2 are in K, and the vertices x_i, $\overline{x_i}$, t_k^j can be in K, $i \in \{1, ..., n\}$, $j \in \{1, ..., m\}$, $k \in \{1, 2, 3\}$. The only possible forbidden edges between these vertices are: the edge $\overline{x_i}x_i$ which does not have both endnodes in K, because $\overline{x_i} \in K$ if and only if the variable x_i is true and $x_i \in K$ if and only if x_i is false; and the edges $x_i t_k^j, \overline{x_i} t_k^j$, by the above placement, we never have both vertices in K. Hence, there is no E^3 edge with both endnodes in K. And this ends the proof of Lemma 2. □

3 (2, 2)-Graph Sandwich Problem

In this section we prove that the $(2,2)$-GRAPH SANDWICH PROBLEM is *NP*-complete by reducing the *NP*-complete problem 3SAT to $(2,2)$-GRAPH SANDWICH PROBLEM.

$(2,2)$-GRAPH SANDWICH PROBLEM
Instance: Vertex set V, forced edge set E^1, forbidden edge set E^3.
Question: Is there a graph $G = (V, E)$, such that $E^1 \subseteq E$ and $E \cap E^3 = \emptyset$, and G is $(2,2)$?

Theorem 2. *The* $(2,2)$-GRAPH SANDWICH PROBLEM *is NP-complete.*

Proof. In order to reduce 3SAT to $(2,2)$-GRAPH SANDWICH PROBLEM we need to construct a particular instance (V, E^1, E^3) of $(2,2)$-GRAPH SANDWICH PROBLEM from a generic instance (X, C) of 3SAT, such that C is satisfiable if and only if (V, E^1, E^3) admits a sandwich graph $G = (V, E)$ which is $(2,2)$. First we describe the construction of a particular instance (V, E^1, E^3) of $(2,2)$-GRAPH SANDWICH PROBLEM; second we prove that every graph $G = (V, E)$ satisfying $E^1 \subseteq E$ and $E \cap E^3 = \emptyset$ and such that G is $(2,2)$, defines a truth assignment for (X, C); third we prove that every truth assignment for (X, C) defines a graph $G = (V, E)$ which is $(2,2)$ satisfying $E^1 \subseteq E$ and $E \cap E^3 = \emptyset$. The particular instance is explained in detail below. The required equivalence can be established following the steps of Theorem 1 and details are omitted. □

Construction of Particular Instance of (2, 2)-GRAPH SANDWICH PROBLEM

The vertex set V contains: an auxiliary set of vertices: $B_1 = \{k_1, k_2, s_{11}, s_{12}, s_{21}, s_{22}\}$; $B_2 = \{k_3, k_4, s_{31}, s_{32}, s_{41}, s_{42}\}$; for each variable x_i, $1 \leq i \leq n$, two vertices x_i, $\overline{x_i}$, corresponding to its literals and a vertex p_i; for each clause $c_j = (l_1^j \vee l_2^j \vee l_3^j)$, $1 \leq j \leq m$, three corresponding vertices t_1^j, t_2^j, t_3^j. See Figure 1, where solid edges denote forced E^1-edges and dashed edges denote forbidden E^3-edges.

The Forced Edge Set E^1 contains: edges between auxiliary vertices $\{k_1k_2$, k_3k_4, k_1s_{11}, k_1s_{12}, k_2s_{21}, k_2s_{22}, k_3s_{31}, k_3s_{32}, k_4s_{41}, k_4s_{42}, $s_{11}s_{12}$, $s_{21}s_{22}$, $s_{31}s_{32}$, $s_{41}s_{42}\}$; for each variable x_i, $1 \leq i \leq n$, the set $\{x_is_{11}$, $\overline{x_i}s_{12}$, x_ip_i, $\overline{x_i}p_i\}$; for each clause c_j, $1 \leq j \leq m$, the set $\{t_1^jt_2^j, t_1^jt_3^j, t_2^jt_3^j\}$.

The Forbidden Edge Set E^3 contains: edges between auxiliary vertices $\{k_1s_{21}$, k_1s_{22}, k_2s_{11}, k_2s_{12}, $s_{11}s_{21}$, $s_{11}s_{22}$, $s_{12}s_{21}$, $s_{12}s_{22}\} \cup \{k_3s_{41}$, k_3s_{42}, k_4s_{31}, k_4s_{32},

$s_{31}s_{41}$, $s_{31}s_{42}$, $s_{32}s_{41}$, $s_{32}s_{42}\} \cup \{uv : u \in B_1 \text{ and } v \in B_2\} \cup \{uv : u \in B_2 \text{ and } v \in V \setminus (B_1 \cup B_2)\}$; for each variable x_i,$1 \leq i \leq n$, the set $\{x_i\overline{x_i}, p_ik_2\}$; for each clause c_j, $1 \leq j \leq m$, $\{t_1^j l_1^j$, $t_2^j l_2^j$, $t_3^j l_3^j\}$.

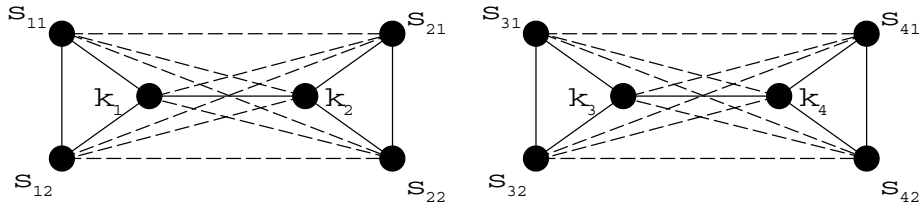


Fig. 3. $(2,2)$ Base Graph - all non-represented edges are E^3 edges.

Call $(2,2)$ *Base graph* the subgraph of $G^2 = (V, E^2)$ induced by $\{k_1, k_2, s_{11}, s_{12}, s_{21}, s_{22}, k_3, k_4, s_{31}, s_{32}, s_{41}, s_{42}\}$ (see Figure 3). As in the previous problem, we have two kinds of gadgets: *Variable gadget* (Figure 1(VG)) and *Clause gadget* (Figure 1(CG)). The special instance has a property similar to Theorem 1: if the particular instance (V, E^1, E^3) admits a $(2,2)$ sandwich graph $G = (V, E)$, then any $(2,2)$ partition (S_1, S_2, K_1, K_2) for G satisfies $S_1, S_2, K_1, K_2 \neq \emptyset$. Without loss of generality assume $k_1, k_2 \in K_1$, $k_3, k_4 \in K_2$, $s_{11}, s_{21}, s_{31}, s_{41} \in S_1$, $s_{12}, s_{22}, s_{32}, s_{42} \in S_2$. This implies $K_2 = \{k_3, k_4\}$.

4 (k,l)-Bounded Δ Graph Sandwich Problem

In this section, we consider the complexity of the (k,l)-Graph Sandwich problem when restricted to inputs having G^2 with bounded maximum degree.

(k,l)-Bounded Δ Graph Sandwich Problem ($(k,l) - B\Delta GSP$)
Instance: Vertex set V, forced edge set E^1, forbidden edge set E^3, where G^2 is a graph with no vertex degree exceeding Δ.
Question: Is there a graph $G = (V, E)$ such that $E^1 \subseteq E$ and $E \cap E^3 = \emptyset$ which is a (k,l) graph?

We completely classify the $(k,l) - B\Delta GSP$ as follows: $(k,l) - B\Delta GSP$ is polynomial for $k \leq 2$ or $\Delta \leq 3$, and NP-complete otherwise.

Lemma 3. *If $(k,l) - B\Delta GSP$ is solvable in polynomial time then the $(k,l+1) - B\Delta GSP$ is solvable in polynomial time.*

Proof. Let (V, E^1, E^3) be an instance for $(k,l+1) - B\Delta GSP$. Suppose that there exists a polynomial time algorithm $\mathcal{A}$ to solve the $(k,l) - B\Delta GSP$. We observe that if there exists a sandwich graph for (V, E^1, E^3) which is $(k,l+1)$ then a clique in G is also a clique in G^2. Thus, in order to define a polynomial time algorithm for $(k,l+1) - B\Delta GSP$ we proceed as follows: for each subset S with

less than or equal to $\Delta + 1$ vertices we verify if S induces a clique in G^2. In the affirmative case we apply the algorithm $\mathcal{A}$ to test if there exists a sandwich graph for the instance $(V \setminus S, E^1, E^3)$ of $(k,l) - B\Delta GSP$. Hence, we have designed an algorithm for $(k, l+1) - B\Delta GSP$ which runs in time $\mathcal{O}(n^{\Delta+1}P)$, where P is the order of the algorithm $\mathcal{A}$. □

Lemma 4. *If $k \leq 2$, then $(k,l) - B\Delta GSP$ is solvable in polynomial time.*

Proof. We argue by induction on l. As we said in the Introduction the $(1,0)$ and $(2,0)$-Graph Sandwich Problems are solvable in polynomial time, so are the corresponding problems $B\Delta GSP$. Suppose that for $k \leq 2$ and $l \geq 0$ the $(k,l) - B\Delta GSP$ is solvable in polynomial time. By Lemma 3 we have that the corresponding $(k, l+1) - B\Delta GSP$ is a polynomial time problem. □

Now, consider $k \geq 3$. Note that, as a consequence of Brook's Theorem [4], $(k,0)$ graph recognition is polynomial when restricted to inputs having $\Delta \leq 3$. This implies by Fact 2 that $(k,0) - B3GSP$ is solvable in polynomial time, and by Lemma 3, $(k,l) - B3GSP$ is also polynomial. However, by [8], $(k,0)$ graph recognition is *NP*-complete, even when restricted to inputs having $\Delta \leq 4$, which implies by Fact 1 that $(k,0) - B\Delta GSP$ is *NP*-complete, and as remarked in [1,2], $(k,l) - B\Delta GSP$ is *NP*-complete, for $\Delta \geq 4$.

5 Conclusion

We proved that the (k,l)-Graph Sandwich Problem is *NP*-complete for the cases $k = 1$ and $l = 2$; $k = 2$ and $l = 1$; or $k = l = 2$. We note that the basic idea of the construction of the particular instance of these problems is a simple necessary condition: if a graph is (k,l) then it does not contain $l+1$ independent cliques of size $k+1$. Recently, this condition was established sufficient for the class of the Chordal graphs, as proved by Hell, Klein, Protti and Tito [12]. In addition, we considered the degree Δ constraint subproblem $(k,l) - B\Delta GSP$ and completely classified the problem as follows: $(k,l) - B\Delta GSP$ is a polynomial problem for $k \leq 2$ or $\Delta \leq 3$; and *NP*-complete otherwise.

References

1. A. Brandstädt. *Partitions of Graphs into One or Two Independent Sets and Cliques.* Report **105** (1991), Informatik Berichte, Fern Universität, Hagen, Germany.
2. A. Brandstädt. *Partitions of Graphs into One or Two Independent Sets and Cliques.* Discrete Math., **152** (1996), 47–54. See also Corrigendum, **186** (1998), 295.
3. A. Brandstädt, V. B. Le and T. Szymczak. *The Complexity of some Problems Related to Graph 3-Colorability.* Discrete Appl. Math., **89** (1998), 59–73.
4. R. L. Brooks. *On Coloring the Nodes of a Network.* In *Proc. Cambridge Philos. Soc.*, **37** (1941) 194–197.
5. M. Cerioli, H. Everett, C. M. H. de Figueiredo, and S. Klein. *The homogeneous set sandwich problem.* Inform. Process. Lett., **67** (1998), 31–35.

6. C. M. H. de Figueiredo, S. Klein and K. Vusković. *The Graph Sandwich Problem for 1-Join Composition is NP-complete.* Discrete Appl. Math., **121** (2002) 73–82.
7. T. Feder, P. Hell, S. Klein, and R. Motwani. *Complexity of graph partition problems.* In Proc. of the 31st Annual ACM Symp. on Theory of Computing - STOC'99, (1999) 464–472.
8. M.R. Garey, D.S. Johnson and L.Stockmeyer. *Some Simplified NP-complete Graph Problems.* Theor. Comput. Sci., **1** (1976) 237–267.
9. M. C. Golumbic. *Matrix sandwich problems.* Linear Algebra Appl., **277** (1998), 239–251.
10. M. C. Golumbic, H. Kaplan, and R. Shamir. *Graph sandwich problems.* J. Algorithms, **19** (1995), 449–473.
11. M. C. Golumbic and A. Wassermann. *Complexity and algorithms for graph and hypergraph sandwich problems.* Graphs Combin., **14** (1998), 223–239.
12. P. Hell, S. Klein, F. Protti and L. Tito. *On Generalized Split Graphs.* In Brazilian Symp. on Graphs, Algorithms and Combinatorics, Electronic Notes in Discrete Math. **7** (2001).
13. H. Kaplan and R. Shamir. *Pathwidth, bandwidth, and completion problems to proper interval graphs with small cliques.* SIAM J. Comput., **25** (1996), 540–561.
14. H. Kaplan and R. Shamir. *Bounded degree interval sandwich problems.* Algorithmica, **24** (1999), 96–104.

Algorithms and Models for the On-Line Vertex-Covering (Extended Abstract)

Marc Demange[1] and Vangelis Th. Paschos[2]

[1] ESSEC, France
demange@essec.fr
[2] LAMSADE, Université Paris-Dauphine, Paris, France
paschos@lamsade.dauphine.fr

Abstract. In on-line computation, the instance of a problem is revealed step-by-step and one has, at the end of each step, to irrevocably decide on the part of the final solution dealing with this step. We first study the minimum vertex-covering problem under two on-line models corresponding to two different ways vertices are revealed. The former one implies that the input-graph is revealed vertex-by-vertex. The second model implies that the input-graph is revealed per clusters, i.e., per induced subgraphs of the final graph. Under the cluster-model, we then relax the constraint that the choice of the part of the final solution dealing with each cluster has to be irrevocable, by allowing backtracking. We assume that one can change decisions upon a vertex membership of the final solution, this change implying, however, some cost depending on the number of the vertices changed. Finally we study simple model where instance is revealed edge-by-edge. Most of the results we present are tight and optimal, or asymptotically optimal.

1 Introduction

On-line computation is very natural in real world applications since there exist situations modeled as problems for which the final data-set is not a priori known. In other words, data are revealed step-by-step. Frequently, when one tries to solve problems modeling such situations, many types of constraints (for example, deadlines on the final solution delivery, deadlines on the implementation of the solution computed) force her/him to start problem's solution before the whole set of data is completely revealed. On the other hand, these constraints may be strict enough forcing so the problem solver to irrevocably decide on the part of the final solution dealing with each part of data revealed, or may be relatively weak, allowing her/him to go back over decisions previously taken about the partial solution computed at each step.

Let Π be an **NP** optimization graph-problem. The *on-line* version of Π will be denoted by LΠ. An *on-line* algorithm $\mathtt{A}$ decides at each step which of the data (vertices or edges) revealed during this step will belong to the final solution. Its performance is measured in terms of the so-called competitive ratio $c_{\mathtt{A}}$ defined,

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 102–113, 2002.

for an instance G, as the ratio of the value of the solution computed by it when running on G to the value of a solution computed off-line, i.e., by an algorithm running once the final graph is completely known. In this paper we deal with deterministic on-line algorithms. On-line graph problems studied until now are, to our knowledge, the traveling salesman ([1]), the graph-coloring ([2,3,4]) and the independent set ([5]).

In this paper we study an almost paradigmatic computer science problem, the minimum vertex-covering problem, denoted by VC in the sequel, and defined as follows: *given a graph* $G(V, E)$*, compute the minimum-cardinality set* $V' \subseteq V$ *such that,* $\forall v_i v_j \in E$*, at least one of the* v_i, v_j *belongs to* V'. We consider that G (we set $n = |V|$ and suppose n known at the beginning of the game) is revealed per non-empty clusters, i.e., per induced subgraphs $G_1(V_1, E_1), G_2(V_2, E_2), \ldots$ of G (we denote by n_i the order of G_i, $i = 1, 2, \ldots$). Every time a new cluster G_i is revealed, the edges linking the vertices of G_i with the vertices of G_j, $j < i$, are also revealed. We denote by t the number of clusters needed so that the whole graph is completely revealed.

We first focus ourselves on the case where graph is revealed by means of its vertices and consider $t = n$, i.e., that G is revealed vertex-by-vertex. We establish the competitive ratio of a very simple but very natural on-line algorithm and show that its competitive ratio is asymptotically optimal..

Next, we generalize our study assuming $t < n$ and study the competitive ratio of (more complicated) on-line algorithms for LVC against an optimal off-line algorithm. Here we distinguish and analyze two cases: $2 < t < n$ and $t = 2$ and we provide analyses shown to be optimal or asymptotically optimal.

We then assume non-irrevocability in the construction of the on-line solution, i.e., by allowing backtracking. This means that the algorithm can interchange a number of vertices in the solution computed by a number of vertices not included in it. But we consider that changes performed imply a cost on the vertices changed. We study the competitiveness (against an optimal off-line algorithm) of two algorithms under two cost models. The former implies that the cost paid for any change is fixed, while the latter implies that for any vertex changed one has to pay a cost equal to the total number of vertices changed. Our analyses are, here also, asymptotically optimal.

Finally, we study a slightly different on-line model, where we assume that the input-graph is revealed edge-by-edge. Together with the arrival of a new edge, are revealed the links of its endpoints with the ones of the edges already revealed. Here also we devise an on-line algorithm and study its competitive ratio against an optimal off-line one. Let us note that for this part of the paper also results are quasi-optimal.

Of course, the study of on-line vertex-covering is interesting per se. However, as we show in [6], LVC is not simply a toy-problem since real production planning problems (described in [6]) are modeled as instances of LVC. Furthermore, all the cases of LVC studied here correspond to natural versions of these problems.

For reasons of length of the paper, basic notions used in the sequel as *matching*, *exposed vertex*, *augmenting path*, *independent set* are not defined here. Their

respective definitions can be found in [7]. Finally, a set will be called minimal (resp., maximal) with respect to a property π if it satisfies π, while deletion (resp., insertion) of an element from (resp., in) S results in a set not satisfying π.

In what follows, we denote by Δ the maximum degree of $G(V, E)$, by $\tau(G)$ the cardinality of a minimum vertex cover of G, by M (resp., M_i) a maximum matching of G (resp., G_i), by m (resp., m_i) the cardinality of M (resp., M_i), and by P (resp., P_i) the set of the exposed vertices of G (resp., G_i) with respect to M (resp., M_i). Denote also by $X(M)$ (resp., $X(M_i)$) the set of the endpoints of M (resp., M_i). Given $V' \subseteq V$, we denote by $G[V']$ the subgraph of G induced by V'. Finally, for $v \in V$, we denote by $\Gamma(v)$ the set of neighbors of v, i.e., $\Gamma(v) = \{u : uv \in E\}$, $\Delta = \max\{|\Gamma(v)| : v \in V\}$.

Note 1. Consider a graph $G(V, E)$, fix a maximal matching M of G and let $P = V \setminus X(M)$. Then, any (maximal) independent set of G is the complement, with respect to V, of a (minimal) vertex-cover of G. Moreover, (i) P is independent for G and (ii) $X(M)$ is a vertex-cover of G with $|X(M)| = 2m$.

2 On-Line Vertex-Covering with $t = n$

We first consider that G is revealed into $t = n$ clusters, i.e., vertex-by-vertex. Before specifying an on-line algorithm for this case, we establish a general result for any algorithm (on-line or off-line) computing a minimal vertex-cover, i.e., a vertex-cover that cannot be reduced by elimination of some of its vertices.

We denote by `MAX_MATCHING` an algorithm computing a maximum matching M of G (the problem of finding a maximum matching of a graph is polynomial ([8])). By (ii) of note 1, $X(M) = \{v_i, v_j : v_iv_j \in M\}$ is a vertex-cover of size $2|M|$ for G. Denote by m the size of M. Finally, set $p = |P| = |V \setminus X(M)|$. The following lemma will be used in theorem 1, just below, and later.

Lemma 1. *For any graph without isolated vertices, $p \leqslant m(\Delta - 1)$. If, in addition, the graph contains ι isolated vertices, then $p - \iota \leqslant m(\Delta - 1)$.*

Proof (Sketch). Fix an edge $v_iv_j \in M$ such that at least one of v_i, v_j has neighbors in P. Let $P_i = P \cap \Gamma(v_i) = \{p_{i_1}, p_{i_2}, \ldots, p_{i_{|P_i|}}\}$; P_j is defined similarly. Suppose $|P_i| \geqslant |P_j|$. Then, if $P_j \neq \emptyset$, the following holds ([6]): $P_j \subseteq P_i$ and $|P_j| = 1$.

Consider now the following graph $B(N, E_B)$ constructed as follows: for every edge of M we draw a vertex (let N_M be the set of vertices so drawn); we also consider vertices of P as vertices of N; in other words, $N = N_M \cup P$; let n_{ij} be the vertex of N_M associated with $v_iv_j \in M$; if there exists an edge linking either v_i or v_j with a vertex $p_k \in P$, then $n_{ij}p_k \in E_B$.

By the discussion above, any $n_{ij} \in N_M$ is linked with at most $\Delta - 1$ vertices of P (the maximum degree of G is Δ and one edge – the matching one – links v_i with v_j). This remains true if $P_j = \emptyset$. Consequently, $p \leqslant m(\Delta - 1)$.

On the other hand, if G contains a set I of ι isolated vertices, then the argument developed above remains valid on the graph $G'(V \setminus I, E)$, q.e.d. □

Since $n = 2m + p$ and, by lemma 1, $p \leqslant m(\Delta - 1)$ (resp., $p - \iota \leqslant m(\Delta - 1)$), one easily gets $n \leqslant m(\Delta + 1)$ (resp., $n - \iota \leqslant m(\Delta + 1)$) and reaches the following lemma.

Lemma 2. *In any graph with no isolated vertices, $m \geqslant n/(\Delta + 1)$. If the graph has ι isolated vertices, then $m \geqslant (n - \iota)/(\Delta + 1)$.*

Theorem 1. *For any graph G, the ratio of the size of any minimal vertex-cover to the size of the vertex-cover induced by `MAX_MATCHING(G)` is bounded above by $\Delta/2$.*

Proof (Sketch). Assume first that G has no isolated vertices, denote by C a minimal vertex cover of G and by s the size of the independent set associated with C, i.e., $|V \setminus C| = s$. Denote by m the size of a maximum matching M of G. By (ii) of note 1, algorithm `MAX_MATCHING` induces a vertex-cover of size $2m$. Recall finally that, by (i) of note 1, $P = V \setminus X(M)$ is independent.

Since C is supposed minimal, the independent set $V \setminus C$ is maximal. Consequently, using $s \geqslant n/(\Delta + 1)$ ([7]), we get $|C| \leqslant \Delta s$.

We now distinguish the following two cases depending on the values of s and m: (i) $s \leqslant m$, (ii) $s > m$. For case (i), using the expression for $|C|$, we get: $|C|/2m \leqslant \Delta/2$. For case (ii), using lemma 1, we also get: $|C|/2m \leqslant \Delta/2$.

Consider now that G contains a set I of isolated vertices. Then, C being minimal, it does not contain any isolated vertex. Furthermore, `MAX_MATCHING`$(G[V \setminus I])$ = `MAX_MATCHING`(G). Hence, the analysis performed just above remains valid. □

It is well-known ([7]) that, for any graph G and for any maximal matching M (of cardinality m) of G,

$$\tau(G) \geqslant m \tag{1}$$

Corollary 1. *For any graph, the ratio of the size of any minimal vertex-cover to the size of an optimal one is bounded above by Δ.*

We now analyze the competitiveness of a natural on-line algorithm, denoted by `OLVC`. It works as follows: suppose that vertices are numbered in the order they arrive; in step i, vertex v_i is revealed; `OLVC` puts it in the solution C, if there exists v_j, $j < i$, not included in C, linked to v_i. Obviously, the cover C so constructed is minimal.

Proposition 1. *The competitive ratio of algorithm `OLVC` against an optimal off-line algorithm for VC is bounded above by Δ and this bound is tight.*

Proof (Sketch). The ratio claimed is deduced by application of corollary 1 and of theorem 1.

Fix now a $\Delta \in \mathbb{N}$, consider a star $S_{\Delta+1}$ on $\Delta + 1$ vertices. Obviously, $\tau(S_{\Delta+1}) = 1$. Suppose that its center is the first vertex revealed; the rest of vertices can be revealed in any order. Then, algorithm `OLVC` will not include the star-center in C, while it will include all the remaining vertices of $S_{\Delta+1}$. Therefore, the competitive ratio achieved in this case is equal to Δ. □

In the rest of the section, we will provide lower bounds on the competitiveness of any algorithm for the case $t = n$. Recall that vertices are numbered in the order they arrive; in step i, vertex v_i is revealed. Also consider that, in step i, $\{v_1, \ldots, v_i\} = C_i \cup S_i$, where C_i draws the vertex-set included in the vertex cover under construction and $S_i = \{v_1, \ldots, v_i\} \setminus C_i$. The final graph is denoted, as usually, by $G(V, E)$ and its maximum degree by Δ. Our purpose is to provide limits in the competitiveness (against an optimal off-line algorithm) of any on-line algorithm solving LVC with $t = n$ (over all the ways the input-graph is revealed). Let us consider the solution of LVC as a two-players game, where the first one (player 1) reveals the instance and the second one (player 2) constructs the solution. Then, we prove the following theorem.

Theorem 2. *1. No algorithm can achieve competitive ratio better than Δ, even if an isomorphic of G is known in advance. 2. No algorithm can achieve competitive ratio strictly better than $\Delta - 2$, even if G is a tree and n is known in advance.*

Proof (Sketch). We first sketch the proof of 1. The isomorphic of G revealed in advance consists of a disjoint collection of p stars, each of order $\Delta + 1$ and of $\Delta - 1$ isolated vertices, where Δ and p are fixed integers. Obviously, the degree of G is Δ and its order $n = p(\Delta + 1) + \Delta - 1$. Assume that player 1 reveals the graph with respect to the following rules: **[i]** if C_i contains Δ isolated vertices (for the graph already revealed), then v_{i+1} is linked to all these vertices; **[ii]** if $v_i \in S_i$ (in other words, v_i has not been taken in the solution) and v_i is not linked to any vertex v_j, $j < i$, and if $i \leqslant n - \Delta$, then vertices $v_i, v_{i+1}, \ldots, v_{i+\Delta}$ form a star rooted in v_i; **[iii]** if p stars have been revealed, the rest of the vertices revealed are isolated; **[iv]** if rules **[i]** and **[ii]** cannot be applied and if $i \leqslant n - 1$, then vertex v_{i+1} is isolated with respect to the graph already revealed.

Applications of rules **[i]** to **[iv]** above implies that player 2 cannot do better than covering edges of any star by its leaves, while optimal off-line solution consists of the star-centers. Therefore a ratio of Δ is achieved at best.

We now sketch the proof of 2. Let Δ be an integer greater than, or equal to, 3 and set $n = \Delta(\Delta + 1) + 1$. Consider that player 1 reveals the graph according to the following rules: **(i)** if C_i contains Δ isolated vertices (with respect to the graph already revealed) and if $i \leqslant n - 2$, then v_{i+1} is linked to all these isolated vertices; **(ii)** if $v_i \in S_i$ (in other words, v_i has not been taken in the solution) and if v_i is not linked to any vertex v_j, $j < i$, and if $i \leqslant n - \Delta - 1$, then vertices $v_i, v_{i+1}, \ldots, v_{i+\Delta}$ form a star rooted in v_i; **(iii)** consider $v_i \in S_i$, v_i isolated with respect to the graph already revealed, and $n - 2 \geqslant i \geqslant n - \Delta$; set $A = \{v_j : j < i, v_j \in C_i, \forall k \leqslant i, v_j v_k \notin E\}$ (i.e., A is the set of the isolated vertices, at instant i, taken in C_i) and $B = \{v_{i+2}, \ldots, v_{n-1}\}$; then: **(iiia)** v_{i+1} is linked to v_i and to any element of set A and **(iiib)** the elements of B form an independent set and are linked to v_i; **(iv)** if rules **(i)** and **(ii)** do not apply and if $i \leqslant n - \Delta$, then vertex v_{i+1} is isolated with the graph already revealed; **(v)** v_n is linked to Δ vertices of degree 1 picked in the several connected components of the graph revealed until step $n - 1$.

If rule **(iii)** is not applied, then in step $n-1$, the graph contains Δ stars, their vertices of degree 1 making part of the solution constructed by player 2. In this case, $\tau(G) = \Delta+1$ (the roots of the stars plus vertex v_n), while the solution constructed is of size $\Delta(\Delta+1)$. The competitive ratio is in this case Δ. Suppose now that rule **(iii)** is applied. Then in step i, the graph consists of k stars (their leaves making part of the solution constructed by player 2) plus the vertices of $A \cup \{i\}$. Then, as we prove in [6], $\tau(G) = \Delta - 1 + 3 = \Delta + 2$, while the solution finally constructed by player 2 has at least $(\Delta-1)\Delta + \Delta = \Delta^2$ vertices. The competitive ratio implied is then at least $\Delta - 2$. □

3 On-Line Vertex Covering with $n > t \geqslant 2$

We assume in this section that G is revealed by non-empty clusters G_i, $i = 1, \ldots, t$, with $2 \leqslant t < n$. We assume first $n > t > 2$. For this case, we propose the following algorithm denoted by `t_OLVC`: when G_1 arrives, `t_OLVC` puts in the cover C the endpoints of a maximum matching on G_1; then for $i = 2, \ldots, t$ it includes in C the endpoints of a maximum matching M_i on G_i as well as exposed vertices of V_i with respect to M_i if they are linked to vertices of $\cup_{1 \leqslant j \leqslant i-1} V_j$ not included in C (the inclusion of these latter vertices is performed greedily).

Obviously, the set C finally computed by `t_OLVC` is a vertex-cover, although not necessarily minimal. So, proposition 1 does not represent the worst case for its competitive ratio. Note that for the case where clusters are assumed without any restriction, setting $|C| \leqslant n - |I|$ (where I denotes the set of isolated vertices, if any) and using lemma 2, competitive ratio $\Delta + 1$ is immediately deduced.

In theorem 3 just below, we suppose that for $i = 1, \ldots, t$, cluster G_i can eventually contain isolated vertices (with respect to G_i) when it arrives. The fact that the final graph G contains or does not contain isolated vertices does not change neither the result nor its proof.

Theorem 3. *Let λ_i be the number of the isolated vertices of G_i introduced in C and set $\lambda = \sum_{i=1}^{t} \lambda_i$. Denote by A_i, $i = 2, \ldots, t$, the sets of the exposed vertices, with respect to M_i, introduced in C by algorithm `t_OLVC` and set $A = \cup_{i=2}^{t} A_i$ and $\rho = \lambda/|A|$. Then, the competitive ratio of algorithm `t_OLVC` against an optimal VC algorithm is bounded above by $2 + (\Delta - 2)/(2 - \rho)$.*

Proof (Sketch). Observe that vertex-set A is exposed with respect to the (non-maximum) matching $\cup_{i=1}^{t} M_i$; moreover, it does not contain any isolated vertex. Observe also that any isolate vertex is exposed with respect to any matching of G; hence, $\rho \leqslant 1$. Denote by M_i, $i = 1, \ldots, t$, a maximum matching of G_i, set $m_i = |M_i|$, $i = 2, \ldots, t$. Let E' be the set of edges that have entailed introduction of the vertices of A in C and denote by $B(A \cup (V_1 \setminus X(M_1)), E')$ the partial subgraph of G induced by $A \cup (V_1 \setminus X(M_1))$ and by E'. Also, denote by M_B a maximum matching of B and by m_B the cardinality of M_B. Then, the cardinality of the on-line solution C computed by `t_OLVC` is $|C| = 2\sum_{i=1}^{t} m_i + |A|$, while $\tau(G) \geqslant (\sum_{i=1}^{t} m_i) + m_B$. Consequently, $c_{\texttt{t\_OLVC}} \leqslant 2 + (|A| - 2m_B)/(\sum_{i=1}^{t} m_i + m_B)$. Using lemma 2, we get $m_B \geqslant |A|/\Delta$. Also, we prove in [6] that $\sum_{i=1}^{t} m_i \geqslant (|A| - \lambda)/(\Delta - 1)$. So, $c_{\texttt{t\_OLVC}} \leqslant 2 + (\Delta - 2)/(2 - \rho)$. □

Since $\rho \leqslant 1$ and the competitive ratio obtained is increasing with ρ, setting $\rho = 1$ the following result is immediately obtained.

Corollary 2. *The competitive ratio of algorithm* `t_OLVC` *against an optimal VC algorithm is bounded above by* Δ.

Note that the solution C computed by algorithm `t_OLVC` is not necessarily minimal. Consequently, the result of corollary 2 cannot be derived by direct application of corollary 1. On the other hand, consider that clusters arrive without isolated vertices. In this case, for any $i = 1, \ldots, t$, $\lambda_i = 0$, so, $\rho = 0$ and the following holds.

Corollary 3. *Whenever clusters arrive without isolated vertices, the competitive ratio of algorithm* `t_OLVC` *against an optimal VC algorithm is bounded above by* $(\Delta + 2)/2$.

We show in [6] that the bound of the corollary 3 can be slightly improved by $(\Delta + 1)/2$.

Consider now case $t = 2$ and suppose that the input graph is revealed within two clusters $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$. Assume also that n, the order of the final graph, is known at the beginning of the game. We recall that, following our assumptions, one has to decide which vertices of the first cluster will belong to the final solution before the arrival of the second cluster. We distinguish two sub-cases for $t = 2$ depending on whether G has or has not isolated vertices.

We first deal with the latter one and suppose that no additional hypotheses are admitted on the forms of the clusters. We analyze the competitive ratio of the following algorithm, denoted by `2_OLVC`: if $|V_1| \leqslant n/2$, then the solution C returned by `2_OLVC` is the union of V_1 together with the endpoints of a maximum matching M_2 of G_2; otherwise, the solution returned by `2_OLVC` is the union of the endpoints of a maximum matching M_1 of G_1 together with V_2.

Theorem 4. *If G has no isolated vertices, then the competitive ratio of algorithm* `2_OLVC` *against an optimal VC algorithm verifies* $c_{\texttt{2\_OLVC}} \leqslant (\Delta+5)/2$. *This ratio is asymptotically tight.*

Proof (Sketch). Denote by m_i the size of M_i, $i = 1, 2$ and suppose that $C = V_1 \cup X(M_2)$; $|C| \leqslant n/2 + 2m_2$. Combining expression for C with expression (1) and taking into account lemma 2 and the fact that $m_2 \leqslant m$, then $c_{\texttt{2\_OLVC}} \leqslant (\Delta + 5)/2$. Suppose instead that $|V_2| \leqslant n/2$; then $C = X(M_1) \cup V_2$. In this case also, the arguments previously developed hold and the competitive ratio achieved is always $(\Delta + 5)/2$.

Let us show that the analysis above is asymptotically tight. Consider a graph $G(V, E)$, collection of R stars $S_{\Delta+1}$. Consider the subgraph G_1 of G consisting of a set of $n/2$ exposed vertices with respect to a maximum matching M of G. Remark that M contains one edge per star and that $V(G_1)$ is a set of isolated vertices of size not larger than $n/2$. Set $G_2 = G[V \setminus V(G_1)]$ and assume that G is revealed per clusters G_1 and G_2. Then, $|C| = (n/2) + 2n/(\Delta + 1)$, $\tau(G) = n/(\Delta + 1)$ and, consequently, $|C|/\tau(G) = (\Delta + 5)/2$. □

Assume now that clusters G_i arrive with no isolated vertices. Let n_i be the order of G_i. Denote by M_i, $i = 1, 2$, a maximum matching of G_i, by P_i the set of the exposed vertices with respect to M_i, by p_i its cardinality. The algorithm proposed, denoted by `C2_OLVC(n,`ϵ`)` for a fixed $\epsilon > 1$ works as follows: when G_1 arrives, it firstly computes M_1 and, if $n_1 \leqslant n/\epsilon$, then it sets $C = V_1$, else it sets $C = X(M_1)$; when G_2 arrives, it computes M_2 and adds $X(M_2)$ in C; it finally completes C with vertices of $V_2 \setminus X(M_2)$ (i.e., vertices of V_2 exposed with respect to M_2) linked to vertices of V_1 that have not been included in C (this last set of vertices added in C is denoted by A_2; we assume that its vertices are chosen greedily).

Theorem 5. *Under the hypothesis that clusters arrive with no isolated vertices, there exists ϵ_0, the largest among the roots of the polynomial $\epsilon^2 - 3\epsilon + 1$, such that the competitive ratio of algorithm `C2_OLVC(n,`ϵ_0`)` against an optimal VC algorithm is bounded above by $2 + ((\Delta + 1)/\epsilon_0)$.*

Proof (Sketch). Set, for $i = 1, 2$, $m_i = |M_i|$. If $n_1 \leqslant n/\epsilon$, then $A_2 = \emptyset$; therefore, the final covering C satisfies $C = V_1 \cup X(M_2)$. In this case, using expression (1) and lemma 2, we get:

$$\frac{|C|}{\tau(G)} \leqslant \frac{\Delta + 1}{\epsilon} + 2 \tag{2}$$

Let us now suppose that $n_1 > n/\epsilon$ instead. Then, the set C finally computed in by algorithm `C2_OLVC` verifies: $|C| = 2(m_1 + m_2) + |A_2|$.

Denote by $Q_1 \subseteq V_1 \setminus X(M_1)$ the set of vertices of V_1 that has entailed the introduction of set A_2 in C, and by $B(Q_1, A_2, E_B)$, the subgraph of G induced by $Q_1 \cup A_2$. Since they are both independent (subsets of P_1 and P_2, respectively), B is bipartite. Denote also by M_B a maximum matching of B and set $m_B = |M_B|$. Since $M_1 \cup M_2 \cup M_B$ is a matching for G, $\tau(G) \geqslant m_1 + m_2 + m_B$.

Consider the set $X(M_B) \cap Q_1$; obviously, $|X(M_B) \cap Q_1| = m_B$. Since G_1 is supposed without isolated vertices, any vertex of $X(M_B) \cap Q_1$ has at most $\Delta - 1$ neighbors in A_2. On the other hand, M_B being maximum for B, any vertex of A_2 receives edges from at least one vertex of $X(M_B) \cap Q_1$. So, $|A_2| \leqslant m_B(\Delta - 1)$. Also, since G_1 and G_2 are both assumed without isolated vertices, application of lemma 2 gives: $m_1 \geqslant n_1/(\Delta + 1)$ and $m_2 \geqslant n_2/(\Delta + 1)$.

The discussion above together with some easy algebra gives

$$\frac{|C|}{\tau(G)} \leqslant \frac{2\,(m_1 + m_2) + |A_2|}{m_1 + m_2 + m_B} = 2 + \frac{|A_2| - 2m_B}{m_1 + m_2 + m_B} \leqslant 2 + \frac{\frac{\Delta-3}{\Delta-1}\,|A_2|}{\frac{n}{\Delta+1} + \frac{|A_2|}{\Delta-1}} \tag{3}$$

Recall that we are currently considering the case $n_1 \geqslant n/\epsilon$, i.e., $n_2 \leqslant n(\epsilon - 1)/\epsilon$. Then, $|A_2| \leqslant p_2 = n_2 - 2m_2 \leqslant n(\Delta - 1)(\epsilon - 1)/((\Delta + 1)\epsilon)$. Remark also that expression (3) is increasing with $|A_2|$; hence,

$$\frac{|C|}{\tau(G)} \leqslant 2 + \frac{\frac{(\Delta-3)(\epsilon-1)}{(\Delta+1)\epsilon}\,n}{\frac{n}{\Delta+1} + \frac{(\epsilon-1)n}{\epsilon(\Delta+1)}} = 2 + \frac{(\Delta - 3)(\epsilon - 1)}{2\epsilon - 1} \tag{4}$$

Note that, for a fixed ϵ, $(\Delta-3)(\epsilon-1)/(2\epsilon-1) \leqslant (\Delta+1)(\epsilon-1)/(2\epsilon-1)$ and that expression (2) is decreasing with ϵ, while expression (4) is increasing. These two expressions asymptotically coincide when $(\Delta+1)/\epsilon = (\Delta+1)(\epsilon-1)/(2\epsilon-1)$, i.e., when $\epsilon^2 - 3\epsilon + 1 = 0$. Since, $\epsilon > 1$, the admissible root of the above equation is $\epsilon_0 \backsimeq 2.62$.

Setting $\epsilon_0 = 2.62$, we get $c_{\texttt{C2\_OLVC}} \leqslant (\Delta + 6.24)/2.62$. This, for large values of Δ, is asymptotically equal to $\Delta/2.62$. □

We provide in the next theorem upper bounds for the case where the number of clusters needed for the revealing of the whole graph is $O(\log n\sqrt{n})$ and for the case $t = 2$. As before, we bring to the fore graphs and strategies for revealing it in either $t = O(\log n\sqrt{n})$ steps (point 1), or in $t = 2$ steps (point 2), such that any on-line VC-algorithm cannot achieve competitive ratios better than the bounds claimed. The proofs for both cases, being quite technical and lengthy, are not given here. They can be found in [6]. In any case, the bounds presented in theorem 6 show that the results presented in theorems 3, 4 and 5 are asymptotically optimal.

Theorem 6. *1. When $t = O(\log n\sqrt{n})$ and any cluster is non-empty, no on-line algorithm for LVC can achieve competitive ratio smaller than $\Delta - 2$ against an optimal off-line algorithm, even if the input-graph is a tree and n is known in advance. 2. For $t = 2$ and for all $\Delta \geqslant 2$, no algorithm can achieve competitive ratio strictly better than $(\Delta + 1)/2$ for a graph of maximum degree Δ, even if it is bipartite with no isolated vertices, both clusters have the same size and an isomorphic of the input-graph is known in advance.*

4 Allowing Backtracking

In this section we somewhat change the working hypotheses adopted and suppose that one can go back over the solution constructed during previous steps. We assume that one can change this solution but she/he has to pay some cost for doing it.

Our on-line algorithm for the case of the backtracking is basically algorithm `t_OLVC`. The spirit of our thought process can be outlined as follows. The best approximation ratio known for VC is bounded above by 2. On the other hand, LVC being computationally harder, it is a priori worse approximated than VC. So, one can "restrain" her/himself in searching for competitive ratios as near as possible to 2. The maximum matching performed on each cluster of G by algorithm `t_OLVC` obviously guarantees approximation ratio 2 on any cluster. The fact that the whole competitive ratio is finally "deteriorated" is due to the vertices of the graph B that have to be taken into account in order to cover cross-edges, i.e., edges between clusters. We so analyze the following algorithm, denoted by `Bt_OLVC`: the solution-set C is initially assigned with the endpoints of maximum matchings M_i of G_i, $i = 1, \ldots, t$; next the graph B induced by the union of the vertex-sets $V_i \setminus X(M_i)$, $i = 1, \ldots, t$, is constructed and C is completed by the endpoints of a maximum matching M_B of B.

Let us denote by $\kappa > 1$ the cost due to the change of the status of a non-covering vertex to a covering one. Also, as previously, we denote by m_i the cardinality of M_i, $i = 1, \ldots, t$ and by m_B the cardinality of M_B.

Theorem 7. *The competitive ratio of algorithm* `Bt_OLVC` *against an optimal off-line algorithm for VC is bounded above by* 2κ.

Proof (Sketch). The vertices changed are the ones of the set $\cup_{i=1}^{t}(V_i \setminus X(M_i))$. Among these vertices, exactly $|X(M_B)| = 2m_B$ vertices pass from non-covering to covering ones. Suppose that for each of them a cost κ has to be paid. Then, $c_{\texttt{Bt\_OLVC}} = ((2\sum_{i=1}^{t} m_i) + 2\kappa m_B)/((\sum_{i=1}^{t} m_i) + m_B) \leqslant 2\kappa$. □

Theorem 7 draws the general case where no further specification of the type of the cost is given. We now assume two cost-models: (i) κ is a fixed constant, and (ii) κ is at most equal to the total number of vertices changed. For the first cost-model, using directly theorem 7, the following result is directly proved.

Corollary 4. *If a fixed cost has to be paid for any vertex-status modification, then the competitive ratio of algorithm* `Bt_OLVC` *against an optimal off-line VC-algorithm is constant.*

Let us now focus ourselves on the second of the cost-models specified above. For this case, we assume n known in advance and deal with the following algorithm, denoted by `Mt_OLVC`: (1) once G_1 arrives, the solution-set C receives the endpoints of a maximum matching M_1 of G_1; (2) for $i = 2, \ldots, t$, compute a maximum matching M_i of G_i, construct the subgraph B of G induced by the set $\cup_{j=1}^{i}(V_j \setminus X(M_j))$ and compute a maximum matching M_B of B: (2a) if $m_B \leqslant \sqrt{n}$, then add to C the endpoints of M_i; (2b) on the other hand, if $m_B > \sqrt{n}$, then add to C the whole set V_i; (3) let i_0 be the last $i \in \{2, \ldots, t\}$ for which step (2a) has been executed; construct the subgraph B' of G induced by the vertex-set $\cup_{j=1}^{i_0}(V_j \setminus X(M_j))$ and add to C the endpoints of a maximum matching $M_{B'}$ of B'.

Theorem 8. *If for any vertex changed, the cost of the change equals the total number of vertices changed, then the competitive ratio of algorithm* `Mt_OLVC` *against an optimal off-line VC-algorithm is bounded above by* $3\sqrt{n}$.

Proof (Sketch). Remark first that the only vertex-changes performed by algorithm `Mt_OLVC` are on $X(M_{B'})$ and, furthermore, that $m_{B'}$ always satisfies $m_{B'} \leqslant \sqrt{n}$.

If step (2b) is not executed at all, i.e., if $i_0 = t$, then $m_{B'} \leqslant \sqrt{n}$. Consequently, using theorem 7 and the expression for $m_{B'}$, we get $c_{\texttt{Mt\_OLVC}} \leqslant 2m_{B'} \leqslant 2\sqrt{n}$. On the other hand, suppose that step (2b) is executed at least once. Then, $m_B > \sqrt{n}$.

Using for $m_{B'}$ the expression given above, setting $\kappa = m_{B'}$ and denoting by $v(C)$ the value of the set C (in $v(C)$ any vertex non changed counts 1 and any vertex changed counts κ), we get: $v(C) \leqslant n - 2m_{B'} + 2\kappa m_{B'} \leqslant n + 2m_{B'}^2$. Denote by m the cardinality of a maximum matching of G. Then, $c_{\texttt{Mt\_OLVC}} = v(C)/\tau(G) \leqslant (n + 2m_{B'}^2)/m \leqslant (n/m_B) + 2m_{B'} \leqslant 3\sqrt{n}$. □

We now show that the result of theorem 8 is quite tight, since no on-line algorithm can achieve competitive ratio (against an optimal off-line one) better than $O(\sqrt{n})$.

Theorem 9. *Under the hypotheses of theorem 8, no on-line algorithm for VC can achieve, against an optimal off-line algorithm, competitive ratio $\sqrt{2n}$, even if the input-graph is bipartite without isolated vertices and n is known in advance.*

Proof (Sketch). Let $(\Delta, n_1) \in \mathbb{N} \times \mathbb{N}$ and set $n = (1+\Delta)n_1$. At the first step, V_1 is an independent set of size n_1. If player 2 selects some vertices of V_1, then the whole instance is a graph without any edge. In this case, the optimal value $\tau^*(G)$ is 0, whereas the on-line value is positive. The resulting ratio equals ∞ and the theorem holds.

Consequently, we can focus ourselves on the case where player 2 selects no vertices during the first step. In this case, the instance graph consists of n_1 stars of size $(1+\Delta)$ rooted in V_1, one star per vertex in V_1. Then, the optimal value satisfies (recall that $n = (1+\Delta)n_1$) $\tau^*(G) = n/(\Delta+1) = n_1$.

Denote by V_1' the set of vertices of V_1 that are changed in order to be included in the final solution (i.e., the vertices introduced in the solution after the backtracking). Then, solution C can be written as $C = V_1' \cup V_2 \setminus \Gamma(V_1')$ for a total cost of $v(C) = |V_1'|^2 + \Delta(n_1 - |V_1'|)$. Therefore, player 2 chooses, at best, a set V_1' of cardinality $\beta^* \in \mathrm{Argmin}_{\lambda \in [0, n/(\Delta+1)]}\{\beta^2 - \Delta\beta + ((\Delta n)/(\Delta+1))\}$. Define $\Delta = 2n_1$. One can easily show that the expression for β^* implies $\beta^* = n_1$. So, combining the expression for $\tau^*(G)$ with the one for $v(C)$ (with $\beta^* = n_1$), we get (after some easy functional analysis) $v(C)/\tau^*(G) = n_1 = (-1+\sqrt{1+8n})/4 \geqslant \sqrt{2n}$.

□

5 A Simple On-Line Model Based Upon Edges

We consider in this section an on-line model supposing that the input-graph is revealed by means of its edges. They arrive one at a time. For every new edge, the links of its endpoints with the endpoints of the edges already present are also revealed. We suppose that $|E|$ is known in advance, we set $E = \{e_1, \ldots, e_{|E|}\}$, where e_i are numbered in order of their arrival, and devise the following algorithm, dented by `E_OLVC`: for any edge arriving, if no endpoint of it is already in the solution-set C, then put in C both of its endpoints.

As one can see from the algorithm above the irrevocability in the construction of the on-line solution deals with the endpoints of an edge as a whole. With respect to a model based upon arrival of vertices it is as one allows, for every edge arriving, a backtracking of level one.

Proposition 2. *The competitive ratio of algorithm `E_OLVC` against an optimal off-line algorithm is bounded above by 2. This bound is tight.*

Proof (Sketch). In order to prove the competitive ratio claimed, just remark that the `FOR`-loop of algorithm `E_OLVC` computes a maximal matching of G.

Consider a star revealed edge-by-edge. Algorithm `E_OLVC` will introduce in C the endpoints of the first edge revealed and no new vertex will be introduced in C later. The optimal vertex-cover for any star consists of its center. So here, the bound 2 is attained. □

It is easy to see that the on-line model just described is equivalent to the one where all vertices are present from the beginning of the game and edges are presented one-by-one. Here, whenever an edge arrives none of the endpoints of which are in C, then both of its endpoints enter C.

Observe that in the model considered in this section, the competitiveness of the on-line algorithm proposed matches the approximation of the best known heuristic for the off-line version of VC.

6 Conclusions

The vertex-covering problem dealt in this paper is one of the central problems in combinatorial optimization in its off-line version. It remains very natural even in its on-line version. We have studied algorithms for several on-line models of LVC. The positive results obtained combined with the negative ones show that the analyses presented are "quasi-optimal" in the sense that no spectacular improvements are to be expected for the models dealt here. A further generalization of the vertex-covering is the one where we consider weights on the vertices of the input-graph and we search for a minimum total-weight vertex cover. Analysis of on-line algorithms for this weighted version of LVC seems to us a very interesting open problem.

References

1. Ausiello, G., Feuerstein, E., Leonardi, S., Stougie, L., Talamo, M.: Algorithms for the on-line traveling salesman problem. Algorithmica **29** (2001) 560–581
2. Gyárfás, A., Lehel, J.: Online and first-fit colorings of graphs. J. Graph Theo. **12** (1988) 217–227
3. Halldórsson, M.M., Szegedy, M.: Lower bounds for on-line graph coloring. Theoret. Comput. Sci. **130** (1994) 163–174
4. Lovász, L., Saks, M., Trotter, W.: An on-line graph coloring algorithm with sublinear performance ratio. Discrete Math. **75** (1989) 319–325
5. Demange, M., Paradon, X., Paschos, V.T.: On-line maximum-order induced hereditary subgraph problems. In Hlaváč, V., Jeffery, K.G., Wiedermann, J., eds.: SOFSEM 2000—Theory and Practice of Informatics. Volume 1963 of Lecture Notes in Computer Science., Springer-Verlag (2000) 326–334
6. Demange, M., Paschos, V.T.: On-line vertex-covering. Technical Report 183, LAMSADE, Universit Paris-Dauphine (2001) Available on www_address: `http://www.lamsade.dauphine.fr/cahdoc.html#cahiers`
7. Berge, C.: Graphs and hypergraphs. North Holland, Amsterdam (1973)
8. Papadimitriou, C.H., Steiglitz, K.: Combinatorial optimization: algorithms and complexity. Prentice Hall, New Jersey (1981)

Weighted Node Coloring: When Stable Sets Are Expensive
(Extended Abstract)

Marc Demange[1], D. de Werra[2], J. Monnot[3], and Vangelis Th. Paschos[3]

[1] ESSEC, France
demange@essec.fr
[2] Ecole Polytechnique Fédérale de Lausanne, Switzerland
dewerra@dma.epfl.ch
[3] LAMSADE, Université Paris-Dauphine, Paris, France
{monnot,paschos}@lamsade.dauphine.fr

Abstract. A version of weighted coloring of a graph is introduced: each node v of a graph $G = (V, E)$ is provided with a positive integer weight $w(v)$ and the weight of a stable set S of G is $w(S) = \max\{w(v) : v \in V \cap S\}$. A k-coloring $\mathcal{S} = (S_1, \ldots, S_k)$ of G is a partition of V into k stable sets $S_1, \ldots, S_k$ and the weight of $\mathcal{S}$ is $w(S_1) + \ldots + w(S_k)$. The objective then is to find a coloring $\mathcal{S} = (S_1, \ldots, S_k)$ of G such that $w(S_1) + \ldots + w(S_k)$ is minimized. Weighted node coloring is **NP**-hard for general graphs (as generalization of the node coloring problem). We prove here that the associated decision problems are **NP**-complete for bipartite graphs, for line-graphs of bipartite graphs and for split graphs. We present approximation results for general graphs. For the other families of graphs dealt, properties of optimal solutions are discussed and complexity and approximability results are presented.

1 Introduction

A k-coloring of $G = (V, E)$ is a partition $\mathcal{S} = (S_1, \ldots, S_k)$ of the node set V of G into stable sets S_i. The objective is here to determine a node coloring minimizing k. A natural generalization of the problem, denoted by WC in what follows, is the one where a strictly positive integer weight $w(v)$ is considered for any node $v \in V$, and where the weight of stable set S of G is $w(S) = \max\{w(v) : v \in S\}$. Then, the objective is to determine $\mathcal{S} = (S_1, \ldots, S_k)$ a node coloring of G minimizing the quantity $\sum_{i=1}^{k} w(S_i)$. This problem is easily shown **NP**-hard; it suffices to consider $w(v) = 1$, $\forall v \in V$ and WC becomes the classical node coloring problem.

In [1] we show that WC is not a toy problem. In terms of scheduling, a weighted node coloring amounts to assigning each job v to a time-slot (or period) i in such a way that no two jobs u, v assigned to the some time slot i are incompatible. In our situation, the lengths of the time slots $1, 2, \ldots, k$ are not given in advance; assuming that the jobs scheduled in time slot i may be processed simultaneously, the amount of time needed will be given by $w(S_i) =$

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 114–125, 2002.

$\max\{w(v) : v \in S_i\}$. As a consequence, the total amount of time needed to complete all jobs will be $\text{val}(\mathcal{S}) = \sum_{i=1}^{k} w(S_i)$. where $\mathcal{S} = (S_1, \ldots, S_k)$. The problem then amounts to finding for a weighted graph $G_w = (V, E, w)$ a coloring $\mathcal{S} = (S_1, \ldots, S_k)$ such that $\text{val}(\mathcal{S})$ is minimum. This problem is related to the *batch scheduling* problem which has been studied by several authors (see for instance [2] for a survey, or [3] for a special case). In the papers on batch scheduling, there are usually incompatibility constraints between operations belonging to a same job, or precedence constraints. The general case of incompatibility requirements represented by an arbitrary graph is formulated in [4], where they consider the complement of our graph: edges indicate compatibilities and they partition the node set into cliques. On the other hand, several types of requirements are introduced, like sequencing constraints or limitations in the size of a batch.

After establishing approximation results for the weighted coloring problem in general graphs, we examine some special cases, dealing with bipartite graphs, split graphs and cographs. We also study the weighted edge coloring problem in bipartite graphs. For all these cases, complexity issues as well as approximability will be discussed. For graph theoretical terms not defined here, the reader is referred to [5].

2 General Properties

The following proposition describes a general property which will be needed later.

Proposition 1. *Consider an instance of WC given by a weighted graph* $G = (V, E, w)$ *and a coloring* $\mathcal{S}'$. *We can always construct in polynomial time a* k*-coloring* $\mathcal{S} = (S_1, \ldots, S_k)$ *verifying* $\text{val}(\mathcal{S}) \leqslant \text{val}(\mathcal{S}')$ *and* $k \leqslant \Delta(G) + 1$.

Proof (Sketch). Set $\mathcal{S}' = (S'_1, S'_2, \ldots)$ and rank the S'_i's in decreasing weight order. Take S_i (the ith component of the coloring $\mathcal{S}$) such that $S_i \supseteq S'_i$ is a maximal stable set in $G \setminus S'_1 \setminus \ldots \setminus S'_{i-1}$. □

In particular, this result holds for an optimal weighted node coloring of G. If H is the line-graph of G, denoted by $L(G)$, we have the following.

Corollary 1. *If* $G = L(H)$, *then the solution* $\mathcal{S}$ *of Proposition 1 verifies* $k \leqslant 2\Delta(H) - 1$.

We can easily show that in Corollary 1 we have $k \leqslant p(\omega(G) - 1) + 1$ where $\omega(G)$ is the maximum cardinality of a clique in G and p is the maximum number of (maximal) cliques in which one node of G is contained. If G is a line-graph $L(H)$ then $p = 2$ and $\omega(G) = \Delta(G)$, so Corollary 1 follows. Also, it follows from Proposition 1 that the number k of colors in an optimal k-coloring $\text{val}(\mathcal{S})$ can be bounded above by any bound on the chromatic number which is derived by a sequential coloring algorithm which gives maximal stable sets in the subgraph generated by the colored nodes. In particular the bounds of Welsh-Powell and of Matula are valid for k (see, for instance, [6]).

We can also establish the following property of optimal k-colorings $\mathcal{S}$ in a weighted graph $G = (V, E, w)$ for $w(v) \in \{t_1, \ldots, t_r\}$ with $t_1 > \ldots > t_r$ for each node v.

Proposition 2. *Let $G = (V, E, w)$ be a r-valued weighted graph and let $q = \chi(G)$ be its chromatic number. Then every optimal k-coloring $\mathcal{S}^* = (S_1^*, \ldots, S_k^*)$ satisfies: $w(S_i^*) > w(S_{i+q-1}^*)$, for any $i \leqslant k - q$. In particular, $k \leqslant 1 + r(q-1)$. This bound is tight.*

Proof (Sketch). Assume that there exists an index $i \leqslant k - q$ such that $w(S_i^*) = \ldots = w(S_{i+q-1}^*)$; $S_i^* \cup \ldots \cup S_k^*$ induces a subgraph G' verifying $\chi(G') \leqslant \chi(G)$. Thus, we can change sets $S_i^*, \ldots, S_k^*$ by q other sets to obtain a $q+i-1$-coloring with a lower cost, a contradiction. □

3 Approximating Weighted Coloring in General Graphs

In this section, we establish approximability results for the weighted coloring problem defined in section 1. We use two approximation-quality criteria called in what follows *standard approximation ratio* and *differential approximation ratio*, respectively. Consider an instance I of an **NP**-hard optimization problem Π and a polynomial time approximation algorithm $\mathtt{A}$ solving Π; we will denote by $\text{worst}(I)$, $\text{val}_{\mathtt{A}}(I)$ and $\text{opt}(I)$ the values of the worst solution of I, of the approximated one (provided by $\mathtt{A}$ when running on I), and the optimal one for I, respectively. If Π is a maximization (resp., minimization) problem, the value $\text{worst}(I)$ is in fact the optimal solution of a minimization (resp., maximization) problem Π' having the same objective function and the same constraint set as Π. Let us note that computation of the solution realizing $\text{worst}(I)$ can be easy for some **NP**-hard problems (this is the case of graph coloring) but for other ones (for example, for traveling salesman, or for optimum satisfiability, or for minimum maximal independent set) this computation is **NP**-hard. Commonly, the quality of an approximation algorithm for Π is expressed by the ratio (called standard in what follows) $\rho_{\mathtt{A}}(I) = \text{val}_{\mathtt{A}}(I)/\text{opt}(I)$. On the other hand, the differential approximation ratio measures how the value of an approximate solution is placed in the interval between $\text{worst}(I)$ and $\text{opt}(I)$. More formally, it is defined as $\delta_{\mathtt{A}}(I) = |\text{worst}(I) - \text{val}_{\mathtt{A}}(I)|/|\text{worst}(I) - \text{opt}(I)|$. A very optimistic configuration for both standard and differential approximations is the one where an algorithm achieves ratios bounded below by $1-\epsilon$ ($1+\epsilon$ for the standard approximation for minimization problems), for any $\epsilon > 0$. We call such algorithms *polynomial time approximation schemes*. The complexities of such schemes may be polynomial or exponential in $1/\epsilon$ (they are always polynomial in the sizes of the instances). When they are polynomial in $1/\epsilon$ the schemes are called *fully polynomial time approximation scheme*.

The standard approximation result presented in this section is based upon the so-called *master-slave approximation strategy*. Consider an **NP**-hard minimization covering graph-problem consisting in covering the nodes of the input graph G, of order n, by subgraphs G' verifying a certain property π. Most of these

problems can be approximated by the following strategy: (a) find a maximum subgraph G' of G verifying π; (b) delete $V(G')$ from V; repeat steps (a) and (b) in the remaining graph until $V = \emptyset$. The maximization problem solved at step (a) is called the *slave*, while the original minimization problem is called the *master*. These terms are due to [7] who points out the fact that if the slave problem is polynomial then the master problem is approximable within $O(\log n)$. A classical example of master-slave approximation for graph coloring, using maximum stable set as slave problem, is given in [8].

Proposition 3. *([9]) In the master-slave approximation game for weighted problems, if the weighted slave problem is approximable within ratio $\rho(n)$, then the weighted master problem is approximable within standard ratio $O(\log n/\rho(n))$.*

For our problem, the (maximization) slave problem, denoted by SLAVE_WC, consists of *determining a stable S^* maximizing quantity $|S|/w(S)$, over any stable set S, where $w(S) = \max\{w(v) : v \in S\}$*. Consequently, the overall algorithm `W_COLOR` we devise for weighted coloring can be outlined as follows: (1) solve SLAVE_WC in G; let $\hat{S}$ be the solution obtained; set $V = V \setminus \hat{S}$, $G = G[V]$; (2) color the nodes of $\hat{S}$ with a new color; repeat steps (1) and (2) until all the nodes of the input graph are colored.

Lemma 1. *SLAVE_WC is approximable in polynomial time within standard ratio $O(\log^2 n/n)$.*

Proof (Sketch). Consider the following algorithm, called `SLAVE_WC` in the sequel: (1) rank the nodes of V in nonincreasing weight-order; let L the list obtained; (2) for any $v \in L$ do: (2a) set $V_v = \{u \in L : w(u) > w(v)\}$, $V = V \setminus (V_v \cup (\{v\} \cup \Gamma(v)))$, $G = G[V]$; (2b) run the maximum stable algorithm of [10] on G; let S_v be the stable set computed; store set $S^v = S_v \cup \{v\}$ as candidate solution for SLAVE_WC; (2c) return to the original graph G; (3) among the sets stored in step (2b), choose one, denoted by $\hat{S}$, maximizing quantity $|S^v|/w(v)$. We prove in [1] that algorithm `SLAVE_WC` achieves, for problem SLAVE_WC the same ratio, $O(\log^2 n/n)$, as the algorithm of [10], called in step (2b) for stable set, this ratio being the best known, in terms of n for the latter problem. □

Using Proposition 3 and Lemma 1, the following holds for algorithm `W_COLOR`.

Proposition 4. *The weighted coloring problem can be approximately solved in polynomial time within standard approximation ratio $O(n/\log n)$.*

We now deal with differential approximation and present a polynomial time approximation algorithm guaranteeing a differential approximation ratio bounded below by a fixed constant. Consider a graph $G = (V, E, w)$, where w is the vector of the node-weights of G. Then, our algorithm, denoted by `DW_COLOR` works as follows: [a] construct an edge-weighted graph $\bar{G} = (V, E', w')$ where $\bar{G}$ is the complement of G and for any $e = [v, u] \in E'$, $w'(e) = \min\{w(u), w(v)\}$; [b] compute a maximum-weight matching M^* of $\bar{G}$; [c] color the endpoints of any edge of M^* with a new color; [d] color every exposed node of V (with respect to M^*) with a new color. The solution computed `DW_COLOR` is a collection of stable sets of size 2 and of singletons.

Proposition 5. *The differential approximation ratio achieved by* DW_COLOR *is bounded below by 1/2. This bound is tight.*

Proof (Sketch). Denote by $\mathcal{S}^* = (S_1^*, \ldots, S_p^*)$ an optimal weighted coloring and by $\mathrm{val}_{\bar{G}}(M)$ the value of any maximum weight matching M of $\bar{G}$. For any $G[S_i^*]$, consider a maximum weight matching M_i', set $M' = \cup_{i=1}^{p} M_i'$ and apply steps [b] to [d] of DW_COLOR starting from M'; denote by $\mathcal{S}'$ the coloring so obtained. Then, $\mathrm{val}(\mathcal{S}') = \mathrm{worst}(G) - \mathrm{val}_{\bar{G}}(M') \leqslant (\mathrm{worst}(G) + \mathrm{opt}(G))/2$. Finally, since $\mathrm{val}_{\bar{G}}(M^*) \geqslant \mathrm{val}_{\bar{G}}(M')$, the result claimed is easily deduced. The tightness of the ratio is proved in [1] by considering an 1-valued graph G_m induced by a matching of size m. □

Note that algorithm DW_COLOR computes an optimal solution when $\alpha(G) \leqslant 2$.

We finish this section by two inapproximability results. Consider any class $\mathcal{G}'$ of graphs and a node-weighted graph $G \in \mathcal{G}'$ and suppose that WC is **NP**-complete for any $G \in \mathcal{G}'$. Then, the following holds.

Proposition 6. *For any class $\mathcal{G}'$ of node-weighted graphs: if WC($\mathcal{G}'$) is **NP**-complete, then, unless $\boldsymbol{P} = \boldsymbol{NP}$, for any $c \in \mathbb{N}$, $c \geqslant 1$, no polynomial time algorithm can compute a solution of WC in any class of graphs such that the difference between its value and the optimal value is bounded above by c; furthermore, if WC($\mathcal{G}'$) is strongly **NP**-complete, then, unless $\boldsymbol{P} = \boldsymbol{NP}$, WC($\mathcal{G}'$) cannot be solved neither by a standard nor by a differential fully polynomial time approximation scheme.*

4 The Bipartite Case and Some Related Cases

4.1 The Bipartite Graphs

In this section $G = (V, E, w)$ will be a weighted bipartite graph where L (resp. R) is the "left set" (resp. "right set") of nodes and each edge has one endpoint in L and the other in R. An instance of WC is given by a bipartite weighted graph G with a positive integer q. Let $\mathrm{WC}(G, q)$ be the following problem: does there exist a coloring $\mathcal{S}$ of G with $\mathrm{val}(\mathcal{S}) \leqslant q$?

Proposition 7. *WC(G, q) is **NP**-complete in the strong sense even if G is a bipartite graph of maximum degree at most 14.*

Proof (Sketch). We use a reduction from 1-PrExt ([11]): "given a bipartite graph $G = (V, E)$ with $|V| \geqslant 3$ and three nodes v_1, v_2, v_3, does there exist a 3-coloring (S_1, S_2, S_3) of (the nodes of) G such that $v_i \in S_i$ for $i = 1, 2, 3$?" Consider an instance of 1-PrExt given by a bipartite graph and specific nodes v_1, v_2, v_3. It is immediate to see that we may assume $\{v_1, v_2, v_3\} \subseteq L$. We introduce three new nodes u_1, u_2, u_3 in R and edges $[v_i, u_j]$ for $i \neq j$ and $1 \leqslant i, j \leqslant 3$. In the new bipartite graph G' we associate weights $w(u_i) = w(v_i) = 2^{3-i}$ for $i = 1, 2, 3$ and $w(v) = 1$ for every other node v in G'. Then we set $q = 7$ and we consider problem $\mathrm{WC}(G', 7)$. There exists a coloring $\mathcal{S}$ of G' with $\mathrm{val}(\mathcal{S}') \leqslant 7$ if and only if there exists a 3-coloring (S_1, S_2, S_3) of G with $v_i \in S_i$, $i = 1, 2, 3$. □

As a consequence of Proposition 7, WC is also **NP**-complete if G is a comparability graph (i.e., a graph whose edges can be transitively oriented, see [5]).

Proposition 8. *If $G = (V, E, w)$ is a bipartite weighted graph with bivalued weights, then one can construct an optimal k-coloring $\mathcal{S}$ in polynomial time.*

Proof (Sketch). By Proposition 2, an optimal solution is either a 2- or a 3-coloring. In the former case we can construct it by a greedy algorithm. For the latter case, if any optimal solution is a 3-coloring, then the set $V_{\max}$ of the maximum-weight nodes is stable (if not, there exists an optimal 2-coloring) and $\mathcal{S} = (V_{\max}, L \setminus V_{\max}, R \setminus V_{\max})$. □

We now propose a polynomial time approximation algorithm achieving a constant standard approximation ratio for WC in bipartite graphs. This algorithm, denoted by `BIP_WCOLOR` works as follows: (1) sort the nodes of G in nonincreasing weight order; let $\bar{L} = (v_1, v_2, \ldots, v_n)$ be the list obtained; (2) starting from v_1 color the nodes of $\bar{L}$ with color c whenever it is possible; (3) optimally color the remaining uncolored nodes with at most two new colors b and g following the bipartition of G; store the solution obtained during steps (2) and (3); (4) compute a minimum-weight 2-coloring; store the solution obtained; (5) output the smallest between the solutions stored in steps (3) and (4).

As the bicoloring of a of a connected bipartite graph is unique, a minimum-weight 2-coloring is simply the unique bipartition of V. If the graph is not connected, then a minimum-weight 2-coloring can be easily computed by taking care of assigning the same color to all the heaviest color-classes of the connected components of G. In what follows, we denote by $w_{\max}$ (resp., $w_{\min}$) the largest (resp., smallest) node weight.

Proposition 9. *`BIP_WCOLOR` polynomially solves WC in bipartite graphs within standard approximation ratio bounded above by $4r_w/(3r_w + 2)$, where $r_w = w_{\max}/w_{\min}$. This bound is tight.*

Proof (Sketch). Obviously, the weight of color c equals $w_{\max}$. Suppose now that step (2) stops while a node of weight $w_{\max}/t$, for some $t > 1$, has been encountered. Then, $\mathrm{opt}(G) \geqslant w_{\max} + (w_{\max}/t) + w_{\min}$ (otherwise, the optimal solution for WC on G would be a 2-coloring). On the other hand, $\mathrm{val}_{\texttt{BIP\_WCOLOR}}(G) \leqslant w_{\max}(t + 2)/t$ if the final solution is the one of step (3) and $\mathrm{val}_{\texttt{BIP\_WCOLOR}}(G) \leqslant 2w_{\max}$ if the final solution is the one of step (4). Combination of the expressions above and some algebra show that the common value for both ratios is $4r_w/(3r_w + 2) \leqslant 4/3$. Tightness is shown in [1]. □

In the proof of Proposition 7, one can see that WC is **NP**-complete when $w_{\max} = 4$ and $w_{\min} = 1$. Here, algorithm `BIP_WCOLOR` yields ratio 7/8 and this ratio is the best possible. So the following holds.

Proposition 10. *Unless $\boldsymbol{P} = \boldsymbol{NP}$, for any $\epsilon > 0$ no polynomial time algorithm achieves a standard approximation ratio bounded above by $(8/7) - \epsilon$ for WC in bipartite graphs.*

We now deal with the differential approximation of WC in bipartite graphs. Consider the following algorithm, called C_SCHEME in what follows and run it with parameters G and a fixed constant $\epsilon > 0$: (a) rank the nodes of G in non-increasing weight and set $w_i = w(v_i)$, $i = 1, \ldots, n$; (b) set $\eta = \lceil 1/\epsilon \rceil$; set $S_L = \{v_{4\eta+3}, \ldots, v_n\} \cap L$; set $S_R = \{v_{4\eta+3}, \ldots, v_n\} \cap R$; (c) set $\tilde{\mathcal{S}}$ the best partition into stable sets of the nodes $v_1, \ldots, v_{4\eta+2}$; (d) output $\hat{S} = S_L \cup S_R \cup \tilde{\mathcal{S}}$.

Since η is a fixed constant, the whole complexity of C_SCHEME is linear in n. Denote now by G' the subgraph of G induced by the node-set $\{v_1, \ldots, v_{4\eta+2}\}$ and recall that $\tilde{\mathcal{S}}$ is optimal for G'.

Proposition 11. *For any fixed $\epsilon > 0$, the differential approximation ratio of C_SCHEME when called with inputs G and ϵ, is bounded below by $1 - \epsilon$.*

Proof (Sketch). We can easily see that $|\tilde{\mathcal{S}}| \leqslant 2\eta + 2$ and $\mathrm{val}(\tilde{\mathcal{S}}) = \mathrm{opt}(G') \leqslant \mathrm{opt}(G)$ (the relative proof is given in [1]). Then, $\epsilon(\mathrm{worst}(G') - \mathrm{opt}(G')) \geqslant 2w_{4\eta+2}$. Moreover, $\mathrm{opt}(G') \leqslant \mathrm{opt}(G)$. Hence, $\mathrm{val}_{\texttt{C\_SCHEME}}(G) \leqslant \mathrm{opt}(G') + 2w_{4\eta+2} \leqslant (1 - \epsilon)\mathrm{opt}(G') + \epsilon\mathrm{worst}(G') \leqslant (1 - \epsilon)\mathrm{opt}(G) + \epsilon\mathrm{worst}(G)$. □

4.2 The Split Graphs

To conclude the study of the bipartite case, we have to examine the situation of *split* graphs, i.e., graphs G in which the node set $V(G)$ can be partitioned into a stable set S and a clique K. These graphs can be considered as intermediate between bipartite graphs and complements of bipartite graphs. In this last case, WC is polynomial ($\alpha(G) \leqslant 2$, cf., Proposition 5).

Proposition 12. *WC is **NP**-complete in the strong sense if G is a split graph.*

Proof (Sketch). The reduction is from the Min-Set-Cover: given a collection $\mathcal{C} = (\overline{C}_i : i \in I)$ of subsets $\overline{C}_i$ of a set $\overline{S}$ and a positive integer q ($q \leq |I|$) does there exist a sub-collection $\mathcal{C}' = (\overline{C}_j : j \in J)$ with $|J| \leq q$ and $\cup_{j \in J} \overline{C}_j = \overline{S}$?

Let us construct a split graph G as follows. Each element $\overline{v}$ of $\overline{S}$ becomes a node v of a stable set S; each subset $\overline{C}_i$ in $\mathcal{C}$ corresponds to a node c_i of the clique K of G. The set $N(c_i)$ of neighbors of node c_i is given by: $N(c_i) = \{v : \overline{v} \in \overline{S}\} \setminus \{v : \overline{v} \in \overline{C}_i\}$. The weights are given by $w(c_i) = |I|$, $i \in I$, and $w(v) = |I| + 1$,$v \in S$. Now there exists a cover $\mathcal{C}' = (\overline{C}_j : j \in J) \subset \mathcal{C}$ with $\cup_{j \in J} \overline{C}_j = \overline{S}$ and $|\mathcal{C}'| = |J| \leq q$ if and only if there exists in G a k-coloring $\mathcal{S} = (S_1, \ldots, S_k)$ with $\mathrm{val}(\mathcal{S}) \leq |I|^2 + q$. □

The proof of Proposition 12 shows that the problem is **NP**-complete even if the weights can take only two values. It also follows from this proof that $\mathrm{WC}(G, q)$ is **NP**-complete if G is a chordal graph, since a split graph is a chordal graph ([5]).

5 An Edge Coloring Model

If the weighted graph $G = (V, E, w)$ is a line-graph $L(H)$, then our node coloring problem becomes an edge coloring problem in a graph H where the edges e have weights $w(e)$.

Proposition 13. *WC is **NP**-complete in the strong sense if G is the line-graph $L(H)$ of a regular bipartite multigraph H with $\Delta(H) = 3$.*

Proof (Sketch). We shall start from the following **NP**-complete problem called 2-SIM ([12]): given a bipartite regular multigraph $H = (V, E)$ and two disjoint (partial) matchings M_1^*, M_2^*, does there exist an edge 3-coloring (M_1, M_2, M_3) of H such that $M_i^* \subseteq M_i$ for $i = 1, 2$?

Replace any edge $e = [u, v]$ in M_2^* by edges $[u, v_e]$, $[v_e, u_e]_1$, $[v_e, u_e]_2$ and $[u_e, v]$ where u_e and v_e are new nodes and introduce $[u, v_e]$ and $[u_e, v]$ in M_2^* and $[v_e, u_e]_1$ in M_1^*. The resulting graph is still regular bipartite with degree 3. Let us give weights $w(e) = 2^{3-i}$ to all edges $e \in M_i^*$ for $i = 1, 2$ and weights $w(e) = 1$ to all remaining edges of H. Let $\widehat{H}$ be the resulting weighted graph. Then, by defining the weight $w(M_i)$ of a matching M_i as the maximum of the weights of the edges in M_i, we have the following: $\widehat{H}$ has an edge k-coloring $\widehat{\mathcal{M}} = (\widehat{M}_1, \ldots, \widehat{M}_k)$ with $\text{val}(\widehat{\mathcal{M}}) = w(\widehat{M}_1) + \ldots + w(\widehat{M}_k) \leqslant 7$ if and only if H has an edge 3-coloring $\mathcal{M} = (M_1, \ldots, M_3)$ with $M_i^* \subseteq M_i$ $(i = 1, 2)$. □

In what follows, we denote by $\text{EWC}(G_k, q)$ the edge coloring version of WC in k-regular bipartite graphs $G_k = (L, R, E)$.

Proposition 14. *EWC is strongly **NP**-complete in k-regular bipartite graphs with $k \geqslant 3$.*

Proof (Sketch). The proof is by induction. For $k = 3$, we use Proposition 13 and the gadget of figure 1 showing how one can transform a cubic bipartite multigraph G to a simple cubic bipartite graph B. Note that in any feasible edge coloring of B, $\{\text{color}(a), \text{color}(b)\} = \{\text{color}(a'), \text{color}(b')\}$.

Suppose that strong **NP**-completeness is true for $k-1$. We use the following reduction from $\text{EWC}(G_{k-1}, q)$ to $\text{EWC}(G_k, 3q)$. Consider a $(k-1)$-regular bipartite graph $G_{k-1} = (L, R, E)$ and denote by w_{k-1} is edge-weight vector. Remark that $|L| = |R|$ and let r_i and l_i be for $i = 1, \ldots, |L|$ the nodes of R and L, respectively. Construct a copy $G'_{k-1} = (L', R', E')$ of G_{k-1} $(L = L',\ R = R',\ E = E')$ and denote by r'_i and l'_i the nodes of R' and L', respectively. For $i = 1, \ldots, |L|$ link r_i with l'_i and l_i with r'_i. Set $w_k(e) = w_{k-1}(e)$ for $e \in E \cup E'$ and $w_k(e) = 2q$ for $e \in \{[r_i, l'_i], [l_i, r'_i] : i = 1, \ldots, |L|\}$. Obviously, G_k is k-regular. Then, there exists an edge coloring of weight q in G_{k-1}, iff there exists an edge coloring of weight $3q$ in G_k. □

We now study the special case where edge weights are bivalued.

Proposition 15. *$WC(L(H), q)$ can be solved in polynomial time if H is bipartite with weights $w(e) \in \{a, b\}$ on the edges.*

Proof (Sketch). In order to simplify the sketch, suppose $a = 1$ and $b = t$. Starting from H, we construct a network N and solve a particular flow problem. Let $E(s)$ be the set of edges e with weight $w(e) = s$ for $s = 1, t$. Let $\Delta(s)$ be the maximum degree of the partial graph $H(s)$ generated by the edges in $E(s)$ for $s = 1, t$. Clearly if $\Delta(t) = \Delta(H)$, then any edge $\Delta(H)$-coloring of H is optimal. Construct

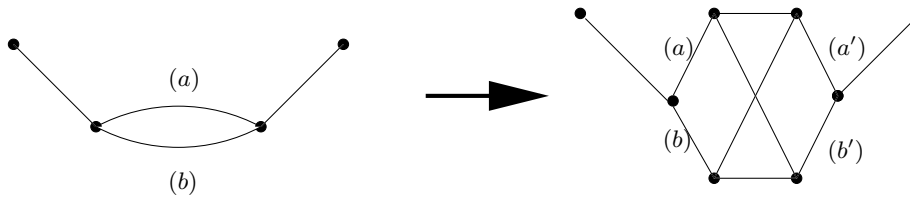


Fig. 1. Transformation of a cubic bipartite multigraph G into a simple cubic bipartite graph B.

a network $N(r)$ as follows: remove from H all edges in $E(t)$ and replace each edge $[u, v]$ in $E(1)$ by an arc $\overrightarrow{e} = (u, v)$ with capacity $c(\overrightarrow{e}) = 1$ and lower bound of flow $l(\overrightarrow{e}) = 0$; here r is a nonnegative integer. Introduce a source s_0 with arc (s_0, u) for each $u \in L$ which is adjacent in H to at least one edge of $E(1)$; set $l(s_0, u) = d_{H(1)}(u) - r$ and $c(s_0, u) = \Delta(t) - d_{H(t)}(u)$. In the same way, introduce a sink t_0 with arc (v, t_0) from each node v of R which is adjacent in H to at least one edge of $E(1)$; set $l(v, t_0) = d_{H(1)}(v) - r$ and $c(v, t_0) = \Delta(t) - d_{H(t)}(v)$. We have to find the smallest possible r for which $N(r)$ contains a feasible flow. Such an r will give us an edge $(\Delta(H(t)) + r)$-coloring $\mathcal{M}$ such that $\mathrm{val}(\mathcal{M}) = \Delta(H(t))t + r$. But such a coloring $\mathcal{M}$ may not be of minimum cost. We have to examine also edge k-colorings $\mathcal{M} = (M_1, \ldots, M_k)$ where $w(M_i) = t$ for the first $\Delta(H(t)) + \ell$ matchings and minimize the number r of matchings M_j with $w(M_j) = 1$. This can be done by the network flow algorithm described above by increasing the capacity of all arcs (s_0, u) and (v, t_0) by ℓ units. We will have to do this for $\ell = 0$ to $\Delta(H) - \Delta(H(t))$. □

In [13] it is shown that WC is **NP**-complete if G is the line graph $L(H)$ of a complete bipartite graph $K_{n,n}$; the nodes of $L(H)$ have degree $2n - 2$. The interest of the above proof is to deal with the case of fixed degrees, for any fixed constant. In addition [13] states Proposition 15 for the special case of the line graph of $K_{n,n}$.

We now deal with the approximation of EWC. Remark first that, by König's theorem ([14]), the optimal solution of the (unweighted) edge covering achieves standard approximation ratio Δ for EWC, for any $\Delta \geqslant 3$, where Δ is the maximum degree of the input graph G.

In what follows in this section, we restrict ourselves to bipartite graphs of maximum degree $\Delta = 3$. We are given a bipartite graph G; denote by w the edge-weight vector and, for $E' \subseteq E$, by $G[E']$ the partial subgraph of G induced by E', and consider the following algorithm `EW_COLOR`, when we assume that the set $E = \{e_1, e_2, \ldots, e_{|E|}\}$ of edges of G is ranked in decreasing weight order and, for any $j \in \{1, \ldots, |E|\}$, we set $E_j = \{e_1, \ldots, e_j\}$: (1) set $M_1^1 = M_2^1 = \ldots = M_{|E|}^1 = \emptyset$; (2) for $i = 1$ to $|E|$ do: set $j_0 = \min\{j = 1, \ldots |E| : M_j^1 \cup \{e_i\}$ is a matching$\}$; set $M_{j_0}^1 = M_{j_0}^1 \cup \{e_i\}$; (3) set $\mathcal{S}_1 = (M_1^1, \ldots, M_{r_1}^1)$ the list of the non-empty matchings of $(M_1^1, M_2^1, \ldots, M_{|E|}^1)$; set $k_0 = \max\{j : G[E_j]$ has maximum degree at most $2\}$; (4) for $\ell = 2$ to k_0 do: (4a) compute

an optimal 2-coloring (M_1^ℓ, M_2^ℓ) for $G[E_\ell]$; (4b) complete (M_1^ℓ, M_2^ℓ) by running steps (1) to (2) in $G \setminus G[E_\ell]$; (4c) set $\mathcal{S}_\ell = (M_1^\ell, M_2^\ell, \ldots, M_{r_\ell}^\ell)$ the edge coloring computed in steps (4a) and (4b); (5) output $\mathcal{S} = \text{argmin}\{\text{val}(\mathcal{S}_\ell) : \ell = 1, \ldots, k_0\}$.

Any set $\mathcal{S}_\ell$ computed by algorithm `EW_COLOR` verify Corollary 1; hence, $r_\ell \leqslant 5$.

Proposition 16. *`EW_COLOR` achieves standard approximation ratio 5/3 in polynomial time. This ratio is tight.*

Proof (Sketch). Following the remark just above on the value of r_ℓ, one can set $\mathcal{S}_\ell = (M_1^\ell, \ldots, M_5^\ell)$, (some of the M_i^ℓ, $i = 1, \ldots, 5$ may be empty). Fix an optimal solution $\mathcal{S}^*$ and denote by M_1^*, M_2^*, M_3^* the three largest matchings of $\mathcal{S}^*$. Set $i_3^* = \min\{j : e_j \in M_3^*\}$. By construction, $G[E_{i_3^*-1}]$ has maximum degree at most 2 and hence $w(M_1^{i_3^*-1}) + w(M_2^{i_3^*-1}) \leqslant w(M_1^*) + w(M_2^*)$ and $w(M_3^{i_3^*-1}) + w(M_4^{i_3^*-1}) + w(M_5^{i_3^*-1}) \leqslant 3w(M_3^*)$. We so finally obtain $\text{val}_{\text{EW\_COLOR}}(\mathcal{S}) \leqslant \text{val}(\mathcal{S}_{i_3^*-1}) \leqslant 5\text{opt}(G)/3$. The proof of the tightness is shown in [1]. □

The same analysis as the one in the proof of Proposition 16 concludes that EWC is approximable within standard approximation ratio bounded above by $(2\Delta - 1)/3$, for any $\Delta \geqslant 3$.

Proposition 17. *Unless* $\mathbf{P} = \mathbf{NP}$*, for any* $\epsilon > 0$ *no polynomial time algorithm achieves approximation ratio bounded above by* $(2^k/(2^k-1)) - \epsilon$*, even in* k*-regular bipartite graphs.*

Proof (Sketch). From the proofs of Propositions 13 and 14, where, in the latter, we change cost $w_k(e)$ to $2\max\{w_{k-1}(e)\}$ (this case remains **NP**-complete), one can see that EWC in regular bipartite graphs of degree at least k is **NP**-complete whenever the optimal value of the instance is at most $2^k - 1$. □

We now give a differential approximation result for EWC. As previously we first assume $G = (L, R, E)$ is a bipartite graph of maximum degree $\Delta = 3$ and with edge-weight vector w, and consider the following algorithm, denoted by `EC_SCHEME` in what follows: set $k = \lceil 1/\epsilon \rceil$; rank the edges in E in decreasing-weight order; set $E = \{e_1, \ldots, e_{|E|}\}$; set $E' = \{e_1, e_2, \ldots, e_{3k+5}\}$; optimally color $G[E']$ and greedily complete the edge coloring of step obtained in order to color E with at most three colors (in other words, omit weights and color the unweighted version of G).

Proposition 18. *Algorithm `EC_SCHEME` is a polynomial time differential approximation scheme for EWC.*

Proof (Sketch). Let $(M_1^*, \ldots, M_r^*)$ be an optimal solution of $G[E']$. By Corollary 1, we can suppose $r \leqslant 5$. So, $\text{worst}(G[E']) - \text{opt}(G[E']) \geqslant 3kw(e_{3k+5}) \geqslant 3w(e_{3k+5})/\epsilon$, and $\text{val}_{\texttt{EC\_SCHEME}}(G) \leqslant \text{opt}(G[E']) + w(e_{3k+6}) + w(e_{3k+7}) + w(e_{3k+8})$. After some algebra and taking into account that edges in E are ranked in decreasing weight order, $\text{val}_{\texttt{EC\_SCHEME}}(G) \leqslant (1-\epsilon)\text{opt}(G) + \epsilon\text{worst}(G)$. □

One can easily see that the result of Proposition 18 holds also for any fixed $\Delta > 3$ and for any graph (not necessarily bipartite).

6 Cographs

The case of cographs (or equivalently graphs containing no induced chain P_4 on four nodes) has to be mentioned. These graphs, also called P_4-free graphs, are a subclass of the perfectly ordered graphs introduced in [15]; for the perfectly ordered graphs, an order θ on the node set V can be defined in such a way that for any induced subgraph G' of the original graph G the *greedy sequential algorithm* (GSC) based on the order θ' induced by θ on the nodes of G' gives a minimum coloring of G'. Here the GSC algorithm based on an order θ consists in examining consecutively the nodes as they occur in θ and coloring them with the smallest possible color. As observed in [6], a graph G is a cograph if and only if for all induced subgraphs G' of G the GSC based upon any order θ gives a coloring of G' in $\chi(G')$ colors.

Lemma 2. *If $G = (V, E, w)$ is a weighted cograph, then all optimal colorings $\mathcal{S} = (S_1, \ldots, S_k)$ satisfy $k = \chi(G)$.*

Proof (Sketch). Assume there exists an optimal k'-coloring $\mathcal{S}' = (S'_1, \ldots, S'_{k'})$ with $k' > \chi(G)$. We can order the nodes of G by taking consecutively the nodes of S'_1, those of S'_2 and so on. Using the resulting order θ we can apply the GSC algorithm which will produce a k-coloring $\mathcal{S} = (S_1, \ldots, S_k)$ with $k = \chi(G)$ (we have ordered $\mathcal{S}$ and $\mathcal{S}'$ by non-increasing costs). Each node $v \in S'_j$ will satisfy $v \in S_i$ with $i \leqslant j$ after applying GSC. Thus, we have $w(S'_i \cup \{v\}) = w(S'_i)$ and $S_{k+1} = \emptyset$, a contradiction. □

We can now show that there is a polynomial algorithm which constructs an optimal k-coloring $\mathcal{S}$; such a result can be expected from graphs like cographs for which several generally difficult coloring problems are easier ([16]).

Proposition 19. *Let $G = (V, E, w)$ be a a weighted cograph. Then, the k-coloring $\mathcal{S}$, constructed by the GSC algorithm based upon any order θ where $u < v$ (u before v in θ) implies $w(u) \geqslant w(v)$, is optimal.*

Proof (Sketch). Let $t_1 > t_2 > \ldots > t_r$ be the values taken by the weights $w(v)$ in G. Every k-coloring $\mathcal{S} = (S_1, \ldots, S_k)$ of G with $k = \chi(G)$ and $w(S_1) \geqslant w(S_2) \geqslant \ldots \geqslant w(S_k)$ satisfies: $w(S_i) \geqslant \max\{t_s : \omega(G(s)) \geqslant i\}$ where $\omega(H)$ denotes the maximum size of a clique in a graph H and $G(s)$ is the subgraph generated by all nodes v with $w(v) \geqslant t_s$. Indeed any such k-coloring will have the first $\omega(G(1))$ sets S_i with $w(S_i) = t_1$; also the first $\omega(G(2))$ sets S_i will have $w(S_i) \geqslant t_2$ and generally the first $\omega(G(s))$ sets S_i will have $w(S_i) \geqslant t_s$.

Now consider the k-coloring $\overline{\mathcal{S}} = (\overline{S}_1, \ldots, \overline{S}_k)$ obtained by applying the GSC algorithm based on any order θ with nonincreasing weights. Let $p(s)$ be the largest color given to a node v with $w(v) = t_s$; let v_0 be such a node. Since cographs are perfectly ordered graphs, it follows by considering the subgraph G' of G generated by v_0 and all its predecessors in θ that there is in G' a clique $K \ni v_0$ with $K \cap \overline{S}_i \neq \emptyset$ for $i = 1, \ldots, p(s)$. So, $\overline{\mathcal{S}}$ satisfies $w(\overline{S}_i) = \max\{t_s : \omega(G(s)) \geqslant i\}$ and thus $\overline{\mathcal{S}}$ is an optimal coloring. □

The above proof shows in fact that if we are given a perfectly ordered graph G and if the order θ of nonincreasing weights in such that the GSC algorithm gives a minimum coloring (i.e., a k-coloring with $k = \chi(G)$), then one can find an optimal k-coloring $\mathcal{S}$ which minimizes val($\mathcal{S}$). For cographs, this condition was satisfied since any order θ could be chosen to construct a minimum coloring.

Proposition 19 is best possible in the following sense. If G is simply a P_4, then we may have no optimal k-coloring $\mathcal{S}$ with $k = \chi(G)$.

References

1. Demange, M., de Werra, D., Monnot, J., Paschos, V.T.: Time slot scheduling of compatible jobs. Cahier du LAMSADE 182, LAMSADE, Universit Paris-Dauphine (2001) Available on www_address: http://www.lamsade.dauphine.fr/cahdoc.html#cahiers.
2. Potts, C.N., Kovalyov, M.Y.: Scheduling with batching: a review. European J. Oper. Res. **120** (2000) 228–249
3. Potts, C.N., Strusevich, V.A., Tautenhahn, T.: Scheduling batches with simultaneous job processing for two-machine shop problems. J. of Scheduling **4** (2001) 25–51
4. Boudhar, M., Finke, G.: Scheduling on a batch machine with job compatibilities. JORBEL (2001) To appear.
5. Berge, C.: Graphs and hypergraphs. North Holland, Amsterdam (1973)
6. de Werra, D.: Heuristics for graph coloring. Computing **7** (1990) 191–208
7. Simon, H.U.: On approximate solutions for combinatorial optimization problems. SIAM J. Disc. Math. **3** (1990) 294–310
8. Johnson, D.S.: Approximation algorithms for combinatorial problems. J. Comput. System Sci. **9** (1974) 256–278
9. Alfandari, L., Paschos, V.T.: Master-slave strategy and polynomial approximation. Comput. Opti. Appl. **16** (2000) 231–245
10. Boppana, B.B., Halldórsson, M.M.: Approximating maximum independent sets by excluding subgraphs. BIT **32** (1992) 180–196
11. Bodlaender, H.L., Jansen, K., Woeginger, G.J.: Scheduling with incompatible jobs. Discrete Appl. Math. **55** (1994) 219–232
12. de Werra, D., Erschler, J.: Open shop scheduling with some additional constraints. Graphs and Combinatorics **12** (1996) 81–93
13. Rendl, F.: On the complexity of decomposing matrices arising in satellite communication. Oper. Res. Lett. **4** (1985) 5–8
14. König, D.: ber graphen und iher anwendung auf determinantentheorie und mengenlehre. Math. Ann. **77** (1916) 453–465
15. Chvtal, V.: Perfectly ordered graphs. In Berge, C., Chvtal, V., eds.: Topics on Perfect Graphs. (Volume 21 of Annals of Discrete Math.) 253–277
16. Jansen, K., Scheffler, P.: Generalized coloring for tree-like graphs. Discrete Appl. Math. **75** (1997) 135–155

The Complexity of Restrictive H-Coloring*

Josep Díaz, Maria Serna, and Dimitrios M. Thilikos

Dept. Llenguatges i Sistemes, Universitat Politecnica de Catalunya,
Jordi Girona Salgado 1–3, 08034 Barcelona, Spain
{diaz,mjserna,sedthilk}@lsi.upc.es

Abstract. We define a variant of the H-coloring problem where the number of preimages of certain vertices is predetermined as part of the problem input. We consider the decision and the counting version of the problem; namely the *restrictive H-coloring* and the *restrictive $\#H$-coloring* problems. We provide a dichotomy theorem characterizing the H's for which the restrictive H-coloring problem is either NP-complete or polynomially solvable. Moreover, we prove that the same criterion discriminates the #P-complete and the polynomially solvable cases of the *restrictive $\#H$-coloring* problem. Finally, we prove that both results apply also to the list versions of the above problems.

1 Introduction

Let us consider the following processing setting, we have a host network H of processors with communication links between them, and a set of jobs with communication demands between them, these jobs and their restrictions in their concurrent execution are modeled by a graph G. The goal is to make a suitable assignment of jobs to processors satisfying all the communication constrains. Historically, one good model to simulate these problems of assignation of interrelated jobs to interconnected processors has been through the H-coloring problem [9].

Formally, given two graphs G and H, a *homomorphism* from G to H is a function $\sigma : V(G) \to V(H)$ where for any edge $\{v, u\} \in E(G)$, $\{\sigma(v), \sigma(u)\}$ is also an edge of H. For fixed H we say that σ is an *H-coloring* of G. The graphs in this paper are finite, undirected, and cannot have multiple edges but can have loops. For a fixed graph H, the *H-coloring problem* asks for the existence of an H-coloring of the input graph G, while the *$\#H$-coloring* asks for the number of the H-colorings of the input graph G. The complexity of these two problems depends on the choice of the particular graph H. It is known that the H-coloring problem is polynomially time solvable if H is bipartite or it contains a loop; otherwise it is NP-complete [12]. Moreover, the $\#H$-coloring problem is polynomially solvable if all the connected components of H are either complete reflexive graphs or

* The work of all the authors was supported by the FET Program of the EU under contract number IST-99-14186 (ALCOM-FT) and by the Spanish CYCIT project TIC-2000-1970-CE. The work of the last author was partially supported by the Ministry of Education and Culture of Spain (Resolucin 31/7/00 - BOE 16/8/00).

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 126–137, 2002.

complete irreflexive bipartite graphs [7]. Recall that a *reflexive* graph has all its vertices looped, and that if none of the vertices of a graph is looped then it is *irreflexive*.

The processing may have further restrictions, in many practical cases, several qualitative restrictions are imposed by the guest network concerning the types of processors that are able to carry out each of the jobs. For this reason, each job may be accompanied with a list of the processors that can perform the task. More formally, for a fixed graph H, and given a graph G, a *list of preferences* is a function $L : V(G) \to 2^{V(H)}$. Given the pair (G, L) a *list H-coloring* of (G, L) is a homomorphism σ from G to H such that for any $v \in V(G)$, $\sigma(v) \in L(v)$. For a fixed graph H, the *list H-coloring* problem asks for the existence of a list H-coloring of the input, formed by a graph G and an associated list of preferences L, while the *list $\#H$-coloring* asks for the number of list H-colorings of the input. It is known that the list H-coloring problem is polynomially time solvable if H is a bi-arc graph [8,9,10]. Moreover, the list $\#H$-coloring problem is polynomially solvable if all the connected components of H are either complete reflexive graphs or complete irreflexive bipartite graphs [13,5].

In real systems the host network wants to keep bounded (or fixed) the load of some processors. Thus, some processors may carry an additional quantitative restriction the *number* of jobs that can be assigned to them. The goal is to make a suitable assignment of jobs to processors satisfying all the communication, load and preference constrains. A variant of the H-coloring problem, the so called *(H, C, K)-coloring* problem and variations whose complexity was studied in [6,5], considers the case in which the quantitative restriction is fixed independently of the graph G. See also [4] and [2] for more results on parameterized version of H-colorings. In this paper we consider the case in which the additional restriction may depend on the graph G, and thus form part of the input.

For a fixed graph H, given a graph G with n vertices, a *weighting function* is a function, $\mathsf{w} : V(H) \to \{0, \ldots, n, \infty\}$. Given the pair (G, w) a *restrictive H-coloring* of (G, w) is an H-coloring σ of G such that for all $a \in V(H)$ with $\mathsf{w}(a) \neq \infty$, $|\{u \in V(G) \mid \sigma(u) = a\}| = \mathsf{w}(a)$, when $\mathsf{w}(a) = \infty$ this set can have any number of vertices. Given a graph G, a list L, $S \subseteq V(H)$ and a weight function w on S a *restrictive list H-coloring* of G is a list H-coloring σ of (G, L) such that σ is also a (G, w) restrictive H-coloring. We introduce the following problems:

Restrictive H-coloring problem: Given a graph G and a weighting function w on H. Does (G, w) have a restrictive H-coloring?

Restrictive list H-coloring problem: Given a graph G, a list L on G, and a weighting function w on H. Does (G, L, w) have a restrictive list H-coloring?

The counting versions of both problems, the restrictive $\#H$-coloring and the restrictive list $\#H$-coloring are defined as usual.

We prove that all these problems are polynomial time solvable if all the connected components of the host graph H are either complete reflexive graphs or complete irreflexive bipartite graphs. Moreover, we prove that in any other case the decision problems are NP-complete and the counting problems are

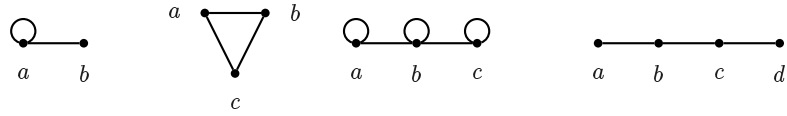


Fig. 1. The four forbidden subgraphs of Lemma 1

#P-complete. Observe that, in contrast to the non restrictive problems, the dichotomy result attained for this problem is the same for both list and non list problems as well as for counting an decision problems.

We use standard notation for graphs and we set $n = |V(G)|$ and $h = |V(H)|$. For a given graph G and a vertex subset $S \subseteq V(G)$, the subgraph induced by S is the graph $G[S] = (S, E(G) \cap S \times S)$.

2 NP-Completeness Results

Notice that, for any given graph G, by setting $\mathsf{w}(a) = \infty$ for all $a \in V(H)$, the restrictive H-coloring problems solves the corresponding H-coloring problem, therefore we can translate all the hardness results to the restrictive problem versions. In particular, the #P-hardness results in [7,13,5] translate as shown in the following result.

Theorem 1. *If H has a connected component that is neither a complete irreflexive bipartite graph nor a complete reflexive clique, then the restrictive #H-coloring and the restrictive list #H-coloring problems are #P-hard.*

In this section we will show that when H has a connected component that is not a complete irreflexive bipartite graph or a complete reflexive clique the restrictive H-coloring problem, and therefore the restrictive list H-coloring problem, are NP-complete. As both problems are clearly in NP, we provide the hardness proofs.

It is well known that the four forbidden subgraphs given in Figure 1 characterize the following property [14]:

Lemma 1. *All the connected components of a graph H are either a complete reflexive graph or a complete irreflexive bipartite graph iff H does not contain as induced subgraphs any of the graphs given in Figure 1.*

Now we can state the result in this section.

Theorem 2. *If H contains any of the graphs in Figure 1 as induced subgraph then, the restrictive H-coloring problem is NP-complete.*

Proof. We will distinguish four cases depending on which of the graphs in Figure 1 appears as an induced subgraph of H. Observe that we can select a particular induced subgraph of H by setting to 0 the number of tasks that a processor can perform.

Case 1. If $\{a, b\}$ is an edge in H where a is looped and b is unlooped, we define

$$\mathsf{w}(v) = \begin{cases} \infty & \text{if } v = a, \\ k & \text{if } v = b, \\ 0 & \text{otherwise.} \end{cases}$$

Then (G, w) has a restrictive H-coloring iff G has an independent set of size at least k.

Case 2. If $\{a, b, c\}$ form a triangle in H, we set

$$\mathsf{w}(v) = \begin{cases} \infty & \text{if } v \in \{a, b, c\}, \\ 0 & \text{otherwise.} \end{cases}$$

Then (G, w) has a restrictive H-coloring iff G is 3-colorable.

Case 3. Let now $\{a, b, c\}$ be an induced reflexive path in H. We will reduce the following problem to the restrictive H-coloring problem.

Balanced Separator: Given a graph G and positive integer $k \leq n$. Is there a partition of $V(G)$ in three sets A, B, C, such that $|C| = k$, that removing C leaves a graph with no edges between A and B, and such that $\max\{|A|, |B|\} \leq |V|/2$.

The above problem can be shown NP-complete by a slight variation of the NP-hardness proof given in [1] for the *minimum B-vertex separator* problem.

Let G be an input of the above problem, we construct a new graph $\tilde{G}$ with $k + 1$ new vertices, $V(\tilde{G}) = V(G) \cup \{u_0, \ldots, u_k\}$, and with edge set $E(\tilde{G}) = E(G) \cup \{\{u_0, x\} \mid x \in V(G)\} \cup \{\{u_0, u_i\} \mid 1 \leq i \leq k\}$.

For any $v \in V(H)$, we set

$$\mathsf{w}(v) = \begin{cases} n/2 & \text{if } v = a, \\ k + 1 & \text{if } v = b, \\ n/2 & \text{if } v = c, \\ 0 & \text{otherwise.} \end{cases}$$

and we can show that G has a balanced separator iff $(\tilde{G}, \mathsf{w})$ has a restrictive H-coloring (see [3] for a detailed proof).

Case 4. Let now $\{a, b, c, d\}$ be an induced irreflexive path in H. We consider the following NP-complete problem, see [11]:

Balanced Complete Bipartite Subgraph: Given a bipartite connected graph $G = (V_1, V_2, E)$ and positive integer k, such that $k \leq |V_1| + |V_2|$. Does G contain $K_{k,k}$ as an induced subgraph?

Let (G, k) be an input of the above problem. Assume that V_1 and V_2 is the partition of $V(G)$, observe that as G is connected and bipartite this partition is unique. Let u_1, u_2 be two new vertices not belonging in $V(G)$. We construct a new bipartite graph $\tilde{G} = (W, F)$ with bipartition $W_1 = V_1 \cup \{u_1\}$ and $W_2 = V_2 \cup \{u_2\}$, and with edge set $F = \{\{u_1, x\} \mid x \in V_2\} \cup \{\{x, u_2\} \mid x \in V_1\} \cup \{\{u_1, u_2\} \cup \{\{x, y\} \mid x \in V_1, y \in V_2, \text{ and } \{x, y\} \notin E(G)\}$. Notice that $\tilde{G}$ is the bipartite complement of G with two new adjacent vertices u_1 and u_2, such that u_1 is connected with all the vertices in one part and u_2 with all the vertices in the other.

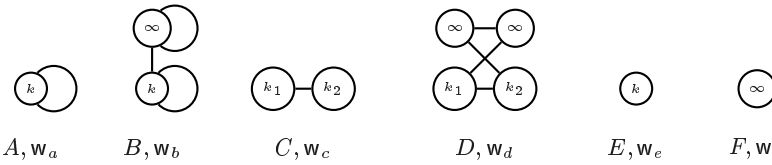


Fig. 2. The six basic cases for counting H-colorings

For all $v \in V(H)$, we set

$$\mathsf{w}(v) = \begin{cases} k & \text{if } v = a, \\ \infty & \text{if } v = b, \\ \infty & \text{if } v = c, \\ k & \text{if } v = d, \\ 0 & \text{otherwise.} \end{cases}$$

Then G contains $K_{k,k}$ as a subgraph iff $\tilde{G}$ has a restrictive H-coloring (see [3] for a detailed proof).

Using the fact that the restrictive list H coloring problem can be used to solve the restrictive H-coloring problem we get,

Theorem 3. *If H has a connected component that is neither a complete irreflexive bipartite graph nor a complete reflexive clique then the restrictive H-coloring and the restrictive list H-coloring problems are NP-hard.*

3 *H*-Coloring: The Connected Case

In this section we solve in polynomial time the counting version of the restrictive H-coloring problem in the case that H does not contain as subgraph any of the forbidden subgraphs in Figure 1 and furthermore G is connected.

Let us first show that for any of the different graphs and weighting functions in Figure 2 the number of restrictive H-colorings of a graph G can be computed in polynomial time.

Given two graphs G, H and a weighting function w on $V(H)$, let $\mathsf{H}(G, H, \mathsf{w})$ denote the number of restrictive H-colorings of (G, w). We set $n = |V(G)|$, and for a connected bipartite graph G, we set n_1, n_2 to be the sizes of the two partitions. It is easy to check that, for the graphs and weighting functions given in Figure 2, the following algorithms compute correctly the number of restrictive colorings.

algorithm $\mathsf{H}(G, A, \mathsf{w}_a)$
begin
  if $n = k$ **then ret** 1 **else ret** 0 **end if**
end

algorithm $\mathsf{H}(G, B, \mathsf{w}_b)$
begin
  if $n \leq k$ **then ret** $\binom{n}{k}$ **else ret** 0 **end if**
end

algorithm $\mathsf{H}(G, C, \mathsf{w}_c)$
begin
  if G is not bipartite
    then ret 0
    elseif ($n_1 \neq k_1$ and $n_1 \neq k_2$) or ($n_2 \neq k_1$ and $n_2 \neq k_2$) **then ret** 0
    elseif ($n_1 = k_1$ and $n_1 \neq k_2$ and $n_2 = k_2$) **then ret** 1
    elseif ($n_1 \neq k_1$ and $n_1 = k_2$ and $n_2 = k_1$) **then ret** 1
    else ret 2
  end if
end

algorithm $\mathsf{H}(G, D, \mathsf{w}_d)$
begin
  if G is not bipartite
    then ret 0
    elseif ($n_1 < k_1$ and $n_1 < k_2$) or ($n_2 < k_1$ and $n_2 < k_2$) **then ret** 0
    elseif ($n_1 \geq k_1$ and $n_1 < k_2$ and $n_2 \geq k_2$) **then ret** $\binom{n_1}{k_1}\binom{n_2}{k_2}$
    elseif ($n_1 < k_1$ and $n_1 \geq k_2$ and $n_2 \geq k_1$) **then ret** $\binom{n_2}{k_1}\binom{n_1}{k_2}$
    else ret $\binom{n_1}{k_1}\binom{n_2}{k_2} + \binom{n_2}{k_1}\binom{n_1}{k_2}$
  end if
end

algorithm $\mathsf{H}(G, E, \mathsf{w}_e)$
begin
if G has no edges and $k = 1$ **then ret** 1**else ret** 0 **end if**
end

algorithm $\mathsf{H}(G, F, \mathsf{w}_f)$
begin
  if G has no edges **then ret** 1 **else ret** 0 **end if**
end

Lemma 2. *Let H be a reflexive clique, given a connected graph G and a weighting function w on H, then $\mathsf{H}(G, H, \mathsf{w})$ can be computed in polynomial time.*

Proof. Let $C = \{a \in V(H) \mid \mathsf{w}(a) \neq \infty\}$, let $k = \sum_{a \in C} \mathsf{w}(a)$, and let $\alpha = |V(H) - C|$. We will consider two cases.

Case 1. $C = V(H)$. In this case, "collapsing" all the vertices in H into a single vertex and assigning weight k to it, we get a type A graph with a weighting function w_a. Observe, that any restrictive A-coloring of (G, w_a) can be extended in $k!$ ways to obtain a valid restrictive H-coloring of (G, w), and any valid H-

coloring of (G, w) can be contracted to provide a valid restrictive A-coloring of (G, w_a).

Case 2. $C \neq V(H)$. In this case, "collapsing" all the vertices in C to a vertex with weight k and all the remaining vertices to a vertex with weight ∞, we obtain a type B graph with weighting function w_b. Observe that any restrictive B-coloring of (G, w_b) can be extended in $k!\alpha^{n-k}$ ways to obtain a valid restrictive H-coloring of (G, w), and any valid H-coloring of (G, w) can be contracted to provide a valid restrictive B-coloring of (G, w_b).

Therefore the following algorithm, which takes polynomial time, computes $\mathsf{H}(G, H, \mathsf{w})$.

algorithm $\mathsf{H}(G, H, \mathsf{w})$
begin
  Compute n, k and α;
  if $\alpha = 0$
    then if $n = k$ **then** **output** $k!$ **else ret** 0 **end if**
    elseif $n \leq k$
      then ret $k!\,\alpha^{n-k}\binom{n}{k}$
      else ret 0
  end if
end

Lemma 3. *Let H be a complete irreflexive bipartite graph with more than one vertex. Given a connected graph G and a weighting function w on H, $\mathsf{H}(G, H, \mathsf{w})$ can be computed in polynomial time.*

Proof. Let $H = (V_1, V_2, E)$. For $i = 1, 2$, let $C_i = \{a \in V_i \mid \mathsf{w}(a) \neq \infty\}$, let $k_i = \sum_{a \in C_i} \mathsf{w}(a)$, and let $\alpha_i = |V_i - C_i|$. We will consider two cases.

Case 1. $C_1 = V_1$ and $C_2 = V_2$. In this case collapsing all the vertices in V_1 to a vertex with weight k_1 and collapsing all the vertices in V_2 to a vertex with weight k_2 we obtain a type C graph and a weighting function w_c. Observe that any restrictive C-coloring of (G, w_c) can be extended in $k_1!\,k_2!$ ways to obtain a valid restrictive H-coloring of (G, w), and any valid H-coloring of (G, w) can be contracted to provide a valid restrictive C-coloring of (G, w_c).

Case 2. $C_1 \neq V_1$ or $C_2 \neq V_2$. In this case by collapsing all the vertices in C_i to a vertex with weight k_i and all the remaining vertices in V_i to an unbounded vertex, we obtain a type D graph with a weighting function w_d. Observe that any restrictive D-coloring of (G, w_d) can be extended in $k_1!\,k_2!\,\alpha_1^{n_1-k_1}\,\alpha_2^{n_2-k_2}$ ways to obtain a valid restrictive H-coloring of (G, w), and that any valid H-coloring of (G, w) can be contracted to provide a valid restrictive D-coloring of (G, w_d).

Therefore, the following algorithm computes $\mathsf{H}(G, H, \mathsf{w})$ in polynomial time.

algorithm $\mathsf{H}(G, H, \mathsf{w})$
begin
  if G is not bipartite

then ret 0
else
Compute $n_1, n_2, k_1, k_2, \alpha_1$ and α_2;
if $\alpha_1 = \alpha_2 = 0$
then
if ($n_1 \neq k_1$ and $n_1 \neq k_2$) or ($n_2 \neq k_1$ and $n_2 \neq k_2$)
then ret 0
elseif ($n_1 = k_1$ and $n_1 \neq k_2$ and $n_2 = k_2$) **then ret** $k_1!\,k_2!$
elseif ($n_1 \neq k_1$ and $n_1 = k_2$ and $n_2 = k_1$) **then ret** $k_1!\,k_2!$
else ret $2\,k_1!\,k_2!$;
end if
elseif ($n_1 < k_1$ and $n_1 < k_2$) or ($n_2 < k_1$ and $n_2 < k_2$)
then ret 0
elseif ($n_1 \geq k_1$ and $n_1 < k_2$ and $n_2 \geq k_2$)
then ret $k_1!\,k_2!\,\alpha_1^{n_1-k_1}\,\alpha_2^{n_2-k_2}\binom{n_1}{k_1}\binom{n_2}{k_2}$
elseif ($n_1 < k_1$ and $n_1 \geq k_2$ and $n_2 \geq k_1$)
then ret $k_1!\,k_2!\,\alpha_1^{n_1-k_1}\,\alpha_2^{n_2-k_2}\binom{n_2}{k_1}\binom{n_1}{k_2}$
else ret $k_1!\,k_2!\,\alpha_1^{n_1-k_1}\,\alpha_2^{n_2-k_2}\left(\binom{n_1}{k_1}\binom{n_2}{k_2} + \binom{n_2}{k_1}\binom{n_1}{k_2}\right)$.
end if
end if
end if
end

In the case that H is an isolated vertex, G must also be an isolated vertex and we can compute, in polynomial time, $\mathsf{H}(G, E, \mathsf{w}_e)$ or $\mathsf{H}(G, F, \mathsf{w}_f)$.

Now we can show the main result in this section.

Theorem 4. *If all the connected components of H are either a complete irreflexive bipartite graph or a complete reflexive clique, then the restrictive #H-coloring problem can be solved in polynomial time.*

Proof. Assume that H has l connected components. Given a weighting function w on H, let w_j denote the restriction of w to the vertices in H_j. As the given graph G is connected, it can be mapped only to one connected component of H, therefore we only have to count the number of restrictive H_j colorings of (G, w_j) that fulfill the weight bounds, with an empty assignment of vertices in G to the remaining components.

We classify the connected components of H as follows; H_j is *free* if $\mathsf{w}(H_j) = \{\infty\}$, H_j is *forbidden* if $\mathsf{w}(H_j) = \{0\}$, otherwise H_j is *restricted*. Therefore, we have

$$\mathsf{H}(G, H, \mathsf{w}) = \begin{cases} 0 & H \text{ has two restricted components,} \\ \sum_{1 \leq j \leq l} \mathsf{H}(G, H_j, \mathsf{w}_j) & \text{if all the c.c. are free or forbidden,} \\ \mathsf{H}(G, H_j, \mathsf{w}_j) & \text{if } H_j \text{ is the unique restricted c.c.} \end{cases}$$

By Lemmas 2 and 3 the last formula can be evaluated in polynomial time.

As counting in polynomial time implies deciding in polynomial time we have,

Corollary 1. *If all the connected components of H are either a complete irreflexive bipartite graph or a complete reflexive clique, then the restrictive H-coloring problem can be solved in polynomial time.*

4 H-Coloring: The General Case

Now we show how to compute the number of restrictive H-colorings in the general case in which the graph G may not be connected. Observe that different components of G can fill partially the weighted vertices of H, therefore we are forced to consider H-colorings of components of G that are not restrictive H-colorings, for the given pair (G, w), but that all together provide one such restrictive H-coloring.

Theorem 5. *If all the connected components of H are either a complete irreflexive bipartite graph or a complete reflexive clique then the restrictive #H-coloring problem can be solved in polynomial time.*

Proof. Given (G, w) and assuming, as usual, that $k \leq \infty$ holds for any integer k, let $E(\mathsf{w})$ be the set

$$E(\mathsf{w}) = \{f : V(H) \to \{0, \ldots, n\} \mid f(a) \leq \mathsf{w}(a) \text{ for all } a \in V(H)\}.$$

Given two functions w_1 and w_2 from $V(H)$ to $\{0, \ldots, n\}$, for any $a \in V(H)$, its sum is defined as usual: $(\mathsf{w}_1 + \mathsf{w}_2)(a) = \mathsf{w}_1(a) + \mathsf{w}_2(a)$.

To keep uniform notation, we will assume that all the weighting functions are defined over $V(H)$. To fullfil this goal, any weighting function of a connected component H_i is extended to H by assigning the weight 0 to all the vertices outside $V(H_i)$. We say that a weighting function w defined over H is *proper* for H_i, if for all $u \in V(H) - V(H_i)$, $\mathsf{w}(u) = 0$. For any component i, $1 \leq i \leq l$, let $P(i)$ be the set of proper functions for component i.

Assume that G has m connected components $G_1, \ldots, G_m$, G^i denotes the graph formed by the disjoint union of $G_1, \ldots, G_i$. For given G and w, $\mathsf{H}(G, H, \mathsf{w})$, we compute initially a table $T[i, j, m]$, such that for any $1 \leq i \leq m$, $1 \leq j \leq l$ and $f \in E(\mathsf{w})$)

$$T[i, j, f] = \begin{cases} \mathsf{H}(G_i, H_j, f) & f \in P(j), \\ 0 & \text{otherwise.} \end{cases}$$

For each f, $T(i, j, f)$ can be computed in polynomial time using Theorem 4. As $|V(H)| = h$ and $|V(G)| = n$, the size of $E(\mathsf{w})$ is at most n^h, therefore polynomial because H is fixed, so the whole table can be computed in polynomial time.

Using dynamic programming techniques, we can compute a table $S[i, f]$, for $1 \leq i \leq m$ and $f \in E(\mathsf{w})$, such that $S[i, f]$ counts the number of restrictive H-colorings of (G^i, f). To get the formula we take into account that a connected component of G must be mapped entirely to a unique connected component of H. Therefore, for any $f \in E(\mathsf{w})$

$$S[1, f] = \sum_{1 \leq j \leq l} T[1, j, f],$$

and, for any $1 < j \leq m$,

$$S[j, f] = \sum_{1 \leq j \leq l, f_1 + f_2 = f} S[j-1, f_1] * T[i, j, f_2].$$

As the size of $E(\mathsf{w})$ is polynomial, table S can be computed in polynomial time.

To finish, let $A(\mathsf{w}) = \{f \in E(\mathsf{w}) \mid \mathsf{w}(a) = f(a) \text{ for all } a \in H \text{ with } \mathsf{w}(a) \neq \infty\}$ be the set of weighting bounded functions fullfiling the restrictions, then

$$\mathsf{H}(G, H, \mathsf{w}) = \sum_{\{f \in A(\mathsf{w})\}} S[m, f],$$

which again can be computed in polynomial time.

Putting together Theorems 1, 3 and 5, we get the dichotomy result.

Theorem 6. *If all the connected components of H are either a complete irreflexive bipartite graph or a complete reflexive clique then the restrictive H-coloring and the restrictive $\#H$-coloring problems can be solved in polynomial time, otherwise they are NP-complete, #P-complete, respectively.*

5 The Restrictive List H-Coloring Problem

In his section we extend the previous result to the problem of counting restrictive list H-colorings. The main difficulty here is that the vertices in a connected component of H cannot be "collapsed" to a single vertex, as this may place together vertices that are not in the same vertex list. Once we have solved the connected case, the second step is identical to the general case for the restrictive H-coloring.

We will consider the two main types of connected components and show that a dynamic programming approach allow us to compute the number of restrictive list H-colorings. Making an abuse of notation we will represent by $\mathsf{H}(G, H, \mathsf{w}, L)$ the number of restrictive list H-colorings of a triple (G, w, L).

Lemma 4. *Let H be a reflexive clique. Given a connected graph G, a weighting function w on H and a list selection for G, then $\mathsf{H}(G, H, \mathsf{w}, L)$ can be computed in polynomial time.*

Proof. As H is a reflexive clique we can assign a vertex of G to any vertex in H, provided that the additional restrictions are fulfilled. Let $V(G) = \{u_1, \ldots, u_n\}$, and let $E(\mathsf{w}) = \{f : V(H) \to \{0, \ldots, n\} \mid \text{ for all } a \in V(H)\ f(a) \leq \mathsf{w}(a)\}$. For any $a \in H$ define f_a by

$$f_a(b) = \begin{cases} 1 & \text{if } b = a, \\ 0 & \text{otherwise.} \end{cases}$$

We want to compute a table $R[i, f]$, $1 \leq i \leq n$, $f \in E(\mathsf{w})$, which counts the number of restrictive list H-colorings for $(G[\{u_1, \ldots, u_i\}], f, L)$. For any $f \in E(\mathsf{w})$, consider the following recurrence

$$R[1, f] = \textbf{if } f = f_a \text{ for some } a \in L(u_1) \textbf{ then } 1 \textbf{ else } 0,$$

$$R[i, f] = \sum_{\substack{f_1 + f_2 = f, \\ f_2 = f_a \text{ for some } a \in L(u_i)}} R[j-1, f_1], \text{ for } 1 < j \leq m.$$

As the size of $E(\mathsf{w})$ is polynomial, we can fill the table R in polynomial time.

Setting $A(\mathsf{w}) = \{f \in E(\mathsf{w}) \mid \mathsf{w}(a) = f(a) \text{ for all } a \in H \text{ with } \mathsf{w}(a) \neq \infty\}$ we have

$$\mathsf{H}(G, H, \mathsf{w}, L) = \sum f \in A(\mathsf{w})R[n, f].$$

Lemma 5. *Let H be a complete irreflexive bipartite graph with more than one vertex, given a connected graph G, a weighting function w on H, and a list selection L for G, $\mathsf{H}(G, H, \mathsf{w})$ can be computed in polynomial time.*

Proof. If H is bipartite, then G must be bipartite. In this case, we can work separately with the two possible assignments of partitions of G with partitions of H. Notice that once the global assignment is set, any vertex can be mapped to any one in the assigned partition, thus working as in the previous lemma we can compute $\mathsf{H}(G, H, \mathsf{w})$ in polynomial time.

Using the same technique of Section 4, we can obtain the polynomial time result. This, together with Theorems 1 and 3, give the dichotomy for the list version of the problem.

Theorem 7. *If all the connected components of H are either a complete irreflexive bipartite graph or a complete reflexive clique, then the restrictive list H-coloring and the restrictive list $\#H$-coloring problems can be solved in polynomial time, otherwise they are NP-complete and #P-complete, respectively.*

6 Further Variations and Conclusions

We can consider a generalized version of the restrictive H-coloring problem in which each processor gets a list of desired ranges. Then, a valid restrictive H-coloring is an H-coloring σ of G, such that, for any $a \in H$, $|\sigma^{-1}(a)|$ falls inside one of a prescribed ranges.

Observe that in this generalized version, we can arrange the list to keep a unique value or keep all possible values, so this generalized version contains as a subproblem the restrictive H-coloring. Using Theorem 5 we can compute $\mathsf{H}(G, H, f)$ for any function $f \in E(u)$. Starting from this information, we can compute in polynomial time the number of restrictive H-colorings satisfying any given list of desired ranks.

We can add to all the above a list of preferences and attain the same dichotomy result for the generalized list problem.

We have dealt through the paper in H-colorings that correspond to embeddings with dilation one. As far as H remains fixed we can solve the restrictive embedding problem for any prefixed maximum dilation d. To do so we just enhance H with edges joining vertices at distance d or less.

References

1. T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is NP-hard. *Information Processing Letters*, 42(3):153–159, 1992.

2. J. Díaz. H-colorings of graphs. *Bulletin of the European Association for Theoretical Computer Science*, (75):82–92, 2001.
3. J. Díaz, M. Serna, and D. M. Thilikos. The complexity of restrictive H-coloring. TR LSI-02-22-R, Software Dept., Universitat Politcnica de Catalunya, 2002.
4. J. Díaz, M. Serna, and D. M. Thilikos. Recent results on parameterized H-colorings. In J. Nešetřil and P. Winkler, editors, *Graphs, Morphisms and Statistical Physics*, DIMACS series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 2002. To appear.
5. J. Díaz, M. Serna, and D. M. Thilikos. Counting list H-colorings and variants. TR LSI-01-27-R, Software Dept., Universitat Politcnica de Catalunya, 2001.
6. J. Díaz, M. Serna, and D. M. Thilikos. (H, C, K)-colorings: Fast, easy and hard cases. In *Mathematical Foundations of Computer Science 2001, MFCS-2001*, volume 2136 of *LNCS*, pages 304–315. Springer-Verlag, 2001.
7. M. Dyer and C. Greenhill. The complexity of counting graph homomorphisms. *Random Structures Algorithms*, 17:260–289, 2000.
8. T. Feder and P. Hell. List homomorphisms to reflexive graphs. *Journal of Combinatorial Theory (series B)*, 72(2):236–250, 1998.
9. T. Feder, P. Hell, and J. Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19:487–505, 1999.
10. T. Feder, P. Hell, and J. Huang. Bi-arc graphs and the complexity of list homomorphism. Manuscript, 2001.
11. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
12. P. Hell and J. Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory (series B)*, 48:92–110, 1990.
13. P. Hell and J. Nešetřil. Counting list homomorphisms and graphs with bounded degrees. In J. Nešetřil and P. Winkler, editors, *Graphs, Morphisms and Statistical Physics*, DIMACS series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 2002. To appear.
14. J. Nešetřil, 2001. personal communication.

A New 3-Color Criterion for Planar Graphs
(Extended Abstract)

Krzysztof Diks, Lukasz Kowalik, and Maciej Kurowski

Institute of Informatics, Warsaw University
Banacha 2, 02-097 Warsaw, Poland
{diks,kowalik,kuros}@mimuw.edu.pl

Abstract. We present a new general 3-color criterion for planar graphs. Applying this criterion we characterize a broad class of 3-colorable planar graphs and provide a corresponding linear time 3-coloring algorithm. We also characterize fully infinite 3-colorable planar triangulations.

1 Introduction

The problem of vertex coloring of a graph using few colors has given rise to one of the most intensively studied areas of the graph theory. A frequently encountered special case is that in which the graph to be colored is planar. Computing a coloring that uses the smallest possible number of colors is known to be an NP-complete problem, even when restricted to the class of planar graphs and 3 colors. More precisely it is an NP-complete problem to decide whether a given planar graph is 3-colorable [GJS]. On the other hand the famous "4-color theorem" says that every planar graph is 4-colorable. Hence it is natural to characterize those planar graphs which are 3-colorable. The first 3-color criterion was formulated by Heawood in 1898 and it is known as the Three Color Theorem [Hea,Ste]: *A maximal planar graph is vertex colorable in three colors if and only if all its vertices have even degrees.* Obviously this theorem implies a very simple algorithm for checking 3-colorability of maximal planar graphs. As the problem of 3-colorability of planar graphs is NP-complete one cannot rather expect any "effective" 3-color criterion for general planar graphs. On the other hand there is a "simple", general 3-color criterion which does not lead to an efficient 3-coloring algorithms: *A planar graph is vertex colorable in three colors if and only if it is a subgraph of a maximal planar graph in which all vertices have even degrees.* This theorem was already known to Heawood as well as it was discovered independently by several authors – see the comprehensive survey written by Steinberg [Ste].

In this paper we introduce a new general 3-color criterion which can be efficiently checked for a broad class of planar graphs. Our criterion generalizes the Heawood's Three Color Theorem. In order to get this result we define a new class of planar graph colorings, so called *edge-side colorings*, and prove that the new type of coloring is equivalent to the ordinary vertex 3-coloring. The criterion allows to characterize 3-colorable triangulations with holes, i.e. plane graphs in

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 138–149, 2002.

which each vertex touches at most one non-triangular face. We provide a linear time algorithm for 3-coloring such graphs. Our criterion allows also to formulate sufficient and necessary conditions for 3-coloring infinite planar triangulations.

2 Basic Definitions and Notation

It is known that a graph is 3-colorable iff all its biconnected components are 3-colorable. In the sequel, if it is not stated explicitly, saying a graph we mean a biconnected, finite multigraph of at least three vertices but without selfloops.

A *plane graph* is a graph whose vertex set is a point set in the plane and the edges are Jordan curves such that two different edges have at most end points in common. A graph is called *planar* if it can be embedded in the plane, i.e. if it is isomorphic to a plane graph.

Let C be a simple cycle in a plane graph G. The cycle C divides the plane into two disjoint open domains – the *interior C-domain* (homeomorphic to an open disc) and the *exterior C-domain*. The set consisting of all vertices of G belonging to the interior C-domain and of all edges crossing this domain is denoted by $\operatorname{Int} C$. If v is a vertex on C then the number of the graph neighbors of v lying in the interior C-domain is called the *internal degree* of v with respect to C and it is denoted by $dIn(C, v) = |\{(v, w) \in E(G) \ : \ w \in \operatorname{Int} C\}|$. We define the internal degree of the cycle C as the sum of the internal degrees of all its vertices. This sum is denoted by $dIn(C) = \sum_{v \in C} dIn(C, v)$.

A *face* in a plane graph G is a C-domain (interior or exterior), for some cycle C, without any vertices and edges inside. Only one face is unbounded and it is called the *outer* face. Similarly, its boundary cycle is called the *outer* one.

A *triangulation* is a plane graph in which the boundary cycle of every face is a triangle (3-cycle). A biconnected plane graph in which all the boundary cycles, except at most one, are triangles is called a *near-triangulation*. W.l.o.g. we will consider this boundary cycle to be the outer one.

A graph is *even* if all its vertices have even degrees. A near-triangulation is *internally even* if all its vertices different from those on the outer cycle have even degrees.

3 A New 3-Color Criterion

In 1898 Heawood [Hea] proved a theorem characterizing (finite) 3-colorable triangulations:

Theorem 1 (Three Color Theorem). *A (finite) triangulation is 3-colorable if and only if it is even.*

This criterion applies only to the *maximal* planar graphs which are isomorphic to triangulations. It allows to check in a very simple manner whether a given maximal planar graph is 3-colorable. One can ask a natural question: can the criterion be generalized to all planar graphs? Unfortunately, since checking

3-colorability is an NP-complete problem even in the planar case, we cannot rather expect any polynomially checkable criterion for general planar graphs. However there are general criteria which allow checking 3-colorability of a given planar graph in some special cases. As stated in [Ste] such a general criterion was already known to Heawood [Hea]. Nevertheless it was not widely known and has been independently discovered and proved several times, e.g. in [Kr1], [Kr2], [Mar]. The criterion follows:

Theorem 2 (Heawood's 3-Color Criterion). *Let G be a plane graph. The following two conditions are equivalent:*

(i) *G is 3-colorable.*
(ii) *There exists an even triangulation H such that G is a subgraph of H, i.e. $H \supseteq G$.*

Moreover, every 3-coloring of a plane graph G can be extended to a 3-coloring of some even triangulation $H \supseteq G$.

As we can see the 3-color criterion stated above tells us nothing about the structure of the graph under consideration. In this section we provide a new type of graph coloring, called *edge-side coloring*, which is equivalent to the vertex 3-coloring but additionally reflects some structural properties of a given graph. This new feature will allow us to characterize a new, broad class of 3-colorable planar graphs which are recognizable and 3-colorable in a linear time.

We start from a few indispensable definitions.

Let G be a plane graph, f a face in G and e an edge on the boundary cycle of f. The pair $s = (e, f)$ is called a *side of edge e in face f* (or shortly a *side*). We say also that side s touches face f. If vertex v is an end point of e then side s is said to be incident with v. Observe that in a biconnected graph every edge has exactly two sides.

Let G be a plane graph and S be the set of all sides in G. *Edge-side coloring* of G is an arbitrary function

$$m : S \longrightarrow \{black, white\}.$$

Edges with one side black and the other side white are called *b-w edges*. The other edges are called *one-color edges* and can be of type *b-b* or *w-w* depending on the colors of their sides, black or white respectively.

We say that an edge-side coloring of a plane graph G is *proper* if and only if the following two conditions are satisfied:

(i) for each face f in G the numbers of white and black sides touching f are congruent (equal) mod 3;
(ii) each vertex v in G is incident with an even number of one-color edges.

We say that a plane graph G is *edge-side colorable* if its edge-sides can be properly colored.

Now we can state the main theorem of the paper.

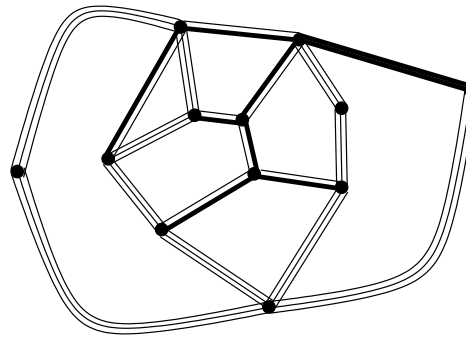


Fig. 1. A proper edge-side coloring. Light (dark) lines indicate white (black) sides

Theorem 3 (3-Color Criteria). *Let G be a (biconnected) plane graph. The following three conditions are equivalent:*

(i) *G is 3-colorable.*
(ii) *There exists an even triangulation $H \supseteq G$.*
(iii) *G is edge-side colorable.*

The equivalence of conditions (i) and (ii) was proved by Heawood (see Theorem 2). The proof of the equivalence of (ii) and (iii) is our main contribution to this paper. We start from a few observations on internally even near-triangulations.

Lemma 1. *Every internally even near-triangulation is 3-colorable.*

Proof. Let C be the outer cycle of a near-triangulation G. Let us take a separate embedding G' of G in which the cycle C' corresponding to C is not longer the outer one. Now we build a new plane graph H from G' placing the entire graph G in the interior C'-domain and identifying the corresponding vertices and edges of the cycles C and C' as shown in Fig. 2. One can easily check that the graph H is an even plane triangulation and hence it is 3-colorable by the Heawood's Three Color Theorem. Since $G \subseteq H$ it is also 3-colorable. □

Lemma 2. *Let G be an even near-triangulation with the outer cycle C. Then* $|C| \equiv 0 \pmod 3$.

Proof. By Lemma 1, G is 3-colorable. Let $C = v_0 v_1 \dots v_{|C|-1} v_0$ and let $\mathcal{K}$ be an arbitrary 3-coloring of G. We will show that one can rename the colors in $\mathcal{K}$ in such a way that $\mathcal{K}(v_i) = (i \bmod 3) + 1$, for every $i = 0 \dots |C| - 1$.

Let v be an arbitrary vertex in C and let x and y be its neighbors such that x, y and v are incident with the same internal (not unbounded) triangular face in G. Vertices x, y, and v have different colors. Now one can observe that every two successive neighbors of v, in a sequence of all neighbors listed in the clockwise order, have different colors. Since the degree of v is even its neighbors

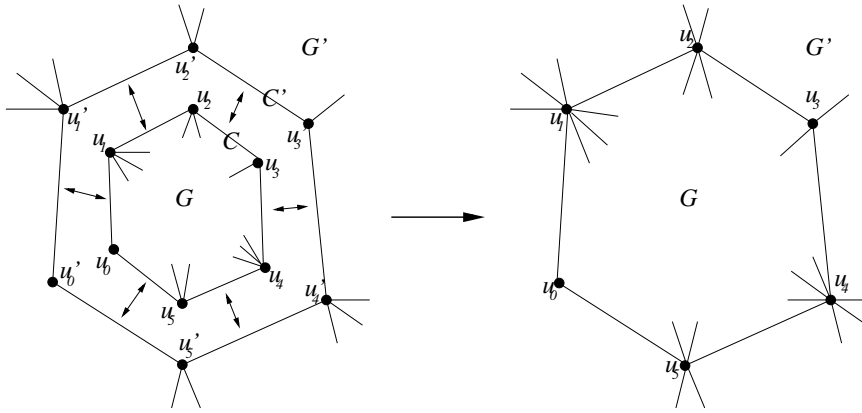


Fig. 2. Proof of lemma 1

on C have different colors. As a result we get $\mathcal{K}(v_i) = (i \bmod 3) + 1$ (possibly after renaming the colors) what implies $|C| \equiv 0 \pmod 3$. □

Lemma 3. *For every $i \geq 3$ such that $i \equiv 0 \pmod 3$ there exists a finite even near-triangulation with the outer cycle of length i.*

Proof. The proof is by induction on i. For $i = 3$ it suffices to take K_3.

Inductive step: by the induction hypothesis there exists a finite, even near-triangulation G_0 with the outer cycle of length $i - 3$. Let v_1, v_2 be arbitrary adjacent vertices in the outer cycle of G_0. Then $G = G_0 \cup \{v_3, v_4, v_5\} \cup \{v_2 - v_3, v_3 - v_4, v_4 - v_5, v_5 - v_1, v_2 - v_4, v_1 - v_4\}$ is an even near-triangulation and its outer cycle has length i. □

Let C be a simple cycle and let m_C be an arbitrary edge 2-coloring of C, $m_C : E(C) \longrightarrow \{black, white\}$. We say that coloring m_C is *balanced* if and only if $|m_C^{-1}(black)| \equiv |m_C^{-1}(white)| \pmod 3$.

Let G be a graph and let C be a simple cycle in G. We say that a balanced coloring m_C of C *corresponds* with G if the following holds: for every vertex v in C the edges of $E(C)$ incident with v have different colors if and only if the degree $d_G(v)$ of v in G is odd.

Lemma 4 (Key Lemma).

(i) For every internally even near-triangulation G with the outer cycle C there exists a balanced 2-coloring m_C of C corresponding with G.

(ii) For every balanced 2-coloring m_C of a cycle C there exists an internally even near-triangulation G with the outer cycle C and such that m_C corresponds with G.

Proof (i). Since G is internally even, the number of vertices of C with odd degrees is even. Let $v_1, v_2, \ldots, v_{2k-1}, v_{2k}$ be a list of all such vertices given in

the clockwise order. For each $i = 1, 2, \ldots, k$, we color black edges on C between v_{2i-1} and v_{2i}. The remaining edges are colored white (see Fig. 3). Observe that vertices in C are incident with edges of different colors if and only if they have odd degrees.

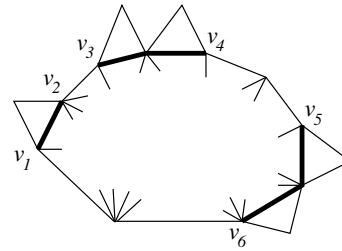


Fig. 3. Constructing a balanced 2-coloring

Let b be the number of black edges on C and let w be the number of white edges on C. After extending G by triangles formed in the outer (unbounded) face and with the black edges as the triangle bases (as shown in Fig. 3) we get an even near-triangulation. By Lemma 2 the outer cycle of this triangulation has length $\equiv 0 \pmod 3$. Hence $2b + w \equiv 0 \pmod 3$ and finally $b \equiv w \pmod 3$. □

Proof (ii). Denote the number of black and white edges of C by b and w respectively. We form a triangle on each black edge e in the interior C-domain as shown in Fig. 4. As the result we get a graph H. Observe that vertex v of H has odd degree if and only if it is incident on C with edges of different colors.

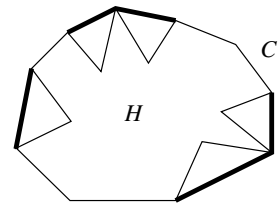


Fig. 4. The graph H with the outer cycle C – constructing a near-triangulation

Let f be the only face of H different from the added triangles and placed in the interior C-domain. The length of the f's boundary cycle is $2b + w \equiv 0 \pmod 3$. By Lemma 3, one can triangulate this face and obtain a required internally even near-triangulation G. □

Lemma 5. *Let G be a finite, biconnected plane graph. Let H be an even triangulation (possibly infinite but locally finite) such that $G \subseteq H$. If for every face*

f of G with facial cycle C_f there exists a balanced edge 2-coloring corresponding with $C_f \cup \mathrm{Int}_H C_f$, the graph G is edge-side colorable.

Proof. For every face f of G with the facial cycle C_f let m_{C_f} be a balanced edge 2-coloring corresponding with $C_f \cup \mathrm{Int}_H C_f$. Let us take an edge-side coloring assigning each side (f, e) the color $m_{C_f}(e)$. Obviously this coloring satisfies the first condition of the definition of the proper edge-side coloring. In order to prove the other one let us consider an arbitrary vertex v in G. Let f' be an arbitrary face in G with v on its facial cycle. Denote this cycle by C'. Let e_1, e_2 be the edges of C' incident with v. Since $m_{C_{f'}}$ corresponds with $C_{f'} \cup \mathrm{Int}_H C_{f'}$ the sides of e_1, e_2 in f' have the same color if and only if the degree $dIn_H(C', v)$ is even. Let $B(v)$ denote the number of black sides incident with v. If $d_G(v)$ is odd then there is an odd number of faces f' incident with v and such that $dIn_H(C', v)$ is odd. On the other hand if $d_G(v)$ is even the number of faces f' incident with v and such that $dIn_H(C', v)$ is odd is even. It follows that $d_G(v) + B(v)$ is always even which is equivalent to the statement that the number of one-color edges incident with v is even.

We have just showed that the second condition in the definition of proper edge-side coloring is satisfied, which completes the proof. □

Now we are ready to prove the part (ii)↔(iii) of our main theorem.

Proof.
(ii)⟶(iii)
Assume that there exists an even triangulation $H \supseteq G$. Observe that since G is biconnected, every face is bounded by a simple cycle. For each face f with the facial cycle C_f we can apply lemma 4 to get a balanced edge 2-coloring m_C of $E(C)$ corresponding with the near-triangulation $C_f \cup \mathrm{Int}_H C_f$. Now we can apply lemma 5 to obtain a proper edge-side coloring of G, what completes the proof of (ii)⟶(iii).

(ii)⟵(iii)
Assume that G is properly edge-side colored. By Lemma 4 one can triangulate (i.e. divide into triangles) each face into an internally even near-triangulation getting a triangulation $H \supseteq G$. Let v be an arbitrary vertex of G. Denote by $F(v)$ the number of faces incident with v for which the odd number of edges ending in v was added during the process of triangulation. Similarly as in the proof of lemma 5 one can show that $d_G(v) + F(v)$ is even. It implies finally that for every vertex v, $d_H(v)$ is even, what means that H is an even triangulation.
□

As the result we get a new 3-color criterion for general planar graphs. In fact, using this criterion for checking whether an arbitrary plane graph is 3-colorable is equally hard as trying to find a proper 3-coloring of a given graph. However we can apply our theorem to show a few classes of planar graphs for which the new criterion can be effectively checked.

4 Applications

One can expect that the criterion formulated in section 3 can be effectively checked for graphs that are "highly triangulated", i. e. when *a lot* of faces are triangles. Moreover, if such a graph has a special structure it can be colored using a greedy algorithm. We define formally a class of graphs for which the greedy algorithm works well. A plane graph G is called *triangle connected* if each vertex of G is incident with a triangular face and the subgraph of the graph dual to G induced by the triangular faces is connected.

In the following subsections we present the greedy algorithm and three classes of graphs for which effective 3-color criteria can be formulated. Our general criterion can be also used to show that plane graphs with face lengths of multiple of three are 3-colorable.

4.1 The Greedy Algorithm

Given a planar, triangle connected graph (without its planar embedding) the algorithm below computes its 3-coloring or reports that such a coloring doesn't exist. The algorithm runs in a linear time. For each vertex v set PossibleColors(v) contains colors which are still admissible for v; S represents the set of vertices for which set PossibleColor contains at most one color. Algorithm uses operation `RESTRICT(v)` which restricts the set of admissible colors for neighbors of v.

```
OPERATION RESTRICT(v)::
for each u in Neighbors(v) do
  if Col(u) = -1 then
  begin
    PossibleColors(u).Remove(Col(v))
    if |PossibleColors(u)| ≤ 1 then S.Add(u)
  end


ALGORITHM GREEDY::
for each v in V(G) do
begin
  PossibleColors(v) := {1, 2, 3}
  Col(v) := -1 {undefined}
end
S := ∅
(p, q) := an arbitrary edge of an arbitrary triangle in G
Col(p) := 1
Col(q) := 2
RESTRICT(p)
RESTRICT(q)

while not S.Empty do
```

```
begin
  v := S.Remove
  if |PossibleColors(v)| ≠ 1 then
    Exit{G is not 3-colorable}
  else
  begin
    Col(v) := PossibleColors(v).Get
    RESTRICT(v)
  end
end while
```

Finding a triangle in a planar graph without its embedding in the plane can be easily done in a linear time, see [Chr]. During each iteration algorithm chooses a triangle with two vertices already colored and colors the third vertex. It can be easily shown that in every embedding one of edges of the initial triangle bounds a triangular face. As graph is triangle connected algorithms stops when all vertices are properly colored.

4.2 Triangulations with Holes

A biconnected plane graph is called a *triangulation with holes* if every of its vertices is incident with at most one non-triangular face, i.e. a face of length at least 4.

Proposition 1. *Every triangulation with holes is triangle connected.*

Proof. Let G be a triangulation with holes. Consider two triangular faces f, h sharing a common vertex v. Vertex v is incident with at most one non-triangular face and subsequently f and h can be connected by a sequence formed of triangular faces, where each two successive faces share a common edge.

Now let f and h be two arbitrary triangular faces of G. Since G is connected, f and h can be connected by a path $e_1, e_2, \ldots, e_k$, where e_i are edges of G. Each edge belongs to at least one triangular face. Denote such face for e_i by t_i. Every two successive faces t_i, t_{i+1} share a vertex. Additionally t_1 shares a vertex with f and t_k shares a vertex with h. Hence we conclude that f and h are connected by a path formed of triangular faces, where each two successive faces share a common edge. □

Triangulations with holes have the following interesting property. Let G a triangulation with holes and let f be a face in G. Let C_f be the facial cycle of f. Then for every vertex v in C_f and an arbitrary even triangulation $H \supseteq G$ the parity of $dIn_H(C_f, v)$ is the same. It follows that there is exactly one edge-side coloring for every triangulation with holes (if not to consider isomorphic ones). This implies a very simple 3-color criterion for triangulations with holes.

We say that a triangulation with holes is *internally even* when the degree of every vertex incident with triangular faces only is even.

Theorem 4. *A triangulation with holes G is 3-colorable if and only if*

(i) it is internally even,
(ii) for every non-triangular face f with the facial cycle C_f there exists a balanced edge 2-coloring m_{C_f} of C_f corresponding with G.

Proof. The proof follows easily from Theorem 3. Let G be a triangulation with holes satisfying conditions *(i)* and *(ii)*. We shall show that G is 3-colorable. For each triangular face f, we color all its sides (e, f) black. For each non-triangular face f with facial cycle C_f we color every side (e, f) with color $m_{C_f}(e)$. The constructed edge-side coloring is balanced for each face and it is easy to check that every vertex is incident with an even number of one-color edges. Now it suffices to use theorem 3.

Now we will show the other implication. Let G be a 3-colorable triangulation with holes. Using theorem 3 we can obtain its proper edge-side coloring m. We recolor black all sides of all triangular faces obtaining a new edge-side coloring m'. One can see that m' is also proper. Now all vertices touching only triangular faces are ends of only one-color (black) edges. Hence they have even degrees. Moreover one can verify that for every non-triangular face f with the facial cycle C_f the coloring $m_{C_f}(e) = m'(f, e)$ corresponds with G. □

4.3 Near-Triangulations and Outerplanar Graphs

Obviously near-triangulations are triangulations with holes. Informally speaking a near-triangulation is a *triangulation with only one hole*. It gives a very simple 3-color criterion for near-triangulations:

Theorem 5. *A near-triangulation is 3-colorable if and only if it is internally even.*

Proof. Lemma 4 implies that the second condition of the Theorem 4 is always satisfied for near-triangulations. □

As every outerplanar graph is a subgraph of a certain internally even near-triangulation we immediately get the following known result:

Corollary 1. *Outerplanar graphs are 3-colorable.*

4.4 Plane Graphs with Faces Which Lengths Are of Multiple of 3

The following theorem ([Ore], [Ste]) easily follows from our criterion:

Theorem 6. *Let G be a graph embedded in the plane in such a way that the number of edges in the boundary of each face is a multiple of 3. If G is even then G is 3-colorable.*

Proof. It suffices to color all edge-sides in G black and to apply Theorem 3. □

4.5 Infinite Triangulations

It is surprising that we can apply our criterion to infinite plane graphs. An *infinite triangulation* is an infinite plane graph with all faces being triangles. We will consider only locally finite triangulation where degrees of all vertices are finite. An *edge accumulation point* (shortly EAP) of an infinite plane graph G is a point P such that for every positive real number ϵ there are infinitely many edges of G with Euclidean distance from P less than ϵ. We will show that the Three Color Theorem holds also for EAP-free infinite triangulations.

Theorem 7. *An EAP-free infinite triangulation is 3-colorable if and only it is even.*

Proof. Assume that EAP-free infinite triangulation G is 3-colorable. Let v be a vertex of odd degree. Since arbitrary two successive neighbors (in clockwise order) of v are adjacent they have different colors. As there is an odd number of neighbors of v we need 3 colors to color them and there is no color left for v. We have just proved implication ($\longrightarrow$).

Assume that G is even. Let v_0 be an arbitrary vertex of G. We define a sequence of graphs

$$G_0 \subset G_1 \subset G_2 \subset G_3 \ldots$$

Let $V(G_0) = \{v_0\}$ and $E(G_0) = \emptyset$. Let W_i be the set of vertices with the graph distance at most i from v. Since graph $G(W_i)$ is finite, it has the outer face f_i with the facial cycle C_i. Obviously there are no cut vertices in $G(W_i)$. Therefore C_i is a simple cycle. We define G_i as $C_i \cup \mathrm{Int}_G C_i$. Since G is EAP-free for every natural i, G_i is a finite graph. Moreover G_i is an internally even near-triangulation. It follows from Theorem 5 that graphs G_i are 3-colorable. Since $G_{i-1} \subseteq G_i$, for $i > 1$, and 3-colorings of G_i and G_{i-1} are unique (i. e. they define the unique partition of the vertices into 3 independent subsets) we can construct 3-colorings $\mathcal{K}_i$ for graphs G_i, $i = 0, 1, 2, \ldots$, such that $\mathcal{K}_{i|G_{i-1}} = \mathcal{K}_{i-1}$. Now we can define a 3-coloring of G as $\mathcal{K}(u) = \mathcal{K}_{d(v_0,u)}(u)$, where $d(v_0, u)$ denotes the graph distance from v_0 to u. □

It is easy to show examples of infinite even triangulations with EAP that are even, but *not* 3-colorable. The construction of such triangulation is shown in Fig. 5. One can see that even first graph in this sequence is not 3-colorable.

In the sequel we use the following well-known fact.

Fact. Let G be an infinite graph. If every finite subgraph of G is k-colorable then G is k-colorable.

Theorem 8. *Let G be an infinite but locally finite triangulation (not necessarily EAP-free). G is 3-colorable if and only if*

(i) G is even,
(ii) for every simple cycle C in G there exists a balanced edge 2-coloring of C corresponding with $C \cup \mathrm{Int}_G C$.

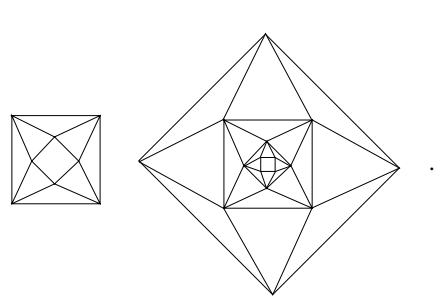


Fig. 5. A construction of not 3-colorable even infinite triangulation

Proof. Assume that G is 3-colorable. Then obviously G must be even. Now let us consider an arbitrary cycle C in G. Let $V_H \subset V$ be a set of vertices defined as follows: $v \in V_H$ if and only if v has a neighbor in $V(C)$ and $v \in C \cup \mathrm{Int}_G\, C$. Let $H = G(V_H)$. As $H \subset G$, H is 3-colorable. It is easy to see that H is a biconnected graph. Therefore we can apply Theorem 3 and get a required edge 2-coloring m_C corresponding with $C \cup \mathrm{Int}_G\, C$.

Now we prove that if (i) and (ii) hold then G is 3-colorable. It suffices to prove that if G satisfies (i) and (ii) then every finite subgraph of G is 3-colorable. Let F be a finite subgraph of G. W.l.o.g. one can assume that F is biconnected. If not, F is a subgraph of a certain biconnected graph $G(W_i)$ defined in the proof of the previous theorem. Now we can use Lemma 5 to get a proper edge-side coloring of F and finish the proof using Theorem 3. □

References

GJS. M. R. Garey, D. S. Johnson, and L. Stockmeyer, Some simplified NP-complete graph problems, *Theoret. Comput. Sci.*, 1 (1976), pp. 237-267.

Hea. P. J. Heawood, On the four-color map theorem, *Quart. J. Pure Math.* **29** (1898) 270-285

Ore. O. Ore, *The Four-Color Problem*, Academic Press, New York, Chapter 13 (1967).

Ste. R. Steinberg, The state of the three color problem [in:], Quo Vadis, Graph Theory? *Annals of Discrete Mathematics*, **55** (1993) 211-248

Kr1. H. Król, On a sufficient and necessary condition of 3-colorableness for the planar graphs. I, *Prace Naukowe Inst. Mat. i Fiz. Teoret. P. Wr.*, Seria Studia i Materialy, No. 6 Grafy i hypergrafy, (1972) 37-40

Kr2. H. Król, On a sufficient and necessary condition of 3-colorableness for the planar graphs. II, *Prace Naukowe Inst. Mat. i Fiz. Teoret. P. Wr.*, Seria Studia i Materialy, No. 9 Grafy i hypergrafy, (1973) 49-54

Mar. N. I. Martinov, 3-colorable planar graphs, *Serdica*, **3**, (1977) 11-16

Chr. M. Chrobak, D. Eppstein, Planar orientations with low out-degree and compactions of adjacency matrices *Theoretical Computer Science*, **86**, (1991) 243-266

An Additive Stretched Routing Scheme for Chordal Graphs

Yon Dourisboure

LaBRI, Université Bordeaux
Yon.Dourisboure@labri.fr

Abstract. This paper concerns routing with succinct tables in chordal graphs. We show how to construct in polynomial time, for every n-node chordal graph of maximum clique size k, a routing scheme using routing tables of $O(k \log n)$ bits per node and $O(\log n)$ bit addresses such that the length of the route between any two nodes is at most the distance between the nodes in the graph plus two. This is complemented by a recent lower bound that shows that if the shortest paths and $O(\log n)$ bit addresses are required, every routing strategy for this class needs $\Omega(2^k \log(n/2^k))$ bits per node.

Keywords: Chordal graph, compact routing tables, tree-decomposition

1 Introduction

Delivering messages between pairs of processors is a basic activity of any distributed communication network. This task is performed using a routing scheme, which is a mechanism for routing messages in the network. The routing mechanism can be invoked at any origin node and be required to deliver a message to some destination node.

It is naturally desirable to route messages along paths that are as short as possible. The efficiency of a routing scheme is measured in terms of its *multiplicative stretch* (or *additive stretch*), namely, the maximum ratio (or surplus) between the length of a route produced by the scheme for some pair of nodes, and their distance. A straightforward approach to achieving the goal of guarantees optimal routes is to store a *complete routing table* in each node v in the network, specifying for each destination u the first edge (or an identifier of that edge, indicating the output port) along some shortest path from v to u. However, this approach may be too expensive for large systems since it requires $O(n \log d)$ memory bits for a node of degree d in an n-node network. Thus, an important problem in large scale communication networks is the design of routing schemes that produce efficient routes and have relatively low *memory requirements*.

The routing problem can be presented as requiring to assign two kinds of labels to every node of a graph. The first is the *address* of the node, whereas the second label is a data structure called the *local routing table*. The labels are assigned in such a way that at every source node v and given the address of any destination node u, one can decide the output port of an edge outgoing of

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 150–163, 2002.

v that leads to u. The decision must be taken locally in v, based solely on the two labels of v and with the address label of u, the latter label being forwarded with the message allowing each intermediate node to proceed similarly.

There are several strategies to decrease the size of the routing tables while keeping route lengths close to the shortest paths. One of the most popular (because quite efficient on tree networks) is called *interval routing scheme*, IRS for short, introduced by [SK85,vLT87]. The addresses range in $[1, n]$ with n the number of nodes. At each edge e outgoing of a node u is assigned one or more sub-intervals of $[1, n]$, corresponding to the set of destinations whose the routes from v traverse e (the destination sets assigned to distinct outgoing edges of v must be disjoint). For a tree, using a depth-first search traversal for assigning the addresses, each destination set consists in exactly one sub-interval of $[1, n]$ (modulo n), so that each node of degree d has to maintain a table of $O(d \log n)$ bits storing the boundaries of the intervals. This size has to be compared with the $O(n \log d)$ bound for the complete routing table approach. (An overview of the IRS technique is developed in [Gav00].) Actually, the IRS strategy can be improved for trees if addresses on slightly more than $\lceil \log n \rceil$ bits are used. More precisely, it is constructed in [FG01a] a routing scheme using routing tables and addresses on $c \log n$ bits, for a small constant $c > 1$, even for arbitrary large degree node. Actually, [TZ01] showed that the factor c can be even reduced to $1 + o(1)$ for n large enough.

Unfortunately, such schemes do not hold for general graphs. In [PU89] it is shown that every routing strategy that guarantees a multiplicative s stretched routing scheme for every n-node graph requires $\Omega(n^{1+1/(2s+4)})$ bits in total, so $\Omega(n^{1/(2s+4)})$ for local routing tables, for some worst-case graphs. For the case of optimal stretch (multiplicative 1 or additive 0 stretch), it is shown in [GP96] that for every shortest path routing strategy and for all d and fixed $\epsilon > 0$ such that $3 \leqslant d \leqslant (1 - \epsilon)n$, there exists a graph of degree bounded by d for which $\Omega(n \log d)$ bit routing tables are required simultaneously on $\Theta(n)$ nodes, matching with the memory requirements of complete routing tables. Both lower bounds assume that routes and $O(\log n)$ bit addresses can be computed and optimized by the routing strategy in order to decrease the memory requirement.

These lower bounds are motivations for the design of routing strategies with compact tables on more specific class of graphs. Here we non exhaustively list some of them. Regular topologies (as hypercubes, tori, cycles, complete graphs, etc.) have specific routing schemes (cf. [Lei92]), but one can design also for them an IRS with few intervals as shown for instance in [vLT87]. For non-regular topologies, several trade-offs between the stretch and the size of the routing tables have been achieved, in particular for c-decomposable graphs [FJ90] (including bounded tree-width graphs), planar graphs [FJ89], and bounded genus graphs [GH99]. More recently, a multiplicative $1 + \epsilon$ stretched routing scheme for every planar graph, for every $\epsilon > 0$, with $\log^{O(1)} n$ bit addresses and routing tables has been announced in [Tho01]. For more detailed presentation of these schemes and an overview of the other strategies and techniques, see [Gav01] and [Pel00].

In this paper we investigate *chordal* graphs, namely the class of graphs containing no induced cycles of length greater than 3. Two such graphs are depicted on Fig. 1. Based on the construction of a multiplicative 3-spanner [PS89] with $O(n \log n)$ edges (namely, a spanning subgraph whose the distance between any two nodes does not exceed 3 times the original distance in the graph), [PU89] have constructed a multiplicative 3 stretched routing strategy for chordal graphs using $O(n \log^2 n)$ bits in total for tables and $O(\log^2 n)$ bit addresses. However the scheme does not produce balanced routing tables and $\Omega(n \log n)$ bits might be required at some nodes. If we insist on shortest path (i.e., optimal stretch), no strategy better than complete routing tables is known. Nevertheless, every chordal graph whose all its maximal cliques are of size $k+1$ exactly, namely every k-tree, supports an IRS using at most 2^{k+1} intervals per outgoing edge [NN98]. Actually, as we will show in Section 2, the result can be easily extended to every chordal graph of maximum clique of size at most $k+1$. Derived from the result of [NN98], every chordal graph with maximal clique $k+1$ has shortest path routing tables of $O((2^k + d) \log n)$ bits per node of degree d, and using addresses $\in [1, n]$.

In this paper we show that, while keeping addresses on $O(\log n)$ bits (more precisely on $(2+o(1)) \log n$ bits), we construct a routing scheme for every chordal graph of maximum clique $k+1$ with $O(k \log n)$ bits for local routing tables. This is performed under the "designer-port model", that is the designer of the scheme can permute, during the preprocessing of the graph and the construction of the scheme, the port numbers of all the links attached to the nodes (this assumption is also done in [FG01b,TZ01]). This result is achieved by the use of two main ingredients: 1) the well known tree-decomposition in maximal cliques of chordal graphs; and 2) the recent compact and distributed data structures for trees, in particular answering efficiently routing queries and ancestor queries with small labels [AR01,KM01,KMS02,FG01a,TZ01]. At this step, it is worth to observe that additive r stretched routing scheme on chordal graphs cannot be reduced to the problem of routing in a suitable spanning tree of the graph. Indeed, as mentioned in [Pri97,BCD99], for every fixed integer r there is a chordal graph without tree r-spanners (additive as well as multiplicative).

The paper is organized as follows. In Section 2, we show how to reformulate the result of [NN98] on k-trees to work on chordal graph of maximum clique k. In Section 3, we describe our routing strategy. We propose some open problems in Conclusion.

2 The Extended Scheme of k-Trees

We study at first the strategy presented in [NN98] for k-trees, and show how to adapt it for chordal graphs. The results of [NN98] is based on the following definition of k-trees.

Definition 1. *For integral $k > 0$, the set of k-trees is the smallest set of graphs satisfying: 1) A clique of size k is a k-tree; and 2) let G be a k-tree on n nodes and K be a clique of size k in G. Then the graph on $n+1$ nodes formed by taking G and introducing a new node u adjacent to all of K is a k-tree.*

In this definition, the clique K is named *attachment clique* of the node u, and all the nodes of K the *parents* of u. The node u is a *child* of each nodes of K. The notions of *ancestors* and *descendants* are analogously defined. For each node u, the *depth* of u is either one greater than the maximum depth of any parent of u, or 0 if u has no parents. Fig. 1(a) depicts a k-tree with depth represented at each node. Finally, using the notion of *cluster* of a node, [NN98] proposed for all k-trees a node-labeling from 1 to n supporting a shortest path IRS with at most 2^{k+1} sub-intervals of $[1, n]$ per outgoing edge. Formally, *cluster*(u) is the set of descendants of u, equidistant from all of the parents of u. This non trivial bound results of a long series of lemmas in [NN98] that cannot be presented here. But, roughly speaking, labels of the nodes in *cluster*(u) form a single interval, and [NN98] were able to bound the number of clusters needed at each outgoing edge in order to route along shortest path.

In this paper, we are interested in chordal graphs (a superset of k-trees). Let $\mathcal{C}_k$ denote the family of chordal graphs of maximum clique at most $k+1$. The result of [NN98] can be naturally extended to $\mathcal{C}_k$ with the following adaptations.

Definition 2. (cf. [BLS99]) *Let G be a graph. The node $v \in V(G)$ is* simplicial *in G if the set of neighbors of v induced a clique in G. An ordering $(v_1, v_2, \ldots, v_n)$ of the nodes of G is a* perfect elimination ordering *if for every $i \in \{1, \ldots, n\}$, the node v_i is simplicial in the subgraph of G induced by $\{v_i, \ldots, v_n\}$.*

Proposition 1. (cf. [BLS99]) *A graph G is chordal if and only if it has a perfect elimination ordering.*

Proposition 2. *For integral $k > 0$, $\mathcal{C}_k$ is the smallest set of graphs satisfying: 1) A clique of size at most k is in $\mathcal{C}_k$; and 2) Let $G \in \mathcal{C}_k$ with n nodes and K be a clique of size at most k in G. Then the graph on $n+1$ nodes formed by taking G and introducing a new node adjacent to all of K is a graph of $\mathcal{C}_k$.*

Proof. Every graph G satisfying points 1) and 2) are in $\mathcal{C}_k$. Conversely, if $G \in \mathcal{C}_k$, by Proposition 1, G has a perfect elimination ordering $(v_1, \ldots, v_n)$. $\{v_n\}$ is a clique of size at most $k > 0$, and using successively $v_{n-1}, \ldots, v_i, \ldots, v_1$ we can reconstruct G by connecting v_i to a clique of G, a clique induced by $\{v_i, \ldots, v_n\}$ say K. As the biggest clique in G is of size $k+1$, the size of the neighborhood of v_i, the size of K, is at most k. □

Using the constructive definition of $\mathcal{C}_k$ (Proposition 2), we can extend all notions introduced in [NN98] for k-trees. Namely, the *parents* of u are the nodes of the clique K used to introduce u. From that, we can define naturally, the *children*, *ancestors* and *descendants* of u. The notions of *depth* and of *cluster* of u are defined as previously for k-trees. Note that the only difference with k-trees is that the number of parents of any node is at most k and not exactly k.

Fig. 1(b) represents a graph in $\mathcal{C}_k$ constructed with the same node ordering to the k-tree depicted in (a). Thanks to these extensions, we can check that all lemmas in [NN98] can be rewritten for $\mathcal{C}_k$. So, every $G \in \mathcal{C}_k$ supports a shortest

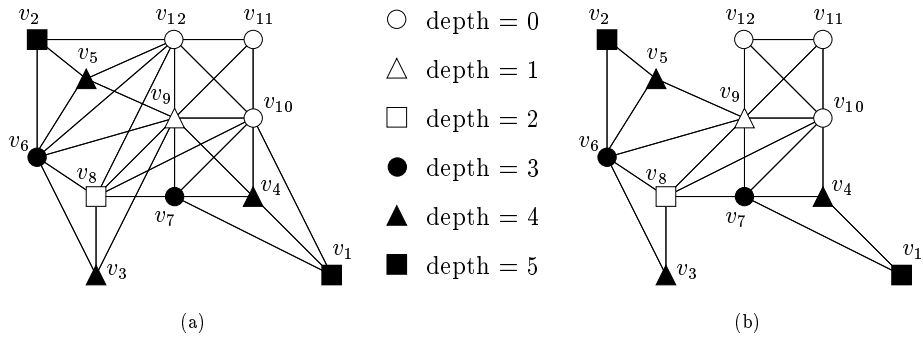


Fig. 1. A 3-tree and a graph in $\mathcal{C}_3$ with same ordering on nodes.

path IRS with at most 2^{k+1} intervals per edge. Actually [NN98] have shown a stronger result: the total number of intervals assigned at the outgoing edges of a node of degree d is at most $S = d - k + \sum_{i=1}^{k} 2^{k-i+2}$. It is known that a such interval routing scheme can be implemented at each node by a table of $O(S \log n)$ bits allowing a routing queries of $O(\log S)$ time per node [Gav00]. Therefore we have:

Theorem 1. *For every n-node graph $G \in \mathcal{C}_k$, there exists a shortest path routing scheme using labels of $\log n$ bits and $O((2^k + d) \log n)$ bits of information in each node of degree d. Moreover, this scheme is polynomial-time constructible and the routing function is computable in $O(k + \log d)$ time.*

3 Additive Stretched Routing Scheme

3.1 Preliminaries

We need the notion of *tree-decomposition* used by Robertson and Seymour in their work on graphs minors [RS86].

Definition 3. *A tree-decomposition of a graph G is a tree T whose nodes are subsets of $V(G)$, such that (see an example on Fig. 2):*

1. $\bigcup_{X \in V(T)} X = V(G)$;
2. *for all $\{u, v\} \in E(G)$, there exists $X \in V(T)$ such that $u, v \in X$; and*
3. *for all $X, Y, Z \in V(T)$, if Y is on the path from X to Z in T then $X \cap Z \subseteq Y$.*

Proposition 3. (cf. [Die00]) *A graph G is a chordal graph if and only if there exists a tree-decomposition of G (polynomial-time constructible) such that for all $X \in V(T)$, X induced a maximal clique in G.*

From now we consider an arbitrary connected graph $\mathcal{G} \in \mathcal{C}_k$ with n nodes. According to Proposition 3, let $\mathcal{T}$ be a tree-decomposition of $\mathcal{G}$ such each one of

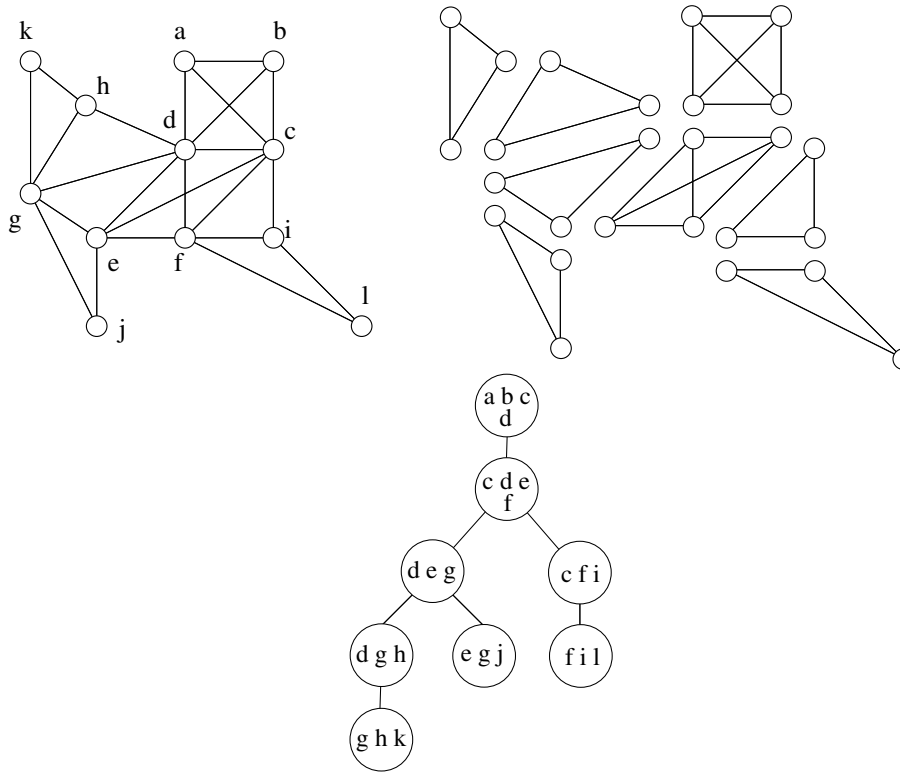


Fig. 2. From left-to-right: a chordal graph G, its set of maximal cliques, and a tree-decomposition of G satisfying Proposition 3.

its nodes induced a maximal clique in $\mathcal{G}$. We assume that $\mathcal{T}$ is rooted. Finally, let S be an arbitrary shortest path spanning tree of $\mathcal{G}$ rooted some node taken from the root of $\mathcal{T}$. (see Fig. 3).

Thanks to the trees S and $\mathcal{T}$, we will show how to construct compact routing tables for $\mathcal{G}$. To prove this fact we need some notations. We will use the standard notions of *children*, *parent*, *ancestors*, *descendants* and *depth* in rooted trees. For simplicity we assume that a node is an ancestor of itself. For every node u of $\mathcal{G}$, the *ball* of u, denoted by $B(u)$, is a node X of $\mathcal{T}$ of minimum depth such $u \in X$. Observe that, once $\mathcal{T}$ has been fixed, $B(u)$ is unique for each u by Rule 3 of Definition 3.

The two following Propositions are very important to understand why the routing scheme we will describe in Paragraph 3.3 is correct.

Proposition 4. *Let u, v be two adjacent nodes of $\mathcal{G}$. One of these statements is true:*

1. *$B(u)$ is an ancestor of $B(v)$ in $\mathcal{T}$ and then $u \in B(v)$.*
2. *$B(v)$ is an ancestor of $B(u)$ in $\mathcal{T}$ and then $v \in B(u)$.*

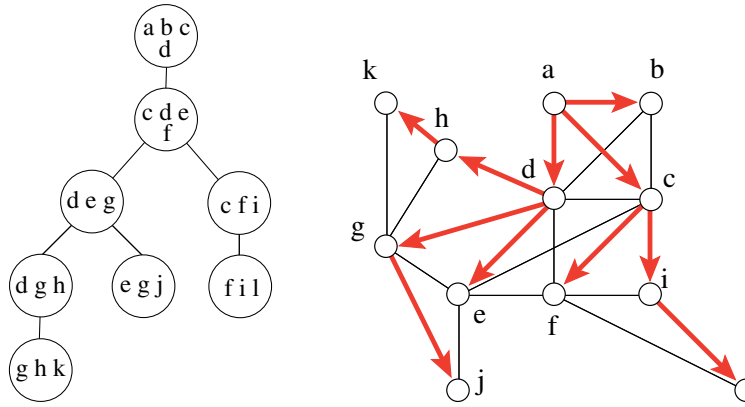


Fig. 3. The tree-decomposition $\mathcal{T}$ and a spanning tree S of $\mathcal{G}$ (directed edges) rooted at a.

Proof. Let u, v be two adjacent nodes of $\mathcal{G}$. If $B(u) = B(v)$ then the two statements are trivially trues (recall that we consider that a node is an ancestor of itself).

So, assume that $B(u) \neq B(v)$. By minimality of the depth of the balls and by definition of $\mathcal{T}$, every ball X containing a node x is a descendant of $B(x)$ in $\mathcal{T}$. Moreover by Rule 2 of Definition 3, there exists a ball Y containing u and v. Thus Y is a descendant of $B(u)$ and of $B(v)$. Thus either $B(u)$ is an ancestor of $B(v)$ or the reverse. If $B(u)$ is an ancestor of $B(v)$ then $B(v)$ is on the path from $B(u)$ to Y in $\mathcal{T}$, by Rule 3 of Definition 3, $u \in B(v)$. Similarly if $B(v)$ is an ancestor of $B(u)$ then $v \in B(u)$. □

Proposition 5. *Let u, v be two nodes of $\mathcal{G}$ such that $B(u)$ is an ancestor of $B(v)$ in $\mathcal{T}$. There exists at least one node $w \in B(u)$, and at most two, such that w is an ancestor of v in S.*

Proof. As $\mathcal{T}$ is a tree-decomposition of $\mathcal{G}$, every path in $\mathcal{G}$ from any node of the root of $\mathcal{T}$ to v, has to use some nodes of $B(u)$. So at least one node of $B(u)$ is ancestor of v in S.

Assume that there exists w_1, w_2, w_3 in $B(u)$ such that they are ancestors of v in S. We can suppose w.l.o.g. that in S: w_1 is an ancestor of w_2, and w_2 is an ancestor of w_3. Say in other words, the path of S traversing w_1, w_2, w_3 is a shortest path from w_1 to w_3 in S and thus in $\mathcal{G}$ (by construction of S). There is a contradiction because $B(u)$ is a clique so $\{w_1, w_3\}$ is an edge. □

3.2 The Routing Scheme

Our scheme associate to every node u of $\mathcal{G}$ two labels: its *address* denoted by *address*(u), and a *local routing table* denoted by *table*(u). The *length* of a label

(*address* or *table*) is the length in bits of its binary representation. A *port-labeling* of $\mathcal{G}$ is a labeling of each directed edge (u, v) of $\mathcal{G}$ by an output port number, an integer denoted by $port(u, v)$, such that $port(u, v) \in [1, \deg(u)]$ and is distinct for every neighbor v of u.

The address of u in $\mathcal{G}$ is defined by $address(u) = \langle ancestor(B(u)), route(u)\rangle$, where:

- $ancestor(X)$, defined for every ball $X \in V(\mathcal{T})$, is a binary label such that X is an ancestor of Y in $\mathcal{T}$ if and only if $f_1(ancestor(X), ancestor(Y)) = true$, for a suitable computable function f_1.
- $route(u)$, defined for every node u of $\mathcal{G}$, is a binary label such that $f_2(route(u), route(v))$, for every $v \neq u$, returns the output port number of the first edge of the path from u to v in S, for a suitable computable function f_2 and a suitable port-labeling of S. Moreover, $f_2(route(u), route(v)) = 1$ if and only if v is a not a descendant of u.

Proposition 6. *There is a suitable port-labeling of S such that the length of $address(u)$ is $(2 + o(1)) \log n$ bits for all u. Moreover, the addresses are polynomial-time constructible.*

Proof. The number of maximal cliques in $\mathcal{G}$ is at most n, thus $|V(\mathcal{T})| \leqslant n$. By the result of [AR01], the length of $ancestor(B(u))$ is at most $\log n + O(\sqrt{\log n})$ bits for every u. In [TZ01], it is shown that the length of $route(u)$ is $\log n + O(\log n / \log\log n)$ bits for a suitable port-labeling of S. Both labelings are polynomial-time constructible. □

We construct for $\mathcal{G}$ a port-labeling as follows: for every edge (u, v) of S, $port(u, v)$ is determined by Proposition 6, and for all edges (u, w) not in S, $port(u, w)$ are distinct integers ranging in $[k + 1, \deg(u)]$ where k is the degree of u in S.

Each node u of $\mathcal{G}$ has a finite set of algorithms (including the functions f_1 and f_2) representing a constant number of bits, and its table label. More precisely, $table(u)$ containing the address of every node $v \in B(u)$ and the port number of the edge (u, v). More precisely,

$$table(u) \;=\; \{(address(v), port(u, v)) \mid v \in B(u)\} \;.$$

W.l.o.g. entries of $table(u)$ are sorted such that the i-th entry $(address(v_i), port(u, v_i))$ is such that in $\mathcal{T}$, $depth(B(v_i)) \geqslant depth(B(v_{i+1}))$.

Proposition 7. *The length of $table(u)$ is $O(k \log n)$ bits, for every $u \in V(\mathcal{G})$.*

Proof. The length of $address(v)$ is $O(\log n)$, $port(u, v)$ can be represented on $O(\log n)$ bits, and as $|B(u)| \leqslant k + 1$, the length of $table(u)$ is $O(k \log n)$ bits. □

Moreover propositions 3 and 6 implies that:

Proposition 8. *The routing scheme is polynomial-time constructible.*

3.3 The Routing Algorithm

Consider u, v two nodes of $\mathcal{G}$, u the sender and v the receiver. We define the procedure SEND(u, v) that returns $port(u, v)$, the port number of the edge on which the message has to be sent. For that we define two subsets of $B(u)$:

$$I = \{w \in B(u) \mid \ w \text{ is an ancestor of } v \text{ in } S\}$$
$$J = \{w \in B(u) \mid \ B(w) \text{ is an ancestor of } B(v) \text{ in } \mathcal{T}\}$$

Recall that we assume that a node is ancestor of itself. Note that for every $v \in B(u)$, the labels $route(v)$ and $ancestor(B(v))$ are contained in $table(u)$. Thus, thanks to functions f_1 and f_2, u is able to construct I and J.

Procedure SEND(u, v): (the four cases are evaluated sequentially)

1. If u is ancestor of v in S then route to v in S.
2. If $I \neq \varnothing$ then route to w such that $w \in I$ and w is of maximum depth in S.
3. If $J \neq \varnothing$ then route to w such that $w \in J$ and $B(w)$ is of maximum depth in $\mathcal{T}$.
4. Otherwise route to w such that $w \in B(u)$ and $B(w)$ is of minimum depth in $\mathcal{T}$.

We now give the correctness of the routing algorithm. Let $\rho(u, v)$ denote the length of the route produced by SEND from u to v. We want to show that $\rho(u, v)$ is bounded by $d(u, v) + 2$, where $d(u, v)$ denotes the distance between u and v in $\mathcal{G}$.

Lemma 1. *If u is an ancestor of v in S, then $\rho(u, v) = d(u, v)$.*

Proof. In this case, SEND routes through the tree S and this produced a shortest path. □

Lemma 2. *Let u, v be two nodes of $\mathcal{G}$ such that $I \neq \varnothing$.*

1. *If there exists a shortest path $u, v_1, \ldots, v_l = v$ in $\mathcal{G}$ from u to v and with $v_1 \in B(u)$, then $\rho(u, v) \leqslant d(u, v) + 1$;*
2. *otherwise $\rho(u, v) \leqslant d(u, v) + 2$.*

Proof. Let u, v be two nodes of $\mathcal{G}$ such that $I \neq \varnothing$ and w be the node of $B(u)$ chosen by the algorithm SEND. w is an ancestor of v in S. So, by Lemma 1, $\rho(w, v) = d(w, v)$. This clearly implies that $\rho(u, v) \leqslant d(u, v) + 2$.

Suppose now that there exists $v_1, \ldots, v_l$ such that $v_1 \in B(u)$ and $u, v_1, \ldots, v_l = v$ is a shortest path in $\mathcal{G}$ from u to v. As w and v_1 are both in $B(u)$ there is an edge between w and v_1. Thus $d(w, v) \leqslant d(v_1, v) + 1 = d(u, v)$ and then $\rho(u, v) \leqslant d(u, v) + 1$. □

Statement 1 of Lemma 2 occurs for instance in Fig. 3, when computing SEND(e, l). The route is e,c,i,l whereas e,f,l is a shortest path. Statement 2 occurs for instance for SEND(g, k) that produces the route g,d,h,k whereas {g, k} is an edge.

Lemma 3. *Let u, v two nodes of $\mathcal{G}$ such that $I = \varnothing$ but $J \neq \varnothing$, then $\rho(u, v) \leqslant d(u, v) + 2$.*

Proof. Let u, v be two nodes of $\mathcal{G}$ such that $I = \varnothing$ but $J \neq \varnothing$, let w be the node of $B(u)$ chosen by the algorithm SEND and let $u, v_1, \ldots, v_l = v$ be a shortest path in $\mathcal{G}$ from u to v.

Note that by the choice of w, $B(w)$ is an ancestor of $B(v)$ in $\mathcal{T}$. Thus Proposition 5 shows that in w we can apply Lemma 2. Moreover by Proposition 4, $B(w)$ is an ancestor of $B(u)$ in $\mathcal{T}$. Applying Proposition 4 between u and v_1, either $B(u)$ is an ancestor of $B(v_1)$ in $\mathcal{T}$ or the reverse. Thus there are only two different cases:

- $B(v_1)$ is an ancestor of $B(w)$ in $\mathcal{T}$ and $v_1 \in B(u)$.
  In this case, Rule 3 of Definition 3 shows that $v_1 \in B(w)$ and then $w, v_1, v_2, \ldots, v_l = v$ is a path in $\mathcal{G}$ from w to v. If it is not a shortest path then $d(w, v) = d(u, v) - 1$, and by Statement 2 of Lemma 2: $\rho(w, v) \leqslant d(w, v) + 2$ thus $\rho(u, v) \leqslant d(u, v) + 2$. Otherwise $d(w, v) = d(u, v)$, but by Statement 1 of Lemma 2: $\rho(w, v) \leqslant d(w, v) + 1$ and here again we have $\rho(u, v) \leqslant d(u, v) + 2$.
- $B(w)$ is a strict ancestor of $B(v_1)$ in $\mathcal{T}$.
  As $I = \varnothing$ and thanks to Proposition 5, $B(u)$ is not an ancestor of $B(v)$ in $\mathcal{T}$. Let X be the nearest common ancestor in $\mathcal{T}$ of $B(u)$ and $B(v)$. As w is the node of J of maximum depth in $\mathcal{T}$, $v_1 \notin J$, thus $v_1 \notin X$. As $\mathcal{T}$ is a tree-decomposition of $\mathcal{G}$, there exists $i \in \{2, \ldots, l\}$ such that $v_i \in X$. Moreover, as $w \in J$, $B(w)$ is an ancestor of X in $\mathcal{T}$. By Rule 3 of Definition 3, $w \in X$. X is a clique so there is an edge between w and v_i. Therefore $u, w, v_i, \ldots, v_l = v$ is a shortest path from u to v, thus $d(w, v) = d(u, v) - 1$. Finally by Lemma 2, we have $\rho(w, v) \leqslant d(w, v) + 2$ and thus $\rho(u, v) \leqslant d(u, v) + 2$. □

Lemma 4. *Let u, v be two nodes of $\mathcal{G}$ such that $I = \varnothing$ and $J = \varnothing$, then $\rho(u, v) \leqslant d(u, v) + 2$.*

Proof. Let u, v be two nodes of $\mathcal{G}$ such that $I = \varnothing$ and $J = \varnothing$, let w be the node chosen by the algorithm SEND (w is the node of $B(u)$ such $B(w)$ is of minimum depth in $\mathcal{T}$), and let $u, v_1, \ldots, v_l, v$ be a shortest path in $\mathcal{G}$ from u to v.

By Proposition 4, $B(w)$ is an ancestor of $B(u)$. Moreover, $B(w)$ is an ancestor of $B(v_1)$. Indeed, applying Proposition 4 between u and v_1, either $B(u)$ is an ancestor of $B(v_1)$ and thus $B(w)$ is an ancestor of $B(v_1)$, or $B(v_1)$ is an ancestor of $B(u)$ and $v_1 \in B(u)$ and by choice of w, $B(w)$ is an ancestor of $B(v_1)$ as well.

Now there exists $i \in \{2, \ldots, l\}$ such that $B(v_i)$ is an ancestor of $B(w)$ because $B(w)$ is not an ancestor of $B(v)$. W.l.o.g. assume that i is minimum (i.e., $B(w)$ is an ancestor of $B(v_{i-1})$). Applying Proposition 4 between v_{i-1} and v_i, we have $v_i \in B(v_{i-1})$. It follows that $v_i \in B(w)$, and that $u, w, v_i, \ldots v_l = v$ is a shortest path from u to v. So $d(w, v) = d(u, v) - 1$.

Step by step we find a sequence $w = w_1, w_2, w_3, \ldots$ each one closer to v (i.e., with $d(w_i, v) = d(w_{i+1}, v) + 1$), or we get a node w_i that falls in Case 1, 2 or 3 of Procedure SEND. So, $\rho(u, v) = i + \rho(w_i, v) \leqslant i + d(w_i, v) + 2 = d(u, v) + 2$ as claimed. □

Proposition 9. *For all u, v, the port returned by* SEND(u, v) *is computable in $O(k)$ time. Moreover, adding a data structure of $O(k \log n)$ bits to u,* SEND(u, v) *can be performed in $O(\log^2 k)$.*

Proof. Let u be a node of $\mathcal{G}$ and *address*(v) the address of a node v of $\mathcal{G}$.

The address of u is the first entry in *table*(u) (recall that entries are sorted), so the label *route*(u) and *route*(v) can be extracted in $O(1)$ time. Then testing if u is ancestor of v in S and routing if it is necessary is done in $O(1)$ time using $p = f_2(route(u), route(v))$.

If $p = 1$, we need to construct I. For that we can test function f_2 on *route*(v) and *route*(w) for every node $w \in B(u)$ (i.e., each entries in *table*(u)). It takes $O(k)$ time. If $I \neq \varnothing$, we find w in $O(1)$ time since I is of size at most 2 (Proposition 5).

In Case 3 of Procedure SEND does not need to construct the whole set J. Just the node of maximum depth in J is required. As *table*(u) is sorted by depth in $\mathcal{T}$ and observing that if $depth(B(v_i)) \geqslant depth(B(v_{i+1}))$ and $B(v_i) \in J$ then $B(v_{i+1}) \in J$, we only need to make $O(\log k)$ tests to find w if it exists. Thus this step needs of $O(\log k)$ time.

In Case 4, w is the last entry in *table*(u). So, it takes $O(1)$ time to find it.

Actually, the time for searching w in I can be improved as follows. We build a search tree M spanning all the nodes of $B(u)$. The root of M is a node of S whose removal consists of a forest whose each tree contains at most $k/2$ nodes of $B(u)$. (We check that this is doable by assigning to each node of S a 0-1 weight: the weight of x is 1 if and only if x is in $B(u)$.) We apply recursively such a decomposition in each tree of the forest, and at each step we label the port number of the nodes of M with the original port numbers in S. The number of nodes in M is $O(k)$ (because the degree of each internal node of M is at least two) and its depth is $h = O(\log k)$. So storing in u the whole tree M with all the labels costs $O(k \log n)$ bits. Using the sublabel *route*(v) in *address*(v) one can "route" in M from its root (this is done virtually in the node u), and find (if it exists) an ancestor of v in $B(u)$. More precisely, at the current node x visited in M we compute $p = f_2(route(x), route(v))$, and test in this order: (1) if p is the port label of an incident edge (x, y) in M, then continue the test in the node y; (2) otherwise (no incident edge of x has label p), then either $x \notin B(u)$ (we test its weight) and then $I = \varnothing$, or $x \in B(u)$ and then we have found $w = u$. The number of tests to perform is at most h, and each test costs $O(\log k)$ time. □

Propositions 6, 7, 8, 9 and Lemmas 1, 2, 3, 4, can be gathered in the following theorem.

Theorem 2. *For every chordal graph of maximum clique of size $k+1$ there exist an additive 2 stretched routing scheme using addresses of size $(2 + o(1)) \log n$ bits and $O(k \log n)$ bits of information in each node, under the assumption that output port numbers can be permuted. Moreover, this scheme is polynomial-time constructible and the routing function is computable in $O(\log^2 k)$ time.*

In the model proposed in this paper (headers cannot be modified), any routing scheme on chordal graphs (stretched or not) must route along short-

est path in trees. And trees are chordal graphs of maximal clique size 2, thus our scheme routes optimally in trees. Observe that if output port numbers cannot be permuted, the size of the tables and of the addresses must be at least $\Omega(\log^2 n/\log\log n)$, by the recent lower bound of [**?**] on trees. Therefore, for every $k = O(\log n/\log\log n)$, every routing scheme on $\mathcal{C}_k$ (stretched or not) that provides better memory requirements than Theorem 2 (tables and addresses) must permute the output port numbers.

4 Conclusion and Discussions

In this paper we have constructed a routing scheme on chordal graphs generating routes close to the shortest paths up to an additive factor two. Our scheme uses pre-computed addresses of $O(\log n)$ bits and routing tables of $O(k \log n)$ bits per node, where $k+1$ is the size of the maximum clique of the graph. The time to route a message in a node is $O(\log k)$.

We note that our scheme is under the "designer-port model", that is the designer of the scheme can permute, during the preprocessing of the graph and the construction of the scheme, the port numbers of all the links attached to the nodes. However, this assumption can be easily relaxed according to scheme based on trees (see [TZ01,FG01a]). Our scheme can be adapted to the fixed-port model (port numbers are fixed in advance and cannot be permuted) with an increasing on the addresses and routing tables size by a $O(\log n/\log\log n)$ factor, the length of the routes remaining the same.

Another direction for improvement is the length of the routes vs. the size of the routing tables. A natural problem would be to find for each integral parameter r, an additive r stretched scheme using, say, $\log^{O(1)} n$ bit addresses and $O(f_r(n,k))$ bit routing tables for a function f_r. Derived from [NN98], we have that $f_0(n,k) \leqslant 2^k \log n$. Our contribution in that paper is a proof for $f_2(n,k) \leqslant k \log n$. These results are complemented with the lower bounds of [Gav02] that shows that $f_0(n,k) \geqslant 2^k \log(n/2^k)$ and that $f_1(n,k) \geqslant 2^k/k$. The question of determining a tight bound for $f_2(n,k)$ is open, and more fundamentally we are wondering if there is a constant r such that $f_r(n,k) = \log^{O(1)} n$, meaning that chordal graphs support additive constant stretched routing scheme with $\log^{O(1)} n$ bit addresses and tables.

Acknowledgments

The author is thankful to Cyril Gavoille for fruitful discussions.

References

AR01. Stephen Alstrup and Theis Rauhe. Improved labeling scheme for ancestor queries. In 13^{th} *Symposium on Discrete Algorithms (SODA)*. ACM-SIAM, January 2001. To appear.

BCD99. Andreas Brandstädt, Victor Chepoi, and Feodor Dragan. Distance approximating trees for chordal and dually chordal graphs. *Journal of Algorithms*, 30:166–184, 1999.

BLS99. Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes – A survey*. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 1999.

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

FG01a. Pierre Fraigniaud and Cyril Gavoille. Routing in trees. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, 28^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2076 of Lecture Notes in Computer Science, pages 757–772. Springer, July 2001.

FG01b. Pierre Fraigniaud and Cyril Gavoille. Routing in trees. Research Report RR-1252-01, LaBRI, University of Bordeaux, 351, cours de la Libération, 33405 Talence Cedex, France, January 2001. Sumitted.

FJ89. Greg N. Frederickson and Ravi Janardan. Efficient message routing in planar networks. *SIAM Journal on Computing*, 18(4):843–857, August 1989.

FJ90. Greg N. Frederickson and Ravi Janardan. Space-efficient message routing in c-decomposable networks. *SIAM Journal on Computing*, 19(1):164–181, February 1990.

Gav00. Cyril Gavoille. A survey on interval routing. *Theoretical Computer Science*, 245(2):217–253, 2000.

Gav01. Cyril Gavoille. Routing in distributed networks: Overview and open problems. *ACM SIGACT News - Distributed Computing Column*, 32(1), March 2001. To appear.

Gav02. Cyril Gavoille. Space lower bounds for routing in chordal graphs with additive stretch, 2002. In preparation.

GH99. Cyril Gavoille and Nicolas Hanusse. Compact routing tables for graphs of bounded genus. In Jiří Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, 26^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1644 of Lecture Notes in Computer Science, pages 351–360. Springer, July 1999.

GP96. Cyril Gavoille and Stéphane Pérennès. Lower bounds for interval routing on 3-regular networks. In Nicola Santoro and Paul Spirakis, editors, 3^{rd} *International Colloquium on Structural Information & Communication Complexity (SIROCCO)*, pages 88–103. Carleton University Press, June 1996.

KM01. Haim Kaplan and Tova Milo. Short and simple labels for small distances and other functions. In 7^{th} *International Workshop on Algorithms and Data Structures (WADS)*, volume 2125 of Lecture Notes in Computer Science, pages 32–40. Springer, August 2001.

KMS02. Haim Kaplan, Tova Milo, and Ronen Shabo. A comparison of labeling schemes for ancestor queries. In 14^{th} *Symposium on Discrete Algorithms (SODA)*. ACM-SIAM, January 2002.

Lei92. Frank Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes*. Morgan Kaufmann, 1992.

NN98. Lata Narayanan and Naomi Nishimura. Interval routing on k-trees. *Journal of Algorithms*, 26(2):325–369, February 1998.

Pel00. David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications, 2000.

Pri97. Erich Prisner. Distance approximating spanning trees. In 14^{th} *Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1200 of Lecture Notes in Computer Science, pages 499–510. Springer, 1997.

PS89. David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.

PU89. David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM*, 36(3):510–530, July 1989.

RS86. Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

SK85. Nicola Santoro and Ramez Khatib. Labelling and implicit routing in networks. *The Computer Journal*, 28(1):5–8, February 1985.

Tho01. Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. In 42^{th} *Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, October 2001.

TZ01. Mikkel Thorup and Uri Zwick. Compact routing schemes. In 13^{th} *Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 1–10, Hersonissos, Crete, Greece, July 2001. ACM PRESS.

vLT87. Jan van Leeuwen and Richard B. Tan. Interval routing. *The Computer Journal*, 30(4):298–307, 1987.

Complexity of Pattern Coloring of Cycle Systems*

Zdeněk Dvořák, Jan Kára, Daniel Král', and Ondřej Pangrác

Department of Applied Mathematics and
Institute for Theoretical Computer Science**
Charles University
Malostranské nám. 25, 118 00 Prague, Czech Republic
{rakdver,kara,kral,pangrac}@kam.ms.mff.cuni.cz

Abstract. A k–cycle system is a system of cyclically ordered k–tuples of a finite set. A pattern is a sequence of letters. A coloring of a k–cycle system with respect to a set of patterns of length k is proper iff each cycle is colored consistently with one of the patterns, i.e. the same/distinct letters correspond to the same/distinct color(s). We prove a dichotomy result on the complexity of coloring a given cycle system with a fixed set of patterns $\mathcal{P}$ by at most l colors and discuss possible generalizations.

1 Introduction

Coloring problems for different combinatorial objects are among intensively studied problems. Problems dealing with colorings of graphs have been generalized to hypergraphs and the original notion of proper colorings of hypergraphs, demanding that no edge of a hypergraph is monochromatic, has been generalized to lots of other structures: Steiner triple and quadruple systems ([1,3,9,10]), mixed hypergraphs ([4]), mixed hypertrees ([5,6]), mixed multigraphs ([7]), block–pattern cycle systems and designs ([11]).

A *k–cycle system* is a pair $\mathcal{C} = (V_\mathcal{C}, C_\mathcal{C})$ where $V_\mathcal{C}$ is a finite set and $C_\mathcal{C}$ is set of cyclically ordered k–tuples of $V_\mathcal{C}$. The members of $V_\mathcal{C}$ are called *vertices* and the members of $C_\mathcal{C}$ are called *cycles*. A k–cycle system is a *k–cycle design* if for each pair of vertices u and v there is exactly one cycle containing uv or vu. Some of the properties of k–cycle systems, especially of k–cycle designs, can be found in [8], e.g. the 4–cycle designs on n vertices exist precisely for $n \bmod 8 = 1$.

A *pattern* of length k is a sequence of letters of length k. The *coloring* c of vertices of a k–cycle system $\mathcal{C}$ with the set of patterns $\mathcal{P}$ of length k is *proper* iff for each cycle C of $\mathcal{C}$ there is a pattern $p \in \mathcal{P}$ which satisfies the following: There is a rotation of C such that the vertices on the positions with the same letters of p are colored by the same color and the vertices on the positions with mutually different letters of p are colored by mutually different colors. This notion of

* The research was done as a part of DIMACS/DIMATIA REU 2001 programme. The REU programme was supported by KONTAKT ME 337.

** Supported by Ministry of Education of Czech Republic as project LN00A056.

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 164–175, 2002.

pattern coloring was introduced in [11] and is actually generalization of several previously introduced notions. Coloring of 2–cycle systems with the pattern AB is just graph coloring: The pattern forces the vertices in the same cycle to be colored by different colors and thus the cycles correspond to the edges of a graph. Coloring of 3–cycle systems with the pattern AAB is just bicoloring of (Steiner) triple systems considered in [1,9,10]. The pattern forces each triple to be colored by exactly two colors. Coloring k–cycle systems with suitable set of patterns also correspond to coloring uniform mixed bihypergraphs of [4].

We address the following complexity problem in this paper: "What is the complexity of decision whether a given cycle system $\mathcal{C}$ with the pattern set $\mathcal{P}$ can be colored by at most l colors ($\mathcal{P}$ and l are fixed)?" We describe all the pattern sets $\mathcal{P}$ and the numbers l for which this problem can be solved in polynomial time and we prove NP–completeness for the remaining cases.

The problem is trivial for $l = 1$. We study the problem for $l \geq 3$ in Section 2. It is, not very surprisingly, NP–complete for any $l \geq 3$ and any set $\mathcal{P}$ of patterns omitting the monochromatic pattern — see Theorem 1. We deal with the remaining case $l = 2$ in Section 3. It is enough to consider only the pattern sets $\mathcal{P}$ containing only patterns consisting of at most two letters. It is not hard to see that if we get an affine subspace over $\mathbb{GF}(2)$ through replacing the letters in the patterns of $\mathcal{P}$ by ones and zeroes and taking all the rotations of the patterns of $\mathcal{P}$, the problem of finding a coloring of the given cycle system is reduced to the problem of finding a solution of a system of linear equations (see the beginning of the Section 3 and Lemma 2 for more details). Lemma 5 states that all the remaining cases are NP–complete (except for $\mathcal{P}$ containing the monochromatic pattern). The results are summarized in Theorem 2.

Theorem 2 is general and it does not provide any examples of patterns for which the problem can be solved in polynomial time. We address the following question in Section 4: "For which patterns p consisting of l distinct letters, the decision problem whether a given cycle system with the pattern set $\{p\}$ can be colored by at most l colors is solvable in polynomial time?" The results of Section 2 and 3 imply that this is possible only for the pattern consisting of one or two letters. In the former case, the problem is actually trivial. In the latter case, these are exactly the patterns whose rotations induce affine subspaces over $\mathbb{GF}(2)$. We list all such patterns in Theorem 3 of Section 4; these patterns are exactly the following ones: A^k, $(AB)^k$, $(AABB)^k$ and $(AAAB)^k$ for all k's (we write X^k for the concatenation of k copies of X). We would like to point the attention of the reader to a quite interesting linear algebra Lemma 6 in Section 4.

We relate our results to previous results in the last section, Section 5. Namely, another proof of one of our main theorems, Theorem 2, is sketched and a more general notion of coloring systems of k-tuples is introduced and counterparts of Theorem 1 and Theorem 2 are stated. Detailed proofs of these two theorems are omitted due to space limitations.

We introduce additional notation: A pattern is called an *l–pattern* if it consists of at most l different letters. The *monochromatic pattern* of length k is A^k (we write throughout the paper for shortness A^k instead of the sequence consist-

ing of k A's), the *alternating pattern* of length k is $(AB)^{k/2}$ (for k even) and the *multichromatic pattern* of length k is a pattern consisting of k mutually different letters, e.g. the multichromatic pattern of length 3 is ABC. We say that the pattern p is *periodic* if it is a concatenation of two or more copies of another pattern. We say that the pattern is *aperiodic* if it is not periodic.

2 NP–Completeness of Coloring of Cycle–Systems

We first prove that it is enough to deal with multichromatic patterns:

Lemma 1. *Let $\mathcal{P}$ be any fixed set of l–patterns of length $k \geq 2$ omitting the monochromatic pattern of length k and $m \geq l$ a fixed integer. Then there exists a k–cycle system $\mathcal{C}$ with m special vertices $v_1, \ldots, v_m$ such that any proper coloring of $\mathcal{C}$ with at most m colors with respect to the pattern set $\mathcal{P}$ assigns the vertices $v_1, \ldots, v_m$ distinct colors.*

Proof. We create the k–cycle system $\mathcal{C}$ on km^2 vertices v_i^j for $1 \leq i \leq m$ and $1 \leq j \leq km$. We add to $\mathcal{C}$ all the k–cycles such that the coloring $c_0(v_i^j) := i$ colors them properly with respect to the set of patterns $\mathcal{P}$. Let $V_i = \{v_i^j | 1 \leq j \leq km\}$.

Let c be any proper coloring of $\mathcal{C}$ using at most m colors. Each V_i contains at least k vertices $U_i \subseteq V_i$ colored by c with the same color γ_i due to the pigeonhole principle. Assume that there is a vertex $v \in V_{i'}$ such that $c(v) = \gamma_i$ for $i \neq i'$. Let p_1 be a pattern of P which contains the greatest number of occurrences of the same letter and let λ_1 be this number. Let C be a cycle containing λ_1 vertices of U_i and the vertex v of $V_{i'}$ — there is certainly such a cycle colored properly by the coloring c_0 with respect to the (non–monochromatic) pattern p_1 for any λ_1–tuple of vertices of V_i and any single vertex of $V_{i'}$. But the cycle C contains at least $\lambda_1 + 1$ vertices colored by c with the same color γ_i which is impossible due to the choice of p_1. Hence the vertices colored by c with the color γ_i are only in V_i. Since c uses at most m colors, all the vertices of V_i have to be colored by c with the color γ_i and thus c_0 is upto renaming the colors the only proper coloring of $\mathcal{C}$. Choosing vertices $v_1^1, \ldots, v_m^1$ to be the special vertices of $\mathcal{C}$ completes the proof.

It is quite easy to prove the NP–completeness result of this section using Lemma 1.

Theorem 1. *Let $\mathcal{P}$ be any fixed set of l–patterns of length $k \geq 2$ omitting the monochromatic pattern of length k and let $l' \geq \max\{3, l\}$ be a fixed integer. Then the decision problem whether a given k–cycle system with the pattern set $\mathcal{P}$ can be colored by at most l' colors is NP–complete.*

On the other hand, if $\mathcal{P}$ contains the monochromatic pattern, any cycle system with $\mathcal{P}$ can be colored by one color and the problem is trivial.

Proof. We may w.l.o.g. assume that $k = l = l' \geq 3$ and $\mathcal{P}$ contains only the multichromatic pattern of length k due to Lemma 1. We present the reduction

from the well–known NP–complete problem (see [2]) whether a given graph can be colored by at most k colors. Let G be a graph with vertices $v_1, \ldots, v_n$ and edges $e_1, \ldots, e_m$. We create a k–cycle system $\mathcal{C}$. $\mathcal{C}$ contains $m(k-2)+n$ vertices $v_1, \ldots, v_n$ and w_i^j for $1 \leq i \leq m$ and $1 \leq j \leq k-2$. We add a cycle $u, v, w_i^1, \ldots, w_i^{k-2}$ for each edge $e_i = uv$, $1 \leq i \leq m$. It is easy to see that G can be colored by at most k colors iff $\mathcal{C}$ can be colored by at most k colors.

The condition that $l' \geq l$ in the statement of the theorem is not restrictive because the patterns containing more than l' letters can be removed from the set $\mathcal{P}$ without changing the complexity of the problem.

3 Complexity of Two–Coloring of Cycle Systems

We develop a connection between colorings using two colors and linear algebra. The calculations are done over the field $\mathbb{GF}(2)$ and the elements of $\mathbb{GF}(2)$ represent the colors. Let $\mathcal{P}$ be the set of 2–patterns of length k. Let $\mathcal{A}(\mathcal{P})$ be the set of all the vectors over $\mathbb{GF}(2)$ of length k such that they are consistent with the set of patterns $\mathcal{P}$, i.e., there is a cyclic rotation p of a pattern of $\mathcal{P}$ such that the vector contains zeroes exactly in those positions where p has A's and ones where p has B's (or vice versa). We say that $\mathcal{P}$ *can be described by a system of linear equations* iff $\mathcal{A}(\mathcal{P})$ forms an affine subspace of $\mathbb{GF}(2)^k$, i.e., there exists a matrix A of size $k' \times k$ and a vector b of size k' such that $\mathcal{A}(\mathcal{P}) = \{x | Ax = b\}$. We say that the matrix A and the vector b *describe* the set of patterns $\mathcal{P}$ in such case.

Lemma 2. *The decision problem, whether a given k–cycle system with the 2–pattern set $\mathcal{P}$ of length $k \geq 2$ can be colored by at most 2 colors, can be solved in polynomial time if $\mathcal{P}$ can be described by a system of linear equations.*

Proof. Let A be the matrix of size $k' \times k$ and let b the vector of size k' which describe $\mathcal{P}$. $\mathcal{A}(\mathcal{P})$ forms an affine subspace of $\mathbb{GF}(2)^k$. Let $\mathcal{C}$ be a given k–cycle system with the pattern set $\mathcal{P}$, let $v_1, \ldots, v_n$ be the vertices of $\mathcal{C}$ and let m be the number of cycles of $\mathcal{C}$. We form a system of mk' equations with n variables $x_1, \ldots, x_n$. We add for each cycle $v_{i_1}, \ldots, v_{i_k}$ of $\mathcal{C}$ the following k' equations:

$$A \begin{pmatrix} x_{i_1} \\ \vdots \\ x_{i_k} \end{pmatrix} = b$$

The solutions $x_1, \ldots, x_n$ of these equations one–to–one correspond to proper 2–colorings c of $\mathcal{C}$ through equalities $c(v_i) = x_i$ for $1 \leq i \leq n$. Hence the decision problem from the statement of the lemma can be solved in polynomial time.

Lemma 3. *The decision problem whether a given 3–cycle system with the pattern AAB can be colored by at most 2 colors, is NP–complete.*

Proof. We present an easy reduction from the well–known NP–complete problem of not–all–equal satisfiability (NAE-SAT) (cf. [2]). The problem is to decide whether for a given formula there exists a variable assignment such that each clause contains both a positive and a negative literal. The problem is NP–complete even for formulas with all the clauses consisting of exactly three literals.

Let Φ be a given formula with clauses of sizes exactly three. Let $x_1, \ldots, x_n$ be the variables of Φ. Let $\mathcal{C}$ be a 3–cycle system with the pattern AAB with two vertices which are forced to be colored by different colors described in Lemma 1. We take n copies of $\mathcal{C}$ with special vertices $v_1, \ldots, v_n$ and $v'_1, \ldots, v'_n$ (the vertices v_i and v'_i belong to the same copy of $\mathcal{C}$). We add for each clause of the formula Φ a 3–cycle which contains v_i iff the clause contains x_i and v'_i iff the clause contains the negation of x_i. The constructed 3–cycle system can be 2–colored iff the formula Φ can be NAE–satisfied: Let c be a proper 2–coloring and let the colors used by c be 0 and 1. We set x_i to false if $c(v_i) = 0$ and to true otherwise ($c(v_i) = 1$). The obtained truth assignment NAE–satisfies the formula Φ: Coloring of a vertex v_i (v'_i) by 0/1 represents that the value of (the negation of) x_i is false/true. The same correspondence also works in the opposite direction, i.e. when constructing a proper coloring from a NAE–satisfying assignment.

Lemma 4. *Let $\mathcal{P}$ be a set of 2–patterns of length $k \geq 2$ without the monochromatic pattern. If there is a k–cycle system $\mathcal{C}$ with $\mathcal{P}$ with vertices u, v and w such that there are exactly 3 (up to renaming the colors) ways in which proper 2–colorings color u, v and w, then the decision problem whether a given k–cycle system can be colored by at most 2 colors with respect to $\mathcal{P}$ is NP–complete.*

Proof. The possible ways of coloring u, v and w with colors A and B are either AAB, ABA, BAA, ABB, BAB, BBA or AAA, BBB, AAB, BBA, ABA, BAB. In the first case, $\mathcal{C}$ forces the vertices to be colored consistently with the pattern AAB and the problem is NP–complete due to Lemma 3. In the latter case, we use Lemma 1 which provides a k–cycle system $\mathcal{C}_{AB}$ with two special vertices u' and u'' which are forced to be colored by different colors. We create a cycle system $\mathcal{C}'$ from a copy of $\mathcal{C}$ and a copy of $\mathcal{C}_{AB}$ by identifying the vertices u'' and u. The possible ways of coloring its vertices u', v and w are BAA, ABB, BAB, ABA, BBA, AAB and we reduced the latter case to the former one.

We prove that Lemma 2 actually describes all the polynomial cases:

Lemma 5. *If a set $\mathcal{P}$ of 2–patterns of length $k \geq 2$ omitting the monochromatic pattern cannot be described by a system of linear equations, the decision problem, whether there is 2–coloring of a given k–cycle system with the pattern set $\mathcal{P}$, is NP–complete.*

On the other hand, if $\mathcal{P}$ can be described by a system of linear equations, and the dimension of $\mathcal{A}(\mathcal{P})$ is k', then there exists $\beta_1, \ldots, \beta_{k'+1} \in \mathbb{GF}(2)$ such that the following system of k equations describes $\mathcal{P}$:

$$\sum_{i=1}^{k'+1} \beta_i x_{1+(j+i-2) \bmod k} = 1 \quad \text{for } 1 \leq j \leq k$$

Proof. We find a system of equations describing $\mathcal{P}$ or we prove that the decision problem from the statement of the lemma is NP–complete for the given set $\mathcal{P}$.

Let $\mathcal{A}(\mathcal{P})$ be the set of zero–one vectors defined in the beginning of this section. We understand the vectors of $\mathcal{A}(\mathcal{P})$ as sequences of length k. We say that $\alpha_1, \ldots, \alpha_\kappa$ *uniquely determines* the $(\kappa+1)$-th coordinate iff the $(\kappa+1)$-th coordinate of each vector of $\mathcal{A}(\mathcal{P})$, whose first κ coordinates are equal to $\alpha_1, \ldots, \alpha_\kappa$, is uniquely determined. If both $\alpha_1, \ldots, \alpha_\kappa$ and $\alpha_1, \ldots, \alpha_{i-1}, \alpha_i + 1, \alpha_{i+1}, \ldots, \alpha_\kappa$ uniquely determine the $(\kappa+1)$-th coordinate, we say that α_i is *essential* $(1 \leq i \leq \kappa)$ iff the $(\kappa+1)$-th coordinate differs for $\alpha_1, \ldots, \alpha_\kappa$ and $\alpha_1, \ldots, \alpha_{i-1}, \alpha_i + 1, \alpha_{i+1}, \ldots, \alpha_\kappa$. We say that α_i is *non–essential* otherwise.

Let κ be the smallest number such that there exists a sequence $\alpha_1, \ldots, \alpha_\kappa$ such that it uniquely determines the $(\kappa+1)$-th coordinate; if it does not exist, $\mathcal{A}(\mathcal{P})$ contains all the vectors of length k. For each sequence $\alpha'_1, \ldots, \alpha'_\kappa$ there exists a vector of $\mathcal{A}(\mathcal{P})$ whose first κ coordinates are equal to $\alpha'_1, \ldots, \alpha'_\kappa$: Otherwise, let $\alpha'_1, \ldots, \alpha'_\kappa$ be the sequence such that there is not a vector of $\mathcal{A}(\mathcal{P})$ whose first κ coordinates are equal to $\alpha'_1, \ldots, \alpha'_\kappa$. Let κ' be the largest number such that there is a vector of $\mathcal{A}(\mathcal{P})$ whose first κ' coordinates are equal to $\alpha'_1, \ldots, \alpha'_{\kappa'}$; such a number exists because $\mathcal{A}(\mathcal{P})$ is closed under negations. Hence $1 \leq \kappa' < \kappa$. The sequence $\alpha'_1, \ldots, \alpha'_{\kappa'}$ determines the $(\kappa'+1)$-th coordinate (it forces it to be $\alpha'_{\kappa'+1} + 1$) and this contradicts the choice of $\alpha_1, \ldots, \alpha_\kappa$ and κ.

We prove: Each sequence $\alpha'_1, \ldots, \alpha'_\kappa$ uniquely determines the $(\kappa+1)$-th coordinate (unless the problem is NP–complete). Assume the opposite and let $\alpha'_1, \ldots, \alpha'_\kappa$ be the sequence with the longest initial subsequence common with $\alpha_1, \ldots, \alpha_\kappa$ which does not uniquely determine the $(\kappa+1)$-th coordinate. Let κ' be the smallest number such that $\alpha_{\kappa'} \neq \alpha'_{\kappa'}$; $\kappa' \geq 2$ since $\mathcal{A}(\mathcal{P})$ contains together with each vector also its negation. Due to Lemma 1 there exists a k–cycle system $\mathcal{C}_{AB}$ which forces two of its vertices to have different colors. Thus we can force two vertices to have different colors and (since we work with 2–colorings) we can also force two vertices to have the same color by using $\mathcal{C}_{AB}$ twice. We create a k–cycle system $\mathcal{C}$ with vertices $v_1, \ldots, v_k$ as follows: We add for each $1 < i \leq \kappa, i \neq \kappa'$ either one copy of $\mathcal{C}_{AB}$ or two copies of $\mathcal{C}_{AB}$ as follows:

- If $\alpha_i = \alpha'_i = \alpha_1 = \alpha'_1$, we force the colors of v_i and v_1 to be the same.
- If $\alpha_i = \alpha'_i \neq \alpha_1 = \alpha'_1$, we force the colors of v_i and v_1 to be different.
- If $\alpha_i \neq \alpha'_i$, $\alpha_i = \alpha_{\kappa'}$ and $\alpha'_i = \alpha'_{\kappa'}$, we force the colors of v_i and $v_{\kappa'}$ to be the same.
- If $\alpha_i \neq \alpha'_i$, $\alpha_i = \alpha'_{\kappa'}$ and $\alpha'_i = \alpha_{\kappa'}$, we force the colors of v_i and $v_{\kappa'}$ to be different.

Next, we add a cycle $v_1, \ldots, v_k$. The vertices $v_1, \ldots, v_\kappa$ are colored consistently with either $\alpha_1, \ldots, \alpha_\kappa$ or $\alpha'_1, \ldots, \alpha'_\kappa$ by any proper coloring using at most two colors. In the first case, the colors of v_1 and $v_{\kappa'}$ uniquely determine the color of $v_{\kappa+1}$, in the second case the color of $v_{\kappa+1}$ can be arbitrary. $\mathcal{C}$ is a k–cycle system such that the vertices v_1, $v_{\kappa'}$ and $v_{\kappa+1}$ can be colored in 3 different ways (upto permutation of the colors). Thus the decision problem of 2–colorability of k–cycle systems with the pattern set $\mathcal{P}$ is NP–complete due to Lemma 4.

We summarize: **Unless the problem is NP–complete, there exists κ such that each sequence $\alpha_1, \ldots, \alpha_\kappa$ uniquely determines the $(\kappa+1)$-th coordinate.**

The preceding immediately implies, since $\mathcal{A}(\mathcal{P})$ is closed under rotations, that $\mathcal{A}(\mathcal{P})$ contains exactly 2^κ vectors — the first κ coordinates uniquely determines the rest of the vector. We prove that α_i is either essential for the $(\kappa+1)$-th coordinate for all the choices of $\alpha_1, \ldots, \alpha_\kappa$ or non–essential for all the choices (unless the problem is NP–complete). Let us assume the opposite: Let $\alpha_1, \ldots, \alpha_\kappa$ be the sequence where α_i is essential for $\alpha_{\kappa+1}$ and $\alpha'_1, \ldots, \alpha'_\kappa$ be the sequence where α'_i is non–essential for $\alpha'_{\kappa+1}$. Assume that $\alpha_1 = \alpha'_1$, because $\mathcal{A}(\mathcal{P})$ contains together with each vector also its negation, and assume also that the $(\kappa+1)$-th coordinates determined by $\alpha_1, \ldots, \alpha_\kappa$ and by $\alpha'_1, \ldots, \alpha'_\kappa$ are the same (otherwise we could change α_i in the first sequence to $\alpha_i + 1$ and α'_i in the second one to $\alpha'_i + 1$—note that this preserves $\alpha_1 = \alpha'_1$ if $i = 1$); let $\alpha_{\kappa+1}$ be this value. Let κ' be the smallest number different from i such that $\alpha_{\kappa'} \neq \alpha'_{\kappa'}$. We create a k–cycle system $\mathcal{C}$ with vertices $v_1, \ldots, v_k$ similarly to the above paragraph. We add for each $1 < j \leq \kappa+1, j \notin \{\kappa', i\}$ either one or two copies of $\mathcal{C}_{AB}$ as follows:

- If $\alpha_j = \alpha'_j = \alpha_1 = \alpha'_1$, we force the colors of v_j and v_1 to be the same.
- If $\alpha_j = \alpha'_j \neq \alpha_1 = \alpha'_1$, we force the colors of v_j and v_1 to be different.
- If $\alpha_j \neq \alpha'_j$, $\alpha_j = \alpha_{\kappa'}$ and $\alpha'_j = \alpha'_{\kappa'}$, we force the colors of v_j and $v_{\kappa'}$ to be the same.
- If $\alpha_j \neq \alpha'_j$, $\alpha_j = \alpha'_{\kappa'}$ and $\alpha'_j = \alpha_{\kappa'}$, we force the colors of v_j and $v_{\kappa'}$ to be different.

Next, we add a cycle $v_1, \ldots, v_k$. Each proper coloring of $\mathcal{C}$ with at most two colors colors the vertices $v_1, \ldots, v_\kappa, v_{\kappa+1}$ consistently with either $\alpha_1, \ldots, \alpha_\kappa$, $\alpha_{\kappa+1}$ or with $\alpha'_1, \ldots, \alpha'_{i-1}$, 0, $\alpha'_{i+1}, \ldots, \alpha'_\kappa, \alpha'_{\kappa+1}$ or with $\alpha'_1, \ldots, \alpha'_{i-1}$, 1, $\alpha'_{i+1}, \ldots, \alpha'_\kappa$, $\alpha'_{\kappa+1}$ (recall that $\alpha_{\kappa+1} = \alpha'_{\kappa+1}$). Thus $\mathcal{C}$ is a k–cycle system with the pattern set $\mathcal{P}$ such that the vertices v_1, v_i and $v_{\kappa'}$ can be colored by colorings using at most two colors only in three different ways upto renaming the colors. Thus the problem of 2–colorability of k–cycle systems with the pattern set $\mathcal{P}$ is NP–complete due to Lemma 4.

We can summarize: **Unless the problem is NP–complete, there exists κ such that each sequence $\alpha_1, \ldots, \alpha_\kappa$ uniquely determines the $(\kappa+1)$-th coordinates and α_i is either essential or non–essential for the $(\kappa+1)$-th coordinate regardless the choice of $\alpha_1, \ldots, \alpha_\kappa$.**

Let I be the set of all i's for which α_i is essential for the $(\kappa+1)$-th coordinate and let γ be the value of the $(\kappa+1)$-th coordinate determined by the sequence of κ zeroes. Let $\alpha_1, \ldots, \alpha_\kappa$ be any sequence of length κ, then this sequence forces the $(\kappa+1)$-th coordinate to be $\gamma + \sum_{i \in I} \alpha_i$, since each change of α_i for $i \in I$ from 0 to 1 changes the value of the $(\kappa+1)$-th coordinate and the change of α_i for $i \notin I$ from 0 to 1 does not affect the $(\kappa+1)$-th coordinate. Let $I' = I \cup \{\kappa+1\}$. The previous can be restated ($\mathcal{A}(\mathcal{P})$ is closed under rotation) as follows:

$$\sum_{i \in I'} \alpha_{1+(i+j-2) \bmod k} = \gamma \quad \text{for all } 1 \leq j \leq k$$

The first $k-\kappa$ equations above, for $1 \leq j \leq k-\kappa$, are linearly independent and since $\mathcal{A}(\mathcal{P})$ contains 2^κ vectors these equations describe $\mathcal{A}(\mathcal{P})$. Since the all–zero vector does not belong to $\mathcal{A}(\mathcal{P})$, the constant γ has to be one (otherwise the all–zero vector would belong to $\mathcal{A}(\mathcal{P})$). The above system of equations can be clearly rewritten to the form from the statement of the lemma by setting $k' = \kappa$, $\beta_i = 1$ for $i \in I'$ and $\beta_i = 0$ for $i \notin I'$.

Lemma 2 and Lemma 5 immediately imply the following theorem:

Theorem 2. *Let $\mathcal{P}$ be a set of 2–patterns of length $k \geq 2$. The decision problem whether a given k–cycle system with the pattern set $\mathcal{P}$ can be colored by at most 2 colors is solvable in polynomial time if and only if at least one of the following two conditions holds (unless P=NP):*

- *$\mathcal{P}$ contains the monochromatic pattern of length k.*
- *$\mathcal{P}$ can be described by a system of linear equations.*

Otherwise, the problem is NP–complete.

4 Single Polynomial Patterns

We answer the question for which single 2–patterns there is a polynomial–time algorithm for deciding whether a given cycle system can be 2–colored. We first state and prove an interesting linear algebra lemma:

Lemma 6. *Let λ be a power of two and let $\alpha_1, \ldots, \alpha_\lambda$ be any sequence of 0's and 1's which contains at least one 1. Then the following system of equations has a solution over $\mathbb{GF}(2)$:*

$$\begin{pmatrix} \alpha_1 & \alpha_2 & \ddots & \alpha_{\lambda-1} & \alpha_\lambda \\ \alpha_\lambda & \alpha_1 & \ddots & \alpha_{\lambda-2} & \alpha_{\lambda-1} \\ \ddots & \ddots & \ddots & \ddots & \ddots \\ \alpha_3 & \alpha_4 & \ddots & \alpha_1 & \alpha_2 \\ \alpha_2 & \alpha_3 & \ddots & \alpha_\lambda & \alpha_1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{\lambda-1} \\ x_\lambda \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}$$

Proof. The proof proceeds by induction on λ. The statement is trivial for $\lambda = 1$; let $\lambda \geq 2$. Let A be the matrix consisting of α_i's from the statement of the lemma. Let A_1 and A_2 be the $\lambda/2 \times \lambda/2$ matrices forming the matrix A:

$$A = \begin{pmatrix} A_1 & A_2 \\ A_2 & A_1 \end{pmatrix}$$

We construct the solution recursively. We distinguish two cases:

- **$A_1 + A_2$ is a non–zero matrix ($A_1 \neq A_2$).** Then the matrix $A_1 + A_2$ is the matrix for the sequence $\alpha_1 + \alpha_{\lambda/2+1}, \ldots, \alpha_{\lambda/2} + \alpha_\lambda$. Let $x_1, \ldots, x_{\lambda/2}$ be the solution for the sequence $\alpha_1 + \alpha_{\lambda/2+1}, \ldots, \alpha_{\lambda/2} + \alpha_\lambda$. Setting $x_{(\lambda/2)+1} = x_1, \ldots, x_\lambda = x_{\lambda/2}$ yields a solution of the original system of linear equations.

– A_1+A_2 **is a zero matrix** ($A_1 = A_2$)**.** Then $\alpha_i = \alpha_{i+\lambda/2}$ for all $1 \leq i \leq \lambda/2$. The matrix $A_1 = A_2$ is the matrix obtained for the sequence $\alpha_1, \ldots, \alpha_{\lambda/2}$. Let $x_1, \ldots, x_{\lambda/2}$ be the solution for the sequence $\alpha_1, \ldots, \alpha_{\lambda/2}$. Setting $x_{\lambda/2+1} = \ldots = x_\lambda = 0$ yields a solution of the original system of linear equations.

We prove that aperiodic patterns for which the above stated problem can be solved in polynomial time have only lengths equal to powers of two:

Lemma 7. *If $\mathcal{P}$ is a set containing a single aperiodic 2–pattern of length k which can be described by a system of linear equations, then k is either $2^{k'-1}$ or $2^{k'}$ where k' is the dimension of the affine subspace $\mathcal{A}(\mathcal{P})$ considered over $\mathbb{GF}(2)$.*

Proof. The affine subspace $\mathcal{A}(\mathcal{P})$ contains all the rotations and negations of the only pattern contained in $\mathcal{P}$. If it can be described by a system of linear equations, then the size of $\mathcal{A}(\mathcal{P})$ is $2^{k'}$ where where k' is its dimension. If all the vectors of $\mathcal{A}(\mathcal{P})$ can be obtained just by the rotations of the pattern, then $k = 2^{k'}$. Otherwise, each rotation of the pattern adds to $\mathcal{A}(\mathcal{P})$ itself and its negation and thus the size of $\mathcal{A}(\mathcal{P})$ is $2k$ and $k = 2^{k'-1}$.

We describe the single patterns allowing a polynomial–time algorithm:

Theorem 3. *The only 2–patterns p (upto rotation) for which the decision problem whether a given k–cycle system with the pattern set $\{p\}$ can be colored by at most 2 colors can be solved in polynomial time (unless P=NP) are the following:*

– $p = A^k$
– $p = (AB)^{k/2}$ *for k even*
– $p = (AABB)^{k/4}$ *for k divisible by four*
– $p = (AAAB)^{k/4}$ *for k divisible by four*

The problem is NP–complete for all the remaining ones.

Proof. Let us assume that there is a polynomial algorithm for 2–coloring of k–cycle systems with the set of patterns $\mathcal{P} = \{p\}$ where p is not the monochromatic pattern. In this case, $\mathcal{P}$ can be described by a system of linear equations due to Theorem 2. If the only pattern p of $\mathcal{P}$ is periodic, i.e. of the form p'^κ where p' is an aperiodic 2–pattern and $\kappa \geq 2$ is an integer, then also $\{p'\}$ can be described by a system of linear equations: Let $\beta_1, \ldots, \beta_{k'+1}$ be the coefficients of the linear equations from Lemma 5 for $\mathcal{P}$. The following system of linear equations clearly describes $\{p'\}$:

$$\sum_{i=1}^{k'+1} \beta_i x_{1+(i+j-2) \bmod (k/\kappa)} = 1 \quad \text{for } 1 \leq j \leq k/\kappa$$

On the other hand, if $\{p'\}$ can be described by a system of linear equations, then clearly also each pattern p'^κ for $\kappa \geq 2$ can be described by a system of linear equations. Thus it is enough to prove that the only aperiodic patterns (except for the monochromatic one) which can be described by a system of linear equations

are AB, $AABB$ and $AAAB$. These patterns can be indeed described by a system of linear equations through the following choice of β_i's:

$$\begin{aligned}
&\text{AB}: \quad \beta_1 = 1, \beta_2 = 1\\
&\text{AABB}: \beta_1 = 1, \beta_2 = 0, \beta_3 = 1\\
&\text{AAAB}: \beta_1 = 1, \beta_2 = 1, \beta_3 = 1, \beta_4 = 1
\end{aligned}$$

We prove that the three above mentioned aperiodic patterns are the only ones which can be described by a system of linear equations. We assume that $p = p'$. Let k' be the dimension of $\mathcal{A}(\mathcal{P})$. The length of p is either $2^{k'-1}$ or $2^{k'}$ due to Lemma 7. We prove that there exists a pattern q of length $K = 2^{\lceil \log_2(k'+1) \rceil}$ such that the vector corresponding to $q^{k/K}$ is a solution of the system of equations for $\mathcal{A}(\mathcal{P})$. Such an aperiodic pattern p of length k cannot exist for $K < k$. Since $2^{k'-1} \leq k$ and $K = 2^{\lceil \log_2(k'+1) \rceil}$, there are no aperiodic patterns for $k' \geq 5$. It is routine to check that the only aperiodic patterns which can be described by a system of linear equations for $k' = 1, 2, 3, 4$ are AB, $AABB$ and $AAAB$.

Note that $K \geq k'+1$ due to choice of K. Let us set $\alpha_i = \beta_i$ for $1 \leq i \leq k'+1$ and $\alpha_i = 0$ for $k'+2 \leq i \leq K$. Let $x_1, \ldots, x_K$ be the solution of the following system of equalities (its existence follows from Lemma 6):

$$\begin{pmatrix}
\alpha_1 & \alpha_2 & \ddots & \alpha_{K-1} & \alpha_K \\
\alpha_K & \alpha_1 & \ddots & \alpha_{K-2} & \alpha_{K-1} \\
\ddots & \ddots & \ddots & \ddots & \ddots \\
\alpha_3 & \alpha_4 & \ddots & \alpha_1 & \alpha_2 \\
\alpha_2 & \alpha_3 & \ddots & \alpha_K & \alpha_1
\end{pmatrix}
\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{K-1} \\ x_K \end{pmatrix} =
\begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}$$

Let q be the pattern corresponding to x through changing zeroes to A's and ones to B's. Then the periodic pattern $q^{k/K}$ has to belong to $\mathcal{P}$ contradicting the fact that $\mathcal{P}$ contains only a single aperiodic pattern.

5 Relation to Other Results

Schaefer studied in [12] the complexity of the satisfiability problem with predicates of restricted types. Namely, he proved the following dichotomy result (we freely use term "described by a system of linear equations" in the obvious meaning through this section even if not defined precisely before for certain types of objects):

Theorem 4. *A satisfiability problem for formulas which are conjunctions of predicates of restricted types is NP-complete unless one of the following six conditions holds:*

1. *A predicate of each type is true when all its variables are set to false.*
2. *A predicate of each type is true when all its variables are set to true.*

3. *A predicate of each type can be defined by a CNF formula consisting only of clauses containing at most one negated variable.*
4. *A predicate of each type can be defined by a CNF formula consisting only of clauses containing at most one unnegated variable.*
5. *A predicate of each type can be defined by a CNF formula consisting only of clauses of size at most two.*
6. *A predicate of each type can be described by a system of linear equations.*

Another way of proving Theorem 2 is deducing it from Theorem 4: The two colors assigned to the vertices of a cycle systems can be represented by false/true values (each vertex gets a single variable). Each predicate in the formula correspond to a single cycle of the system and the predicate is true iff the corresponding cycle is colored properly. Another proof of Theorem 2 may be in showing that these predicates (unless a monochromatic pattern is present) cannot fall into any of the polynomial cases except for the last one.

Theorem 4 suggests the following generalization of cycle systems and their coloring: A *k–tuple system* is a pair $\mathcal{T} = (V_\mathcal{T}, T_\mathcal{T})$ where $V_\mathcal{T}$ is a finite set of vertices and $T_\mathcal{T}$ is set of ordered k–tuples of $V_\mathcal{T}$. The difference between cycle systems and tuple systems is replacing a cyclic ordering with an usual ordering. A *coloring* c of vertices of a k–tuple system $\mathcal{T}$ with the set of patterns $\mathcal{P}$ of length k is *proper* iff for tuple T of $\mathcal{T}$ there is a pattern $p \in \mathcal{P}$ such that the vertices on the positions with the same letters of p are colored by the same color and the vertices on the positions with mutually different letters of p are colored by mutually different colors. Cycle systems may be viewed as tuple systems with pattern sets closed under rotations of the patterns.

In a similar way, the following counterparts of Theorem 1 and Theorem 2 may be proved:

Theorem 5. *Let $\mathcal{P}$ be any fixed set of l–patterns of length $k \geq 2$ omitting the monochromatic pattern of length k and let $l' \geq \max\{3, l\}$ be a fixed integer. Then the decision problem whether a given k–tuple system with the pattern set $\mathcal{P}$ can be colored by at most l' colors is NP–complete.*

On the other hand, if $\mathcal{P}$ contains the monochromatic pattern, any tuple system with $\mathcal{P}$ can be colored by one color and the problem is trivial.

Theorem 6. *Let $\mathcal{P}$ be a set of 2–patterns of length $k \geq 2$. The decision problem whether a given k–tuple system with the pattern set $\mathcal{P}$ can be colored by at most 2 colors is solvable in polynomial time if and only if at least one of the following two conditions holds (unless P=NP):*

- *$\mathcal{P}$ contains the monochromatic pattern of length k.*
- *$\mathcal{P}$ can be described by a system of linear equations.*

Otherwise, the problem is NP–complete.

We omit the proofs of these two theorems due to space limitations though they are not straightforward generalizations of the proofs of Theorem 1 and Theorem 2.

Acknowledgement

The research was done as a part of the DIMACS/DIMATIA Research Experience for Undergraduates programme in Summer 2001. This is a joint programme of DIMACS at Rutgers University and DIMATIA at Charles University. Our REU supervisor was Jeff Kahn from Rutgers University. The authors would like to thank Jan Kratochvíl and Gerhard Woeginger for pointing out the reference [12].

References

1. C. Colbourn, J. Dinitz, A. Rosa: Bicoloring Triple Systems, Electronic J. Combin. 6# 1, 1999, paper 25, 16 pages.
2. M. R. Garey, D. S. Johnson: Computers and Intractability, A Guide to the Theory of NP–completeness, Freeman, San Franscisco, Cal., 1979.
3. L. Haddad: On the chromatic numbers of Steiner triple systems, J. Combinat. Designs 7, 1999, 1–10.
4. T. Jiang, D. Mubayi, Zs. Tuza, V. Voloshin and D. B. West: The Chromatic Spectrum of Mixed Hypergraphs Graphs Comb. 18 2, 2002, 309–318.
5. D. Král', J. Kratochvíl, A. Proskurowski, H.-J. Voss: Coloring mixed hypertrees, Proceedings 26th Workshop on Graph-Theoretic Concepts in Computer Science, LNCS vol. 1928, 2000, 279–289.
6. D. Král': On Complexity of Colouring Mixed Hypertrees, Proceedings 13th Symposium Fundamentals of Computation Theory, 1st Workshop on Efficient Algorithms, LNCS vol. 2138, 2001, 516–524.
7. D. Král', J. Kratochvíl, H.-J. Voss: Mixed Hypergraphs with Bounded Degree: Edge–Colouring of Mixed Multigraphs, to appear in Theoretical Computer Science.
8. C. C. Lindner, C. A. Roger: Decomposition into cycles II: Cycle Systems, Contemporary Design Theory: A Collection of Surveys, J. H. Dinitz and D. R. Stinson (eds.), John Wiley and Son, New York, 1992, 325–369.
9. L. Milazzo and Zs. Tuza: Upper chromatic number of Steiner triple and quadruple systems, Discrete Math. 174 (1997), 247–259.
10. L. Milazzo and Zs. Tuza: Strict colorings for classes of Steiner triple systems, Discrete Math. 182 (1998), 233–243.
11. G. Quattrocchi: Colouring 4-cycle systems with specified block colour patterns: the case of embedding P_3–designs, Electronic J. Combinatorics 8, 2001, #R24.
12. T. J. Schaefer: The complexity of satisfiability problems, Proc. of the Tenth Annual ACM Symposium on Theory of Computing (STOC), 1978, 216–226.

Safe Reduction Rules for Weighted Treewidth

Frank van den Eijkhof* and Hans L. Bodlaender**

Institute of Information and Computer Science, Utrecht University
P.O.Box 80.089, 3508TB Utrecht, The Netherlands

Abstract. Several sets of reductions rules are known for preprocessing a graph when computing its treewidth. In this paper, we give reduction rules for a weighted variant of treewidth, motivated by the analysis of algorithms for probabilistic networks. We present two general reduction rules that are safe for weighted treewidth, which generalise many of the existing reduction rules for treewidth. Experimental results show that these reduction rules can significantly reduce the problem size for several instances of real-life probabilistic networks.

1 Introduction

For many graph problems, it is useful and important to find a tree decomposition of minimal treewidth [3,9,11]. Often these problems can be solved in linear or polynomial time when a tree decomposition of bounded treewidth is known.

The problem of finding a tree decomposition with minimum treewidth is NP-hard [1], and it is also hard to approximate the treewidth [8]. Preprocessing techniques can help reducing the sizes of instances of these problems.

In [5], we give a set of reduction rules that can be used to preprocess a graph when computing its treewidth. Each of the rules reduces the number of vertices of the graph and a tree decomposition for the reduced graph with minimum treewidth can easily be extended to a tree decomposition for the original graph that also has minimum treewidth. We call such a rule safe. To allow more safe rules, we maintain a variable *low* that invariantly is a lower bound on the treewidth of the graph. We now say that a reduction rule is *safe*, if and only if the maximum of *low* and the treewidth of the graph is not changed. Preprocessing applies a series of safe reduction rules, taken from a set, on a graph until no more reduction rules can be applied. When the preprocessing results in an empty graph, we can trivially find a tree decomposition with minimum treewidth for the original graph. When the graph resulting after preprocessing is not empty, either the treewidth can be computed for the remaining graph by an exact, but possibly computational expensive algorithm (like integer linear programming),

* This author was partially supported by the Netherlands Computer Science Research Foundation with financial support from the Netherlands Organisation for Scientific Research.

** This author was partially supported by EC contract IST-1999-14186: Project ALCOM-FT (Algorithms and Complexity – Future Technologies).

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 176–185, 2002.

or the treewidth can be approximated using heuristic algorithms. Given the tree decomposition of the reduced graph, reversing the reductions now yields a tree decomposition of the original graph that has either an exact or an approximate treewidth.

Several modern decision support systems have *probabilistic networks* as underlying technology. These networks model dependences and independencies between statistical variables using a directed acyclic graph. For each statistical variable, represented by a vertex in the graph, a probabilistic function is defined. The most important problem to solve on these networks is *probabilistic inference*, which computes the probability distribution of a variable given a value-assignment to other variables. The most efficient algorithm currently used for probabilistic inference is based upon the use of a tree decomposition, since many probabilistic networks that model real-life situations appear to have small treewidth [7,11][1]. However, the statistical variables in a probabilistic network may have more than two values, and thus a tree decomposition with minimum treewidth may not be optimal for this algorithm. Instead of treewidth, we consider minimising the weighted treewidth, where the weighted treewidth of a tree decomposition is defined as the maximum of $\prod_{v \in X_i} w(v)$ over all bags X_i in the tree decomposition. The function $w(v)$ denotes the weight of vertex v; this is the finite number of values the variable associated with v can attain. The weighted treewidth expresses the maximum time needed to process a node of the tree decomposition.

Note that the treewidth of a graph is one less than the logarithm (with base 2) of the weighted treewidth when all vertices have weight 2 (binary variables).

In Section 3, we present the Simplicial rule, which is a rather straightforward generalisation of the non-weighted case. It removes a simplicial vertex, which is a vertex for which all neighbours form a clique, and updates a variable *low* representing the lower bound for the weighted treewidth of the original graph. We also provide a general rule that comprises many possible reduction rules. For this rule we try to generate a clique that separates some single vertices from the rest of the graph using *contraction* of edges. The contraction operation contracts an edge into a single vertex that is adjacent to the neighbours of the endpoints of the edge. When applying this operation, the safeness property requires that the weight of the resulting vertex is equal to the minimum of the weights of the endpoints.

In Section 4, we show that these rules generalise several known reduction rules, which can now be extended to work on weighted graphs, in particular the rules used for recognising graphs with for non-weighted graphs identified to use for recognising graphs with treewidth at most 1, 2, or 3 [2] and the almost simplicial rule from [5].

Section 5 presents several experiments conducted by applying a subset of our reduction rules on 23 real-life probabilistic networks. Our experiments reveal that for several networks a decomposition with minimal weighted treewidth can

[1] More precisely, the algorithms work on a tree decomposition of the so called moralised graph of the probabilistic network. We refer to e.g., [7,11] for details.

be found, while most remaining networks are reduced significantly. With Section 6 we conclude the paper.

2 Definitions and Preliminaries

In this paper, we assume that graphs are undirected, and simple, and have a weight function $w : V \to \mathbf{N}^+$. We use the notation $G = (V, E, w)$. We assume familiarity with standard graph notions, like independent set, clique, etc.

Let $G = (V, E, w)$ be a graph. The set of neighbours of a vertex v is denoted $N(v)$. The *degree* of a vertex is denoted as $deg(v) = |N(v)|$. A vertex $v \in V$ is *simplicial* when $N(v)$ is a clique. A *subgraph* $H(G)$ of G is a graph $H = (V', E', w[V'])$ with $V' \subseteq V$, $E' \subseteq (V' \times V') \cap V$, and $w[V'] : V' \to \mathbf{N}^+$ a function that assigns for every vertex $v \in V'$ the value $w(v)$ to $w[V'](v)$. For a set of vertices $W \subseteq V$, the *subgraph induced by* W is the subgraph $G[W] = (W, (W \times W) \cap E, w[W])$. For the sets of vertices $W, X \subset V$, W and X disjoint, the set X *separates* W when every path between a vertex in W and a vertex in $V \backslash (W \cup X)$ uses a vertex in X.

Let $G = (V, E, w)$ be a graph. The *neighbourhood weight* or $nw(v)$ of a vertex $v \in V$ is $nw(v) = w(v) \cdot \prod_{u \in N(v)} w(u)$. The weight of a set of vertices $S \subseteq V$ is $w(S) = \prod_{v \in S} w(v)$.

A *tree decomposition* of a graph $G = (V, E)$, or a weighted graph $G = (V, E, w)$, is a pair $(\{X_i | i \in I\}, T = (I, F))$ with T a tree and for every $i \in I$ a *bag* $X_i \subseteq V$, such that for each vertex $v \in V$ there exists a bag with $v \in X_i$, for each edge $(v, u) \in E$ there exists a bag with $v, u \in X_i$, and for each vertex $v \in V$ the induced graph $T[S_v]$, with $S_v = \{i \in I | v \in X_i\}$, is a tree.

The *weighted treewidth* of a tree decomposition $(\{X_i | i \in I\}, T = (I, F))$ of a weighted graph $G = (V, E, w)$ equals $\max_{i \in I} w(X_i)$; the *weighted treewidth* of a graph G, denoted $\tau_w(G)$, is the minimum weighted treewidth over all tree decompositions of G.

Let $G = (V, E, w)$ be a graph. A *contraction* of an edge $(v, u) \in E$ with $w(v) \leq w(u)$ makes v adjacent to $(N(v) \cup N(u)) \backslash \{v\}$, and removes vertex u and edge (v, u). Rephrased more intuitively, an edge is contracted to the endpoint that has the smallest weight.

A *minor* of a graph G is a graph G' that is obtained from G by a sequence of zero or more vertex removals, edge removals, and/or edge contractions. Using that edges are always contracted to the endpoint with the smaller weight, one can prove the following lemma similar to the unweighted case (e.g.: [4, Lemma 16]).

Lemma 1. *Let G' be a minor of G. Then $\tau_w(G') \leq \tau_w(G)$.*

3 General Reduction Rules

In this section, we define two reduction rules. The first rule deletes simplicial vertices, and the second rule is based upon sets of edges that can be contracted

such that they form cliques in the graph. This rule generalises some reduction rules known for non-weighted graphs.

When reversing a reduction, for each vertex deleted in the reduction a new bag is added containing this vertex and its neighbours. Because the neighbours form a clique in the reduced graph, the bag can be included in the tree decomposition. We must be able to guarantee that this bag will not increase the weighted treewidth to a value above the weighted treewidth of the original graph. Therefore we maintain a variable *low* that represent the largest lower bound we know for the weighted treewidth of the original graph. When a non-simplicial vertex is deleted, its neighbourhood weight must have a value equal or smaller than *low*. When deleting a simplicial vertex, the value of *low* can be increased to the neighbourhood weight of the vertex.

A reduction rule is called *safe* when application of the rule changes a graph G and its associated variable low_G, into G' and its associated variable $low_{G'}$, such that $\max(low_G, \tau_w(G)) = \max(low_{G'}, \tau_w(G'))$. Safeness of a reduction rule implies that when we start with *low* any value that is at most the weighted treewidth of the original graph, the rule does not increase the weighted treewidth of the graph above that of the original graph.

We will first introduce the Simplicial rule.

Definition 1. *Let G be a graph, and let v be a simplicial vertex in G with neighbourhood weight $nw(v) \geq 0$. The* Simplicial rule *increases low to $\max(low, nw(v))$, and removes vertex v from G.*

Theorem 1. *The Simplicial rule is safe.*

Proof. As G contains a clique of size $nw(v)$, we know that $\tau_w(G) \geq nw(v)$. Furthermore, because $G - v$ is a minor of G, we know that $\tau_w(G) \geq \tau_w(G - v)$. Therefore, $\tau_w(G) \geq \max(nw(v), \tau_w(G - v))$.

Now let T' be a tree decomposition for $G - v$ with weighted treewidth $k \leq \max(nw(v), \tau_w(G - v))$. We create a tree decomposition T by adding a bag X_i to T' as follows. Bag X_i consists of $N(v) \cup \{v\}$, and we connect X_i to a bag X_j in T' with $i \neq j$ and $N(v) \subseteq X_j$. Since $N(v)$ is a clique in G', X_j exists (Lemma 3.1 in [6]). The weighted treewidth of T now equals $\max(nw(v), k)$, and thus $\tau_w(G) \leq \max(nw(v), \tau_w(G - v))$, hence $\tau_w(G) = \max(nw(v), \tau_w(G - v))$. □

In Figure 1 an application of the Simplicial rule is illustrated. Solid lines represent edges, and the dotted lines connect to the remainder of the graph. The numbers represent the weights on the vertices. For this example, *low* will become at least 120.

The second reduction rule is based upon edge contraction.

Definition 2. *A* contraction-reduction *is a 4-tuple (G, X, Y, S) with $G = (V, E, w)$ a weighted graph, X, Y disjoint sets of vertices $\subset V$, X an independent set, Y not a clique, and S a set of edges, each with one endpoint in X and one endpoint in Y such that for each $(x, y) \in S$, $x \in X$, $y \in Y$: $w(x) \geq w(y)$, and such that contraction of all edges in S will turn Y into a clique.*

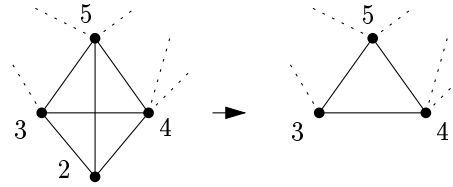


Fig. 1. An instance of the *Simplicial rule*.

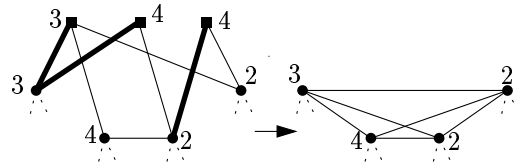


Fig. 2. A contraction-reduction

Note that for a contraction-reduction (G, X, Y, S) we know that $\tau_w(G) \geq w(Y)$, because Y is a clique in a minor of G. See Figure 2 for an example of a contraction reduction; the edges in S are drawn by fatter lines.

Definition 3. *Let $C = (G, X, Y, S)$ be a contraction-reduction. When $low \geq \max_{x \in X} nw(x)$, the* Contraction-reduction rule *denoted by C removes all vertices of X from G and turns Y into a clique.*

Theorem 2. *The Contraction-reduction rule is safe.*

Proof. Let (G, X, Y, S) be a Contraction-reduction rule, and let $low \geq 0$ be the value of the variable low during the application of the rule. Let G' be the minor of G obtained by contracting all edges in S. Then, we have that $\tau_w(G') \leq \tau_w(G)$. Combined with the precondition $low \geq \max_{x \in X} nw(x)$, we have that $\max(low, \tau_w(G)) \geq \max_{x \in X}(nw(x), \tau_w(G'))$.

Now let T' be a tree decomposition of G'. We obtain T from T' by adding a bag X_x to T' for each vertex $x \in X$ as follows. Bag X_x consists of $N(x) \cup \{x\}$, and we connect X_x to a bag X_j in T' with $x \neq j$ and $N(x) \subseteq X_j$. Since $N(x) \subseteq Y$ is a clique in G', X_j exists (see [6, Lemma 3.1]).

The weighted treewidth of T is at most the maximum of $\tau_w(T')$ and $\max_{x \in X} nw(x)$, which is $\tau_w(G) \leq \max_{x \in X}(nw(x), \tau_w(G'))$. Hence, $\tau_w(G) = \max_{x \in X}(nw(x), \tau_w(G'))$. So $\max(\tau_w(G), low) = \max(\tau_w(G'), low)$. □

Note that for a Contraction-reduction rule (G, X, Y, S), we also can increase low to $\max(w(Y), low)$, since Y is a clique in a minor of G. However, this observation is only useful when no vertex $x \in X$ exists with $N(x) = Y$, as otherwise this rule already requires that $low \geq \max_{x \in X} nw(x)$, which then equals $w(Y)$. Note that the contraction-reduction in Figure 2 is safe when $low \geq 72$.

4 Existing and New Rules as Instances of General Rules

Unfortunately, we can show that determining the existence of a contraction-reduction in a given graph is NP-complete. This proof will be given in the full version. For preprocessing the need for reduction rules with low computational cost for finding them is obvious, and therefore we search for easily identifiable instances of the Contraction-reduction rule. So far, several of those instances are known for non-weighted graphs: including the Simplicial rule, Arnborg and Proskurowski have identified the complete set of reduction rules for non-weighted graphs with treewidth at most three [2], i.e. the Single, Twig, Series, Triangle, Buddy and Cube rule. The Single and Twig rule are instances of the Simplicial rule, because they regard vertices with degree 0 and 1. Sanders [13] provided an extension to this set for non-weighted graphs with treewidth at most four. Only a subset of his reduction rules are instances of our contraction-reduction rule; many of his rules are not reversible easily. (See also [10].) Furthermore, we generalised the Series and Triangle rule to the Almost simplicial rule for non-weighted graphs [5].

We will now show that all mentioned rules, except for the Simplicial rule, can also be written as a Contraction-reduction rule, that is, extended to weighted graphs. Thus, we have established sufficient conditions for safeness on the weights of the vertices involved in the following rules. We define the Almost simplicial rule as follows.

Definition 4. *Let G be a graph with vertices v and u, where v is not simplicial and $N(v)\backslash\{u\}$ forms a clique. Let low $\geq nw(v)$ and $w(v) \geq w(u)$. The* Almost simplicial rule *is defined as the Contraction-reduction rule $(G, \{v\}, N(v), \{(v, u)\})$.*

Corollary 1. *The Almost simplicial rule is safe.*

Proof. Let $(G, \{v\}, N(v), (v, u))$ be an Almost simplicial rule with a lower bound for $\tau_w(G)$ as defined. When applying the Almost simplicial rule, contraction of (v, u) turns $N(v)$ into a clique. Because $N(v)$ separates v, Theorem 2 can be used. □

Figure 3 shows an instance of the Almost simplicial rule; it can be carried out when $low \geq 288$.

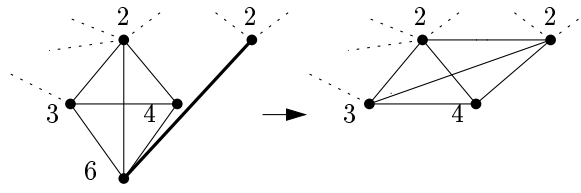


Fig. 3. An instance of the *Almost simplicial rule.*

The Series and Triangle rules are special cases of the Almost simplicial rule. When a vertex v has neighbours x and y, with x and y not adjacent, v can be removed and edge (x, y) can be added when the weight of v is larger than the weight of x or the weight of y, and when *low* is equal or larger than the multiplication of the weights of v, x, and y.

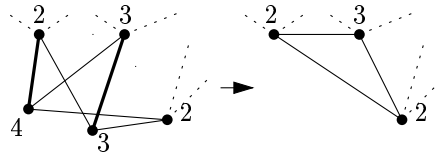


Fig. 4. An instance of the *Buddies rule*.

The Buddies rule is defined as follows.

Definition 5. *Let G be a graph with vertex sets $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_{n+1}\}$, $n \geq 0$, with $\forall_{1 \leq i \leq n} : N(x_i) = Y$. Let low $\geq \max_{1 \leq i \leq n}(nw(x_i))$, and let $\forall_{1 \leq i \leq n} : w(x_i) \geq w(y_i)$. The* Buddies rule *is defined as $(G, X, Y, \{(x_i, y_i) \mid 0 \leq i \leq n\})$.*

The Buddies rule can be proven safe using Theorem 2. The Buddies rule (G, X, Y, S) with $|X| = 2$ and $|Y| = 3$ is also known as the Buddy rule. The instance of the Buddies rule shown in Figure 4 can be applied when $low \geq 48$.

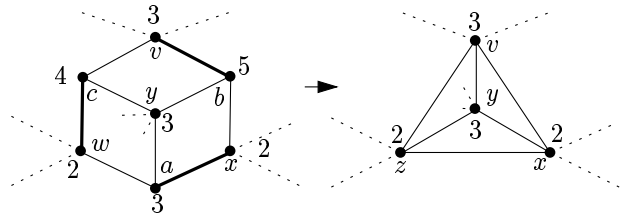


Fig. 5. An instance of the *Cube rule*.

The last reduction rule from Arnborg and Proskurowski [2] is the Cube rule.

Definition 6. *Let $G = (V, E, w)$ be a graph with vertex sets $Y = \{v, x, y, z\} \subseteq V$, $X = \{a, b, c\} \subseteq V$. Let $N(a) = \{z, y, x\}$, $N(b) = \{x, y, v\}$, $N(c) = \{v, y, z\}$. When low $\geq \max(nw(a), nw(b), nw(c))$ and $w(z) \leq w(a)$, $w(x) \leq w(b)$, and $w(v) \leq w(c)$, the* Cube rule *is defined as the Contraction-reduction rule $(G, X, Y, \{(z, a), (x, b), (v, c)\})$.*

The Cube rule can be proven safe with Theorem 2. We will not generalise the Cube rule, since our experiments on the unweighted case [5] indicate that this

reduction rule seldom occurs. In Figure 5 an instance of the Cube rule is given, which can be applied when $low \geq 90$.

All reduction rules mentioned are built upon existing rules. However, it is also possible to derive new reduction rules with help of Theorem 2. Consider Figure 6. If for $1 \leq i \leq 5$, we have $w(x_i) \geq w(y_i)$, then the subgraph of Figure 6 depicts a contraction-reduction; and it cannot be reduced using the Simplicial, Almost simplicial, Buddies, or Cube rule.

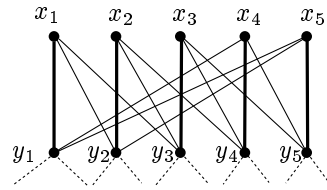


Fig. 6. A contraction-reduction when $w(x_i) \geq w(y_i)$ for $1 \leq i \leq 5$.

Testing for and applying of the Simplicial and Almost simplicial rules on a graph $G = (V, E, w)$ can be done in $O(|V|^2|E|)$ time, and of the Cube rule in $O(|V|)$ time [2,5,12]. For the Buddies rule (G, X, Y, S) the time bound is $O(|V|^2)$ when $|X|$ is bounded by some constant.

5 Experiments

In the previous sections we introduced several reduction rules for preprocessing a graph. We now report the results of some experiments with our reduction rules. Our experiments are conducted on 23 probabilistic networks developed for real-life problems. The Alarm, BOBLO, Diabetes, Link, Munin, Oesoca, PigNet2, Pigs, VSD, and Wilson networks are taken from medical applications; several versions exist of the Munin and Oesoca networks. The Barley and Mildew networks are used for agricultural purposes, the Water network models a water purification process and the OOW-trad, OOW-bas, OOW-solo, and Ship-ship networks are developed for maritime use.

Table 1 shows some results of our preprocessing technique. The preprocessing consists of two phases; for each phase a set of reduction rules is employed. In the first phase the set Prepro-1 = {Simplicial rule} is used, and for the second phase we applied the set Prepro-2 = Prepro-1 ∪ {Almost simplicial rule, Buddy rule}. For each instance the size of the graph is shown after moralisation, and after the first and second phase. After each phase we also provide the number of vertices deleted by each reduction rule in the set, and the value of variable *low*, representing the lower bound currently known for the weighted treewidth of the instance. Our table reveals for example, that the second phase reduces six graphs to the empty graph, meaning that for these instances we find a tree decomposition with minimum weighted treewidth. The Buddies and Cube rule

Table 1. Preprocessing results for several probabilistic networks

instance	moral		Prepro-1				Prepro-2					
	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$	#smpl	*low*	$\|V\|$	$\|E\|$	#smpl	#asmp	#bud	*low*
Alarm	37	65	11	19	26	32	0	0	32	5	0	32
Barley	48	126	35	92	13	40320	29	83	14	5	0	40320
BOBLO	221	328	71	132	150	687820	0	0	186	35	0	687820
Diabetes	413	819	335	665	78	7056	332	662	78	3	0	7056
Link	724	1738	494	1349	230	128	339	1194	230	155	0	128
Mildew	35	80	20	40	15	280000	15	31	15	5	0	280000
Munin1	189	366	108	241	81	600	90	220	82	17	0	600
Munin2	1003	1662	449	826	554	600	317	674	564	122	0	600
Munin3	1044	1745	419	790	625	600	172	443	652	208	12	2000
Munin4	1041	1843	436	920	605	600	271	724	614	156	0	2000
Munin-KGO	1066	1730	298	549	768	1280	95	252	803	168	0	2000
Oesoca+	67	208	30	141	37	1536	27	131	37	3	0	1536
Oesoca	39	67	5	7	34	240	0	0	38	1	0	240
Oesoca42	42	72	6	10	36	240	0	0	40	2	0	240
OOW-trad	33	72	27	59	6	900	25	57	6	2	0	900
OOW-bas	27	54	19	37	8	5400	17	35	8	2	0	5400
OOW-solo	40	87	31	68	9	5400	29	66	9	2	0	5400
PigNet2	3032	7264	1643	4556	1389	81	1051	3835	1418	563	0	81
Pigs	441	806	163	305	278	27	126	265	282	33	0	27
Ship-ship	50	114	39	92	11	600	38	91	11	1	0	600
VSD	38	62	12	21	26	240	0	0	32	6	0	240
Water	32	123	24	101	8	3072	22	96	8	2	0	3072
Wilson	21	27	6	8	15	36	0	0	19	2	0	108

are not yet implemented completely; addition of the Buddy rule only resulted in the deletion of 12 vertices from the Munin3 network. The lower bound value for Mildew is caused by a clique of three vertices with weights 100, 100, and 28; the weights in the BOBLO network have values between 2 and 595.

Our preprocessing method takes little time: the algorithm (implemented in C++) used at most a few seconds per network on a standard modern workstation.

6 Conclusions

Instances of NP-hard problems often can be reduced to equivalent but smaller instances using preprocessing techniques. For the problem of finding a tree decomposition for a weighted graph with minimal weighted treewidth, we provided such a technique. Our method consists of the application of a set of reduction rules that allows for reduction of a weighted graph without increasing its weighted treewidth, and for which every reduction is easily reversible. We showed that several of the known reduction rules can be generated from the generic Contraction-reduction rule, and that new feasible reduction rules can be created.

Experiments were conducted on the graphs of a set of probabilistic networks taken from real-life applications. These experiments revealed that a subset of

the identified reduction rules were able to reduces the graphs significantly, or even completely. Therefore, preprocessing by applying reduction rules is a very useful technique for the weighted treewidth problem.

Acknowledgements

We thank Linda van der Gaag for her comments and guidance for this paper; furthermore we thank Kristian Kristensen, Anders L. Madsen, Kristian G. Olesen, Claus Skaaning Jensen, and Linda van der Gaag for providing instances of probabilistic networks.

References

1. S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.
2. S. Arnborg and A. Proskurowski. Characterization and recognition of partial 3-trees. *SIAM J. Alg. Disc. Meth.*, 7:305–314, 1986.
3. S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Disc. Appl. Math.*, 23:11–24, 1989.
4. H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comp. Sc.*, 209:1–45, 1998.
5. H. L. Bodlaender, A. M. C. A. Koster, F. van den Eijkhof, and L. C. van der Gaag. Pre-processing for triangulation of probabilistic networks. In J. Breese and D. Koller, editors, *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, pages 32–39, San Francisco, 2001. Morgan Kaufmann.
6. H. L. Bodlaender and R. H. Möhring. The pathwidth and treewidth of cographs. *SIAM J. Disc. Math.*, 6:181–188, 1993.
7. F. V. Jensen. *Bayesian Networks and Decision Graphs*. Statistics for Engineering and Information Science, Springer-Verlag, New York, 2001.
8. T. Kloks. *Treewidth. Computations and Approximations*. Lecture Notes in Computer Science, Vol. 842. Springer-Verlag, Berlin, 1994.
9. A. M. C. A. Koster, S. P. M. van Hoesel, and A. W. J. Kolen. Solving frequency assignment problems via tree-decomposition. Technical Report RM/99/011, Faculty of Economics and Business Administration, Universiteit Maastricht, Maastricht, the Netherlands, 1999.
10. J. Lagergren. The nonexistence of reduction rules giving an embedding into a k-tree. *Disc. Appl. Math.*, 54:219–223, 1994.
11. S. J. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *The Journal of the Royal Statistical Society. Series B (Methodological)*, 50:157–224, 1988.
12. J. Matoušek and R. Thomas. Algorithms for finding tree-decompositions of graphs. *J. Algorithms*, 12:1–22, 1991.
13. D. P. Sanders. On linear recognition of tree-width at most four. *SIAM J. Disc. Math.*, 9(1):101–117, 1996.

Graph Separator Algorithms: A Refined Analysis

Henning Fernau

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

Universität Tübingen
Wilhelm-Schickard-Institut für Informatik
Sand 13, D-72076 Tübingen, Germany
fernau@informatik.uni-tuebingen.de

Abstract. This paper continues our research on the use of graph separator theorems for designing fixed parameter algorithms started with the COCOON'01 contribution [2], showing how a more elaborated use of these theorems can bring down the algorithmically relevant constants. More precisely, if a $c^{\sqrt{k}}$-algorithm is obtainable with the help of applying the well-known Lipton/Tarjan planar separator theorem, our new approach will lead to a $c^{2/3\sqrt{k}}$-algorithm, this way also improving on the direct use of the "best" known planar separator theorem. For several problems, the constants can be even improved more by analyzing other separator theorems.

1 Graph Separator Based Parameterized Algorithms

In [2], it was shown how to use classical graph separator theorems for developping recursive fixed parameter algorithms for NP-hard (planar) graph problems. Typically, these algorithms have running time $c^{\sqrt{k}}n^{O(1)}$ (hence, these are $c^{\sqrt{k}}$-*algorithms* for short) for a constant c, where k is the parameter of the problem. Special attention was paid to figure out the constant c depending on two constants α and β related to separator theorems and σ and d depending on the nature of the concrete problem. We explain these constants in more detail in the following two subsections.

The focus of the present paper is to bring down the algorithmically relevant constants. More precisely, if a $c^{\sqrt{k}}$-algorithm is obtainable with the help of applying the well-known Lipton/Tarjan planar separator theorem, our new approach will lead to a $c^{2/3\sqrt{k}}$-algorithm, this way also improving on the direct use of the "best" known planar separator theorem. For several problems, the constants can be even improved more by analyzing other separator theorems.

1.1 Separator Theorems

Let $G = (V, E)$ be an undirected graph. A *separator* $S \subseteq V$ of G divides V into two *parts* $A_1 \subseteq V$ and $A_2 \subseteq V$ such that $A_1 + S + A_2 = V$ and no edge joins

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 186–197, 2002.

vertices in A_1 and A_2. The triple (A_1, S, A_2) is also called a *separation* of G. According to Lipton and Tarjan [17], an *$f(\cdot)$-separator theorem (with constants $\alpha < 1$, $\beta > 0$)* for a class $\mathbb{G}$ of graphs which is closed under taking vertex-induced subgraphs is a theorem of the following form:

> If G is any n-vertex graph in $\mathbb{G}$, then there is a separation (A_1, S, A_2) of G such that neither A_1 nor A_2 contains more than αn vertices and S contains no more than $\beta f(n)$ vertices.

Clearly, these notions can be generalized to the case where a separator partitions the vertex set into ℓ subsets instead of only two.

The techniques we develop here all are based on the existence of "small" graph separators. Here, "small" means that $|S|$ is bounded by $o(|V|)$. Known results on planar separator theorems are summarized in Table 1.

Table 1. Summary of various $\sqrt{\cdot}$-separator theorems with their constants α and β. Here, $r(\alpha, \beta)$ denotes the ratio $r(\alpha, \beta) = \beta/(1 - \sqrt{\alpha})$, which is of central importance to the running time analysis of our algorithms, cf. Theorem 1. The last row contains the derived exponential base for VERTEX COVER.

bounds for β	$\alpha = \frac{2}{3}$	$r(\frac{2}{3}, \beta)$	$\alpha = \frac{1}{2}$	$r(\frac{1}{2}, \beta)$	$\alpha = \frac{3}{4}$	$r(\frac{3}{4}, \beta)$
upper	$2\sqrt{2}$ [17]	15.41	$7 + \frac{1}{\sqrt{3}}$ [20]	25.87	$\sqrt{\frac{2\pi}{\sqrt{3}}} \cdot \frac{1+\sqrt{3}}{\sqrt{8}}$ [19]	13.73
	$\sqrt{6}$ [13]	13.35	$\sqrt{24}$ [6,8]	16.73		
	$\sqrt{4.5}$ [7]	11.56				
	$\sqrt{\frac{2}{3}} + \sqrt{\frac{4}{3}}$ [12]	10.74				
lower	1.55 [13]	8.45	1.65 [19]	5.63	1.42 [19]	10.60
best c for VC	$2^{15.19} \approx 37381$		$2^{23.66} \approx 13254694$		$2^{19.42} \approx 701459$	

We will call a separator S of a planar graph G *cycle separator* if there exists a triangulation $\hat{G}$ of G such that S forms a simple cycle in $\hat{G}$. In fact, the current "record holder" in the case of $\alpha = 2/3$ yields a cycle separator, see [12].

Similar separator theorems for planar weighted graphs exist (sometimes with slightly worse constants), which can be used for designing parameterized algorithms, as well.

As can be seen by looking closer at the column for $\alpha = 2/3$, a certain race has been going on to bring down $r(\alpha, \beta)$; this indeed influences directly the performance of straightforward algorithmic applications of the mentioned separator theorems, as shown in the third main row of Table 1. We will argue in this paper that alternative ways of making use of separator theorems exist which yield better constants, even when starting off from the conceptually simplest separator theorem, which is the one due to Lipton and Tarjan. So, from an algorithmic perspective, it might be worthwhile starting this sort of race again, improving the exponential bases by more sophisticated applications of (other) separator theorems.

Finally, we think that also the lower bound argument(s) yielding the figures indicated in Table 1 applies to our approach, so that we do not see any real barrier for improving separator-based algorithms. Maybe, the folklore statement that separator-based algorithms are doomed to be impractical (which, according to computer experiments, need not be the case anyways) can be even falsified this way.

1.2 Parameterized Graph Algorithms

Let $\mathcal{L}$ be a parameterized problem, i.e., $\mathcal{L}$ is a subset of $\Sigma^* \times \mathbb{N}$. *Reduction to problem kernel*, then, means to replace instance $(I,k) \in \Sigma^* \times \mathbb{N}$ by a "reduced" instance $(I',k') \in \Sigma^* \times \mathbb{N}$ (which we call *problem kernel*) such that $k' \leq c \cdot k, |I'| \leq p(k)$ with constant c, some function p only depending on k giving the size of the problem kernel, and $(I,k) \in \mathcal{L}$ iff $(I',k') \in \mathcal{L}$. Furthermore, we require that the reduction from (I,k) to (I',k') is computable in polynomial time $T_K(|I|,k)$. Often, the best one can hope for is that the problem kernel is size linear in k, a so-called *linear problem kernel*, i.e., $|I'| \leq d \cdot k$ for some constant d.

A parameterized problem is called *fixed parameter tractable* iff it allows for a solving algorithm running in time $f(k)|I|^{O(1)}$ on input instance (I,k)—where f is an arbitrary function only depending on k—iff it admits a problem kernel. For more details, we refer to the monograph [16].

In [2], the notion of *(slim) glueable select&verify* problems was coined in order to pin down the class of graph problems suitable for a divide and conquer approach. We refer to the mentioned paper for the precise, very technical definition of these notions. For our purposes, it is sufficient to know that, for solving certain problems recursively, σ *colours* are needed to implement the book-keeping handing down information to the sub-problems and receiving partial solutions back.

Let us clarify these notions a bit by discussing one example: In the case of VERTEX COVER VC, applying a separator theorem (recursively) means: (i) find a separation (A_1, S, A_2) of the "current" graph G [1]; (ii) discuss all "colourings" of the vertices of S with 0 (meaning that the vertex in question is considered *not* to belong to the vertex cover) or with 1 (i.e.., the vertex in question is considered to belong to the vertex cover), hence, $2^{|S|}$ many cases have to be treated separately; (iii) in each case, we can modify the graph $G[A_1]$ (and similarly $G[A_2]$) induced by A_1 by erasing all vertices which are neighbouring a vertex in the separator coloured with 0, since the connecting edges must be covered by putting these vertices within the vertex cover to be constructed; this way, we obtain modified graphs G_1 and G_2; (iv) recursively, we have to solve VC on G_1 and G_2; (v) after returning from the recursive descent, we must "combine" the partial VC vertices and select some optimal VC solution (among the possibly $2^{|S|}$ many ones) which will be handed "upwards", either giving a solution to the original problem or testifying that such a solution does not exist (when the upper bound k is surpassed) or forming part of a VC solution of a bigger graph.

[1] For a recent survey on fast separator-finding algorithms, we refer to [5].

In fact, the reasoning just described shows that VERTEX COVER is a select&verify graph problem which is glueable with two colours (referring to the fact that two different colours are enough to "pre-set" the vertices in the separator and to construct ("glue together") a valid vertex cover from partial solutions obtained recursively). Moreover, we have shown that VERTEX COVER is a *slim* problem, since in the recursive step (iii), there is no need to "hand down" graphs which are "much larger" than $G[A_i]$ (in our example, they tend to be even smaller than this by the mentioned graph modifications).

In a similar but more complicated fashion, it is possible to describe a recursive algorithm for DOMINATING SET on planar graphs, this time needing three colours: 1 means that the such-coloured vertex was selected to be part of the dominating set to be constructed, 0_i, $i = 1, 2$, means that the such-coloured vertex will *not* be part of the dominating set to be constructed and it must be dominated by a vertex from A_i. Details are left to the reader (or can be found in [3]). According to the algorithms that we know, DOMINATING SET is *not* a slim problem: somehow, at least the vertices of the separator which are meant to be dominated by vertices of A_i need to be handed down in order to verify the validity of a partial dominating set, which means that in the worst case all separator vertices are considered in the recursive step, formalized by $G[A_i \cup S]$.

Table 2. The results listed in the table are contained in [1,2,9].

Problem	planar VC	planar IS	planar DS
σ	2	2	4
d	2	4	> 300

Table 2 shows the relevant two problem-specific parameters σ and d for three important problems on planar graphs: VERTEX COVER VC, PLANAR INDEPENDENT SET IS, and PLANAR DOMINATING SET DS. All these problems were proven to be glueable select&verify problems [2].

As the main theorem in [2], the following theorem was derived:

Theorem 1. *Suppose that $\mathcal{G}$ admits a problem kernel of polynomial size $p(k)$ on $\mathbb{G}$ computable in time $T_K(n,k)$. Then, there is an algorithm to decide $(G,k) \in \mathcal{G}$, for a graph $G \in \mathbb{G}$, in time*

$$c(\alpha', \beta, \sigma)^{\sqrt{p(k)}} q(k) + T_K(n,k), \quad \textit{where} \quad c(\alpha', \beta, \sigma) = \sigma^{\beta/(1-\sqrt{\alpha'})}, \qquad (1)$$

and $\alpha' = \alpha + \epsilon$ for any $\epsilon \in (0, 1-\alpha)$, holding only for $n \geq n_0(\epsilon)$, where $q(\cdot)$ is some polynomial. If, however, $\mathcal{G}$ is slim or the $\sqrt{\cdot}$-separator theorem yields cycle separators, then the running time for the computation is $c(\alpha, \beta, \sigma)^{\sqrt{p(k)}} q(k) + T_K(n,k)$, which then holds for all k. □

In particular, this means that for glueable select&verify problems for planar graphs that admit a *linear* problem kernel of size dk, we get an algorithm of running time $O\big(c(\alpha', \beta, \sigma, d)^{\sqrt{k}} q(k) + T_K(n,k)\big)$, where $c(\alpha', \beta, \sigma, d) = \sigma^{\beta\sqrt{d}/(1-\sqrt{\alpha'})}$.

Since VERTEX COVER is a slim problem, Theorem 1 yields the concrete constants listed in Table 1. Theorem 1 also gives a $c^{\sqrt{gk}}$-algorithm for the class $\mathbb{G}_g$ of graphs of genus bounded by g, based on a separator theorem derived by Djidjev [15] and a $c^{\sqrt{\ell k}}$-algorithm for the class of ℓ-map graphs, see [10,11]. Interestingly, both g and ℓ is (up to an additive constant) also the size of the largest clique found in graphs belonging to these graph classes, which is intuitively satisfying, since cliques make graphs hard to separate.

Let us mention that Theorem 1 is also applicable in a non-parameterized setting by taking $n = dk$. This way, $c^{\sqrt{n}}$-algorithms can be obtained for many graph problems. The improvements we derive in the next sections also apply to this non-parameterized setting.

2 A Brief Sketch of Separator Theorem Proofs

As pointed out by Venkatesan [20], a small separator, as obtained by Lipton and Tarjan's approach, consists of two ingredients:

- the first ingredient of the separator is composed of all vertices which have the same distance from the root of an assumed minimal height spanning tree (modulo some constant s), and
- the second ingredient is found according to a "special separator theorem" for planar graphs with radius of at most s.

The constant s is chosen such that the overall size of the separator is minimized.

Let us discuss the first ingredient in more details. Consider an n-vertex planar graph $G = (V, E)$, and let a spanning tree T of minimal height of G be given which has root r. A *level* i of a vertex v of G specifies the distance of v from r in the tree T. Choose some integer s "suitably" (this will become clearer later). Let $L_{T,j}$ denote the vertices of G which are at those levels i in T such that $i \bmod s = j$. In particular, $r \in L_{T,0}$. If s does not exceed the number of levels in T, then there exists a level L_{T,i_0} such that

$$|L_{T,i_0}| \leq \lfloor n/s \rfloor. \tag{2}$$

As seen later, the case when s exceeds the number of levels of T is an easy one.

The vertex set L_{T,i_0} will be the first ingredient of the separator we are going to construct. Observe that L_{T,i_0} alone is not a separator of G in general, since there may be edges not belonging to the spanning tree T which connect vertices of the remaining levels. Nevertheless, the graph $G - L_{T,i_0}$ is small when we look at its radius. Therefore, we will also call the procedure of cutting out L_{T,i_0} from G a *folding step*. More precisely, Venkatesan [20, Theorem 1] proved:

Lemma 1. *Assume that s does not exceed the radius h of G. Let T be a spanning tree of height h. For any level $L_{T,j}$ of G, the components of $G - L_{T,j}$ form a subgraph of another planar graph G' which contains $n - |L_{T,j}| + 1$ vertices and which has radius no more than s.* □

The second ingredient of the separator we are going to construct is obtained via a separator theorem for planar graphs of bounded radius, applied to G' form Lemma 1; of course, the obtained small separator will also separate $G - L_{T,j}$. Let us quote the needed "special separator theorem" in the form of a lemma which is due to Lipton and Tarjan [17]. Observe that weights are given to vertices of a graph by means of a function ω which assigns non-negative reals to vertices and, hence, to vertex sets.

Lemma 2. *Let G be a planar graph of radius s, with nonnegative weights on its vertices adding in total to no more than 1. Then, a separation (A_1, S, A_2) of G can be found in linear time such that neither A_1 nor A_2 has weight exceeding $2/3$ and $|S| \leq 2s + 1$. The vertices of S lie on a cycle if G is triangulated.* □

We remark that a statement similar to Lemma 2 is also valid for graphs of bounded genus, compare [14,15], so that our approach can be easily generalized to these graph classes $\mathbb{G}_g$. In addition, if it can be shown that there is a function $\bar{h}$ such that $\bar{h}(k)$ is an upper bound on the radius of any graph instance (G, k) with $G \in \mathbb{G}_g$ for a select&verify graph problem, then this problem (restricted to graphs from $\mathbb{G}_g$) is fixed-parameter tractable.

It is now possible to derive the classical Lipton/Tarjan result from the above considerations:

- If the radius of a given n-vertex planar graph G is less than or equal to $\sqrt{2n}$, then we might apply Lemma 2 directly.
- Otherwise, choose $s = \sqrt{n/2}$. The first ingredient of the separator, delivered by the folding step, then contains at most $n/s = \sqrt{2n}$ vertices according to Equation (2) and the second ingredient less than $2s + 1 = \sqrt{2n} + 1$ vertices according to Lemma 2 (which is applicable due to Lemma 1), summing up to at most $2\sqrt{2n} + 1$ vertices within the separator.

Here, the "magic number" s was determined as to minimize the sum

$$n/s + 2s \tag{3}$$

of the two ingredients of the separator.

3 Optimizing Divide and Conquer Algorithms

We are now going to improve the constants involved in the proof of Theorems 1 considerably by tailoring the use of separator theorems for the divide and conquer approach.

Our main two observations leading to the improvements explained in the next theorem are the following ones:

1. It is unnecessary to perform the folding step over and over again in the recursion, since the graph parts which occur after having applied a classical separator theorem will have small radius. Instead, one can apply Lemma 2

directly several times in a row in the course of the recursion. We cannot expect good results if we only apply Lemma 2, since the radius (as condition of applying Lemma 2) would not decrease in general, which means that the separators we get after several recursive applications of Lemma 2 will be rather large. Therefore, from time to time, a folding step has to be inserted. The goal is now to find the optimal number of recursive applications of Lemma 2 in a row.

2. It is not necessary to minimize a single separator occurring in the recursion, but it might be more reasonable to minimize the sum of all separators which occur on an arbitrary recursion path, since the running time of a divide and conquer algorithm based on separators in graphs basically depends only on the sum of the sizes of the separators accumulated along a fixed recursion path, cf. the discussions preceding Equation (3). Thus, it can be advantageous to have larger separators at certain levels of the recursion tree if this buys us smaller separators at the other levels.

In the following, we are specializing the constants α and β as $\alpha = 2/3$ and $\beta = \sqrt{8}$, because the simple proof structure of the planar separator theorem of Lipton and Tarjan (as sketched in Section 2) allows for easy modifications. It remains a challenging future research topic to see which ideas from other separator theorems could be used in order to obtain better constants in the next theorem. In fact, the results of Section 4 indicate that other separator theorems can be preferable.

Theorem 2. *Let $\mathcal{G}$ be a select&verify problem on planar graphs which is glueable with σ colours, and suppose that $\mathcal{G}$ admits a problem kernel of polynomial size $p(k)$ computable in time $T_K(n,k)$.*

Then, there is an algorithm to decide $(G,k) \in \mathcal{G}$, for an n-vertex planar graph G, in time

$$c(\alpha',\sigma)^{\sqrt{p(k)}}q(k) + T_K(n,k), \quad \text{where} \quad c(\alpha',\sigma) \approx \sigma^{1.80665/(1-\sqrt{\alpha'})},$$

and $\alpha' = 2/3 + \varepsilon$ for any $\varepsilon \in (0,1/3)$, holding only for $k \geq k_0(\varepsilon)$, where $q(\cdot)$ is some polynomial.

If $\mathcal{G}$ is slim, then the running time for the computation is

$$c(2/3,\sigma)^{\sqrt{p(k)}}q(k) + T_K(n,k),$$

which then holds for all k.

Proof. After kernelization, we are left with a graph with at most $p(k)$ vertices. For this graph G', we would like to answer the question whether $(G',k') \in \mathcal{G}$ for some k' linearly depending on k. We will do this in a recursive manner.

In the first step of the recursion, we will basically use the classical separator theorem of Lipton and Tarjan. Recall the outline of its proof in Section 2. Especially, note that the obtained separator consists of two ingredients. We will discuss these ingredients once more in the following.

Analogously to the considerations preceding Lemma 1, let L_{T,i_0} be a level in the vertex set induced by some spanning tree T of minimal height in G' such that $|L_{T,i_0}| \leq \lfloor p(k)/s \rfloor$, where s is an integer we are going to optimize in the following. We use L_{T,i_0} in a folding step. According to Lemma 1, we are now left with a planar graph with radius upperbounded by s. Therefore, we can now recurse the next ℓ steps in a row using only Lemma 2. Then, we will interleave another folding step in order to decrease the radius of the remaining graph parts, which are again handled by ℓ applications of Lemma 2 in a row, and so forth.

Within this basic scheme of recursion, let us consider a fixed recursion path. The time spent in ℓ recursion steps using only Lemma 2 basically is $\sigma^{(2s)\ell}$, since the accumulated size of the separators along the recursion path is upperbounded by $2s\ell$. Namely, in each of these ℓ recursion steps, a new separator of size $2s$ has to be considered. Therefore, the size size_ℓ of all separators along the recursion path after one folding step and ℓ applications of Lemma 2 is upperbounded by:

$$\text{size}_\ell \leq |L_{T,i_0}| + 2s\ell \leq \lfloor p(k)/s \rfloor + 2s\ell. \tag{4}$$

In order to minimize size_ℓ (which directly influences the running time of the recursive algorithm, as noted in the discussion preceding this theorem), we should choose $s = s_\ell := \sqrt{p(k)/(2\ell)}$. This means that we obtain an optimal total separator size of

$$\text{size}_\ell \leq \sqrt{8\ell p(k)}. \tag{5}$$

Let us now compare this new mixed strategy with a direct application of known separator theorems for obtaining divide and conquer algorithms. Firstly, observe that, after ℓ recursion steps according to our new mixed strategy, we are left with a graph whose components contain at most $(2/3)^\ell n$ vertices, if we started with an n-vertex graph. This situation is completely the same if we would have repeatedly applied the currently best separator theorem known for planar graphs, see Table 1. Consider a separator theorem with general constants α and β. After ℓ iterations of this separator theorem, along a fixed recursion path, we would have accumulated separators of total size upperbounded by

$$\beta \cdot \sqrt{n} + \beta \cdot \sqrt{\alpha n} + \cdots + \beta \cdot \sqrt{\alpha^{\ell-1} n},$$

if we started with an n-vertex graph. This geometric sum can be rewritten, yielding the general upper bound

$$\beta \cdot \frac{1 - \alpha^{\ell/2}}{1 - \sqrt{\alpha}} \cdot \sqrt{n} \tag{6}$$

Let us compare our approach therefore with Djidjev's separator theorem with constants $\beta = \sqrt{2/3} + \sqrt{4/3} \approx 1.97$ and $\alpha = 2/3$, see Table 1. After ℓ iterations of this separator theorem, along a fixed recursion path, we would have accumulated separators of total size $\overline{\text{size}}_\ell$, where

$$\overline{\text{size}}_\ell \leq (\sqrt{2/3} + \sqrt{4/3}) \cdot \frac{1 - (2/3)^{\ell/2}}{1 - \sqrt{2/3}} \cdot \sqrt{p(k)} \tag{7}$$

according to Equation (6). Comparing this approach with the previously outlined new mixed strategy can now be done by comparing size_ℓ and $\overline{\text{size}}_\ell$. Simple arithmetics shows that $\text{size}_\ell \leq \overline{\text{size}}_\ell$ iff $\ell \in \{1, \ldots, 12\}$. Therefore, our mixed strategy is better than the direct approach for $\ell \in \{1, \ldots, 12\}$.

Which $\ell \in \{1, \ldots, 12\}$ is the best choice within our mixed strategy? We answer this question by the following approach. Equating the right-hand side of Equation (5) and Expression (6) (in the case when $\alpha = 2/3$ and $n = p(k)$) allows us to compute a β_ℓ of a hypothetic separator theorem whose direct use for divide and conquer matches our mixed strategy. In this way, we obtain the following formula for β_ℓ:

$$\beta_\ell = \sqrt{8\ell} \cdot \frac{1 - \sqrt{2/3}}{1 - (2/3)^{\ell/2}}.$$

Some computations show that β_6 is the minimum of all such β_ℓ for $\ell \in \{1, \ldots, 12\}$. More precisely, we have $\beta_6 = 1.80665$ as required in the theorem. □

Example 1. For Vertex Cover on planar graphs, we obtain by the previous theorem a $2^{13.9234\sqrt{k}} \approx 15537^{\sqrt{k}}$-algorithm, which obviously beats the figures in Table 1.

To further improve our constants, we need a generalized notion of separation as introduced in the following section.

4 Regular Partitions

Djidjev [13] coined the notion of *regular γ-partition*. We will only need the following restricted notion (which Djidjev would term regular 1/2-partition):

Definition 1. *A* regular partition (A_1, A_2, A_3, S) *of an n-vertex graph $G = (V, E)$ is a partition $V = A_1 + A_2 + A_3 + S$ such that*

- *A_i and A_j are not connected to each other for $1 \leq i < j \leq 3$ and*
- *$|A_i| \leq 1/2 \cdot n$ for $i = 1, 2, 3$.*

Hence, in a regular partition, S can be viewed as a separator that splits G into *three* parts A_1, A_2, and A_3. Djidjev has shown the following analogue of Lemma 2:

Lemma 3. *Let G be a planar graph of radius s. Then, a regular partition (A_1, A_2, A_3, S) exists such that $|S| \leq 3s + 1$.* □

With the help of Lemma 3, Venkatesan [20] proved:

Lemma 4. *Let G be a planar n-vertex graph. Then, there exists a regular partition (A_1, A_2, A_3, S) such that $|S| \leq \sqrt{12}\sqrt{n}$.* □

In principle, the proposition was shown in the same way as the separator theorem of Lipton and Tarjan. In the folding step, a level L_{T,i_0} of a spanning tree of G is chosen such that $|L_{T,i_0}| \leq n/s$; then, Lemmas 1 and 3 are used to get a regular partition with $|S| \leq n/s + 3s$; finally, s is chosen to minimize $n/s + 3s$ which yields the claimed constant. Obviously, we can try the same trick as in Theorem 2 to find an optimal ℓ so that Lemma 3 is used ℓ subsequent times in the recursion. Some computations lead to:

Theorem 3. *Let $\mathcal{G}$ be a select&verify problem on planar graphs which is 3-glueable with σ colours, and suppose that $\mathcal{G}$ admits a problem kernel of polynomial size $p(k)$ computable in time $T_K(n,k)$.*

Then, there is an algorithm to decide $(G,k) \in \mathcal{G}$, for an n-vertex planar graph G, in time

$$c(\alpha',\sigma)^{\sqrt{p(k)}}q(k) + T_K(n,k), \quad \textit{where} \quad c(\alpha',\sigma) \approx \sigma^{2.7056/(1-\sqrt{\alpha'})},$$

and $\alpha' = 1/2 + \varepsilon$ for any $\varepsilon \in (0,1/2)$, holding only for $k \geq k_0(\varepsilon)$, where $q(\cdot)$ is some polynomial.

If $\mathcal{G}$ is slim, then the running time for the computation is

$$c(1/2,\sigma)^{\sqrt{p(k)}}q(k) + T_K(n,k),$$

which then holds for all k. □

We need to explain one more term used in the formulation of the preceding theorem: we call a problem *x-glueable with σ_x colours* if we are referring to an algorithmic use of a separator theorem splitting graphs into x chunks where σ_x colours are needed in the recursion; so, our previous notion of glueability with σ colours would now become 2-glueability with σ colours.

Example 2. For VERTEX COVER on planar graphs, we obtain in this way a $2^{13.0639\sqrt{k}} \approx 8564^{\sqrt{k}}$-algorithm, which again beats the figure of Example 1, since VERTEX COVER is 3-glueable with 2 colours.

As to DOMINATING SET, four colours are now needed instead of three to specify which of the three graph chunks is made "responsible" for dominating certain not yet dominated vertices of the separator; so, we have 3-glueability with 4 colours versus 2-glueability with 3 colours. Hence, Theorem 3 is not always yielding better algorithms compared with Theorem 2.

5 Conclusions

In this abstract, we sketched how an interleaved use of different separator theorems could yield improved running time estimates for algorithms based on recursively bisecting graphs. Interestingly, the basic scheme of these algorithms is not affected at all. This scheme can be sketched as follows:
If the currently examined graph piece is not small enough, then

– Find a small separator.
– For all assignments of "colours" to the vertices of the separator, recurse.

Since this scheme is very general, in particular it not only applies to parameterized algorithms.

Therefore, it keeps being a challenging question to bring down the constants derived in Theorem 3 by an even more sophisticated analysis of the use of separator theorems. It would be also interesting to see such optimizing ideas applied to other graph classes as graphs of bounded genus or map graphs.

A somewhat different venue to use graph separation for the design of parameterized algorithms was taken in [4], relying even more on properties of planar graphs. It would be interesting to see whether both approaches can be combined in order to get algorithms with better provable worst-case upper bounds.

Finally, let us mention that it is also possible to devise algorithms based on n/ε separation theorems, see [20]. This way, we can prove the following result:

Theorem 4. *Let $\mathcal{G}$ be a select&verify problem on planar graphs. We make the following further assumptions:*

– *$\mathcal{G}$ is glueable with σ colours.*
– *$\mathcal{G}$ admits a problem kernel of linear size dk computable in time $T_K(n,k)$.*
– *There is an $O(\rho^n)$ algorithm for solving the (possibly precoloured) graph decision problem under consideration for a planar graph with n vertices.*

Then, there is an algorithm to decide $(G,k) \in \mathcal{G}$, for an n-vertex planar graph G, in time

$$O(dk \cdot 2^{\theta(\sigma,\rho,d)\cdot k^{2/3}} + T_K(n,k)), \quad \textit{where} \quad \theta(\sigma,\rho,d) = 2\log(\rho)\left(\sqrt{24}d\frac{\log(\sigma)}{\log(\rho)}\right)^{2/3}.$$

□

Example 3. In the case of VERTEX COVER, Robson's algorithm [18] gives $\rho \approx 1.21$, so that Theorem 4 yields a $c^{k^{2/3}}$-algorithm with

$$c = 2^{(2\cdot\log 1.21\cdot(2\sqrt{24}/\log 1.21)^{2/3})} \approx 2^{5.27} \approx 38.59.$$

Compare this running time with the best known $c^{k^{1/2}}$-algorithm ($c = 2^{4\sqrt{3}} \approx 121.79$ was shown in [4]).

Acknowledgements

We are grateful for fruitful and long discussions on this topic with Jochen Alber and with Rolf Niedermeier.

References

1. J. Alber, M. Fellows, and R. Niedermeier. Efficient data reduction for DOMINATING SET: a linear kernel for the planar case. In M. Penttonen and E. Meineche Schmidt, editors, *Algorithm Theory – SWAT 2002*, volume 2368 of *LNCS*, pages 150–159. Springer, 2002.
2. J. Alber, H. Fernau, and R. Niedermeier. Graph separators: a parameterized view. In J. Wang, editor, *Computing and Combinatorics, Proceedings COCOON 2001*, volume 2108 of *LNCS*, pages 318–327. Springer, 2001.
3. J. Alber, H. Fernau, and R. Niedermeier. Parameterized complexity: exponential speedup for planar graph problems. Technical Report TR01–023, ECCC Reports, Trier, March 2001.
4. J. Alber, H. Fernau, and R. Niedermeier. Parameterized complexity: exponential speedup for planar graph problems. In F. Orejas, P. G. Spirakis, and J. v. Leeuwen, editors, *International Colloquium on Automata, Languages and Programming ICALP 2001*, volume 2076 of *LNCS*, pages 261–272. Springer, 2001.
5. L. Aleksandrov, H. Djidjev, H. Guo, and A. Maheshwari. Partitioning planar graphs with costs and weights. In D. M. Mount and C. Stein, editors, *Algorithm Engineering and Experiments – ALENEX 2002*, volume 2409 of *LNCS*, pages 98–110. Springer, 2002.
6. L. G. Aleksandrov and H. N. Djidjev. Linear algorithms for partitioning embedded graphs of bounded genus. *SIAM J. Disc. Math.*, 9:129–150, 1996.
7. N. Alon, P. D. Seymour, and R. Thomas. Planar separators. *SIAM J. Disc. Math.*, 2:184–193, 1994.
8. H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45, 1998.
9. J. Chen, I. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. *Journal of Algorithms*, 41:280–301, 2001.
10. Z.-Z. Chen. Approximation algorithms for independent sets in map graphs. *Journal of Algorithms*, 41:20–40, 2001.
11. Z.-Z. Chen, M. Grigni, and C. H. Papadimitriou. Map graphs. *Journal of the ACM*, 49:127–138, 2002.
12. H. Djidjev and S. Venkatesan. Reduced constants for simple cycle graph separation. *Acta Informatica*, 34:231–243, 1997.
13. H. N. Djidjev. On the problem of partitioning planar graphs. *SIAM J. Algebraic Discrete Methods*, 3(2):229–240, 1982.
14. H. N. Djidjev. A linear algorithm for partitioning graphs of fixed genus. *Serdica*, 11:369–387, 1985.
15. H. N. Djidjev. A separator theorem for graphs of fixed genus. *Serdica*, 11:319–329, 1985.
16. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
17. R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36(2):177–189, 1979.
18. J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7(3):425–440, September 1986.
19. D. A. Spielman and S.-H. Teng. Disk packings and planar separators. In *SCG 96: 12th Annual ACM Symposium on Computational Geometry*, pages 349–358, 1996.
20. S. M. Venkatesan. Improved constants for some separator theorems. *Journal of Algorithms*, 8:572–578, 1987.

Generalized H-Coloring and H-Covering of Trees

Jiří Fiala[1,⋆], Pinar Heggernes[2], Petter Kristiansen[2], and Jan Arne Telle[2]

[1] DIMATIA and ITI⋆⋆, Charles University, 118 00 Praha 1, Czech Republic
fiala@kam.mff.cuni.cz
[2] Department of Informatics, University of Bergen, N-5020 Bergen, Norway
{pinar,petterk,telle}@ii.uib.no

Abstract. We study $H(p,q)$-colorings of graphs, for H a fixed simple graph and p,q natural numbers, a generalization of various other vertex partitioning concepts such as H-covering. An H-cover of a graph G is a local isomorphism between G and H, and the complexity of deciding if an input graph G has an H-cover is still open for many graphs H. In this paper we show that the complexity of $H(2p,q)$-COLORING is directly related to these open graph covering problems, and answer some of them by resolving the complexity of $H(p,q)$-COLORING for all acyclic graphs H and all values of p and q.

1 Introduction

Colorings of graphs is a well-studied subject. In some cases the 'colors' to be assigned are the vertices of a fixed graph H, and model a situation where certain pairs of colors are treated specially. For example, in the well-known H-COLORING problem we ask for an assignment of 'colors' to the vertices of an input graph G such that adjacent vertices of G obtain adjacent 'colors', defining a homomorphism between G and H. H-COLORING is known to be solvable in polynomial time for bipartite H, and $\mathcal{NP}$-complete otherwise [9]. In the H-COVER problem a vertex $v \in V(G)$ is assigned a 'color' $u \in V(H)$ of the same degree, in such a way that the set of 'colors' assigned to the neighbors of v is exactly the set of 'colors' adjacent to u, defining a local isomorphism between G and H. Graph coverings come from algebraic graph theory [3], and form a special case of covering spaces from algebraic topology [14], with applications in topological graph theory [8]. The first applications in computer science were to graph recognition by parallel networks of processors [2,6]. The question of the computational complexity of H-COVER was first posed in 1989 [4], a variety of results have been shown, see e.g. [10,11,12,7], but it is still unclear what characterizes the class of simple graphs H that lead to polynomial time H-COVER problems.

⋆ Research supported in part by NATO science fellowship, administered through Norwegian Research Council project no. 143192/410 during his postdoctoral fellowship at the University of Bergen, and by Czech research grant GAČR 201/99/0242.
⋆⋆ Supported by the Ministry of Education of the Czech Republic as project LN00A056.

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 198–210, 2002.

A wide generalization of both H-coloring and H-covering is given by the so-called $H(\sigma, \rho)$-colorings for subsets of natural numbers σ and ρ [13], and in this paper we focus on the case of $|\sigma| = |\rho| = 1$. For a fixed graph H and natural numbers p and q, the $H(p, q)$-COLORING problem studied in this paper asks if an input graph G has a mapping $f : V(G) \to V(H)$ where the neighbors of any $v \in V(G)$ are mapped to the closed neighborhood of $f(v)$, with exactly p neighbors mapped to $f(v)$, and exactly q neighbors mapped to each neighbor of $f(v)$. From the definition it is clear that $H(0, 1)$-coloring is equivalent to H-covering, but it is maybe more surprising that for any simple graph H and any $p \geq 0, q \geq 1$ there exists a multigraph M such that $H(2p, q)$-coloring is equivalent to M-covering. In fact, the first graph covering problem shown to be $\mathcal{NP}$-complete in [1] was equivalent to $P_2(2, 1)$-COLORING, P_2 an edge. Moreover, it is known that resolving the complexity of M-COVER for all multigraphs M (where the covering definition is somewhat more complicated) is necessary and sufficient for resolving the complexity of H-COVER for all simple graphs H [10]. Via this link, the results in this paper, on the complexity of $H(p, q)$-COLORING, contribute directly towards the partial solution of the open problem mentioned above. See Fig. 1 for an example of a tree T and values of p and q (for which $T(p, q)$-COLORING is $\mathcal{NP}$-complete), the corresponding simple graph H (for which H-COVER then is $\mathcal{NP}$-complete), and the linking multigraph M.

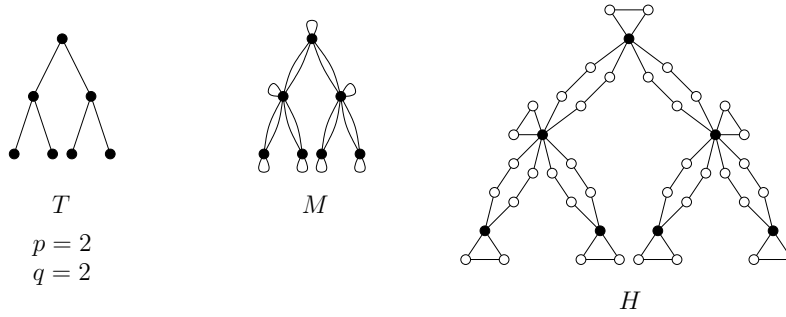


Fig. 1. A tree T (for which $T(2, 2)$-COLORING is $\mathcal{NP}$-complete), the corresponding simple graph H (for which H-COVER then is $\mathcal{NP}$-complete), and the linking multigraph M

The degree refinement matrix $\mathbf{M}_G$ of a graph G gives crucial information for these problems, and it is defined as follows: The *degree refinement* of a multigraph G is the partition of its vertices into the minimum number of blocks $\mathcal{B}_G = \{B_1(G), \ldots, B_t(G)\}$ for which there are constants m_{ij}, such that for all $1 \leq i, j \leq t$ each vertex in B_i has exactly m_{ij} neighbors in B_j. For a given, canonical, ordering of degree refinement blocks, the $t \times t$ matrix $\mathbf{M}_G$, $\mathbf{M}_G[i, j] = m_{ij}$, is called the *degree refinement matrix*.

We can now state our main theorem.

Theorem 1. *$T(p,q)$-COLORING, for a tree T and natural numbers p and q, is solvable in polynomial time if either:*

- *(trivial cases) all blocks [in the degree refinement of T] have size 1, or $q = 0$, or $p = 0, q = 1$,*
- *$p = 0$, $q = 2$ and either all blocks containing non-leaves have size 1, or they all have size 2,*
- *$p = 0$, $q \geq 3$ and all blocks have size 2,*
- *$p = 1$, $q = 1$ and all entries in [the degree refinement matrix] $\mathbf{M}_T$ are at most 2, or*
- *$p \geq 2$, $q = 1$, all entries in $\mathbf{M}_T$ are at most 2, and no block contains an induced edge,*

and is $\mathcal{NP}$-complete in all other cases.

The next section contains all formal definitions, and several important observations. The remainder of the paper is then devoted to the proof of Theorem 1, which has three different ingredients, split into three sections: Sect. 3 contains polynomial-time algorithms, based on finding factors in regular graphs and on reductions to 2SAT, Sect. 4 $\mathcal{NP}$-completeness reductions, and Sect. 5 various characterizations of trees, to show that all cases have been accounted for.

2 Definitions and Basic Observations

Let H be a fixed simple graph with k vertices $V(H) = \{u_1, u_2, \ldots, u_k\}$, and let p and q be natural numbers. An *$H(p,q)$-coloring* of a simple graph G is a partition $V_1, V_2, \ldots, V_k$ of $V(G)$, such that for all $1 \leq i, j \leq k$

$$\forall v \in V_i : |N_G(v) \cap V_j| = \begin{cases} p & \text{if } i = j \\ q & \text{if } u_i u_j \in E(H) \\ 0 & \text{otherwise} \end{cases},$$

where $N_G(v)$ denotes the open neighborhood of v in G. We will also view the partition as a vertex mapping $f : V(G) \to V(H)$, with $f(v) = v_i$ for $v \in V_i$. Note that this is equivalent to the definition given in Sect. 1.

In this paper we investigate the complexity of the following problem.

***H*(*p*, *q*)-COLORING**
INSTANCE: Simple graph G.
QUESTION: Does G have an $H(p,q)$-coloring?

For a simple graph H, $H(0,1)$-COLORING is precisely the same as the H-COVER problem. Moreover, any $H(2p,q)$-coloring will correspond to an M-covering for some multigraph M, as we now show. The usual definition of topological covering spaces requires that to cover a multigraph M by a graph

G we must specify, in addition to a vertex mapping $f : V(G) \to V(M)$, also an edge mapping $g : E(G) \to E(M)$. The edge map must respect the vertex map, and for every vertex $v \in V(G)$, and every edge e incident to $f(v)$, there must be exactly one edge incident to v that is mapped to e. Moreover, for every self-loop s on $f(v)$ there must be exactly two edges (or one self-loop) incident to v mapped to s.

For a simple graph H and natural numbers p and q, let H_q^p be the multigraph obtained from H by adding p self-loops to each vertex of H, and replacing each edge of H by q multiple edges.

Observation 1 *A simple graph G has an $H(2p, q)$-coloring if and only if it has an H_q^p-cover.*

Proof. The forward direction of the proof follows from [10] which shows that even if multigraph-covering requires an edge map, this edge map always exists if the vertex map f obeys the cardinality constraint that for every vertex $v \in V(G)$ the number of neighbors of v mapping to a vertex $u \in V(H)$ is the same as the number of multiple edges between u and $f(v)$. Likewise, the number of neighbors of v mapping to $f(v)$ should be twice the number of self-loops on $f(v)$. Clearly, an $H(2p, q)$-coloring of G does satisfy these cardinality constraints as imposed by H_q^p. The other direction of the proof is trivial. □

The degree refinement and degree refinement matrix of a graph are computed in polynomial time by stepwise refinement. Start with the vertices partitioned by their degrees and refine the partition as long as two vertices in the same block do not have the same number of neighbors in some other block. Note that the center of a tree, consisting of vertices whose greatest distance from any other vertex is as small as possible, contains either one or two vertices. The following fact is folklore.

Fact 1 *If G has an H-cover f, then $\mathbf{M}_G = \mathbf{M}_H$, and f is a one-to-one mapping of the degree refinement blocks.*

Theorem 2. *If G has an $H(p, q)$-coloring, and H is a simple, connected graph, then*

$$\mathbf{M}_G = q\mathbf{M}_H + p\mathbf{I} \ .$$

Proof. If $p = 2r$ we know from Observation 1 and Fact 1 that $\mathbf{M}_G = \mathbf{M}_{H_q^r}$, and first show that $\mathbf{M}_{H_q^r} = q\mathbf{M}_H + 2r\mathbf{I}$. When computing the degree refinement of H_q^r, self-loops are inconsequential, since all vertices $u \in V(H_q^r)$ have exactly r such loops. Likewise, the edge multiplicity is also inconsequential, since for all distinct, adjacent vertices $u, u' \in V(H_q^r)$, there are exactly q edges between u and u'. This implies that the degree refinements of H and H_q^r have the same block structure. Moreover, since every vertex in $V(H)$ gets r self-loops, and every edge in $E(H)$ is replaced by q multiple edges, we have $\mathbf{M}_{H_q^r} = q\mathbf{M}_H + 2r\mathbf{I}$. The argument for odd p differs only by some technical details. □

The following fact is now evident.

Fact 2 *If G has an $H(p,q)$-coloring f, and H is connected, then $|f^{-1}(u)| = |f^{-1}(u')|$ for all $u, u' \in V(H)$.*

The *simple (p,q)-cover of G* is the graph $(G \bowtie K_q) \,\square\, K_{p+1}$, where the graph products $\bowtie$ and $\square$ are defined as follows (in so called Nešetřil convention):

- $V(G \bowtie G') = V(G \,\square\, G') = V(G) \times V(G')$,
- $(v,v')(w,w') \in E(G \bowtie G')$ if and only if $vw \in E(G)$, and ($v'w' \in E(G')$ or $v' = w'$),
- $(v,v')(w,w') \in E(G \,\square\, G')$ if and only if ($vw \in E(G)$ and $v' = w'$), or ($v'w' \in E(G')$ and $v = w$).

For an example see the simple $(2,2)$-cover of P_3 depicted in Fig. 2, P_3 the path on three vertices.

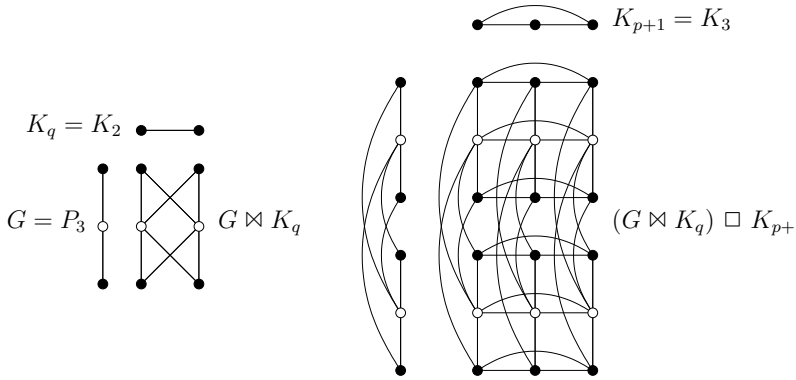


Fig. 2. Simple $(2,2)$-cover of P_3

Observe, that the operation $G \bowtie K_q$ replaces an edge of $vw \in E(G)$ with a complete bipartite graph $K_{q,q}$ on q copies of the vertices v and w. Similarly the operation $G \,\square\, K_{p+1}$ forms a clique K_{p+1} on $p+1$ copies of every vertex $v \in V(G)$, while the edges inside the $p+1$ copies of G are maintained. Blocks in the degree refinement of the simple (p,q)-cover correspond to the blocks in the degree refinement of the original graph, as indicated in Fig. 2 by the black and white vertex colors.

Lemma 1. *For every graph H and every $p \geq 0$, $q \geq 1$, the simple (p,q)-cover of H has an $H(p,q)$-coloring.*

Proof. The mapping $f : ((u,a),b) \to u$, where $u \in V(H)$, $a \in V(K_q)$, $b \in V(K_{p+1})$ is an $H(p,q)$-coloring. Every vertex has p neighbors mapped to the same target, those that differ only in the b-coordinate; and for every neighbor u' of u, $((u,a),b)$ is adjacent to $((u',a'),b)$, for every $1 \leq a' \leq q$. □

3 Polynomial Cases

Theorem 3. *If $\mathbf{M}_H = \mathbf{A}_H$, the adjacency matrix of H, i.e. if all blocks in the degree refinement of H are of size 1, then $H(p,q)$-COLORING is solvable in polynomial time.*

Proof. From Theorem 2 we know what the degree refinement matrix must be. Theorem 2 describes the necessary condition, and when $\mathbf{M}_H = \mathbf{A}_H$ this is also the sufficient condition. □

As an aside we mention that for a random graph H of the model $G(n,p)$, it is almost always the case that $\mathbf{M}_H = \mathbf{A}_H$ when $0 < p = p(n) \leq \frac{1}{2}$ is such that $p^5 n/(\log n)^5 \to \infty$ [5, ch. 3].

Lemma 2. *$T(0,q)$-COLORING, $q \geq 2$, is solvable in polynomial time for every tree T whose degree refinement blocks are all of size 2.*

Proof. We reduce to 2SAT. Let G be an instance of $T(0,q)$-COLORING, $q \geq 2$, where T is a tree whose degree refinement blocks are all of size 2. We construct a formula ϕ, such that ϕ has a satisfying truth assignment if and only if G has a $T(0,q)$-coloring. It will be obvious how to transform ϕ into a set V of variables and a collection C of two-literal clauses, to form a 2SAT instance (V,C).

First, compute the degree refinements of G and T. If they do not obey the constraints of Theorem 2, reject the input G. Otherwise, let $\mathcal{B}_G = \{B_1(G), \ldots, B_t(G)\}$ and $\mathcal{B}_T = \{B_1(T), \ldots, B_t(T)\}$ be the degree refinements of G and T, respectively. Let $B_i(T) = \{\text{left}_i, \text{right}_i\}$, and note that for each vertex $v \in B_i(G)$ we must decide whether it should map to left_i (variable v FALSE) or right_i (variable v TRUE). When all blocks in the degree refinement of T are of size 2, the center of T is an edge uu', and both u and u' belong to the same degree refinement block $B_1(T)$. $B_1(T)$ is the only block containing adjacent vertices. For every pair of adjacent vertices $v, w \in B_1(G)$ we insert the subformula $(v \not\Leftrightarrow w)$ into ϕ, and for every vertex $x \in B_i(G)$, with a neighbour $y \in B_j(G)$, $j \neq i$, we insert the the subformula $(x \Leftrightarrow y)$ into ϕ.

Let $f : V(G) \to V(T)$ be a $T(0,q)$-coloring of G, and label the left and right vertex of each block $B_i(T)$ with 0 and 1, respectively. Viewing the label of $f(v)$ as a truth assignment to variable v, we get a satisfying truth assignment τ for ϕ. In the other direction of the proof we first use the degree refinement of G to determine which block $B_i(T)$ in the degree refinement of T a vertex $v \in V(G)$ must map to. We then use a truth assignment τ for ϕ to determine if v should be mapped to the left or right vertex of $B_i(T)$. □

Let S_k denote the graph $K_{1,k}$, the star on $k+1$ vertices.

Lemma 3. *$S_k(0,2)$-COLORING is solvable in polynomial time for every k.*

Proof. Let G be an instance of $S_k(0,2)$-COLORING. By Theorem 2, the vertices of G must either be of degree 2, or $2k$. The vertices of degree $2k$ must map to the central vertex of S_k. The remaining vertices must map to the k leaves, in

such way that the neighborhoods of the vertices mapping to the center can be split into k disjoint pairs, where both vertices of a pair map to a unique leaf.

Contracting the vertices of degree 2 in a homeomorphic manner results in a $2k$-regular graph G'. By Petersen's theorem, G' can be split into k disjoint 2-factors. This split can be done in polynomial time, and we get an $S_k(0,2)$-coloring of G by mapping the vertices of degree 2 to the same leaf if and only if the corresponding edges belong to the same 2-factor. □

Note that in the case $k = 2$ the lemma also provides a polynomial time algorithm for $P_3(0,2)$-COLORING.

Lemma 4. *$T(0,2)$-COLORING is solvable in polynomial time for every tree T if either all blocks containing non-leaves have size 1, or they all have size 2.*

Proof. Note that the condition on T implies that we have either: (1) all degree refinement blocks of size 2 or more contain only leaves, or (2) all degree refinement blocks of size not equal to 2 contain only leaves.

Case 1: Let G be an instance of $T(0,2)$-COLORING, where T is a tree such that all blocks in the degree refinement of T of size 2 or more contain only leaves.

First, compute the degree refinements of G and T. If they do not obey the constraints of Theorem 2, reject the input G. Otherwise, let $\mathcal{B}_G = \{B_1(G), \ldots, B_t(G)\}$ and $\mathcal{B}_T = \{B_1(T), \ldots, B_t(T)\}$ be the degree refinements of G and T, respectively. For each block $B_i(G)$ corresponding to a block $B_i(T)$ of size 1, all vertices of $B_i(G)$ must map to the single vertex of $B_i(T)$. For each block $B_j(G)$ corresponding to a block $B_j(T)$ of size 2 or more, the vertices of $B_j(G)$ have neighbors in exactly one other block in the degree refinement of G; this block must correspond to a block of size 1 in the degree refinement of T. Thus, for each such $B_j(G)$, the problem can be solved independently, in polynomial time by Lemma 3, and the solutions combined into an overall $T(0,2)$-coloring of G.

Case 2: Let G be an instance of $T(0,2)$-COLORING, where T is a tree such that all blocks in the degree refinement of T of size not equal to 2 contain only leaves.

First, compute the degree refinements of G and T. If they do not obey the constraints of Theorem 2, reject the input G. Otherwise, let $\mathcal{B}_G = \{B_1(G), \ldots, B_s(G), B_{s+1}(G), \ldots, B_t(G)\}$ and $\mathcal{B}_T = \{B_1(T), \ldots, B_s(T), B_{s+1}(T), \ldots, B_t(T)\}$ be the degree refinements of G and T, respectively, with $B_1(T), \ldots, B_s(T)$ as the blocks of size 2, and $B_{s+1}(T), \ldots, B_t(T)$ as the remaining blocks. For the portion of G induced by the blocks $B_1(G), \ldots, B_s(G)$, the vertices can be mapped to the appropriate vertices of the portion of T induced by $B_1(T), \ldots, B_s(T)$ in polynomial time by Lemma 2. The blocks $B_{s+1}(G), \ldots, B_t(G)$ can be handled independently, in polynomial time by Lemma 3, and the solutions combined into an overall $T(0,2)$-coloring of G. □

Lemma 5. *$T(p,q)$-COLORING is solvable in polynomial time for every tree T if either:*

(1) *$p = 1$, $q = 1$ and all entries in $\mathbf{M}_T$ are at most 2, or*
(2) *$p \geq 2$, $q = 1$, all entries in $\mathbf{M}_T$ are at most 2, and no block in the degree refinement of T contains an induced edge.*

Proof. Both problems can be reduced to 2SAT. We only provide proof of case 1, as the proof of case 2 is similar.

Let G be an instance of $T(1,1)$-COLORING, where T is a tree such that all entries in $\mathbf{M}_T$ are at most 2. We construct a set V of variables and a formula ϕ, such that ϕ has a satisfying truth assignment if and only if G has a $T(1,1)$-coloring. It will be obvious how to transform ϕ into a collection C of two-literal clauses, to form a 2SAT instance (V, C).

First, compute the degree refinements of G and T. If they do not obey the constraints of Theorem 2, reject the input G. Otherwise, let $\mathcal{B}_G = \{B_1(G), \ldots, B_t(G)\}$ and $\mathcal{B}_T = \{B_1(T), \ldots, B_t(T)\}$ be the degree refinements of G and T, respectively. Let $B_1(T)$ be the block containing the center of T, and level the blocks of $\mathcal{B}_T$ according to their distance from $B_1(T)$, with $B_1(T)$ as level 1. $\mathcal{B}_G$ is given the same leveling. For each vertex $v \in B_i(G)$ we must decide which vertex of $B_i(T)$ it should map to. When all entries in the degree refinement matrix $\mathbf{M}_T$ are at most 2, a vertex $u \in B_i(T)$ can have at most 2 neighbours in $B_j(T)$, and $B_1(T)$ is the only block that can contain adjacent vertices. A block $B_i(T)$ on level l, $l > 1$, can therefore contain at most 2^l vertices. Every vertex v in a block on level l is represented by l variables $v_1, \ldots, v_l$. For every vertex $v \in B_1(G)$ we insert the subformula (v_1) into ϕ; for every vertex $w \in B_i(G)$ on level l, with two children $x, x' \in B_j(G)$ on level $l+1$, we insert subformulas $(x_1 \Leftrightarrow w_1), \ldots, (x_l \Leftrightarrow w_l), (x'_1 \Leftrightarrow w_1), \ldots, (x'_l \Leftrightarrow w_l), (x_{l+1} \not\Leftrightarrow x'_{l+1})$ into ϕ; and for every vertex $y \in B_i(G)$ on level l, with only one child $z \in B_j(G)$ on level $l+1$, we insert subformulas $(z_1 \Leftrightarrow y_1), \ldots, (z_l \Leftrightarrow y_l), (z_{l+1})$ into ϕ.

Let $f : V(G) \to V(T)$ be a $T(1,1)$-coloring of G, and label the vertices of T as follows: label vertices in the center of T with 1, for all other vertices concatenate the label of its parent with 0 if the vertex is the left child of its parent, in its block, and concatenate the label of its parent with 1 if it is the right, or only, child. For a vertex $v \in B_i(G)$ on level l, viewing the label of $f(v)$ as a truth assignment to the variables $v_1, \ldots, v_l$, digit by digit, we get a satisfying truth assignment τ for ϕ.

In the other direction of the proof we first use the degree refinement of G to determine which block $B_i(T)$ in the degree refinement of T a vertex $v \in V(G)$ must map to. If the center of T is an edge, T consists of two subtrees T_{left} and T_{right}, and we must decide whether a vertex v should map to the left or right subtree. In this case $B_1(G)$ induces a 2-regular graph, which is split by following the cycles, mapping two adjacent vertices to the left vertex of the center, two to the right, and so on. This split of $B_1(G)$ propagates down through the rest of G, and we map one part to T_{left}, the other to T_{right}. Finally, we use a truth assignment τ for ϕ and its restriction to variables $v_1, \ldots v_l$, to determine which

vertex $u \in B_i(T)$ a vertex $v \in B_i(G)$ on level l must map to, in the same manner as described above. □

4 $\mathcal{NP}$-Complete Cases

In this section all remaining $T(p,q)$-COLORING problems are shown to be $\mathcal{NP}$-complete. Due to lack of space we only give the full proof of Lemma 6. To resolve the complexity of the open H-COVER problems mentioned in the introduction, it will probably be necessary to generalize Lemma 6 to all graphs, hence its proof is of special interest.

Recall that the simple (p,q)-cover of H has $|V(H)| \cdot q(p+1)$ vertices. For simplicity we write $u_{(i-1)q+j}$ for the vertex $((u,a),b)$, when a is the i-th vertex of K_q, and b is the j-th vertex of K_{p+1}.

Lemma 6. *If T' is a tree isomorphic to a connected component of the subtree of T induced by some subset of degree refinement blocks $\mathcal{B}'_T \subseteq \mathcal{B}_T$, and the degree refinement of T' is identical to $\mathcal{B}'_T$ restricted to T', then $T'(p,q)$-COLORING reduces to $T(p,q)$-COLORING, for every p and q.*

Proof. We can assume $q \neq 0$, otherwise both $T'(p,q)$-COLORING and $T(p,q)$-COLORING are solvable in polynomial time, and the reduction follows trivially.

For every $u \in T'$ let T_u denote the component of $(V(T), E(T) \setminus E(T'))$ containing u, i.e., the part of T which hangs from u, but which does not belong to T'. Clearly trees T_u and $T_{u'}$ are isomorphic if u and u' belong to the same degree refinement block.

Let G' be an instance of $T'(p,q)$-COLORING. We may assume, that the blocks in the degree refinement of G', $\mathcal{B}_{G'} = \{B_1(G'), \ldots, B_t(G')\}$, correspond to the blocks $\mathcal{B}'_T$, otherwise G' has no $T'(p,q)$-coloring. We construct a graph G which will have a $T(p,q)$-coloring if and only if G' has a $T'(p,q)$-coloring.

For every vertex $v \in V(G')$ there is a gadget F_v. For $v \in B_i(G')$ F_v is constructed by taking the simple (p,q)-cover of T_u, for an arbitrary vertex $u \in B_i(T')$, and removing the edges connecting the $q(p+1)$ copies $u_1, \ldots, u_{q(p+1)}$ of vertex u. Note that gadgets F_v and $F_{v'}$ are isomorphic for vertices v and v' from the same degree refinement block. G is constructed by making $q(p+1)$ disjoint copies of G, where the a-th copy of a vertex $v \in V(G')$ is labeled v_a. For every vertex $v \in V(G)$ we insert the gadget F_v, and identify vertex v_a from the a-th copy of G and vertex u_a from F_v.

For an example of the construction of G see Fig. 3. The example shows a reduction from $P_2(2,1)$-COLORING, where P_2 appears as a block in T indicated by the white vertices. For a white vertex $u \in V(T)$, the tree T_u is depicted in the upper right corner together with the gadget F_v, the dotted edges connecting copies of u are removed. An instance G' and the constructed graph G are depicted in the lower part. The dashed edges are those that belong to the copies of G', while the solid edges belong to the gadgets F_v.

Let $\mathcal{B}_G = \{B_1(G), \ldots, B_t(G)\}$ be the degree refinement of G. We claim, that each $B_i(G)$ corresponds to the block $B_i(T) \in \mathcal{B}_T$. Every vertex $v \in B_i(G')$ is

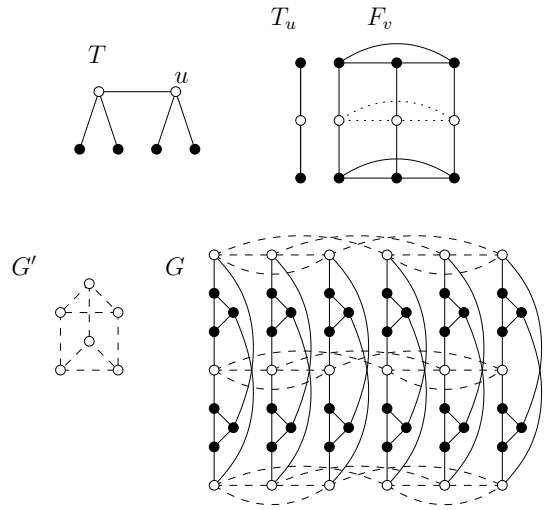


Fig. 3. Constructed graph G has a $T(2,1)$-coloring if and only if G' has $T'(2,1)$-coloring, for $T' = P_2$

connected to exactly p neighbors inside $B_i(G')$, in G; every copy of v_a is therefore connected to the same p neighbors inside the a-th copy of G'. Similarly, v_a has the correct number of neighbors in every other block $B_j(G)$. Take any $u \in B_i(T)$, for $B_j(G') \in \mathcal{B}_{G'}$, $|N(v_a) \cap B_j(G)| = |N(v) \cap B_j(G')| = q \cdot |N(u) \cap B_j(T)|$ holds inside each copy of G', and for $B_j(G) \notin \mathcal{B}_{G'}$ we get the same equality, $|N(v_a) \cap B_j(G)| = q \cdot |N(u) \cap B_j(T)|$, due to the construction of the simple (p,q)-cover F_v of T_u. The properties of F_v assure the same for its vertices.

If G has a $T(p,q)$-coloring f, its restriction to a single copy of G' is a $T'(p,q)$-coloring. Only vertices of T' (or its isomorphic copy in T) may appear as colors of G', because the blocks in the degree refinement of T and G are in one-to-one correspondence if any such $T(p,q)$-coloring exists. Conversely, any $T'(p,q)$-coloring f' of G' can be extended to a $T(p,q)$-coloring f of G; we use the same mapping on each copy of G', i.e. $f(v_a) = f'(v)$ for all $1 \leq a \leq q(p+1)$, and extend it to each F_v as described in Lemma 1. □

Lemma 7. *If no block in the degree refinement of H contains an edge, then $H(p,q)$-COLORING reduces to $H(p+1,q)$-COLORING, for every p and q.*

Lemma 8. *$P_2(p,q)$-COLORING is $\mathcal{NP}$-complete if $p \geq 1$, $q \geq 1$, except for the case $p = q = 1$.*

Lemma 9. *$P_3(p,q)$-COLORING is $\mathcal{NP}$-complete if either:*

(1) *$p = 0$, $q \geq 3$, or*
(2) *$p = 1$, $q = 2$.*

Lemma 10. *$S_k(0,q)$-COLORING is $\mathcal{NP}$-complete for $k \geq 2$ if $q \geq 3$.*

Lemma 11. *$S_k(p,q)$-COLORING is $\mathcal{NP}$-complete for $k \geq 3$ if either:*

(1) *$p = 1$, $q = 1$, or*
(2) *$p = 1$, $q = 2$.*

Lemma 12. *$T(0,2)$-COLORING is $\mathcal{NP}$-complete for every tree T whose degree refinement consists of three blocks, one of size 1 and the others of size at least 2.*

Lemma 13. *If T is a tree with one degree refinement block consisting of exactly one vertex u, and $2T$ is the tree made from two disjoint copies of T, T_1 and T_2, joined by the edge u_1u_2, then $T(p,q)$-COLORING reduces to $2T(p,q)$-COLORING, for every p and q.*

5 Completing the Proof

In the following we write $v \approx_G v'$ if vertices $v, v' \in V(G)$ belong to the same block in the degree refinement of G.

Lemma 14. *For any tree T with degree refinement $\mathcal{B}_T$, one of the following cases applies:*

(1) *All blocks of $\mathcal{B}_T$ contain only one vertex,*
(2) *there exists a block $B_i(T) \in \mathcal{B}_T$ of size 2, whose vertices induce an edge,*
(3) *there exists two adjacent blocks $B_i(T), B_j(T) \in \mathcal{B}_T$ whose vertices induce a disjoint union of stars $S_k, k \geq 3$, or*
(4) *for all pairs of adjacent blocks $B_i(T), B_j(T) \in \mathcal{B}_T$, their vertices either induce a perfect matching or a disjoint union of paths $P_3 = S_2$, and at least one pair inducing a P_3 exists in T.*

Proof. Assume T has at least two distinct vertices $u \approx_T u'$, otherwise the first case applies. Since T is connected, u and u' are connected by a path $P = v_1, \ldots, v_k$ with $u = v_1$ and $u' = v_k$ as its terminals. The fact $u \approx_T u'$ implies $v_i \approx_T v_{k+1-i}$. Therefore, if P is of odd length, its center $v_{\lfloor k/2 \rfloor}, v_{\lceil k/2 \rceil}$ induces an edge and the second case applies.

Otherwise vertices of two adjacent blocks always induce a disjoint union of stars $S_k, k \geq 1$. The last two cases distinguish whether an induced star S_k, $k \geq 3$ exists, or not. □

Proof of Theorem 1. Throughout this case study we assume that the degree refinement of the tree T contains at least one block of size 2 or more, otherwise a polynomial-time algorithm follows from Theorem 3. There are nine cases depending on the values of p and q.

A. $p = 0$.

A1. $q = 1$. This case is equivalent to the tree-isomorphism problem, which is solvable in polynomial time, even if T is not fixed.

A2. $q = 2$. Assume T does not satisfy the conditions of Lemma 4. Since the center of any tree is always of size at most 2, the degree refinement of T contains two adjacent blocks $B_i(T)$ and $B_j(T)$, with $|B_i(T)| < |B_j(T)|$, and neither $B_i(T)$ nor $B_j(T)$ contain leaves. Hence, $B_j(T)$ is adjacent to another block $B_k(T)$, $|B_j(T)| \leq |B_k(T)|$. A connected component T' of the subtree of T induced by $B_i(T) \cup B_j(T) \cup B_k(T)$ satisfies the properties of Lemma 12, possibly with the application of Lemma 13. We apply Lemma 6 to T' to show $\mathcal{NP}$-completeness for T.

A3. $q \geq 3$. Assume T does not satisfy the conditions of Lemma 2. The degree refinement of T must then contain two adjacent blocks $B_i(T)$ and $B_j(T)$, with $|B_i(T)| \neq |B_j(T)|$. Any connected component T' of the subtree induced by $B_i \cup B_j$ is either isomorphic to $S_k, k \geq 2$, or to two such stars linked as described in Lemma 13. $\mathcal{NP}$-completeness for T follows from Lemmata 10 and 6.

B. $p = 1$.

B1. $q = 1$. Assume T does not satisfy the conditions of Lemma 5. Since $m_{ij} \geq 3$, any connected component T' of T, restricted to $B_i(T) \cup B_j(T)$, is either isomorphic to a star S_k, $k \geq 3$, or two such stars linked as described in Lemma 13. $\mathcal{NP}$-completeness for T follows from Lemmata 11 and 6.

B2. $q = 2$. By Lemma 14, T either contains (in the sense of Lemma 6) a block-induced subtree isomorphic to

- P_2, in which case $\mathcal{NP}$-completeness for T follows from Lemma 8,
- S_k, $k \geq 3$, in which case $\mathcal{NP}$-completeness for T follows from Lemma 11, or
- $P_3 = S_2$, in which case $\mathcal{NP}$-completeness for T follows from Lemma 9.

B3. $q \geq 3$. As for B2 above. For a P_2 contained in T $\mathcal{NP}$-completeness follows from Lemma 8, for P_3 from Lemmata 9 and 7, and for S_k, $k \geq 3$ from Lemmata 10 and 7.

C. $p \geq 2$.

C1. $q = 1$. Assume T does not satisfy the conditions of Lemma 5. The degree refinement of T must then either contain a block $B_i(T)$ inducing a P_2, or a pair of blocks $B_i(T)$ and $B_j(T)$ inducing an S_k, $k \geq 3$. In the former case $\mathcal{NP}$-completeness for T follows from Lemma 8, in the latter from Lemmata 10 and 7.

C2. $q = 2$. As for B2 above. For a P_2 contained in T $\mathcal{NP}$-completeness follows from Lemma 8, for P_3 from Lemmata 9 and 7, and for S_k, $k \geq 3$ from Lemmata 11 and 7.

C3. $q \geq 3$. As for C2 above, with Lemma 10 instead of Lemma 11. □

References

1. James Abello, Michael R. Fellows, and John C. Stillwell. On the complexity and combinatorics of covering finite complexes. *Australasian Journal of Combinatorics*, 4:103–112, 1991.
2. Dana Angluin. Local and global properties in networks of processors (extended abstract). In *Conference Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, pages 82–93, Los Angeles, California, April 28–30 1980.
3. Norman Biggs. *Algebraic Graph Theory*. Cambridge University Press, 1974.
4. Hans L. Bodlaender. The classification of coverings of processor networks. *Journal of Parallel and Distributed Computing*, 6:166–182, 1989.
5. Béla Bollobás. *Random Graphs*. Academic Press, 1985.
6. Bruno Courcelle and Yves Métivier. Coverings and minors: Applications to local computations in graphs. *European Journal of Combinatorics*, 15:127–138, 1994.
7. Jiří Fiala and Jan Kratochvíl. Partial covers of graphs. To appear in *Discussiones Mathematicae Graph Theory*.
8. Jonathan L. Gross and Thomas W. Tucker. *Topological Graph Theory*. J. Wiley and Sons, 1987.
9. Pavol Hell and Jaroslav Nešetřil. On the complexity of H-colouring. *Journal of Combinatorial Theory, Series B*, 48(1):92–110, 1990.
10. Jan Kratochvíl, Andrzej Proskurowski, and Jan Arne Telle. Complexity of colored graph covers I. Colored directed multigraphs. In *Graph-Theoretic Concepts in Computer Science, 23rd International Workshop, WG'97, Berlin, Germany, June 18–20 1997, Proceedings*, volume 1335 of *Lecture Notes in Computer Science*, pages 242–257. Springer-Verlag, 1997.
11. Jan Kratochvíl, Andrzej Proskurowski, and Jan Arne Telle. Covering regular graphs. *Journal of Combinatorial Theory, Series B*, 71(1):1–16, 1997.
12. Jan Kratochvíl, Andrzej Proskurowski, and Jan Arne Telle. Complexity of graph covering problems. *Nordic Journal of Computing*, 5(3):173–195, 1998.
13. Petter Kristiansen and Jan Arne Telle. Generalized H-coloring of graphs. In *Algorithms and Computation, 11th International Conference, ISAAC 2000, Taipei, Taiwan, December 18–20 2000, Proceedings*, volume 1969 of *Lecture Notes in Computer Science*, pages 456–466. Springer-Verlag, 2000.
14. William S. Massey. *Algebraic Topology: An Introduction*. Harcourt, Brace and World, 1967. Reprinted 1977, Springer-Verlag.

The Complexity of Approximating the Oriented Diameter of Chordal Graphs (Extended Abstract)*

Fedor V. Fomin[1], Martín Matamala[2], and Ivan Rapaport[2]

[1] Heinz Nixdorf Institute and Paderborn University,
Fürstenallee 11, D-33102 Paderborn, Germany
fomin@uni-paderborn.de

[2] Departamento de Ingeniería Matemática and Centro de Modelamiento Matemático
UMR 2071-CNRS, Universidad de Chile, Casilla 170-3, Correo 3, Santiago, Chile
{mmatamal,irapaport}@dim.uchile.cl

Abstract. The oriented diameter of a (undirected) graph G is the smallest diameter among all the diameters of strongly connected orientations of G. We study algorithmic aspects of determining the oriented diameter of a chordal graph. We

- give a linear time algorithm such that, for a given chordal graph G, either concludes that there is no strongly connected orientation of G, or finds a strongly connected orientation of G with diameter at most twice the diameter of G plus one;
- prove that the corresponding decision problem remains NP-complete even when restricted to a small subclass of chordal graphs called split graphs;
- show that unless $P = NP$, there is neither a polynomial-time absolute approximation algorithm nor an α-approximation (for every $\alpha < \frac{3}{2}$) algorithm computing oriented diameter of a chordal graph.

Keywords: Diameter, orientation, chordal graph, approximation algorithm.

1 Introduction

A variety of different diameter problems are discussed in the literature (see the survey of Chung [3] for further references), including the problem of finding orientations for undirected or mixed graphs to minimize diameters. This problem has a long history. In 1939 Robbins [14] proved that an undirected graph G admits a strongly connected orientation if and only if G is connected and bridgeless.

* The work of IR and MM is partially supported by FONDAP on Applied Mathematics, Fondecyt 1020611 and Fondecyt 1010442. Part of this job was done while FVF was a postdoc at CMM, supported by FONDAP. FVF acknowledges support by EC contract IST-1999-14186, Project ALCOM-FT (Algorithms and Complexity - Future Technologies).

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 211–222, 2002.

More recently, Chung, Garey and Tarjan [4] provided a linear-time algorithm for testing whether a graph has a strong orientation and finding one if it does.

Chvátal and Thomassen [5], studied the following question: How are the diameter of a graph G and the diameter of a strongly connected orientation of G related? This leads to the following problem:

Oriented Diameter Problem (ODP): Given a graph G, find a strongly connected orientation H with the smallest diameter.

This question is very basic and natural. We refer to Chapter 2 of Roberts book [15] and to Chapter 2 of Bang-Jensen & Gutin book [1] for some nice applications and discussions of oriented diameter problem.

1.1 Definitions

Let G be either a simple graph or a digraph with vertex set $V(G)$ and edge set $E(G)$. By $\{u, v\}$ we denote the undirected edge with ends in u and v and by (u, v) we denote the directed arc, directed from u toward v. The distance $d_G(u, v)$ between two vertices u and v of G is the length of the shortest path (the shortest directed path if G is directed) between u and v in the graph G (from u to v if G is directed). If there is no path from u to v then we put $d_G(u, v) = +\infty$. The diameter of a graph G, denoted by $\mathrm{diam}(G)$, is defined to be the maximum distance between two vertices of G. Thus $\mathrm{diam}(G) = \max\{d_G(u, v) : u, v \in V(G)\}$. We denote by $\boldsymbol{d}_H(u, v) = \max\{d_H(u, v), d_H(v, u)\}$.

An *orientation* of an undirected graph G is a directed graph whose arcs correspond to assignments of directions to the edges of G. An orientation H of G is *strongly connected* if every two vertices in H are mutually reachable in H ($\mathrm{diam}(H) < +\infty$). An edge e in a connected graph G is called a *bridge* if $G - e$ is not connected. A connected graph G is *bridgeless* if $G - e$ is connected for every edge e, i.e. there is no bridge in G. For a graph G let us define its oriented diameter.

$$OD(G) = \min\{\mathrm{diam}(H) \colon H \text{ is an orientation of } G\}.$$

As it was proved by Robbins in 1939, if G is not connected or has a bridge, then there is no strongly connected orientation of G. In that case $OD(G) = +\infty$. Further we consider only connected bridgeless graphs and strongly connected orientations.

We say that an algorithm $\mathcal{A}$ is an (α, k)-*approximation algorithm* for ODP if for every graph G it runs in polynomial time and outputs an orientation H of G such that $\mathrm{diam}(H) \leq \alpha OD(G) + k$.

An $(\alpha, 0)$-approximation algorithm for ODP is called an α-*approximation algorithm* for ODP and an $(1, k)-$approximation algorithm for ODP is called an *absolute approximation algorithm* for ODP.

A *chord* of a cycle C in G is an edge not in C that has both end in C. A *chordless cycle* in G is a cycle of length more than three that has no chord. A graph G is *chordal* if it contains no chordless cycle.

1.2 Known Results

Chvátal and Thomassen [5] showed that ODP is NP-hard for general graphs. Therefore, there is a natural interest to investigate the complexity issues of ODP for different graph classes.

Some graph classes for which ODP can be computed are known. However, the direction of previous studies were mainly directed on finding graph classes for which oriented diameter is equal to the diameter of a graph. Koh and Tay [11] study ODP for the Cartesian products of several simple graph classes. Šoltés [18] obtained some results on complete bipartite graphs and Gutin [9] investigated n-partite complete graphs. Konig, Krumme & Lazard [12] study the orientation problem on the torus. The case of planar grids was studied by Roberts & Xu [16].

Chung [3] discusses some implications and results on ODP in context of network routing. The surveys of Hedetniemi, Hedetniemi & Liestman [10] and Fraigniaud & Lazard [7] stress the importance of ODP in study communication networks problems like broadcasting and gossiping. Discussion on one-way street arrangements problems and it relation to ODPcan be found in the book of Roberts [15].

Chordal graphs form very well investigated class of graphs. They have well understood nice properties and many NP hard problems like Coloring, Clique, Independent Set can be solved fast when the input is restricted to chordal graphs. We refer to Golumbic book [8] for the introduction and to Brandstädt, Le, & Spinrad book [2] for more recent results on chordal graphs.

Not much was known about the algorithmic aspects of ODPd for chordal graphs. Chvátal and Thomassen [5] proved that every bridgeless connected graph G admits a strongly connected orientation H with the property

(P) If an edge $\{u, v\}$ belongs to a cycle in G of length k, then (u, v) or (v, u) belongs to a directed cycle in H of length at most $(k-2)2^{\lfloor (k-1)/2 \rfloor} + 2$.

For a graph G where every edge belongs to a triangle, Property (P) tell us that the oriented diameter of G is at most 3 times the diameter of G. Connected bridgeless chordal graphs are graphs for which every edge belongs to a triangle. Therefore, Property (P) suggests the existence of a 3-approximation algorithm for ODP when restricted to chordal graphs. The searching of both better approximation algorithms and hardness results for ODP when restricted to chordal graphs, motivates this paper.

1.3 Our Contribution

In Section 2 we show that for every chordal graph G there exists computable in linear time orientation H such that for every pair of vertices u and v, $\boldsymbol{d}_H(u, v) \leq 2d_G(u, v) + 1$. Notice that this result implies that for chordal graphs ODP is $(2, 1)-$approximable. To show that the bound is sharp we construct an infinite sequence of chordal graphs such that for every graph G from this sequence any orientation of G has diameter at least $2\,\mathrm{diam}(G) + 1$.

In Section 3 we prove that ODP remains NP-hard in the subclass of chordal graphs called split graphs. Moreover, we prove two non approximability results: first, for every $\alpha < \frac{3}{2}$ ODP is not $\alpha-$approximable in the class of split graphs; second, there is no absolute approximation algorithm for ODP when restricted to chordal graphs.

2 Positive Results

Our algorithmical contribution is stated in the following theorem.

Theorem 1. *There is a linear time* $(2,1)$*-approximation algorithm for* ODP *in the class of chordal graphs.*

Theorem 1 follows from the next much stronger result.

Theorem 2. *For every bridgeless connected chordal graph* G *there exists computable in linear time orientation* H *such that, for every pair of vertices* u *and* v, $\boldsymbol{d}_H(u,v) \leq 2d_G(u,v)+1$.

Notice that Theorem 2 is an improvement of the bound of Chvátal and Thomassen applied to chordal graphs. Moreover, as stated in Theorem 3, this bound is the best possible.

Theorem 3. *For every* $n \geq 1$ *there exists a chordal graph* G_n *such that* $\operatorname{diam}(G_n) = 2n+1$ *and* $\operatorname{diam}(H) \geq 2\operatorname{diam}(G_n)+1$ *for every orientation* H *of* G_n.

Proof. Figure 1 shows a chordal graph G_2 of diameter 5 for which there is no orientation with diameter smaller than $2 \cdot 5 + 1$. This construction can be easily generalized to larger graphs. For details, see the full version of this paper. □

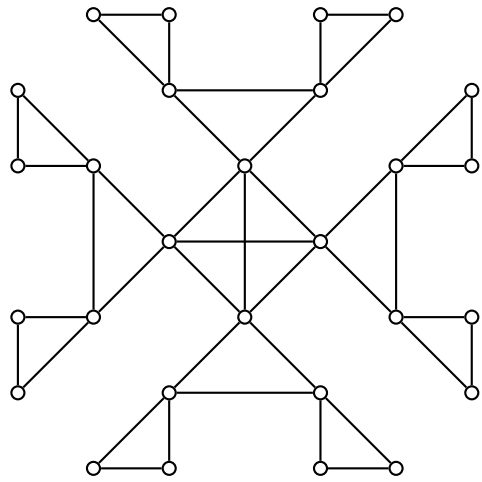


Fig. 1. Connected bridgeless chordal graph of diameter 5.

The rest of this section is devoted to the proof of Theorem 2. The proof of this theorem is indirect. First we prove that every 2-connected chordal graph has a special orientation that can be obtained from his perfect elimination ordering (Lemma 3). Then we use this orientation to prove the theorem for 2-connected graphs and only then extend the result on bridgeless graphs.

Let us begin with some definitions. For a given chordal graph G and an orientation H of its edges we say that an arc in H is *good* if it belongs to a directed triangle and it is *bad* otherwise. A good orientation is an orientation leaving every arc good. Let K_n be the complete graph with n vertices.

In order to orient chordal graphs we need first to construct good orientations of complete graphs K_n for $n \geq 5$.

Lemma 1. *For every $n \geq 5$ there exists a good orientation of K_n. Moreover, every good orientation of K_n can be extended to a good orientation of K_{n+1} and this extension can be found in linear time.*

Proof. Let us think that K_{n+1} is obtained from K_n by adding new vertex v and let H_n be a good orientation of K_n.

If n is even then a good orientation of K_{n+1} can be obtained from orientation of K_n by forming $\frac{n}{2}$ directed triangles using all edges adjacent to v. The orientation of every triangle is induced by the arc from H_n. Clearly, this orientation can be done in $O(n)$ steps.

Suppose that n is odd. Observe first that for any $n \geq 4$ and every orientation H of K_n there are three vertices in K_n inducing a triangle that is not strongly connected. Let a, b and c be such vertices for the orientation H_n. W.l.o.g. we may think that the arcs in H are of the form (a, b), (a, c) and (b, c). The remaining $n-3$ edges adjacent to v are in $(n-3)/2$ triangles, each of the triangles having one arc in K_n. We orient these edges as in the previous case. So to obtain the orientation of K_{n+1} one should choose four arbitrary vertices in K_n and find three vertices that do not undue strongly connected triangle. This can be done in constant number of steps. And the orientation of the remaining $n-3$ edges can be done in $O(n)$ steps. □

Notice that for $n = 4$ the statement of Lemma 1 is not true. In terms of diameter we have the following corollary which will be used in Section 3.

Corollary 1. *For every $n \geq 4$ there exists an orientation of K_n with diameter 2 if $n \geq 5$, and with diameter 3 if $n = 4$.*

We first consider 2-connected chordal graphs. A connected graph G is said to be 2-*connected* if for every vertex v, the graph $G - \{v\}$ is connected. At the end of the section we show how to orient a chordal graph G by using the orientations of its 2-connected components.

A vertex v in a graph G is called *simplicial* if the graph induced by its neighborhood $N_G(v)$ is a clique. By the classical result of Dirac [6], chordal graphs have been characterized as those having a *perfect elimination ordering*. This is a vertex ordering $\{v_1, \ldots, v_n\}$ such that for every $i \in \{1, \ldots, n\}$, the

vertex v_i is simplicial in $G_i := G[v_1, \ldots, v_i]$ (where $G[S]$ denotes the graph induced by the vertex set S). A perfect elimination ordering of a chordal graph can be found in linear time by using LexBFS (see the pioneering paper of Rose, Tarjan & Lueker [17]).

The idea of our construction is to orient all the edges incident to v_i in $G[v_1, \ldots, v_i]$ sequentially (following the perfect elimination ordering). If a chordal graph doesn't contain a maximal clique of size 4 then using Lemma 1 one can construct an orientation of G with diameter at most $2\,\mathrm{diam}(G)$ easily. The main problem we have to deal with is the existence of the non good orientable cliques K_4 in chordal graphs.

Let $\delta = (v_1, \ldots, v_n)$ be a perfect elimination ordering of chordal graph G. We say that vertex v_i is *super-simplicial* (subject to δ) if $N(v_i) \cap \{v_i, v_{i+1}, \ldots, v_n\} = \emptyset$. Notice that every super-simplicial vertex is simplicial but not vice versa.

We need the following technical lemma about super-simplicial vertices.

Lemma 2. *Let $\delta = (v_1, \ldots, v_n)$ be a perfect elimination ordering of a 2-connected chordal graph G. Then if v_i is not super-simplicial and $N(v_i) \cap \{v_1, v_2, \ldots, v_{i-1}\} \neq \emptyset$ then there are $k > i > l$ such that v_k, v_i, v_l is a clique in G.*

Proof. Let v_p and v_q, $p > i > q$, be vertices adjacent to v_i. If $\{v_p, v_q\} \in E(G)$ then $\{v_p, v_q, v_i\}$ induce a clique and the lemma is proved. If $\{v_p, v_q\} \notin E(G)$ then vertices $\{v_p, v_q, v_i\}$ belong to a cycle C in G (G is 2-connected). We choose C to have the shortest length among all cycles containing $\{v_p, v_q, v_i\}$. Notice that the length of C is at least 4. The cycle contains at least one vertex which is before (in δ) v_i and at least one vertex that is after v_i. Therefore, there are two adjacent vertices $v_{p'}$, $v_{q'}$ with $p' > i > q'$. Because C is the shortest cycle, the only chords in this cycle are the edges adjacent to v_i. Then chordality of G implies that v_i is adjacent to $v_{p'}$ and $v_{q'}$ which concludes the proof of the lemma. □

Lemma 3. *There exists a linear time algorithm that given a 2-connected chordal graph G and a perfect elimination ordering $\{v_1, \ldots, v_n\}$ of $V(G)$ computes an orientation H with the following properties.*

(a) Every maximal clique in G has at most one bad arc in H.
(b) If (u, v) is a bad arc in H then u is a simplicial vertex (in the perfect elimination ordering) of $V(G)$.
(c) For every $v \in V(G)$, $\boldsymbol{d}_H(v, v_1) \leq 2d_G(v, v_1)$.
(d) Every clique in H has diameter at most 3.

Proof. Iteratively, for $k = 3, \ldots, n$ we construct an orientation H_k of $G_k = G[v_1, \ldots, v_k]$ with the following properties.

(P1) Every bad arc belongs to a maximal clique (in H_k) of size four or two.
(P2) At most one arc is bad in each maximal clique.
(P3) If (u, v) is a bad arc in H then u is either a super-simplicial vertex in G or the vertices u, v are used in some step $j > k$ to form a new clique, i.e. $u, v \in N_{G_j}[v_j]$ for some $j > k$.

Clearly, for $k = 3$ such an orientation exists. We extend the orientation H_k to an orientation H_{k+1} of G_{k+1} satisfying properties (P1), (P2) and (P3).

Suppose that $N_{G_{k+1}}(v_{k+1}) = \{u_1, ..., u_r\}$. Notice that $r \geq 2$ since G is 2-connected.

1. If $r > 4$ then by (P1) we have that every arc in $H_k[u_1, ..., u_r]$ is good. We use Lemma 1 to get a good orientation of $G[u_1, ..., u_r, v_{k+1}]$.
2. For $r = 4$ we use Lemma 1 to get a good orientation of $G[u_1, ..., u_r, v_{k+1}]$. For $r = 2$ we orient new edges in a directed triangle following the orientation given to $\{u_1, u_2\}$. In both cases the bad arc in H_k (if any) belongs to one of the directed triangles in $H[u_1, ..., u_r, v_{k+1}]$ and is good in H_{k+1}.
3. For $r = 3$ we consider three cases.
   (i) If $H_k[u_1, u_2, u_3]$ contains a bad arc, say (u_1, u_2), then we direct the new edges obtaining the following arcs: (u_2, v_{k+1}) and (v_{k+1}, u_1). Moreover, if $(u_1, u_3) \in H_k$ then we add (u_3, v_{k+1}) to H_{k+1}. Otherwise we add (v_{k+1}, u_3) to H_{k+1}. Then the arcs (u_2, v_{k+1}), (v_{k+1}, u_1) and (u_1, u_2) are in a directed triangle and the arc between v_{k+1} and u_3 is also in a directed triangle.
   (ii) If $H_k[u_1, u_2, u_3]$ has no bad arcs and v_{k+1} is not super-simplicial (with respect to the perfect elimination ordering) then by Lemma 2 at least one edge, say, with ends in $\{v_{k+1}, u_1\}$, is used in a step $j > k$. Then we direct edges $\{v_{k+1}, u_2\}$ and $\{v_{k+1}, u_3\}$ to form a directed triangle with arc (u_2, u_3) (or (u_3, u_2)) and we add the bad arc (v_{k+1}, u_1).
   (iii) If $H_k[u_1, u_2, u_3]$ has no bad arcs and v_{k+1} is super-simplicial then we direct edges $\{v_{k+1}, u_2\}$ and $\{v_{k+1}, u_3\}$ to form a directed triangle with arc (u_2, u_3) (or (u_3, u_2)) and we add the bad arc (v_{k+1}, u_1), where the vertex u_2 has among all u_i the minimum distance in G to v_1.

It is easy to see that the orientation H_{k+1} satisfies properties (P1), (P2) and (P3).

Clearly the orientation H satisfies properties (a) and (b). We prove that H satisfies Property (c) by induction in k. As before let $\{u_1, \ldots, u_r\} = N_{G_{k+1}}(v_{k+1})$. Let us assume that for all $v \in G_k$

$$\boldsymbol{d}_H(v_1, v) \leq 2d_G(v_1, v) \tag{1}$$

If there is no bad arcs in H connecting v_{k+1} with $u_1, \ldots, u_r$ we have that $\boldsymbol{d}_H(u_i, v_{k+1}) \leq 2$ for all $i = 1, \ldots, r$ and (1) holds for G_{k+1}.

If v_{k+1} is connected to some u_i by a bad arc in H then $r = 3$, $\{u_1, u_2, u_3\}$ induces a directed triangle in H and v_{k+1} is a simplicial vertex of G. Therefore $d_G(v_1, v_{k+1}) = d_G(v_1, u_2) + 1$ where u_2 is the vertex having minimum distance to v_1 in G, among all u_i, $i = 1, 2, 3$. By the construction of H, there exists a directed triangle that contains v_{k+1} and u_2 which implies that $\boldsymbol{d}_H(u_2, v_{k+1}) \leq 2$. Therefore $\boldsymbol{d}_H(v_1, v_{k+1}) \leq 2d_G(v_1, u_2) + 2 = 2d_G(v_1, v_{k+1})$. Property (d) follows from (P2).

Finally, we claim that for every k the orientation of the arcs adjacent to v_{k+1} during the extension of orientation H_k to H_{k+1} can be performed in $O(|N(v_k)|)$.

We assume that the set of super-simplicial vertices is known. (Clearly this set can be computed in linear time $O(\sum_{v\in V(G)} |N(v)|) = O(|E(G)|)$.)

If we are in the cases 1 or 2 then the orientation of arcs can be performed in $O(|N(v_k)|)$ steps by Lemma 1. If we are in case 3 then subcase (i) is performed in a constant number of steps. For subcases (ii) we should be able to find an edge adjacent to v_{k+1} that will be turned into a bad arc. To find this edge we need a vertex from $\{u_1, u_2, u_3\}$ which is adjacent to a vertex v_j, $j > k+1$. This can be done in linear time. The subcase (iii) takes constant number of steps.

Finally, the complexity of the algorithm is $O(\sum_{1\le k\le n} |N(v_k)|) = O(|E(G)|)$. □

Lemma 4. *There exists a linear time algorithm that given a 2-connected chordal graph G computes an orientation H such that, for every pair of vertices u and v, $\boldsymbol{d}_H(u,v) \le 2d_G(u,v) + 1$.*

Proof. Given G the algorithm first computes (in linear time) a perfect elimination ordering and then the orientation H (in linear time) given by Lemma 3. We prove that H has the desired property.

Take $u, v \in V(G)$ and let P be a shortest $(u,v)-$path in G. If $d_G(u,v) = 1$ then u, v are in some clique C, $|C| \ge 3$. From Property (d) we have $\boldsymbol{d}_H(u,v) \le 3$.

Suppose that $d_G(u,v) > 1$. Clearly, the inner vertices of P cannot be simplicial; therefore each arc in H associated to some inner edge of P is contained in a directed triangle in H.

If the arc associated to the edge in P incident to u is bad then u is simplicial and the arc is directed from u in the orientation H. Therefore, if the arc e associated to the edge of P incident with v is contained in a directed triangle (in H), then we get $\boldsymbol{d}_H(u,v) \le 2d_G(u,v)$.

If the arc e is bad then v is simplicial and $e = (y,v)$. But $\boldsymbol{d}_H(y,v) \le 3$. Hence, $\boldsymbol{d}_H(u,v) \le \boldsymbol{d}_H(u,y) + \boldsymbol{d}_H(y,v) \le 2d_G(u,y) + 3 = 2d_G(u,v) + 1$. □

Considering the special rôle that vertex v_1 plays in the orientation H obtained in Lemma 3, we say that H is rooted in v_1 and by extension we say that G is rooted in v_1.

Proof of Theorem 2. For a simplicial decomposition $\{v_1, \ldots, v_n\}$, we consider the 2-connected component C_0 of G that contains v_1 and we orient it as in Lemma 4. The set of 2-connected components has a tree-like structure T which we consider rooted in C_0. By the classical result of Tarjan [19], the tree-like structure of 2-connected component can be computed in linear time. Notice that the notion of father and sons of a 2-connected component is well defined. For every 2-connected component C we define its *father* cut vertex as the unique vertex in C which belongs to its father.

To each 2-connected component we assign an orientation as in Lemma 3 rooted in its father cut vertex. Let H be the orientation of G so obtained. In each 2-connected component the construction of H is done in linear time by Lemma 3. the orientation H is computable in linear time.

Let u and v be two vertices of G and let P be a shortest path between u and v in G. If P has no cut points then u and v lay in the same 2-connected component. From Lemma 4 and the construction of H we have $\boldsymbol{d}_H(u,v) \leq 2d_G(u,v)+1$. Otherwise, let $u_1,\ldots,u_r$ be the cut points in P and C_i the 2-connected component containing u_i and u_{i+1} for $i=1,\ldots,r-1$. Notice that for at most one i_0 neither u_{i_0} nor u_{i_0+1} are father cut vertices of C_{i_0}. From the construction of H we know that $\boldsymbol{d}_H(u_i,u_{i+1}) \leq 2d_G(u_i,u_{i+1})$ for all $i=1,\ldots,r$ not equal to i_0 (in this case $\boldsymbol{d}_H(u_{i_0},u_{i_0+1}) \leq 2d_G(u_{i_0},u_{i_0+1})+1$). Therefore, $\boldsymbol{d}_H(u,v) \leq 2d_G(u,v)+1$. □

3 Negative Results

Our first step is to prove the NP-hardness of ODP for chordal graphs. In fact we will prove a stronger result: The NP-hardness of ODP for split graphs. A graph G is a split graph if its vertex set $V(G)$ can be partitioned into sets C and I such that C is a clique and I is an independent set. Split graphs form a subclass of chordal graphs of diameter at most 3.

Our proof, inspired by the one of Chvátal and Thomassen [5], relies on the NP-completeness of the 2-coloring problem for hypergraphs obtained by Lovász [13]. Let us recall that a hypergraph $\mathcal{H}$ is called 2-colorable if its vertices can be colored red and blue in such a way that every edge includes at least one vertex of each color.

Lemma 5. *For every $k \geq 0$ and for every hypergraph $\mathcal{H}$ there exists a chordal graph $G^k_{\mathcal{H}}$ (split graph for $k=0$) such that if $\mathcal{H}$ is 2-colorable then $OD(G^k_{\mathcal{H}}) = 2(k+1)$ and if $\mathcal{H}$ is not 2-colorable then $OD(G^k_{\mathcal{H}}) = 3(k+1)$*

Proof. We first consider the case $k=0$. For a given hypergraph $\mathcal{H}$, we will construct a split graph $G^0_{\mathcal{H}} = G_{\mathcal{H}}$ such that $\mathcal{H}$ is 2-colorable if and only if there is an orientation of $G_{\mathcal{H}}$ of diameter 2. Let $\mathcal{H}$ be a hypergraph with vertex set V of size n and edge set E of size m.

The clique C of $G_{\mathcal{H}}$ contains $n+2m+2$ vertices. More precisely, $C = V \cup Y$ with $V = \{v_1, v_2, \ldots, v_n\}$ being the vertex set of $\mathcal{H}$, and $Y = \{\alpha, \beta\} \cup E_1 \cup E_2$ where E_1 and E_2 are copies of the edge set E of $\mathcal{H}$. The independent set I of $G_{\mathcal{H}}$ contains $m+1$ vertices. More precisely, $I = \{x\} \cup E$.

Now let us explain how to connect the vertices of I with those of C. The vertex x is connected to all the vertices of V. A vertex $e \in E$ is connected to a vertex $v \in V$ if and only if $v \in e$ (in the hypergraph $\mathcal{H}$). Finally, every vertex $y \in Y$ is connected to every vertex e of E. Clearly, $OD(G_{\mathcal{H}}) \leq 3$.

Let D be an orientation of $G_{\mathcal{H}}$ of diameter two. The way we color every vertex v of the hypergraph $\mathcal{H}$ is the following: If according to D, the edge connecting vertex x with v is oriented towards v, then we color it red (otherwise we color it blue). Since for every vertex $e \in E$ the distance $d_D(x,e) = d_D(e,x) = 2$, it follows that every edge e in $\mathcal{H}$ contains a red and a blue vertex.

Now let us suppose that $\mathcal{H}$ is 2-colorable. Let us denote by R and B the set of red and blue vertices in V. We partition the sets R and B as follows:

$R = \{r\} \cup R'$, $B = \{b\} \cup B'$ (we just need the sets R', B', E_1, E_2 to have more than 5 elements; we can assume without loss of generality that the 2-coloring problem is restricted to instances satisfying this). In Corollary 1, we showed that we can orient the internal edges of R', B', E_1, E_2 in order to achieve, for each subgraph, an internal diameter 2.

The rest of the orientation is described in the following $0-1$ matrix. A value 1 in the position (P, Q) means that all the edges connecting the vertices of P with those of Q are oriented from P towards Q.

	x	r	R'	b	B'	α	β	E	E_1	E_2
x		1	1							
r				1		1	1			
R'					1				1	1
b	1					1	1			
B'	1								1	1
α			1		1			1	1	
β			1		1	1				1
E							1			1
E_1		1		1			1	1		
E_2		1		1		1			1	

The only edges that have not been oriented yet are those connecting the subsets R and B with E. By orienting them we will solve the problem of reaching E from x and reaching x from E. The solution is easy: We orient all the edges between R and E from R towards E and all the edges between B and E from E towards B.

The last problem is to reach any $e \in E$ from any other $e' \in E$. For achieving this we slightly modify the orientation between the sets E, E_1 and E_2. We identify m disjoint directed triangles of the form $e \to e_1 \to e_2 \to e$, with $e \in E$, $e_1 \in E_1$ and $e_2 \in E_2$, and we reverse the order getting $e_2 \to e_1 \to e \to e_2$. The reader should be able to verify that the diameter of D is at most 2.

For $k > 0$ the chordal graph $G^k_{\mathcal{H}}$ is constructed from several copies of the split graph $G^0_{\mathcal{H}} = (C, \{e_1, \ldots, e_m, x\})$ arranged in a tree-like structure (see Figure 2). In this way we are able to "amplify the gap" of the diameter of $G^0_{\mathcal{H}}$ according to the colorability of $\mathcal{H}$. The details can be found in the full version of this paper. □

Theorem 4. ODP *is NP-hard for split graphs.*

Proof. Taking $k = 0$ in Lemma 5 the 2-coloring problem for hypergraphs can be reduced to ODP in polynomial time. □

Now we prove two results concerning the hardness of approximating the oriented diameter.

Theorem 5. *Let $\alpha < \frac{3}{2}$. Unless $P = NP$,* ODP *has no α-approximation algorithm for split graphs.*

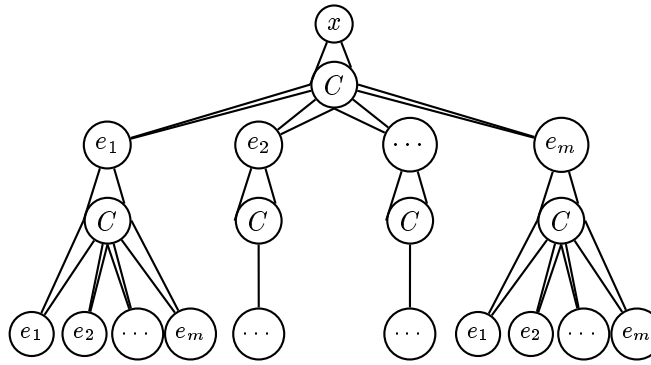


Fig. 2. Tree-like structure of $G^2_{\mathcal{H}}$.

Proof. Let $A(G)$ be the orientation assigned to a graph G by an α-approximation algorithm for ODP. If $\mathcal{H}$ is 2-colorable then $G_{\mathcal{H}}$ has diameter 2. Thus $\text{diam}(A(G_{\mathcal{H}})) \leq 2\alpha < 3$. On the other hand, if $\mathcal{H}$ is not 2-colorable then every orientation of the graph $G_{\mathcal{H}}$ has diameter at least 3. Whence $\text{diam}(A(G_{\mathcal{H}})) \geq 3$. Therefore, since $G_{\mathcal{H}}$ can be constructed in polynomial time, by knowing $\text{diam}(A(G_{\mathcal{H}}))$ we can decide the 2-colorability of $\mathcal{H}$. □

Theorem 6. *Unless $P = NP$, there is no absolute approximation algorithm for* ODP *when restricted to chordal graphs.*

Proof. Let us assume that there exist K and an absolute approximation algorithm for ODP such that $\text{diam}(A(G)) \leq OD(G) + K$. By using this algorithm we could decide the 2-coloring problem for hypergraphs. Let $\mathcal{H}$ be a hypergraph and $k > K$. From Lemma 5 there exists a chordal graph $G^k_{\mathcal{H}}$ computable in polynomial time such that, if $\mathcal{H}$ is 2-colorable, then $G^k_{\mathcal{H}}$ has diameter $2k + 2$. Thus $\text{diam}(A(G^k_{\mathcal{H}})) \leq 2k + 2 + K < 3k + 2$. And, if H is not 2-colorable then $G^k_{\mathcal{H}}$ has diameter $3k + 2$. Thus $\text{diam}(A(G^k_{\mathcal{H}})) \geq 3k + 2$. □

4 Concluding Remarks

In this paper we have provided linear time $(2, 1)$-approximation algorithm for oriented diameter of chordal graphs. From another hand, we proved that for every $\alpha < 3/2$ finding an orientation with diameter at most α times the oriented diameter is NP hard. The challenging question is to decrease the gap between these lower and upper bounds. But even existence of 2-approximation algorithm is an interesting open problem.

References

1. J. Bang-Jensen and G. Gutin, *Digraphs*, Springer-Verlag London Ltd., London, 2001. Theory, algorithms and applications.

2. A. Brandstädt, V. B. Le, and J. P. Spinrad, *Graph classes: a survey*, SIAM Monographs on Discrete Mathematics and Applications, Society for Industrial and Applied Mathematics, Philadelphia, 1999.
3. F. R. K. Chung, *Diameters of communication networks*, in Mathematics of information processing (Louisville, Ky., 1984), Amer. Math. Soc., Providence, R.I., 1986, pp. 1–18.
4. F. R. K. Chung, M. R. Garey, and R. E. Tarjan, *Strongly connected orientations of mixed multigraphs*, Networks, 15 (1985), pp. 477–484.
5. V. Chvátal and C. Thomassen, *Distances in orientations of graphs*, J. Combinatorial Theory Ser. B, 24 (1978), pp. 61–75.
6. G. Dirac, *On rigid circuit graphs*, Abhandl. Math. Sem. d. Univ. Hamburg, 25 (1961), pp. 71–76.
7. P. Fraigniaud and E. Lazard, *Methods and problems of communication in usual networks*, Discrete Appl. Math., 53 (1994), pp. 79–133.
8. M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
9. G. Gutin, *Minimizing and maximizing the diameter in orientations of graphs*, Graphs Combin., 10 (1994), pp. 225–230.
10. S. M. Hedetniemi, S. T. Hedetniemi, and A. L. Liestman, *A survey of gossiping and broadcasting in communication networks*, Networks, 18 (1988), pp. 319–349.
11. K. M. Koh and E. G. Tay, *On optimal orientations of Cartesian products of even cycles*, Networks, 32 (1998), pp. 299–306.
12. J.-C. Konig, D. W. Krumme, and E. Lazard, *Diameter-preserving orientations of the torus*, Networks, 32 (1998), pp. 1–11.
13. L. Lovász, *Coverings and coloring of hypergraphs*, in Proceedings of the Fourth Southeastern Conference on Combinatorics, Graph Theory, and Computing (Florida Atlantic Univ., Boca Raton, Fla., 1973), Utilitas Math., Winnipeg, Man., 1973, pp. 3–12.
14. H. E. Robbins, *A theorem on graphs with an application to a problem of traffic control*, Amer. Math. Monthly, 46 (1939), pp. 281–283.
15. F. S. Roberts, *Graph theory and its applications to problems of society*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, Pa., 1978.
16. F. S. Roberts and Y. Xu, *On the optimal strongly connected orientations of city street graphs. I. Large grids*, SIAM J. Discrete Math., 1 (1988), pp. 199–222.
17. D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic aspects of vertex elimination on graphs*, SIAM J. Comput., 5 (1976), pp. 266–283.
18. L. Šoltés, *Orientations of graphs minimizing the radius or the diameter*, Math. Slovaca, 36 (1986), pp. 289–296.
19. R. E. Tarjan, *Depth first search and linear graph algorithms*, SIAM Journal on Computing, 1 (1972), pp. 146–160.

Radiocolorings in Periodic Planar Graphs: PSPACE-Completeness and Efficient Approximations for the Optimal Range of Frequencies

D.A. Fotakis[2], S.E. Nikoletseas[1], V.G. Papadopoulou[1], and P.G. Spirakis[1,⋆]

[1] Computer Technology Institute (CTI) and Patras University
Riga Fereou 61, 26221 Patras, Greece
{nikole,viki,spirakis}@cti.gr
[2] Max-Planck-Institute für Informatik,
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, fotakis@mpi-sb.mpg.de

Abstract. The Frequency Assignment Problem (FAP) in radio networks is the problem of assigning frequencies to transmitters exploiting frequency reuse while keeping signal interference to acceptable levels. The FAP is usually modelled by variations of the graph coloring problem. The Radiocoloring (RC) of a graph $G(V, E)$ is an assignment function $\Phi : V \to \mathbb{N}$ such that $|\Phi(u) - \Phi(v)| \geq 2$, when u, v are neighbors in G, and $|\Phi(u) - \Phi(v)| \geq 1$ when the distance of u, v in G is two. The range of frequencies used is called *span*. Here, we consider the optimization version of the Radiocoloring Problem (RCP) of finding a radiocoloring assignment of minimum span, called *min span RCP*.
In this paper, we deal with a variation of RCP: that of satisfying frequency assignment requests with some *periodic* behavior. In this case, the interference graph is an (infinite) periodic graph. Infinite periodic graphs model finite networks that accept periodic (in time, e.g. daily) requests for frequency assignment. Alternatively, they may model very large networks produced by the repetition of a small graph.
A *periodic graph* G is defined by an infinite two-way sequence of repetitions of the same finite graph $G_i(V_i, E_i)$. The edge set of G is derived by connecting the vertices of each iteration G_i to some of the vertices of the next iteration G_{i+1}, the same for all G_i. The model of periodic graphs considered here is similar to that of periodic graphs in Orlin [13], Marathe et al [10]. We focus on planar periodic graphs, because in many cases real networks are planar and also because of their independent mathematical interest. We give two basic results:

- We prove that the *min span RCP is $\mathcal{PSPACE}$-complete for periodic planar graphs.*
- We provide an $O(n(\Delta(G_i) + \sigma))$ time algorithm, (where $|V_i| = n$, $\Delta(G_i)$ is the maximum degree of the graph G_i and σ is the number of edges connecting each G_i to G_{i+1}), which obtains a radiocoloring of a periodic planar graph G that *approximates the minimum span within a ratio which tends to* 2 *as* $\Delta(G_i) + \sigma$ *tends to infinity.*

⋆ This work has been partially supported by the EU IST/FET projects ALCOM-FT, FLAGS, CRESCCO and EU/RTN Project ARACNE. Part of the last author's work was done during his visit at Max-Planck-Institute für Informatik (MPI).

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 223–234, 2002.

1 Introduction, Previous Work and Our Results

1.1 The Radiocoloring Problem

The Frequency Assignment Problem (FAP) in radio networks is a well-studied, interesting and well motivated problem, aiming at assigning frequencies to transmitters exploiting frequency reuse while keeping signal interference to acceptable levels. The FAP is usually modeled by variations of the graph coloring problem. The interference between transmitters is usually modelled by the interference graph $G(V, E)$, where the set V corresponds to the set of transmitters and E represents distance constraints. The set of colors represents the available frequencies. In addition, the color of each vertex in a particular assignment gets an integer value which has to satisfy certain inequalities compared to the values of colors of nearby nodes in the interference graph G (frequency-distance constraints). We study an important variation of FAP, the Radiocoloring Problem (RCP):

Definition 1. Radiocoloring Problem (RCP): *Given a graph $G(V, E)$ consider a function $\Phi : V \to N$ such that $|\Phi(u) - \Phi(v)| \geq 2$ if $d(u, v) = 1$ and $|\Phi(u) - \Phi(v)| \geq 1$ if $d(u, v) = 2$, where $d(u, v)$ is the distance between u and v in G. The function Φ is a radiocoloring (RC) of the graph G. We call the problem of finding such an assignment the Radiocoloring Problem (RCP).*

An important parameter of a radiocoloring assignment is the following:

Definition 2. span: *The number $\nu = max_{v \in V}\Phi(v) - min_{u \in V}\Phi(u) + 1$ used in a radiocoloring assignment Φ is called the span of the assignment Φ.*

The optimization version of RCP corresponding to this parameter, considered here, is the following:

Definition 3. min span RCP: *Given a graph G, find a radiocoloring assignment for the graph G of minimum span, denoted by $X_{span}(G)$.*

Note that this version of RCP has been proved to be $\mathcal{NP}$-hard even for planar and other restricted families of ordinary graphs ([5,2]).

1.2 The Min Span RCP in Periodic Graphs

In this work we investigate the min span RCP for an interesting family of infinite planar graphs, called *periodic planar graphs*. A periodic graph G is defined by an infinite sequence of repetitions of the same finite graph $G_i(V_i, E_i)$. The edge set of G is derived by connecting some of the vertices of each iteration G_i to some of the vertices of the next iteration G_{i+1}, the same for all iterations.

Infinite periodic graphs usually represent finite networks that accept periodic (in time, e.g. daily) requests for frequency assignment. We note that periodic interference graphs usually represent networks of great practical interest, since in many networks the requests for frequency assignment exhibit some periodic behavior. That is, the network accepts periodic (e.g. daily) requests for frequency

assignment. Each request has a starting and ending time and a node where it is applied. Two requests interfere if they apply for nearby nodes and their time intervals overlap. The assignment should be such that there is no time overlap between any two nearby requests of the same or the preceding and following periods of requests. Alternatively, infinite periodic graphs can model very large networks produced by the repetition of a small graph. Note in this context that many real networks consist of the repetition of the same component.

We focus here on *planar* periodic graphs, because in many cases real networks are planar and because of the independent mathematical interest of this family of graphs.

Definition 4. Linear Periodic Planar Graph G**:** *A linear periodic planar graph is defined as follows:*

Let $\widetilde{G}$ *be an arbitrary finite connected planar graph. Let* V *the vertex set of* $\widetilde{G}$*. Let also* E_0 *be the edge set of* $\widetilde{G}$*. Let* E_+ *be a specific set of ordered pairs* (u, v) *of the nodes of* $\widetilde{G}$*. Note that* E_+ *must be a set of ordered pairs of vertices whose connection according to the rule* **(c2)** *below leads to planarity preservation.*

Consider the two-way infinite sequence of graphs $\ldots, G_i, G_{i+1}, \ldots$*, where each* G_i *is isomorphic to* $\widetilde{G}$*. The infinite graph* G *is obtained from this sequence as follows:*

(a) *We assume a line (in fact, any 1-dimensional infinite simple curve) on which we select discrete points* $\ldots, i, i+1, i+2, \ldots$*, such that:*
- **(a1)** *Each point in the line is replaced by* $\widetilde{G}$*.*
- **(a2)** *Each edge* $(i, i+1)$ *in the line is replaced by* E_+*.*
- **(a3)** *For any finite subset of consecutive points in the line, replacing the points of the line by graphs* $\widetilde{G}$ *end the edges between them by* E_+*, the resulting graph is planar.*

(b) *The vertex set of* G *is the union of the vertex sets of the sequence* $\ldots, G_i, G_{i+1}, \ldots$*.*

(c) *The edges of* G *are* **(c1)** *The union of edge sets of the sequence of* G_i*s (i.e., the edge set* E_0 *of* $\widetilde{G}$*)* **(c2)** *For each pair of adjacent copies of* $\widetilde{G}$*, call them* G_i*,* G_{i+1}*, we use the* E_+ *specification of* G *to connect the nodes of* G_i *corresponding to the first elements of the pairs in* E_+ *to the nodes of* G_{i+1}*, corresponding to the second elements of the pairs in* E_+*.*

We denote a linear periodic planar graph by $G = (\widetilde{G}(V, E_0), E_+)$*.*

For an iteration i of G, $G_i(V_i, E_i)$, let E_{i-} and E_{i+} the set of edges connecting G_i with the previous and next iterations of G, respectively. Note that both E_{i-} and E_{i+} are the same as E_+ and that G_i is isomorphic to $\widetilde{G}$.

Note that this model of infinite periodic graphs is similar to the model of Orlin [13] for *l-dimensional periodic graphs* with $l = 1$. Our model is in fact equivalent to 1-level-restricted periodic graphs in Marathe et al [10], which is a special case of [13]. We consider the planar case of such graphs.

Definition 5. $(\widetilde{\boldsymbol{G}}, \boldsymbol{E_+})$: *The pair* $\widetilde{G}, E_+$ *is called the finite specification of* G*.*

Observe that there can be infinite periodic graphs which are not linear. For example, the periodic graph whose graph $\widetilde{G}$ in its finite specification, is a cycle connecting vertices a, b, c, d, and $E_+ = \{(a,a), (b,b), (c,c), (d,d)\}$, is not a linear periodic planar graph, since there is no line of discrete points such that each point in the line can be replaced by $\widetilde{G}$ and each edge of it can be replaced by E_+ leading to planarity.

Note 1. All our results refer to linear periodic planar graphs, which we call periodic planar graphs in the sequel.

1.3 Our Results

1. We first prove that *the min span RCP is $\mathcal{PSPACE}$-complete for periodic planar graphs.* (The space is polynomial with respect to the size of the finite specification $\widetilde{G}$.)
2. We provide an $O(n(\Delta(G_i)+\sigma))$ time algorithm, (where $|V_i| = n$, $\Delta(G_i)$ is the maximum degree of the graph G_i and σ is the number of edges connecting G_i to G_{i+1}), which obtains a radiocoloring of a periodic planar graph G that *approximates the minimum span within a ratio which tends to* 2 *as* $\Delta(G_i)+\sigma$ *tends to infinity.*

We remark that, any approximation algorithm for the min span RCP of a finite planar graph G, that achieves a span of at most $\alpha\Delta(G) + constant$, for any α and where $\Delta(G)$ is the maximum degree of G, can be used as a subroutine in our algorithm to produce an approximation for min span periodic planar RCP of asymptotic ratio α for periodic planar graphs.

1.4 Previous Work

The FAP has been considered in e.g. [4,1]. The problem of min span RCP for ordinary planar graphs was proved to be $\mathcal{NP}$-complete in [5,6,2]. In [5,1] the authors provide an efficient approximation algorithm for the min order RCP of planar graphs achieving an approximation ratio tending to 2 and 1.8 (for graphs of large degree ($\Delta(G) \geq 749$)), respectively. The work of [2] presents efficient approximations for some interesting families of graphs: outerplanar, bounded treewidth, permutation and split graphs.

A model for periodic graphs (called *l-dimensional periodic graphs*) was first presented by Orlin in [13]. The model of periodic graphs considered in this work is similar to that of Orlin for the 1-dimensional case, $l = 1$ (also called *1-dimensional periodically specified graphs* or simply *periodically specified graphs*), when restricted to planar instances.

The complexity of various basic problems of periodically specified graphs was studied by Orlin [13] and Wanke [17]. In [13,11,16] it is proved that the problems of Maximum Independent Set (MIS), Hamiltonial Path, Partition into Triangles, SAT, 3-coloring for periodically specified graphs are $\mathcal{PSPACE}$-complete. The appoximability of basic problems on infinite periodic graphs was studied by several researchers ([3,8,14]) giving efficient algorithms for solving problems

such as determining strongly connected components, testing the existence of cycles, bipartiteness, planarity and minimum cost spanning forests for periodically specified graphs.

Marathe et al [10] presented several $\mathcal{PSPACE}$-hardness results and also efficient approximation schemes for partitioning problems including MIS, min vertex cover and max-SAT for periodically specified graphs when restricted to planar instances. However, their approximation technique for periodically specified graphs (illustrated for the MIS problem) can not directly apply for coloring problems, considered here, because it takes the union of partial solutions-subsets of the infinite graph and thus it does not consider all the vertices; something not allowed in coloring problems.

Because of the page limit, we have included full proofs of some Lemmas and Theorems in the full version of the paper [7].

2 Embeddings of Periodic Planar Graphs

In this paper, we use the notion of an *embedding* of a planar graph.

Definition 6. Planar Embedding (of a periodic graph G) *([12]): For each node v of G, there is an adjacency list, such that all neighbours of v appear in clockwise order with respect to an actual drawing of G.*

The following Lemma reveals important information about the structure of a linear periodic planar graph.

Lemma 1. *Any linear periodic planar graph G can be embedded in the plane by interchanging at most two different planar embeddings of the graph obtained by an iteration i of G, G_i (which is isomorphic to $\widetilde{G}$) and the set of edges connecting G_i with the previous and next iterations, sets E_{i-}, E_{i+} (each of which is equal to E_+), called* Extended $\widetilde{G}$.

Proof. Observe first that there are cases where we need to interchange two different embeddings of the graph Extended $\widetilde{G}$ in order to draw a linear periodic planar graph preserving planarity. As an example, consider the periodic graph whose graph $\widetilde{G}$ in its finite specification is a single edge connecting two vertices a, b, and $E_+ = \{(a, b), (b, a)\}$. See the graph of Figure 1(a). We will show that interchanging at most two different embeddings of the graph Extended $\widetilde{G}$ is enough to draw any linear periodic planar graph G preserving planarity.

In order to check whether it is possible that three embeddings to be required, we need to consider any 3 consecutive iterations of G. Recall that different embeddings may be introduced because of the connections between nodes of consecutive iterations. Hence, only nodes of the exterior face of each iteration will be involved. So, we can view the three consecutive iterations as three cycles. Moreover, we can consider three simple lines, since less edges are involved in the embeddings of consecutive iterations in the case of lines compared to cycles.

Finally, observe that, in order for three or more planar embeddings to be needed, we have to consider at least 3 nodes of G_i, assume a, b, c. So, we can consider a line consisting of nodes a, b, c, having edges ab, bc (denoted by L_3).

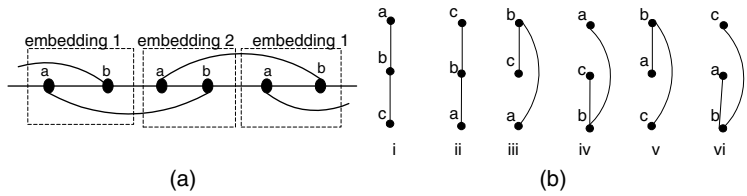


Fig. 1. (a) For this graph we need to interchange two different embeddings of the graph Extended $\widetilde{G}$, in order to draw the periodic graph in the plane without crossings. (b) The six possible 'drawings' of Line L_3.

We distinguish 6 different 'drawings' of a line L_3 in the plane (see Figure 1(b)). Note that some of them are equivalent embeddings with respect to definition 6 of a planar embedding. However, we consider all 6 of them since for an iteration i each such drawing combined with sets of edges E_{i-} and E_{i+} can result in a distinct embedding of the graph Extended $\widetilde{G}$. In our case the Extended $\widetilde{G}$ is the graph obtained by L_3 of iteration i and a set of edges connecting L_3 with the previous and next iterations, (sets E_{i-}, E_{i+}). In the following, we use the term 'drawing' when we refer to any of these 6 drawings and the term 'embedding' when we refer to any of them together with an instance of sets E_{i-} and E_{i+}.

To check whether three embeddings may be needed, we need to check all possible triples of these drawings. For each such triple, we check all possible sets E_+ that can lead to a linear periodic graph. For each such set, we show that interchanging at most two of these six drawings in any two consecutive iterations, is enough to draw the infinite graph in the plane. Recall that two drawings (of consecutive iterations i and $i+1$) combined with the sets of edges connecting each iteration with next and previous iterations (sets E_{i-}, E_{i+} and $E_{(i+1)-}$, $E_{(i+1)+}$) result in two different embeddings of the graph Extended $\widetilde{G}$. Henceforth, interchanging two different embeddings of the graph Extended $\widetilde{G}$ we can draw the infinite periodic planar graph in the plane without crossings.

By exhaustive check of all possible cases (see the full version [7]), we conclude that in all cases, interchanging at most two embeddings of the graph $\widetilde{G}$, is enough to draw the linear periodic graph in the plane, without edge crossings. □

Lemma 2. *Assume that we construct an infinite graph as in Definition 4 except that instead of* **(a3)** *we have that for any three consecutive points of the line, replacing each point by graph $\widetilde{G}$ and the edges among consecutive points by E_+, the resulting graph is planar.*

Then, the infinite graph thus constructed is a linear periodic planar graph.

Proof. The above requirement implies **(a3)** of Def. 4 because of Lemma 1. To see why, observe that there are two possible ways to draw a linear periodic planar graph: using the infinite sequence $\ldots, A, B, A, \ldots$ or $\ldots, B, A, B, \ldots$ (where A, B are the two planar embeddings of $\widetilde{G}$ needed). Our assumption guarantees that the coexistence of AB and BA graphs (together with E_+) has a planar embedding. Thus, by induction, any longer string will also have a planar embedding. □

3 The $\mathcal{PSPACE}$-Completeness of Min Span RCP for Periodic Planar Graphs

We prove have that min span RCP is $\mathcal{PSPACE}$-complete for periodic planar graphs. In order to show this, we need to prove that a number of problems are $\mathcal{PSPACE}$-complete. Our $\mathcal{PSPACE}$-completeness proofs utilize known constructions for the $\mathcal{NP}$-completeness of corresponding problems for ordinary graphs. However, note that applying those constructions on the infinite periodic graph G we need infinite time to get the transformed graph G' with the desired properties. We manage to apply the transformation only to a part of the infinite graph (an iteration) and from the obtained graph to get the transformation of the whole infinite periodic graph, thus the new graph G', in time polynomial to the size of the finite specification of G. Moreover, the new graph fulfills the desired properties. These are achieved by exploiting some 'locality characteristics' that the constructions utilized here exhibit, i.e. the construction applies locally on a part (vertex or edge) of this part involving only information of the neighborhood of the part and it affects only to this neighborhood. This, combined with the repetitive structure of an infinite periodic graph enables us to get constructions of polynomial time in the size of the finite specification of the graph.

Let any iteration $G_i(V_i, E_i)$ of G and let E_{i+} the set of edges connecting any iteration G_i to the next iteration G_{i+1}. A *constant period 4-edge coloring* of a periodic graph G is a 4-edge coloring of G that assigns to each edge uv of E_i and E_{i+} of any iteration G_i the same color as the color assigned to the corresponding edge $u'v'$ of any other iteration G_j of G.

Lemma 3. *The problem of deciding whether a periodic planar graph $G = (\widetilde{G}(V, E_0), E_+)$ is 3-colorable (also called periodic planar 3-coloring) is $\mathcal{PSPACE}$-complete.*

Proof. (a) *Membership in $\mathcal{PSPACE}$:*
The proof that this problem is in $\mathcal{PSPACE}$ is similar to that for periodic SAT [16]: The 3-coloring of two adjacent iterations must repeat after at most 3^{2n} copies, and thus the whole 3-coloring can be assumed to be periodic with exponential period. A polynomial-space machine can then guess and check such a 3-coloring.

More analytically, suppose that the given periodic graph is 3-colorable, and consider a valid 3-coloring of it. This assignment consists of a two-way infinite $\ldots, T_i, T_{i+1}, T_{i+2}, \ldots$ of valid 3-coloring assignments to the various blocks of nodes (one for each iteration). Each T_i is an element of $\{1, 2, 3\}^n$, where $n = |V|$ is the number of vertices of the graph G_i.

The ith *chunk*, where i is any integer, is the pair (T_i, T_{i+1}), of two consecutive valid 3-coloring assignments. Since there are 3^{2n} possible different chunks, there must be two chunks, not further than 3^{2n} from each other, that are identical. That is $(T_i, T_{i+1}) = (T_j, T_{j+1})$ for some i and j between $i + 2$ and $i + 3^{2n}$. But this means that there is a 3-coloring assignment consisting of a two-way infinite repetition of $(T_i, T_{i+1}, \ldots, T_{j-1})$. We conclude that *if a periodic graph G*

is 3*-colorable, then it has a periodic 3-coloring assignment with period at most exponential in the number of nodes of one iteration.*

This crucial observation allows us to solve the problem in polynomial space: Using non-determinism we can guess valid assignments $T_1, T_2, \ldots$, always remembering the last two. After we guess T_i we check that all nodes in iteration $i-1$ are still properly colored. Once we have successfully guessed $T_{3^{2^n}+2}$ we accept: We know that there is a periodic 3-coloring assignment on G.

(b) *The* $\mathcal{PSPACE}$*-completeness proof:*

In order to show the completeness, we reduce from 3-coloring of periodic general graphs, which is known to be $\mathcal{PSPACE}$-complete ([16]). We adapt the construction used by [15] to prove the $\mathcal{NP}$-completeness of PLANAR-3-COLORING from 3-COLORING. See the full version [7] for a complete proof. □

Lemma 4. *The problem of deciding whether a given periodic planar graph* $G = (\widetilde{G}(V, E_0), E_+)$ *of maximal degree four is 3-colorable is* $\mathcal{PSPACE}$*-complete.*

Proof. (outline) We reduce planar 3-coloring of a periodic graph with a maximum degree 4 from periodic planar 3-coloring which was proved to be $\mathcal{PSPACE}$-complete in Lemma 3 adapting the same technique used to prove that the problem of 3-coloring an ordinary planar graph of maximum degree four it is $\mathcal{NP}$-complete of [15]. *For the full proof, see [7].* □

Lemma 5. *The problem of 3-Coloring a periodic planar graph with a given constant period 4-edge coloring is* $\mathcal{PSPACE}$*-complete.*

Proof. (outline) We use a transformation from the 3-coloring for planar periodic graphs with maximum degree four which was proved to be $\mathcal{PSPACE}$-complete in Lemma 4 adapting the reduction used in [15] to show $\mathcal{NP}$-hardness of 3-coloring of planar graphs with maximum degree four. *For the full proof, see [7].* □

Next, we prove the main Theorem using the above results. We will show that it is $\mathcal{PSPACE}$-complete to decide whether $X_{span}(G) \leq 9$ for a given periodic planar graph $G = (\widetilde{G}(V, E_0), E_+)$.

Theorem 1. *The problem of deciding whether a periodic planar graph* $G = (\widetilde{G}(V, E_0), E_+)$*, of maximum degree seven, whose graph* $\widetilde{G}$ *in its finite specification* $(\widetilde{G}, E_+)$ *is a planar bipartite graph, can be radiocolored using a span of size at most* 9*, is* $\mathcal{PSPACE}$*-complete.*

Proof. We prove that the min span radiocoloring for periodic planar graphs is $\mathcal{PSPACE}$-complete, by reducing it from 3-coloring of periodic planar graphs with a given constant period 4-edge coloring which was proved to be $\mathcal{PSPACE}$-complete in Lemma 5. Our reduction adapts the construction by Bodlaender et al in [2] to show $\mathcal{NP}$-hardness of min span RCP of ordinary planar graphs reducing it from the 3-coloring of planar graphs with a given 4-edge coloring. *For the full proof of the Theorem see [7].* □

As in [2], it is possible to generalize the result as follows:

Theorem 2. *Let $r \geq 8$ be an even integer. The problem of deciding whether a periodic planar graph $G = (\widetilde{G}(V, E_0), E_+)$ of maximal degree $r - 2$ can be radiocolored using a span of size at most r is $\mathcal{PSPACE}$-complete.*

4 An Efficient, Constant Ratio Approximation Algorithm for Min Span RCP for Periodic Planar Graphs

We present an efficient time, constant ratio approximation algorithm that approximates the min span radiocoloring problem for periodic planar graphs with the same ratio as the ratio obtained by the best known approximation algorithm for planar graphs (which we use as a subroutine for the finite specification), for the same problem.

4.1 The Modified Graph

Consider the following partition of the periodic graph G: group together every four consecutive iterations of the graph, call the $j-th$ such group $G_{group\ j}$. More specifically,

$$G_{group\ j} = \{G(i) \bigcup G(i+1) \bigcup G(i+2) \bigcup G(i+3)\},\ \forall\ i = \ldots, -7, -3, 1, 5, 9, \ldots$$

where $j = 1, 2, 3, 4, \ldots$, (respectively, i.e. $j = \lceil i/4 \rceil$). Consider the graph $G_{group\ j}$ of G. Denote the first graph of the group as $G_{(j)1}$ or G_1, the second as $G_{(j)2}$ or G_2 and so on until the fourth. The algorithm produces a new graph, called $G'_{group\ j}$ defined as follows: The graph has the same vertex and edge set as the graph $G_{group\ j}$ except from the following modifications on the first and the fourth graphs of group $G_{group\ j}$:

Consider an edge $uv \in E_{4+}$ of graph G_4. Recall that $u \in V_4$ and the vertex v belongs to the next iteration of G, that is $v \in V_{(j*4)+1}$. For each such edge uv of the graph G_4 we do the following:

- Delete edge uv.
- Add a new edge uv' connecting the vertex $u \in G_4$ to vertex v', where $v' \in V_1$ is the corresponding to v ($v \in V_{(j*4)+1}$) vertex in graph G_1. Recall that the graphs $G_1, V_{(j*4)+1}$ are isomorphic.
- Delete edge $u''v'$ of graph G_1, where $v' \in G_1$ and u'' is the corresponding to u vertex ($u \in V_4$) in iteration G_{j*4-1} of G.

An example of the graph obtained by a periodic graph is illustrated in Figure 2. The graph $G'_{group\ j}$ has two critical properties compared to the initial periodic planar graph G: (i) it has the same maximum degree as the initial graph G, i.e. $\Delta(G'_{group\ j}) = \Delta(G)$ and (ii) as the next Lemma proves, it is a planar graph.

Lemma 6. *The modified graph $G'_{group\ j}$ is a planar graph.*

Proof. See the full version of this paper [7]. □

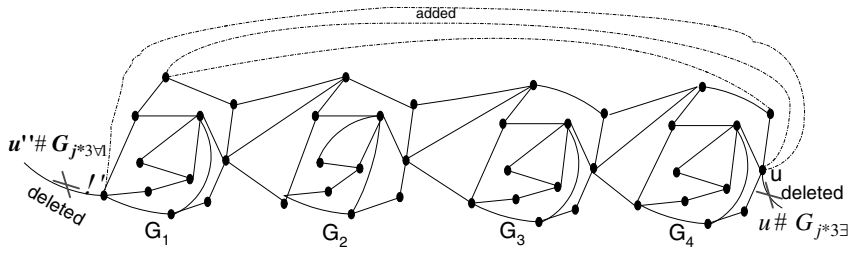


Fig. 2. The graph $G'_{group\ j}$ produced by the *Group j* of the periodic graph G

4.2 The Periodic Radiocoloring Partitioning Algorithm (PRPA)

The following definition is needed by the Algorithm. The definition uses the observation that the optimal span, S^*, of a radiocoloring of a graph G with maximum degree $\Delta(G)$ is clearly $S^* \geq \Delta(G)$.

Definition 7. RC Algorithm: *Let an RC Algorithm be any known min span radiocoloring polynomial time approximation algorithm for finite planar graphs with performance ratio R (when $\Delta(G)$ is used as a lower bound), i.e. if S^* is the optimal span and S_{RC} is the span obtained by the algorithm then there are constants $R > 1$ and b such that*

$$\Delta(G) \leq S^* \leq S_{RC} \leq R \cdot \Delta(G) + b$$

For example the algorithm of [9] is an *RC* algorithm with $R = 2$ and $b = 35$.

Algorithm PRPA:

1. Run an *RC* algorithm, on graph $G'_{group\ j}$.
   Let S_{RC} be the span obtained by *RC* on $G'_{group\ j}$.

2. For all $j = 1, 2, \ldots$ color the four graphs $G_{(j-1)*4+1}$, $G_{(j-1)*4+2}$, $G_{(j-1)*4+3}$, , $G_{(j-1)*4+4}$ of the group $G_{group\ j}$ as follows: Set the color of each vertex of graph $G_{(j-1)*4+k}$, $k = 1, 2, 3, 4$ to the color of its corresponding vertex, in V_k of $V(G'_{group\ j})$.
   Step 2 produces a radiocoloring of the whole periodic graph G with span S_{RC}.

Theorem 3. (**Correctness**) *The algorithm PRPA produces a radiocoloring of a periodic graph G.*

Proof. (outline) It can be proved that there is no conflict either between the colors of vertices inside a group $G_{group\ j}$ or between the colors of vertices in neighbour groups. *For the full proof, see [7].* □

Theorem 4. *The Algorithm PRPA runs in time $O(T(RC))$ and approximates the span within an asymptotic ratio of R, where $T(RC)$ is the time needed for the RC Algorithm to run on $G'_{group\ j}$.*

Proof. Let $\Delta(G)$ the maximum degree of the periodic graph G. Recall that the graph $G'_{group\ j}$ has maximum degree $\Delta(G'_{group\ j}) = \Delta(G)$. Denote by S the span of graph $G'_{group\ j}$ obtained by our algorithm and by $S^*(G'_{group\ j})$ the optimal span of $G'_{group\ j}$. Note that, (a) $S^* \leq S$.

(b) Also that, $S \leq R \cdot \Delta(G'_{group\ j}) + b$, by the definition of the *RC* Algorithm.

(c) Since, $\Delta(G'_{group\ j}) = \Delta(G)$, we get $S^* \leq S \leq R \cdot \Delta(G) + b$

Also, since $S^* \geq \Delta(G)$, we get that, $1 \leq \frac{S}{S^*} \leq R + \frac{b}{\Delta(G)}$

Finally, since $\Delta(G) \geq \Delta(\widetilde{G})$, we have $1 \leq \frac{S}{S^*} \leq R + \frac{b}{\Delta(\widetilde{G})}$.

Also, the algorithm, clearly, needs $O(T(RC))$ time, where $T(RC)$ is the time needed for algorithm *RC* to run on $G'_{group\ j}$. □

Remark. If the *RC* Algorithm is the algorithm in [9], then algorithm *PRPA* has $R = 2$ and $b = 35$ and runs in $O(n(\Delta(G_i) + \sigma))$ time, where $n = |V_i|$, and σ is the number of edges with which each G_i is connected to G_{i+1} and $\Delta(G_i)$ is the maximum degree of G_i.

The *min order RCP* is the problem of finding a radiocoloring assignment that uses a minimum number of distinct frequencies. We conjecture that min order is $\mathcal{PSPACE}$-complete for periodic planar graphs. Note that the following modification of *PRPA* works also for min order RCP: In Step 1, apply a known min order radiocoloring algorithm (instead of a min span RC algorithm), e.g. [5,1], for the radiocoloring of $G'_{group\ j}$. The same analysis gives that the modified *PRPA* algorithm approximates the min order RCP of G within an asymptotic approximation ratio of 2.

Lemma 7. *Any approximation algorithm for the min span RCP of a finite planar graph G, that achieves a span of at most $\alpha\Delta(G) + constant$, for any α, can be used as a subroutine in algorithm PRPA to produce an approximation algorithm for min span periodic planar RCP of asymptotic ratio α.*

Proof. *Using the same arguments as in Theorem 4.* □

References

1. Geir Agnarsson, Magnus M.Halldórsson: Coloring Powers of Planar Graphs. ACM Symposium on Discrete Algorithms (SODA) (2000).
2. Bodlaender, H.L., T. Kloks, R.B. Tan and J. van Leeuwen: Approximations for λ-coloring of graphs. In H. Reichel and S. Tison (Eds.), STACS 2000, Proc. 17th Annual Symp. on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science Vol. 1770, Springer-Verlag, Berlin, 2000, pp. 395-406.
3. E. Cohen and N. Meggido: Strongly polynomial-time and $\mathcal{NC}$ algorithms for detecting cycles in dynamic graphs. J. ACM, 40, pp. 791-830 (1993).

4. D. Fotakis, G. Pantziou, G. Pentaris and P. Spirakis: Frequency Assignment in Mobile and Radio Networks. Networks in Distributed Computing, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 45, American Mathematical Society (1999) 73-90.
5. D.A. Fotakis, S.E. Nikoletseas, V.G. Papadopoulou and P.G. Spirakis: $\mathcal{NP}$-completeness Results and Efficient Approximations for Radiocoloring in Planar Graphs. 25th International Symposium on Mathematical Foundations of Computer Science (MFCS), LNCS 1893, pp 363-372, 2000.
6. D.A. Fotakis, S.E. Nikoletseas, V.G. Papadopoulou and P.G. Spirakis: Hardness Results and Efficient Approximations for Frequency Assignment Problems: Radio Labelling and Radio Coloring. Computing and Informatics (CAI) Journal, Vol 20, pp. 121-180 (2001).
7. D.A Fotakis, S.E. Nikoletseas, V.G. Papadopoulou and P.G. Spirakis: Radiocolorings in Periodic Planar Graphs: PSPACE-Completeness and Efficient Approximations for the Optimal Range of Frequencies, ALCOMFT-TR-01-107, (May 2001) URL: $http://www.brics.dk/cgi-alcomft/db?state=reports$.
8. F. Höfting and E. Wanke: Minimum cost paths in periodic graphs. SIAM J. of Comput. 24 pp. 1051-1067 (1995).
9. J. Van D. Heuvel and S. McGuiness: Colouring the Square of a Planar Graph. CDAM Research Report Series, July (1999).
10. H. Marathe, H. Hunt III, R. Stearns and V. Radhakrishnan: Approximation Algorithm for PSPACE-HARD hierarchically specified and periodically specified problems. Siam J. Comp. Vol. 27,, No 5, pp. 1237-1261, Oct (1998).
11. H. Marathe, H. Hunt III, R. Stearns and V. Radhakrishnan: Complexity of hierachically and 1-dimensioned periodically specified problems. DIMACS Workshop on Satisfiability Problem: Theory and Applications (1996).
12. T. Nishizeki, N. Chiba: Planar graphs: Theory and algorithms. North-Holland Mathematics Studies 140, Annals of Discrete Mathematics (32) (1988).
13. J. B. Orlin: The Complexity of Dynamic/Periodic Languages and Optimization Problems 13th Annual ACM Sympocium on Theory of Computing (STOC), pp. 218-227 (1981).
14. J. B. Orlin: Some problems on dynamic/periodic graphs. Progress on Combinatorial Optimization, pp. 273-293 (1984).
15. M. R. Garey, D. S. Johnson: Computers and Intractability: A guide to the Theory of NP-completeness. W. H./ Freeman and Company (1979).
16. C.H. Papadimitriou. Computational Complexity. Addisson-Wesley Publishing Company. (1994).
17. E. Wanke. Paths and Cycles in Finite Periodic Graphs: In the Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science (MFCS), LNCS 711, Springer-Verlag, pp 751-760 (1993).

Completely Independent Spanning Trees in Maximal Planar Graphs

Toru Hasunuma

Department of Computer Science
The University of Electro-Communications
1–5–1 Chofugaoka, Chofu, Tokyo 182–8585 Japan
hasunuma@cs.uec.ac.jp

Abstract. Let G be a graph. Let $T_1, T_2, \ldots, T_k$ be spanning trees in G. If for any two vertices u, v in G, the paths from u to v in $T_1, T_2, \ldots, T_k$ are pairwise openly disjoint, then we say that $T_1, T_2, \ldots, T_k$ are completely independent spanning trees in G. In this paper, we show that there are two completely independent spanning trees in any 4-connected maximal planar graph. Our proof induces a linear-time algorithm for finding such trees. Besides, we show that given a graph G, the problem of deciding whether there exist two completely independent spanning trees in G is NP-complete.

1 Introduction

1.1 Independent Spanning Trees

Let G be a graph. The vertex set and the edge set of G are denoted by $V(G)$ and $E(G)$, respectively. A subgraph H of G is called spanning if $V(H) = V(G)$. Let P_1 and P_2 be paths from a vertex x to a vertex y in G. If $E(P_1) \cap E(P_2) = \emptyset$ and $V(P_1) \cap V(P_2) = \{x, y\}$, then we say that P_1 and P_2 are *openly disjoint*. Let T' and T'' be trees rooted at a vertex r in G. If for any vertex $v(\neq r) \in V(T') \cap V(T'')$, the paths from r to v in T' and in T'' are openly disjoint, then we say that T' and T'' are *independent*. Let $T_1, T_2, \ldots, T_k$ be spanning trees rooted at r in G. If $T_1, T_2, \ldots, T_k$ are pairwise independent, then we say that $T_1, T_2, \ldots, T_k$ are *independent spanning trees rooted at* r in G. A graph H is called k-*connected* if the removal of any $k-1$ vertices from H results a connected graph with at least two vertices. The following conjecture is a central topic on independent spanning trees.

Conjecture 1. Let G be a k-connected graph. Then there are k independent spanning trees rooted at any vertex in G.

This conjecture was shown to be true for $k \leq 3$, ([10],[1],[20]). Also, Huck proved that the conjecture holds if we restrict ourselves to planar graphs ([5],[7]). The conjecture remains open for general graphs when $k \geq 4$.

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 235–245, 2002.

Three variations of this conjecture are obtained by replacing "openly disjoint" (in the definition of independent) and "k-connected" with "edge-disjoint" and "k-edge-connected", respectively, or considering directed graphs instead of graphs. The directed edge version of this conjecture indeed holds for any $k \geq 1$, which was shown by Edmonds [2]. The directed version originally conjectured by Frank was shown to be true for $k \leq 2$ [19], and for line digraphs [9], but was disproved for general digraphs when $k \geq 3$ [6].

Independent spanning trees have been studied not only from the theoretical point of view but also from the practical point of view. Independent spanning trees have applications to fault-tolerant broadcasting problems in interconnection networks. Broadcasting is a process that send a message originated from a processor to all other processors in a parallel computer, and can be modeled by a spanning tree in the interconnection network. Suppose that we have k independent spanning trees rooted at r in G. Using these spanning trees to broadcast the same message from r, even if any $(k-1)$ processors (different from r) fail, we can correctly broadcast the message to all fault-free processors. Motivated by this application, independent spanning trees in several graph (or digraph) classes related to interconnection networks have been studied (de Bruijn and Kautz digraphs [4] [9], product graphs [15], chordal rings [11]).

1.2 Completely Independent Spanning Trees

In the application of independent spanning trees to fault-tolerant broadcasting, we have to reconstruct independent spanning trees when the root is changed with another vertex. If one set of spanning trees is always a set of independent spanning trees rooted at any given vertex, then we do not have to reconstruct independent spanning trees. Motivated by this point of view, the notion of completely independent spanning trees was introduced in [8]. Let T' and T'' be trees in G. If for any two vertices $u, v \in V(T') \cap V(T'')$, the paths from u to v in T' and in T'' are openly disjoint, then we say that T' and T'' are *completely independent*. Let $T_1, T_2, \ldots, T_k$ be spanning trees in G. If $T_1, T_2, \ldots, T_k$ are pairwise completely independent, then we say that $T_1, T_2, \ldots, T_k$ are *completely independent spanning trees* in G.

It has been shown in [8] that there are k completely independent spanning trees in the underlying graph of a k-connected line digraph. Also, the notion of completely independent spanning trees was characterized in as follows.

Theorem 1. [8] *Let $T_1, T_2, \ldots, T_k$ be spanning trees in a graph G. Then, $T_1, T_2, \ldots, T_k$ are completely independent if and only if $T_1, T_2, \ldots, T_k$ are edge-disjoint and for any vertex v of G, there is at most one spanning tree T_i such that the degree of v in T_i is greater than one.*

From this characterization, we can see that for any given set S of k completely independent spanning trees, the removal of any one vertex can make at most one spanning tree disconnected. Thus, at least one spanning tree in S preserves its connectedness in the case that any $(k-1)$ vertices are removed. By this property,

completely independent spanning trees may be useful not only fault-tolerant broadcasting but also general fault-tolerant network issues.

Completely independent spanning trees are edge-disjoint spanning trees, although independent spanning trees are not always edge-disjoint. On edge-disjoint spanning trees, Nash-Williams [14] showed that there are k edge-disjoint spanning trees in any $2k$-edge-connected graph. Analogously to this fact, we pose the following statement as a conjecture.

Conjecture 2. There are k completely independent spanning trees in any $2k$-connected graph.

As special cases, we can check that the complete bipartite graph $K_{2k-1,2k-1}$ cannot have k completely independent spanning trees, but $K_{2k,2k}$ has k completely independent spanning trees. In this sence, the connectivity condition in the conjecture cannot be weakened. Also, we can easily checked that the complete graph K_{2k} has k completely independent spanning trees. Thus, the conjecture is a sufficient condition for the existence of completely independent spanning trees.

1.3 Our Results

In this paper, we show that Conjecture 2 holds for maximal planar graphs, i.e., there are two completely independent spanning trees in any 4-connected maximal planar graph. From our proof, a linear-time algorithm for finding such trees is obtained. Besides, we show that given a graph G, the problem of deciding whether there exist two completely independent spanning trees in G is NP-complete.

This paper is organized as follows. In Section 2, basic definitions and terminology are given. In Section 3, we show that Conjecture 2 holds for maximal planar graphs. In Section 4, we present an NP-completeness result on completely independent spanning trees. Finally, we conclude the paper in Section 5.

2 Preliminaries

A *planar graph* is a graph which can be embedded in the plane without crossing of edges. A planar graph is *maximal* if we cannot add a new edge to it while preserving its planarity. By a planar embedding of a planar graph, the plane is cut into *faces*. The *outer face* is a unique unbounded face.

In this paper, we treat maximal planar graphs and assume that a maximal planar graph is given along with a fixed planar embedding. A *triangle* is a cycle of length 3. Let G be a maximal planar graph. Every face of G is surrounded by a triangle. A cycle of G divides the plane into its interior and exterior portions. A *region* of a cycle is the portion inside the cycle. A region may contain vertices and edges. A face which is not the outer face is a minimal region. Let R be a region of a cycle of G. Then the cycle is called the *boundary* of R, and denoted by $C(R)$.

Let $S \subseteq V(G)$ and $U \subseteq E(G)$. The subgraphs of G induced by S and U are denoted by $\langle S \rangle_G$ and $\langle U \rangle_G$, respectively. Let $v \in V(G)$. We denote by $N_G(v)$

the set of vertices adjacent to v in G. Also, the set $N_G(v) \cup \{v\}$ is denoted by $N_G^+(v)$.

Let C be a cycle in a maximal planar graph G. An edge inside C joining non-adjacent vertices in C is called a chord of C. Let $w \in V(C)$. The set of vertices inside C which are adjacent to w is denoted by $N_C^{in}(w)$. (Note that $N_C^{in}(w) \cap V(C) = \emptyset$.) Let $x \in V(G)$. From the property of a maximal planar graph, $\langle N_G(x) \rangle_G$ contains a cycle having all vertices in $N_G(x)$ following the clockwise order with respect to a planar embedding, and the cycle is denoted by $C_G(x)$.

A *cut-vertex* of a graph H is a vertex such that removing it from H results a disconnected graph. A connected graph with at least two vertices and no cut-vertex is called a block. A *block* of H is a subgraph of H, which is itself a block and maximal with respect to that property. A *cyclic block* is a block with at least three vertices.

3 Two Completely Independent Spanning Trees in Four-Connected Maximal Planar Graphs

In this section, we show that there are two completely independent spanning trees in any 4-connected maximal planar graph.

Consider a coloring of the vertices and edges in G using k colors such that every set of edges with the same color induces a spanning tree and every internal vertex in each spanning tree has the same color as the edges. From Theorem 1, the set of spanning trees induced by each color is a set of completely independent spanning trees. In what follows, we use blue and red for such a coloring in order to construct two completely independent spanning trees.

We start with the following lemma, which is used as the basis for our inductive proof.

Lemma 1. *Let G be a maximal planar graph with at least four vertices. Let $v \in V(G)$. Then there are two completely independent spanning trees in $\langle N_G^+(v) \rangle_G$.*

Proof. Let $u, w \in N_G(v)$ such that $(u, w) \in E(G)$. We color v and u red, and color all vertices in $N_G(v) \setminus \{u\}$ blue. Also, we color the edge (u, w) and all edges in $\{(v, x) \mid x \in N_G(v) \setminus \{w\}\}$ red, and color the edge (v, w) and all edges in $E(C_G(v)) \setminus \{(u, w)\}$ blue. It is easily checked that the trees induced by the red edges and by the blue edges are completely independent spanning trees in $\langle N_G^+(v) \rangle_G$.

Here, we introduce definitions and terminology used in the proof.

Definition 1. *Let x be a vertex inside a chordless cycle C. If $|N_G(x) \cap V(C)| \geq 2$, then x is called a clipping vertex of C. For a clipping vertex x of C, let $N_G(x, C)$ denote the set $N_G(x) \cap V(C)$.*

By a clipping vertex x and the edges joining x and a vertex in $N_G(x, C)$, the region of C is cut into several regions called *clipped regions* of x. For a clipped

region R of x, a vertex in the boundary of R which is neither x nor a vertex in $N_G(x, C)$ is called a *proper boundary vertex* of R. If v is a proper boundary vertex of a clipped region R of x, then we say that x is a *clipping vertex with respect to* v, and R is a *clipped region with respect to* v. The set of clipping vertices with respect to v is denoted by $V_{cl}(v, C)$. Also, the set of clipped regions with respect to v is denoted by $R_{cl}(v, C)$.

Definition 2. *Let G be a maximal planar graph. Let C be a chordless cycle in G and $v \in V(C)$. The inscribed graph with respect to v and C denoted by $H_G(v, C)$ is defined as follows.*

$$H_G(v, C) = \langle \cup_{w \in V(C) \setminus \{v\}} N_C^{in}(w) \cup \{v\} \rangle_G.$$

A boundary edge of $H_G(v, C)$ is an edge in

$$E(H_G(v, C)) \cap (\cup_{w \in V(C) \setminus \{v\}} E(C_G(w))).$$

Now, we consider the following situation. We have already obtained two completely independent trees outside a cycle C which contain all vertices in $V(C)$ such that all vertices in $V(C)$ have the same color except for one vertex called *the entering vertex*. Several vertices in $V(C)$ different from the entering vertex are not allowed to change their color hereafter. Such vertices in $V(C)$ is called *forbidden vertices*. We try to augment the two completely independent trees to have all vertices inside C while preserving their completely independentness. We will show that we can correctly augment the trees if the configuration of the entering vertex and forbidden vertices in C is one of the following 5 types. (Note that in the description of each type, "u is adjacent to v" means that u is adjacent to v in C, i.e., $(u, v) \in E(C)$.)

- Type 1: There is exactly one forbidden vertex. The entering vertex is adjacent to the forbidden vertex.
- Type 2: The number of forbidden vertices is two. The two forbidden vertices are adjacent. The entering vertex is adjacent to one of the forbidden vertices.
- Type 3: The number of forbidden vertices is two. The two forbidden vertices are not adjacent. The entering vertex is adjacent to one of the forbidden vertices.
- Type 4: C has no chord. The number of forbidden vertices is three. The three forbidden vertices are consecutive. The entering vertex is adjacent to one of the forbidden vertices.
- Type 5: C has no chord. The number of forbidden vertices is three. There are three forbidden vertices v_1, v_2, v_3 such that v_1 and v_2 are adjacent and v_3 is not adjacent to v_1 and v_2. The entering vertex is adjacent to v_3.

Let v be a vertex in the boundary of the outer face of G. Also, let $u, w \in N_G(v)$ such that $(u, w) \in E(G)$. According to the proof of Lemma 2, we can construct two completely independent spanning trees in $\langle N_G^+(v) \rangle_G$. For the cycle $C_G(v)$, let u be the entering vertex and w a forbidden vertex. Then, the configuration is Type 1. Therefore, by proving the next lemma, our main theorem is obtained.

Lemma 2. *Let G be a 4-connected maximal planar graph. Let C be a cycle in G. Suppose that there are two completely independent trees outside C which contain all vertices in $V(C)$ such that all vertices in C except for the entering vertex have the same color, and the configuration of the entering vertex and forbidden vertices is one of the five types defined above. Then the completely independent trees can be augmented to have all vertices inside C while preserving their completely independentness.*

Proof. We show this lemma by induction on the number of vertices inside C. In this proof, we assume that all vertices in C except for the entering vertex have blue color (thus the entering vertex has red color). Let v_e be the entering vertex.

The basis (i.e., the case that there is no vertex inside C) holds vacuously. Suppose that there is a vertex inside C.

Case 1: C has no chord.

We first construct the inscribed graph $H = H_G(v_e, C)$. Then we color all vertices in $V(H) \setminus \{v_e\}$ red. Also, we color all boundary edges in H red. From the definition of H, for every vertex w in $V(H) \setminus \{v_e\}$, there is at least one edge joining w and a vertex in $V(C) \setminus \{v_e\}$. For each vertex $v \in V(H) \setminus \{v_e\}$, we select one edge joining v and a vertex in $V(C) \setminus \{v_e\}$, and color it blue. By this coloring, the graph induced by the blue edges is a tree. On the other hand, the graph induced by the red edges may not be a tree but a connected graph (thus, we can obtain a tree as a subgraph). Every vertex in $V(H) \setminus \{v_e\}$ is colored red and is an end-vertex of the blue tree. Hence, we can correctly augment the trees outside C to have the vertices of H while preserving their completely independentness. If H is a tree, then the augmented trees have all vertices inside C.

Now suppose that H is not a tree. Then we consider the block-cut-vertex tree $\mathcal{T}$ of H. The vertex set of $\mathcal{T}$ consists of cyclic blocks, cut-vertices and end-vertices in H. For $X, Y \in V(\mathcal{T})$, there is an edge between X and Y iff X is a cyclic block and Y is a cut-vertex contained in X, or neither X nor Y is a cyclic block such that X and Y are adjacent in H. If v_e is an end-vertex in H, then let $Z = v_e$. Otherwise, let Z be the cyclic block containing v_e. We regard Z as the root of $\mathcal{T}$.

For each cyclic block, we apply the inductive hypothesis. Before applying such inductive augmentation, we do the following manipulation. For each forbidden vertex v_f in C and for each $x \in V_{cl}(v_f, C)$, if v_e is not contained in the boundary of the clipped region of x with respect to v_f, then we set x as a forbidden vertex. There are four cases with respect to this manipulation.

Case 1.1: The configuration is Type 1. Since the forbidden vertex v_f is adjacent to v_e in C, there is no clipped region in $R_{cl}(v_f, C)$ such that v_e is not contained in the boundary. Thus, we do nothing in the manipulation in this case.

Case 1.2: The configuration is Type 2 or Type 3. Similarly to the case 1.1, we do nothing with respect to the forbidden vertex adjacent to v_e. Thus, it is sufficient to try the manipulation only for the forbidden vertex not adjacent to v_e.

Case 1.3: The configuration is Type 4. There are two adjacent forbidden vertices that are not adjacent to v_e. For two adjacent vertices w_1, w_2 in C,

$R_{cl}(w_1, C) \subseteq R_{cl}(w_2, C)$, or $R_{cl}(w_2, C) \subseteq R_{cl}(w_1, C)$. Therefore it is sufficient to try the manipulation only for one forbidden vertex.

Case 1.4: The configuration is Type 5. In this case, the entering vertex is adjacent to the isolated forbidden vertex. Similarly to the case 1.3, it is sufficient to try the manipulation only for one forbidden vertex.

Therefore, in any type of configuration, we set new forbidden vertices in H with respect to at most one forbidden vertex in C. Hence, the new forbidden vertices in H are on the same path from the root in $\mathcal{T}$. As a result, in any cyclic block of H, at most two forbidden vertices are newly set. In particular, if a cyclic block has two forbidden vertices, then one forbidden vertex is its parent cut vertex in $\mathcal{T}$.

Next, for each vertex X in $\mathcal{T}$, we do the following manipulation from the root (in depth-first or breadth-first order). If X is either an end-vertex or a cut vertex, then we do nothing. Suppose that X is a cyclic block of H. Let $C(X)$ denote the cycle surrounding X. If X is the root, then we set v_e as a forbidden vertex in $C(X)$. Otherwise, we set the parent cut vertex as a forbidden vertex in $C(X)$. (Such a vertex may have already been set as a forbidden vertex by the former manipulation.) Then we select a vertex in $C(X)$ adjacent to a forbidden vertex as the entering vertex and recolor it blue, and uncolor one edge in $C(X)$ incident to it. Since all edges in $C(X)$ have been colored red, uncoloring one edge in the cycle does not make the graph induced by the red edges disconnected. Since there are at most two forbidden vertices in $C(X)$, the configuration of the entering vertex and forbidden vertices in $C(X)$ is one of Types 1,2,3. Hence, by the induction hypothesis, we can correctly augment the two completely independent trees to have all vertices inside $C(X)$.

After we have done such augmentation for X, if a child cut vertex Y of X in $\mathcal{T}$ have been recolored, then we consider the clipped regions $R_1, R_2, \ldots, R_l$ of Y which do not contain v_e as a boundary vertex. We set Y and all vertices in $N_G(Y, C)$ ($N_G(Y, C) \setminus \{v_e\}$ if $v_e \in N_G(Y, C)$) as forbidden vertices. Then, in each $C(R_i)$, the number of forbidden vertices is three such that the forbidden vertices are consecutive. Note that if there was a forbidden vertex which is a proper boundary vertex of R_i, then Y must have been set as a forbidden vertex and cannot be recolored. Also note that $C(R_i)$ has no chord. Hence, unless $|V(C(R_i))| = 3$, we can select a proper boundary vertex of R_i as the entering vertex so that the configuration of the entering vertex and forbidden vertices in $C(R_i)$ becomes Type 4, and by the induction hypothesis, we can augment the two completely independent trees to have all vertices inside $C(R_i)$. If $|V(C(R_i))| = 3$, then there is no vertex inside $C(R_i)$ (i.e., we do not have to do anything), since G is 4-connected. After we have done such augmentation for $C(R_1), C(R_2), \ldots, C(R_l)$, we delete the subtree rooted at Y from $\mathcal{T}$.

Case 2: C has a chord.

In this case, we consider the block-chord tree $\mathcal{S}$. The vertex set of $\mathcal{S}$ consists of chordless subcycles and chords in $\langle V(C)\rangle_G$. For $X, Y \in V(\mathcal{S})$, there is an edge between X and Y iff X is a chordless subcycle and Y is a chord contained in X.

Let Z be a chordless subcycle which contains the entering vertex and a forbidden vertex. (Note that at least one forbidden vertex is adjacent to the entering

vertex in C.) We regard Z as the root of $\mathcal{S}$ and for each vertex X in $\mathcal{S}$, we do the following manipulation from the root (in depth-first or breadth-first order). If X is a chord, then we do nothing. Suppose that X is a chordless subcycle. Also, suppose that X is the root. Then the configuration of the entering vertex and forbidden vertices is one of Types 1,2, and 3. Similarly to Case 1, we can correctly augment the completely independent trees. Suppose that X is not the root. Let Y be the parent vertex of X in $\mathcal{S}$. That is, Y is a chord contained in X. Since the configuration type of C is one of Types 1,2,3, the number of forbidden vertices in C is at most two. Since there is at least one forbidden vertex in Z, there is at most one forbidden vertex in X except for the vertices of Y.

Case 2.1: One vertex in Y has blue color, and the other vertex in Y has red color. (Note that we do not apply any manipulation for a chord, however, since a chord is contained in a subcycle, the vertices of the chord may be recolored in the manipulation for the subcycle.) In this case, we treat the vertex in Y with red color as the entering vertex and the other vertex in Y with blue color as a forbidden vertex. Thus, the configuration in X is one of Types 1,2 and 3.

Case 2.2: Both vertices in Y have blue color. In this case, we theat both the vertices as forbidden vertices. Then, we select the entering vertex so that the configuration type becomes one of Type 2,4, and 5.

It does not happen that both vertices in Y have red color. In Case 1, for each cycle $C(R_i)$ to which the induction hypothesis is applied, the entering vertex is selected from proper boundary vertices. Thus, after we have done the manipulation for a chordless cycle, there is no two consecutive vertices with red color.

Therefore, in any case, we can correctly augment the trees to have all vertices inside C.

From Lemmas 2 and 5, the following theorem is obtained.

Theorem 2. *There are two completely independent spanning trees in any 4-connected maximal planar graph.*

From the proofs in Lemmas 2 and 5, an algorithm for finding two completely independent spanning trees is induced. The algorithm can be implemented to run in linear-time. We omit the details of such an implementation.

4 NP-Completeness Result

In this section, we present an NP-completeness result on completely independent spanning trees.

Theorem 3. *Given a graph G, the problem of deciding whether there exist two completely independent spanning trees in G is NP-complete.*

Proof. We use a reduction from NAE-3SAT (Not-All-Equal 3SAT) [3]; Instance: a set $\mathcal{V}$ of variables, a collection $\mathcal{C}$ of clauses over $\mathcal{V}$ such that each clause $C \in \mathcal{C}$

has 3 literals. Question: Is there a truth assignment for $\mathcal{V}$ such that each clause in $\mathcal{C}$ has at least one true literal and at least one false literal?

Let K_4 denote the complete graph with four vertices. Let DK_4 denote a graph obtained from two copies of K_4 by identifying one edge in one copy and one edge in the other copy (with their two vertices). Two vertices of the identified edge in DK_4 are called *side vertices*. For each variable x, we prepare a graph $G_x \cong DK_4$ and let $V(G_x) = \{v_{x,i} \mid 1 \leq i \leq 6\}$ such that $\langle\{v_{x,i} \mid 1 \leq i \leq 4\}\rangle \cong \langle\{v_{x,i} \mid 3 \leq i \leq 6\}\rangle \cong K_4$. Thus, $v_{x,3}$ and $v_{x,4}$ are the side vertices of G_x. For each clause $C = \{a, b, c\}$, we prepare a graph $G_C \cong K_4$ and let $V(G_C) = \{v_{C,a}, v_{C,b}, v_{C,c}, v_C^*\}$.

Now we construct the following graph H for a given instance $(\mathcal{V}, \mathcal{C})$ of NAE3-SAT, where $\mathcal{V}$ and $\mathcal{C}$ are the set of variables and the set of clauses, respectively.

$$\begin{cases} V(H) = (\cup_{C\in\mathcal{C}} V(G_C)) \cup (\cup_{x\in\mathcal{V}} V(G_x)) \cup \{v_B, v_R\}, \\ E(H) = (\cup_{C\in\mathcal{C}} E(G_C)) \cup (\cup_{x\in\mathcal{V}} E(G_x)) \\ \qquad \cup\{\{v_{C,a}, v_{x,3}\} \mid C \in \mathcal{C}, a \in C, a = x\} \\ \qquad \cup\{\{v_{C,b}, v_{x,4}\} \mid C \in \mathcal{C}, b \in C, b = \bar{x}\} \\ \qquad \cup\{\{v_{x,3}, v_B\}, \{v_{x,4}, v_B\}, \{v_{x,3}, v_R\}, \{v_{x,4}, v_R\} \mid x \in \mathcal{V}\}. \end{cases}$$

We show that $(\mathcal{V}, \mathcal{C})$ is NAE satisfiable iff there are two completely independent spanning trees in H.

Suppose that there is a truth assignment t for $\mathcal{V}$ such that each clause in $\mathcal{C}$ has at least one true literal and at least one false literal. According to the truth assignment t, we construct two trees T_B, T_R as follows. We color all edges in T_B blue and all edges in T_R red.

$$\begin{cases} V(T_B) = V(T_R) = V(H) \setminus \{v_C^* \mid C \in \mathcal{C}\}, \\ E(T_B) = (\cup_{x\in\mathcal{V}, t(x)=1}(E_1(G_x) \cup \{\{v_{x,3}, v_{C,a}\} \mid a \in C \in \mathcal{C}, a = x\} \cup \{v_{x,3}, v_B\}) \\ \qquad \cup(\cup_{x\in\mathcal{V}, t(x)=0}(E_0(G_x) \cup \{\{v_{x,4}, v_{C,a}\} \mid a \in C \in \mathcal{C}, a = \bar{x}\} \cup \{v_{x,4}, v_B\}) \\ \qquad \cup\{\{v_{x^*,i}, v_R\} \mid (t(x^*) = 1 \text{ and } i = 3) \text{ or } (t(x^*) = 0 \text{ and } i = 4)\}, \\ E(T_R) = (\cup_{x\in\mathcal{V}, t(x)=1}(E_0(G_x) \cup \{\{v_{x,4}, v_{C,a}\} \mid a \in C \in \mathcal{C}, a = \bar{x}\} \cup \{v_{x,4}, v_R\}) \\ \qquad \cup(\cup_{x\in\mathcal{V}, t(x)=0}(E_1(G_x) \cup \{\{v_{x,3}, v_{C,a}\} \mid a \in C \in \mathcal{C}, a = x\} \cup \{v_{x,3}, v_R\}) \\ \qquad \cup\{\{v_{x^*,i}, v_B\} \mid (t(x^*) = 1 \text{ and } i = 4) \text{ or } (t(x^*) = 0 \text{ and } i = 3)\}, \end{cases}$$

where x^* is any fixed variable in $\mathcal{V}$, and

$$E_1(G_x) = \{\{v_{x,1}, v_{x,2}\}, \{v_{x,2}, v_{x,3}\}, \{v_{x,3}, v_{x,4}\}, \{v_{x,3}, v_{x,5}\}, \{v_{x,5}, v_{x,6}\}\},$$
$$E_0(G_x) = \{\{v_{x,1}, v_{x,3}\}, \{v_{x,1}, v_{x,4}\}, \{v_{x,2}, v_{x,4}\}, \{v_{x,4}, v_{x,5}\}, \{v_{x,4}, v_{x,6}\}\}.$$

Note that $\langle E_1(G_x)\rangle$ and $\langle E_0(G_x)\rangle$ are completely independent. Also, it is easily checked that T_B and T_R are completely independent. For each clause C, there are three edges joining a vertex in G_C and a vertex in a graph associated by a variable. For these three edges, at least one edge is colored blue and at least one edge is colored red, since t is a truth assignment for $\mathcal{V}$ such that each clause in $\mathcal{C}$ has at least one true literal and at least one false literal. Without loss of generality, we can assume that $v_{C,a}$ and $v_{C,b}$ are incident to blue edges, and $v_{C,c}$ is incident to a red edge. Then, we color $\{v_{C,a}, v_{C,b}\}$, $\{v_C^*, v_{C,a}\}$, $\{v_{C,b}, v_{C,c}\}$

blue, and $\{v_{C,c}, v_{C,a}\}$,$\{v_{C,c}, v_C^*\}$,$\{v_C^*, v_{C,b}\}$ red. By the similar coloring for each clause graph, T_B, T_R can be augmented to spanning trees while preserving their completely independentness.

Conversely, suppose that there are completely independent spanning trees T'_B, T'_R in H. We color all edges and all internal vertices in T'_B and T'_R by blue and red, respectively. Now, consider G_x for any variable x. It does not happen that both side vertices of G_x have the same color. If both side vertices have the same color, then the tree with the other color cannot be connected. By the similar reason, each side vertex must be colored.

For any clause graph G_C, where $C = \{a, b, c\}$, there is a blue path from v_C^* to v_B, and there is a red path from v_C^* to v_R. This means that there exists a vertex $v_{C,B}$ in $\{v_{C,a}, v_{C,b}, v_{C,c}\}$ such that it has blue color and is adjacent to a blue side vertex in a variable graph, and also there exists a vertex $v_{C,R}$ in $\{v_{C,a}, v_{C,b}, v_{C,c}\}$ such that it has red color and is adjacent to a red side vertex in a variable graph.

Now we define an assignment t' as follows. If $v_{x,3}$ is colored blue then let $t'(x) = 1$, otherwise $t'(x) = 0$. Then $v_{C,B}$ corresponds to a true literal, and $v_{C,R}$ corresponds to a false literal. Therefore, t' is a truth assignment for $\mathcal{V}$ such that each clause in $\mathcal{C}$ has at least one true literal and at least one false literal.

5 Conclusion

In this paper, we have shown that there are two completely independent spanning trees in any 4-connected maximal planar graph. Our proof induces a linear-time algorithm for finding such trees. Also, we have proved that the problem of deciding whether there exist two completely independent spanning trees in a given graph is NP-complete.

It remains unknown whether our result can be generalized to 4-connected planar graphs.

References

1. J. Cheriyan and S.N. Maheshwari, Finding nonseparating induced cycles and independent spanning trees in 3-connected graphs. *J. Algorithms* 9 (1988) 507–537.
2. J. Edmonds, Submodular functions, matroids and certain polyhedra, in: R. Guy et al. (Eds.) Combinatorial Structures and Their Applications (Gordon and Breach, New York, 1969) 69–87.
3. M.R. Garey and D.S. Johnson, Computers and Intractability – A Guide to the Theory of NP-Completeness, Freeman, San Francisco, CA 1979.
4. Z. Ge and S.L. Hakimi, Disjoint rooted spanning trees with small depths in de Bruijn and Kautz graphs, *SIAM J. Comput.* 26 (1997) 79–92.
5. A. Huck, Independent trees in graphs. *Graphs and Combin.* 10 (1994) 29–45.
6. A. Huck, Disproof of a conjecture about independent spanning trees in k-connected directed graphs, *J. Graph Theory* 20 (1995) 235–239.
7. A. Huck, Independent trees in planar graphs, *Graphs and Combin.* 15 (1999) 29–77.

8. T. Hasunuma, Completely independent spanning trees in the underlying graph of a line digraph, *Discrete Math.* 234 (2001) 149–157.
9. T. Hasunuma and H. Nagamochi, Independent spanning trees with small depths in iterated line digraphs, *Discrete Applied Math.* 110 (2001) 189–211.
10. A. Itai and M. Rodeh, The multi-tree approach to reliability in distributed networks, *Inform. and Comput.* 79 (1988) 43–59.
11. Y. Iwasaki, Y. Kajiwara, K. Obokata, and Y. Igarashi, Independent spanning trees of chordal rings, *Inform. Process. Lett.* 69 (1999) 155–160.
12. K. Miura, D. Takahashi, S. Nakano, and T. Nishizeki, A linear-time algorithm to find four independent spanning trees in four-connected planar graphs, Proc. Graph-Theoretic Concepts in Computer Science, WG'98, LNCS 1517, pp. 310–323 (1998).
13. S. Nagai and S. Nakano, A linear-time algorithm to find independent spanning trees in maximal planar graphs, Proc. Graph-Theoretic Concepts in Computer Science, WG00, LNCS 1928, pp. 290–301 (2000).
14. C.St.J.A. Nash-Williams, Edge-disjoint spanning trees in finite graphs, *J. London Math. Soc.* 36 (1961) 445–450.
15. K. Obokata, Y. Iwasaki, F. Bao, and Y. Igarashi, Independent spanning trees in product graphs and their construction, *IEICE Trans.* E79–A (1996) 1894–1903.
16. Y. Perl and Y. Shiloach, Finding two disjoint paths between two pairs of vertices in a graph, *J. ACM* 25 (1978) 1–9.
17. N. Robertson and P.D. Seymour, Graph minors XIII: The disjoint paths problem, *J. Combin. Theory*, Ser B, 63 (1995) 65–110.
18. Y. Shiloach, A polynomial solution to the undirected two paths problem, *J. ACM* 27 (1980) 445–456.
19. R.W. Whitty, Vertex-disjoint paths and edge-disjoint branchings in directed graphs, *J. Graph Theory* 11 (1987) 349–358.
20. A. Zehavi and A. Itai, Three tree-paths, *J. Graph Theory* 13 (1989) 175–188.

Facets of the Directed Acyclic Graph Layering Polytope

Patrick Healy and Nikola S. Nikolov

CSIS Department, University of Limerick, Limerick, Republic of Ireland
{patrick.healy,nikola.nikolov}@ul.ie
fax +353-61-202734

Abstract. The drawing of hierarchical graphs is one of the main areas of research in the field of Graph Drawing. In this paper we study the problem of partitioning the node set of a directed acyclic graph into layers – the first step of the commonly accepted Sugiyama algorithm for drawing directed acyclic graphs as hierarchies. We present a combinatorial optimization approach to the layering problem; we define a graph layering polytope and describe its properties in terms of facet-defining inequalities. The theoretical study presented is the basis of a new branch-and-cut layering algorithm which produces better quality drawings of hierarchical graphs.

1 Introduction

The problem we study in this paper is related to the visualization of hierarchical relations between objects in a system. This problem arises in many applications and recently it has become more important because of the fast development of the software industry and the rapid growth of the World Wide Web and the corresponding need to visualize it (in the process of browsing, in the results of search engines, as a web-master tool, etc.). A hierarchy is a directed acyclic graph (DAG) the edges of which are drawn to point unidirectionally, i.e. they form a unidirectional flow. Although there are alternative concepts for visual representing of DAGs, a commonly accepted and used one is to require that all the edges point in the same direction. An example of a hierarchical drawing of a DAG is shown in Figure 1. Virtually every contemporary graph drawing package provides an implementation of an algorithm for drawing DAGs as hierarchies and usually this is the algorithm of Sugiyama et al. [11] which has become a standard tool for drawing hierarchical graphs.

The original idea of Sugiyama et al. is to draw a DAG in three separate and independent steps. Firstly the nodes of the DAG are separated into layers so that all the edges point unidirectionally, then the nodes are ordered within the layers, trying to improve the quality of the drawing through the minimization of edge crossings, and finally, x and y coordinates are assigned to the nodes and the shape of lines, which represent the edges of the DAG are determined. At each of the three steps a separate algorithm is applied which solves the corresponding problem according to certain aesthetic criteria important for the

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 246–257, 2002.

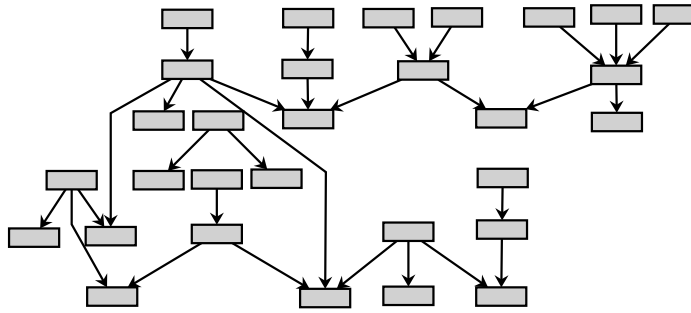


Fig. 1. An example drawing of a DAG as a hierarchy.

final drawing, such as specific dimensions of the drawing, representations of the edges as straight lines, minimization of the number of edge crossings etc. In their initial publication Sugiyama et al. [11] and later Eades and Sugiyama [2] provided a number of independent algorithms which can be applied at each of the three steps. Since then many researchers have proposed new algorithms for the three steps, although most of the research effort has focused on the second step, the reduction of edge crossings (see, for instance, Jünger *et al.* [8] and Healy and Kuusik [6]). In this paper we present a theoretical study of the first step of the Sugiyama algorithm with the aim of developing an algorithm for partitioning the nodes of a DAG into layers which has equivalent quality to the existing algorithms for the second step of the Sugiyama algorithm.

An algorithm that partitions the nodes of a DAG into layers so that all edges point unidirectionally is known as a layering algorithm. A layering algorithm must find a layering of a DAG subject to certain aesthetic criteria. While these may be subjective, some are generally agreed upon [2]: the drawing should be compact; large edge spans should be avoided; and, the edges should be as straight as possible. Compactness can be achieved by specifying bounds W and H on the width and the height of the layering respectively. Short edge spans are desirable aesthetically because they increase the readability of the drawing but also because the forced introduction of dummy nodes at the places where edges cross layers complicates the next steps of the Sugiyama algorithm. Further, these dummy nodes may also permit additional bends on edges since edge bends mainly occur at dummy nodes.

At present there are three layering algorithms which find layerings of a DAG subject to *some* of the above aesthetic criteria. They all have polynomial running time: the longest path algorithm finds a layering with minimal height [2]; the Coffman-Graham algorithm finds a layering of width at most W and height $h \leq (2 - 2/W)h_{min}$, where h_{min} is the minimum height of a layering [1]; and the ILP algorithm of Gansner *et al.* finds a layering with minimum number of dummy nodes [4]. An upper bound on the width of the layering can be specified only in the Coffman-Graham algorithm. In the classical version of the Coffman-Graham algorithm the width of a layer is considered to be the number of real

nodes (each assumed to be of unit width) the layer contains while neglecting the widths of the introduced dummy nodes. The algorithm can be modified to take into account the true widths of the real nodes, but the width of the final drawing may still be much greater than expected. We contend that the widths of dummy nodes introduced by the layering algorithm to the width of a layer – especially if they derive form "thick" edges – contribute to a layer's width, and taking account of them would express the width of the final drawing more accurately. However, layering a graph taking into account simply the width in the traditional sense and the height constraints is an NP-hard problem since the Precedence Constrained Multiprocessor Scheduling problem can be reduced straightforwardly to it [2].

Thus, the goal of this work is to partition the nodes of a DAG into layers subject to specified maximum dimensions, reporting so if no layering within the given bounds exists. The dimensions of the layering should be true in that the width of dummy nodes should not be assumed to be negligible and the layering should minimize the sum of the edge spans. We call this optimization problem *WHS-Layering*. This problem is closely related to the Precedence Constrained Multiprocessor Scheduling problem and to the Capacitated Clustering problem. To the best of our knowledge there is no exact solution technique developed for the former problem. For the latter problem a branch-and-cut algorithm by Ferreira *et al.* [3] and a branch-and-price algorithm by Mehrotra and Trick [9] have been proposed.

In related work [7] we performed a series of experiments that compared each of the previously described algorithms with an Integer Linear Programming formulation on a benchmark set of graphs. This work demonstrated that the ILP model gave solutions of superior quality, at the expense of longer running time. This paper provides a theoretical basis for that formulation by proving that three families of inequalities are facet-defining; thus, the proposed model is a reasonably efficient description of the set of feasible layerings. Further, these facets can be used to find solutions more quickly in a branch-and-cut algorithm.

In the next section we introduce some formal definitions and then in Section 3 we take a combinatorial optimization approach to the layering problem describing the graph layering polytope as a tool for constructing an exact layering algorithm subject to the aesthetics discussed above. Section 4 draws the main conclusions of our work.

2 Preliminaries

Definition 1. *Given a DAG $G = (V, E)$, where each node $v \in V$ has a positive width w_v, a* layering *of G is a partition of its node set V into subsets $V_1, V_2, \ldots, V_h$, such that if $(u, v) \in E$ where $u \in V_i$ and $v \in V_j$ then $i > j$. A DAG with a layering is called a* layered digraph.

Definition 2. *The* height *of a layering is the number of layers, h; traditionally the* width of layer V_k *is defined as $w(V_k) = \sum_{v \in V_k} w_v$ and the* width *of a layered digraph is $w = \max_{1 \leq k \leq h} w(V_k)$.*

Layered digraphs are conventionally drawn so that all nodes in layer V_k lie on the horizontal line $y = k$. A layering algorithm is an algorithm which fixes the y coordinate of each node, and ensures that all edges point downwards (i.e. unidirectionally). The span of an edge (u, v) with $u \in V_i$ and $v \in V_j$ is $i - j$. When edges span multiple layers, it is common to introduce dummy nodes with in- and out-degree 1 in the intermediate layers.

We denote by $d^-(v)$ and $d^+(v)$ the in- and out-degree of node $v \in V$, respectively. For $G = (V, E)$ with unitary edge lengths, define $l_p(v)$ to be the length of the longest path from any node u to v where $d^-(u) = 0$. Similarly, define $l_s(v)$ to be the longest path from v to any node u where $d^+(u) = 0$. That is, the values $l_p(v)$ and $l_s(v)$ refer, respectively, to the length of the longest path from any predecessor to v and to the length of the longest path from v to any successor. Suppose the node set V of G must to be partitioned into at most H layers. Then for each node $v \in V$ there is a set of consecutive layers where potentially the node can be placed if all the edges are required to point downwards. The following three definitions describe this set.

Definition 3. *The* roof *of node v is the number of highest layer node v can be placed in. We denote the roof of v by $\rho(v)$, i.e. $\rho(v) = H - l_p(v)$.*

Definition 4. *The* floor *of v is the lowest level node v can be placed in. We denote the floor of v by $\varphi(v)$, i.e. $\varphi(v) = l_s(v) + 1$.*

Definition 5. *The* layer span *of node v is $L(v) = \{k \in N : \varphi(v) \leq k \leq \rho(v)\}$. That is, $L(v)$ refers to the set of layers in which node v can be placed if all the edges are required to point downwards.*

The roof and the layer span of node v depend on the upper bound on the number of layers H. We do not include H in the notation of $\rho(v)$ and $L(v)$, because normally it is clear what is the value of H from the context.

Let $G = (V, E)$ be a DAG the nodes of which are to be partitioned into at most $H > 0$ layers $V_1, V_2, \ldots, V_H$ of width at most $W > 0$. We construct a *layering DAG* or *LDAG* $\mathcal{L}_G^H = (V_\mathcal{L}, E_\mathcal{L})$ as follows. For each $v \in V$ and each $k \in L(v)$ there exists a node $\lambda_{vk} \in V_\mathcal{L}$ that corresponds to node $v \in V$ placed in layer V_k. The pair $(\lambda_{uk}, \lambda_{vl}) \in E_\mathcal{L}$ if and only if $(u, v) \in E$.

Property 1. Let $\mathcal{L}_G^H$ be an LDAG of $G = (V, E)$, $H > 0$. If $(u, v) \in E$ then $\varphi(u) > \varphi(v)$ and $\rho(u) > \rho(v)$.

Property 2. Let $\mathcal{L}_G^H$ be an LDAG of $G = (V, E)$, $|V| = n$, $|E| = m$, $H > 0$. Then $|V_\mathcal{L}| = O(n^2)$ and $|E_\mathcal{L}| = O(mn^2)$.

Definition 6. *The set $F \subseteq V_\mathcal{L}$* partially represents *$G$ if*

- *$\lambda_{uk} \in F$, $\lambda_{vl} \in F$ and $k \neq l$ implies $u \neq v$;*
- *all the edges of $\mathcal{L}_G^H[F]$, i.e. the subgraph of $\mathcal{L}_G^H$ induced by F, point downwards.*

We call F a partial layering *of G.*

Definition 7. *$F \subseteq V_{\mathcal{L}}$ (fully)* represents *G if F partially represents G and for each node $v \in V$ there is a node $\lambda_{vk} \in F$ for some k.*

Note that if F partially represents G then $\mathcal{L}_G^H[F]$ is a layered digraph, where each node $v \in V$ is represented by at most one node of $\mathcal{L}_G^H$.

3 The Directed Acyclic Graph Layering Polytope

In this section we describe the layering problem as an optimization problem over an independence system taking into account all the aesthetic criteria discussed above as well as the contribution of the dummy nodes to the width of the layering.

3.1 The Layering Problem as an Optimization Problem

Let $G = (V, E)$ be a DAG the nodes of which are to be partitioned into at most $H > 0$ layers $V_1, V_2, \ldots, V_H$ of width at most $W > 0$ and let $\mathcal{L}_G^H = (V_{\mathcal{L}}, E_{\mathcal{L}})$ be the corresponding LDAG. The pair $(V_{\mathcal{L}}, \mathcal{I})$, where $\mathcal{I}$ is the family of all the subsets of $V_{\mathcal{L}}$ which partially represent G and induce layered digraphs of width at most W, is an independence system. The problem of finding a layering of G on at most H layers of width not greater than W and minimum total sum of edge spans can be expressed as an optimization problem over the independence system $(V_{\mathcal{L}}, \mathcal{I})$ as follows: $\min\{C(F) : F \in \mathcal{I}\}$, where $C(F) = \sum_{i \in F} c(i)$ and $c(i)$ is a weight associated with each node $i \in V_{\mathcal{L}}$.

We need such a weight function C with co-domain R, so that $C(F)$ reaches its minimum at a set F which induces a layered digraph with minimum total sum of edge spans. If x is the incidence vector of a subset of $V_{\mathcal{L}}$ then the expression

$$\sum_{(u,v)\in E} \left(\sum_{k=\varphi(u)}^{\rho(u)} k x_{uk} - \sum_{k=\varphi(v)}^{\rho(v)} k x_{vk} \right) = \sum_{v \in V} \sum_{k=\varphi(v)}^{\rho(v)} k \left(d^+(v) - d^-(v) \right) x_{vk}$$

represents the sum of edge spans of $\mathcal{L}_G^H[F]$. If we set $c(\lambda_{vk}) = k\,(d^+(v) - d^-(v))$ then the minimum of the weight function would correspond to a layering with minimum total sum of edge spans. But the minimum can potentially be reached at a partial layering which does not represent G. To ensure that the minimum will be reached at a set that represents G we set $c(\lambda_{vk}) = k\,(d^+(v) - d^-(v)) - M$, where M is an appropriately large positive number, for instance, $M = H \times |E|$. Then the optimization problem over $(V_{\mathcal{L}}, \mathcal{I})$ takes the form

$$\min \left\{ \sum_{\lambda_{vk} \in F} \left[k \left(d^+(v) - d^-(v) \right) - M \right] : F \in \mathcal{I} \right\}$$

We call the polytope associated with the family of subsets $\mathcal{I}$, the *graph layering polytope* and we denote it by $\mathcal{GLP}(\mathcal{L}_G^H, W)$.

Theorem 1. *The dimension of the graph layering polytope $\mathcal{GLP}(\mathcal{L}_G^H, W)$ with $G = (V, E)$ is $|V_{\mathcal{L}}| = \sum_{u \in V} (\rho(u) - \varphi(u) + 1)$, i.e. it is full dimensional. For each node λ_{vk} of $\mathcal{L}_G^H$ the inequalities $x_{vk} \geq 0$ define facets of $\mathcal{GLP}(\mathcal{L}_G^H, W)$.*

Proof. This theorem is a trivial corollary of the properties of independence systems. The properties of independence systems can be found in [5].

Each independence system has a unique system of minimal dependent sets called *circuits*. If $\mathcal{D} = \{D_1, D_2, \ldots, D_{|\mathcal{D}|}\}$ is the system of circuits of $(V_\mathcal{L}, \mathcal{I})$, then the same optimization problem can be expressed as the linear program

$$\min \left\{ c^T y : y \in conv \left\{ Ax \leq b : x \in \{0,1\}^{|V_\mathcal{L}|} \right\} \right\}$$

where $c : V_\mathcal{L} \to Z$, and $c(\lambda_{vk}) = k(d^+(v) - d^-(v)) - M$. A is the incidence matrix of the members of $\mathcal{D}$ with $V_\mathcal{L}$ and $b = (b_1, b_2, \ldots, b_{|\mathcal{D}|})$, where $b_i = |D_i| - 1$ for $i = 1, \ldots, |\mathcal{D}|$. The inequalities $Ax \leq b$ are called *rank inequalities*.

In the next three subsections we take a detailed look at the circuits of the graph layering polytope $\mathcal{GLP}(\mathcal{L}_G^H, W)$.

3.2 Assignment Circuits

If $(V_\mathcal{L}, \mathcal{I})$ is the independence system described above then each node of G corresponds to at most one node of $\mathcal{L}_G^H[F]$ for each $F \in \mathcal{I}$. This property gives a set of circuits of the type $D_a = \{\lambda_{vk}, \lambda_{vl}\}$, $k \neq l$, which we call *assignment circuits*. The rank inequality for such a circuit is $x_{vk} + x_{vl} \leq 1$. We can combine all the rank inequalities related to the node $v \in V$ in one inequality, which is stronger than all of them

$$\sum_{k=\varphi(v)}^{\rho(v)} x_{vk} \leq 1 \tag{1}$$

This is obviously a valid inequality of $\mathcal{GLP}(\mathcal{L}_G^H, W)$, because each set in $\mathcal{I}$ partially represents G.

Theorem 2. *The assignment inequalities (1) are facet defining for the graph layering polytope* $\mathcal{GLP}(\mathcal{L}_G^H, W)$.

Proof. Let $G = (V, E)$ be a DAG the nodes of which are to be partitioned into at most $H > 0$ layers $V_1, V_2, \ldots, V_H$ of width at most $W > 0$; let $\mathcal{L}_G^H = (V_\mathcal{L}, E_\mathcal{L})$ be the corresponding LDAG. It is enough to find $|V_\mathcal{L}|$ affinely independent subsets of $V_\mathcal{L}$ which partially represent G, induce layered digraphs of width at most W, and whose incidence vectors satisfy (1) with equality. (In our case it suffices to look for linearly independent subsets as the supporting hyperplanes do not contain the origin.) Consider the sets $F_k = \{\lambda_{vk}\}$ for all $\varphi(v) \leq k \leq \rho(v)$ and the sets $F_{ul} = \{\lambda_{ul}, \lambda_{vk}\}$ for all $u \neq v$ and $\varphi(u) \leq l \leq \rho(u)$, with $k = \varphi(v)$ in case $(u, v) \in E$ and $k = \rho(v)$ otherwise. Each of these $|V_\mathcal{L}|$ sets partially represents G (see Property 1), have width at most equal to the width of the widest node of G, and their incidence vectors give linearly independent incidence vectors which satisfy (1) with equality.

3.3 Direction Circuits

The requirement that all the edges point unidirectionally leads to a set of *direction circuits* each including two nodes connected by an edge which does not point downwards. For instance, if $(\lambda_{uk}, \lambda_{vl}) \in E_{\mathcal{L}}$ is an edge that does not point downwards, i.e. $k \leq l$, then $D_d = \{\lambda_{uk}, \lambda_{vl}\}$ is such a circuit. The rank inequality which corresponds to it is $x_{uk} + x_{vl} \leq 1$. We can generalize these rank inequalities by stronger inequalities, which we call strong relative-ordering (SRO) inequalities, as follows.

$$\sum_{i=\varphi(u)}^{k} x_{ui} + \sum_{i=k}^{\rho(v)} x_{vi} \leq 1 \tag{2}$$

for each $k \in L(u) \cap L(v)$ and for each edge (u, v). The number of SRO inequalities is $\sum_{(u,v)\in E} \max\{0, \rho(v) - \varphi(u) + 1\}$.

Theorem 3. *Let $G = (V, E)$ be a DAG and $(u, v) \in E$ is a non-transitive edge. Then the SRO inequality (2) with $k \in L(u) \cap L(v)$ is facet-defining for $\mathcal{GLP}(\mathcal{L}_G^H, W)$ for any $W > 0$.*

Proof. We apply sequential lifting (see [10] for details about the technique of sequential lifting) starting with nodes λ_{uk} and λ_{vk}.

Step 1. Consider $\mathcal{L}_G^H[\{\lambda_{uk}, \lambda_{vk}\}]$ - the subgraph of $\mathcal{L}_G^H$, which contains only the nodes λ_{uk} and λ_{vk} and the edge between them. Then obviously

$$x_{uk} + x_{vk} \leq 1 \tag{3}$$

is a facet defining for $\mathcal{GLP}(\mathcal{L}_G^H[\{\lambda_{uk}, \lambda_{vk}\}], W)$. Sequentially lifting (3) with λ_{uj}, $j = \varphi(u) \ldots, k-1$ and λ_{vj}, $j = k+1, \ldots, \rho(v)$ leads to the inequality (2) which defines a facet of $\mathcal{GLP}(\mathcal{L}_G^H[\{\lambda_{uj} : \varphi(u) \leq j \leq k\} \cup \{\lambda_{vj} : k \leq j \leq \rho(v)\}], W)$.

Step 2. If node $w \in V$ is incident neither to u nor to v then we lift x_{wj} with a zero coefficient. Thus we conclude that (3) defines a facet of $\mathcal{GLP}(\mathcal{L}_G^H[F], W)$ for any $H > 0$ and $W > 0$ and $F = \{\lambda_{uj} : \varphi(u) \leq j \leq k\} \cup \{\lambda_{vj} : k \leq j \leq \rho(v)\} \cup \{\lambda_{wj} : w \textit{ is incident neither to } u \textit{ nor to } v,\ j \in L(w)\}$.

Step 3. Then consider λ_{wj}, where w is either u or v and node λ_{wj} has not been considered yet in a previous step. Let, for simplicity of notation, $w = u$. Then $k+1 \leq j \leq \rho(u)$. If node u is placed in layer V_j then node v can be placed in layer V_k and thus lifting variable x_{wj} we add it to the facet-defining inequality with a zero coefficient.

Step 4. Now consider node λ_{wj} with $w \notin \{u, v\}$ and w incident to at most one of the nodes u and v. If node w is placed in layer V_j exactly one of the variables $\{x_{uj} : \varphi(u) \leq j \leq k\} \cup \{x_{vj} : k \leq j \leq \rho(v)\}$ can be equal to 1 and thus we lift x_{wj} with a zero coefficient.

Step 5. The last and most complicated case is when λ_{wj} is a node with w incident to both u and v. The four drawings shown in Figure 2 represent all the possible cases of relations between u, v and w. The drawings in Figures 2(a) and 2(b) represent cases which cannot occur under the conditions of the theorem,

because there is a cycle in the case (a) and edge (u, v) is transitive in case (b). If $j \neq k$ then we can apply zero lifting to x_{wj}. We can also apply zero lifting in any of the cases shown in Figures 2(c) and 2(d).

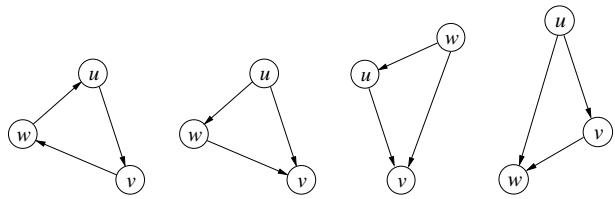


Fig. 2. The four possible cases of relations between nodes u, v and w if there is an edge (u, v) and w is incident to both u and v.

In the five steps above we have lifted all the variables, thus showing that (2) defines a facet of $\mathcal{GLP}(\mathcal{L}_G^H, W)$.

Note that the strong relative ordering inequalities for all the non-transitive edges are sufficient to ensure that all the edges of the layered digraph point downwards.

3.4 Capacity Circuits

The upper bound W on the width of the desired layering gives rise to the set of *capacity circuits* which partially represent G, but the layered digraph induced by them have width greater than W. If any of the nodes of such a circuit is removed then the remainder of the set induces a layered digraph of width less than or equal to W. This side of the graph layering polytope has proved to be the most difficult to describe by facet defining inequalities. This is predictable since the layering problem with unrestricted width of the layering is polynomially solvable, as has been shown by the algorithm of Gansner et al. We identify two types of non-trivial valid inequalities of $\mathcal{GLP}(\mathcal{L}_G^H, W)$ related to the imposed width bound. The first we call *path-augmented layer (PAL) inequalities* and they have the following form.

$$\sum_{i=1}^{r} x_{v_i k} + \sum_{i=1}^{m_p} x_{u_i^p k} + \sum_{i=1}^{m_s} x_{u_i^s k} \leq r - 1 \tag{4}$$

for

- all r-tuples $\{v_1, v_2, \ldots, v_r\}$ of pairwise independent nodes (without a direct path between any two of them) which cannot be placed together into the same layer without causing the width of the layering to exceed the upper bound W;

- $m_p \geq 0$ and nodes $u_1^p, \ldots, u_{m_p}^p$ form a directed path $(u_{m_p}^p, \ldots, u_1^p)$ where u_1^p is a common immediate predecessor to each of $v_1, \ldots, v_r$;
- $m_s \geq 0$ and nodes $u_1^s, \ldots, u_{m_s}^s$ form a directed path $(u_1^s, \ldots, u_{m_s}^s)$ where u_1^s is a common immediate successor to each of $v_1, \ldots, v_r$;
- $k \in \left(\bigcap_{i=1}^{r} L(v_i)\right) \bigcap \left(\bigcap_{i=1}^{m_p} L(u_i^p)\right) \bigcap \left(\bigcap_{i=1}^{m_s} L(u_i^s)\right)$.

Theorem 4. *Let (4) be a PAL inequality for the DAG G such that there are no transitive edges with endpoints among nodes $v_1, v_2, \ldots, v_r$; $u_1^p, \ldots, u_{m_p}^p$ and $u_1^s, \ldots, u_{m_s}^s$. Further, assume that $r \geq 2$, $m_p + m_s \geq 2r - 2$ and the sum of the widths of each r nodes is at most W. Then (4) is facet-defining for the layering polytope associated with G.*

Proof. As in the proof of Theorem 3 we construct $|V_{\mathcal{L}}|$ subsets of $V_{\mathcal{L}}$ which partially represent G, induce layered digraphs of width at most W, satisfy (4) as an equality and whose incidence vectors are linearly independent. We construct the sets in four steps.

Step 1. For each node $\lambda_{v_i k}$ we can choose the set $\{\lambda_{v_1 k}, \ldots, \lambda_{v_{i-1} k}, \lambda_{v_{i+1} k}, \ldots, \lambda_{v_r k}\}$. Suppose, for simplicity of notation, that $\lambda_{v_1 k}, \lambda_{v_2 k}, \ldots, \lambda_{v_r k}$ correspond to the first r components of an incidence vector of a subset of $V_{\mathcal{L}}$. Then the sets described above give the following matrix of incidence vectors.

$$A = \begin{pmatrix} 0\,1\,1 \ldots 1\,0\,0 \ldots 0 \\ 1\,0\,1 \ldots 1\,0\,0 \ldots 0 \\ 1\,1\,0 \ldots 1\,0\,0 \ldots 0 \\ \vdots\,\vdots\,\vdots\ \vdots\ \vdots\,\vdots\,\vdots\ \vdots\ \vdots \\ 1\,1\,1 \ldots 0\,0\,0 \ldots 0 \end{pmatrix}$$

When $r \geq 2$ the rank of A is r, because

$$\begin{vmatrix} 0\,1\,1 \ldots 1 \\ 1\,0\,1 \ldots 1 \\ 1\,1\,0 \ldots 1 \\ \vdots\,\vdots\,\vdots\ \vdots\ \vdots \\ 1\,1\,1 \ldots 0 \end{vmatrix} = (-1)^{r-1}(r-1) \neq 0$$

In other words the incidence vectors of the r subsets chosen above are linearly independent.

Step 2. Consider the nodes $\lambda_{u_1^p k}, \lambda_{u_2^p k}, \ldots, \lambda_{u_{m_p}^p k}$ and $\lambda_{u_1^s k}, \lambda_{u_2^s k}, \ldots, \lambda_{u_{m_s}^s k}$. Since we have assumed that there is no transitive edges with endpoints among these nodes, we can construct the sets $\{\lambda_{u_i^p k}, \lambda_{v_1 k}, \ldots, \lambda_{v_{r-1} k}\}$ for $i = 2, \ldots, m_p$ and $\{\lambda_{u_i^s k}, \lambda_{v_1 k}, \ldots, \lambda_{v_{r-1} k}\}$ for $i = 2, \ldots, m_s$ and another two sets F_p and F_s as follows.

$$F_p = \{\lambda_{u_1^p k}\} \cup \{\lambda_{u_i^p k} : i = 3, 5 \ldots, 2\lceil m_p/2 \rceil - 1\} \cup \{\lambda_{u_i^s k} : i = 2, 4, \ldots, 2\lfloor m_s/2 \rfloor\};$$

$$F_s = \{\lambda_{u_1^s k}\} \cup \{\lambda_{u_i^s k} : i = 3, 5, \ldots, 2\lceil m_s/2 \rceil - 1\} \cup \{\lambda_{u_i^p k} : i = 2, 4, \ldots, 2\lfloor m_p/2 \rfloor\};$$

$$\begin{aligned}|F_p| &= 1 + \lceil m_p/2 \rceil - 1 + \lfloor m_s/2 \rfloor \\ &= \lceil m_p/2 \rceil + \lfloor m_s/2 \rfloor \\ &\geq \lceil m_p/2 + m_s/2 \rceil \\ &\geq \lfloor r - 1 \rfloor \\ &= r - 1.\end{aligned}$$

Similarly $|F_s| \geq r - 1$. Thus we can always choose a subset of F_s and a subset of F_p with exactly $r - 1$ elements. It is easy to see that the incidence vectors of the sets constructed at this step are linearly independent.

Step 3. Consider $\lambda_{xl} \in V_{\mathcal{L}}$ which does not appear in any of the sets we have already constructed. Then the following cases are possible.

- If x is one of $v_1, ..., v_r$, say $x = v_r$ for simplicity of notation, or if there is no edge between x and any of the nodes $v_1, \ldots, v_r$ then we choose the set $\{\lambda_{xl}, \lambda_{v_1 k}, \ldots, \lambda_{v_{r-1} k}\}$.
- If $(x, v_i) \in E$ for some $i = 1, 2, \ldots, r$ and $l > k$, t.e. layer l is above layer k then we can choose the same set $\{\lambda_{xl}, \lambda_{v_1 k}, \ldots, \lambda_{v_{r-1} k}\}$
- If $(x, v_i) \in E$ for some $i = 1, 2, \ldots, r$ and $l \leq k$ then we construct the set $F = \{\lambda_{xl}\} \cup \{\lambda_{u_i^s k} : i = 1, 3, \ldots 2\lceil m_s/2 \rceil - 1\} \cup F_x$, where

$$F_x \subseteq \{\lambda_{u_1^p k}, \lambda_{u_2^p k}, \ldots, \lambda_{u_{m_p}^p k}\}$$

  with no edge between any two nodes in F_x, node x not represented by any node in F_x, and $|F_x| \geq \lfloor m_p/2 \rfloor$. F_x is always possible to construct because
  - if $x \neq u_i^p$ for each $i = 1, \ldots, m_p$ then there is no edge of the type (x, u_i^p), because otherwise the edge (x, v_i) would be transitive. If x is one of u_i^p then there is only one such edge which is a part of the path;
  - there can be no more than one edge of the type (u_i^p, x), because if there is another one, say (u_j^p, x), then either of them would be transitive.

  Hence

$$|F| = 1 + \lceil m_s/2 \rceil + \lfloor m_p/2 \rfloor \geq 1 + \lfloor m_s/2 + m_p/2 \rfloor = 1 + \lfloor r - 1 \rfloor = r$$

  If $|F| > r$ then we can always choose a subset with exactly r elements, leaving λ_{xl} and any other $r - 1$ nodes.
- If $(v_i, x) \in E$ for some $i = 1, 2, \ldots, r$ then we can do constructions analogous to the constructions in the previous two cases.

The second group of inequalities, related to the capacity circuits, are the same *SRO inequalities* described above, but this time for nodes u and v which are not related by an edge and where we have established that u cannot be placed above v (or vice-versa) because it will lead to the existence of a layer of width greater that W.

Theorem 5. *Let $G = (V, E)$ be a DAG, u and v are two independent nodes (without a directed path between them) and u has to be placed above v in order to have a layering of width not greater than W. Then the SRO inequality (2) with $k \in L(u) \cap L(v)$ is facet-defining for $\mathcal{GLP}(\mathcal{L}_G^H, W)$ for any $W > 0$.*

Proof. The proof is available upon request. We omit it from this paper, because it follows the same scheme as the proof of Theorem 3.

4 Conclusions

We have defined the layering problem for DAGs as a formal optimization problem over an independence system and have identified a number of facet-defining inequalities of the graph layering polytope. In a related paper [7] we presented an experimental comparison of the three existing layering algorithms on over 5911 example DAGs and an Integer Linear Programming (ILP) layering algorithm; this paper provides a theoretical basis for the ILP model and results reported there. The results clearly showed that our approach lead to better quality drawings albeit with longer running time. The solution method employed there was CPLEX's canned branch-and-bound ILP solver and it allowed us to solve medium-sized problems (100 vertices) with little difficulty.

In this work we establish that three families of inequalities of that model are indeed facet-defining and now it is possible to implement a competitive branch-and-cut algorithm. The preliminary results of this work are quite encouraging and suggest that we can provide more finely tuned graph layouts while keeping running time within an acceptable bound.

References

1. E. G. Coffman and R. L. Graham. Optimal scheduling for two processor systems. *Acta Informatica*, 1:200–213, 1972.
2. P. Eades and K. Sugiyama. How to draw a directed graph. *Journal of Information Processing*, 13(4):424–437, 1990.
3. C. E. Ferreira, A. Martin, C. C. De Souza, R. Weismantel, and L. A. Wolsey. Formulations and valid inequalities for the node capacitated graph partitioning problem. *Mathematical Programming*, 74:247–266, 1996.
4. E. R. Gansner, E. Koutsofios, S. C. North, and K.-P Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, March 1993.
5. M. Grötschel and M. W. Padberg. Polyhedral theory. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Series in Discrete Mathematics, pages 251–305. John Wiley & Sons, 1985.
6. P. Healy and A. Kuusik. The vertex-exchange graph: A new concept for multi-level crossing minimization. In J. Kratochvíl, editor, *Graph Drawing: Proceedings of 7th International Symposium, GD '99*, volume 1731 of *Lecture Notes in Computer Science*, pages 205–216. Springer-Verlag, 1999.
7. P. Healy and N. S. Nikolov. How to layer a directed acyclic graph. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Graph Drawing: Proceedings of 9th International Symposium, GD 2001*, volume 2265 of *Lecture Notes in Computer Science*, pages 16–30. Springer-Verlag, 2002.

8. M. Jünger, E. K. Lee, P. Mutzel, and T. Odenthal. A polyhedral approach to the multi-layer crossing minimization problem. In G. Di Battista, editor, *Graph Drawing: 5th International Symposium, GD '97*, volume 1353 of *Lecture Notes in Computer Science*, pages 13–24, Rome, Italy, September 1997. Springer-Verlag.
9. A. Mehrotra and M. A. Trick. Cliques and clustering: A combinatorial approach. *Operations Research Letters*, 22(1):1–12, 1998.
10. G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience series in discrete mathematics and optimization. John Wiley & Sons, Inc., 1988.
11. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transaction on Systems, Man, and Cybernetics*, 11(2):109–125, February 1981.

Recognizing When Heuristics Can Approximate Minimum Vertex Covers Is Complete for Parallel Access to NP*

Edith Hemaspaandra[1], Jörg Rothe[2], and Holger Spakowski[2]

[1] Rochester Institute of Technology, Department of Computer Science
Rochester, NY 14627, USA
eh@cs.rit.edu
[2] Heinrich-Heine-Universität Düsseldorf, Institut für Informatik
40225 Düsseldorf, Germany
{rothe,spakowsk}@cs.uni-duesseldorf.de

Abstract. For both the edge deletion heuristic and the maximum-degree greedy heuristic, we study the problem of recognizing those graphs for which that heuristic can approximate the size of a minimum vertex cover within a constant factor of r, where r is a fixed rational number. Our main results are that these problems are complete for the class of problems solvable via parallel access to NP. To achieve these main results, we also show that the restriction of the vertex cover problem to those graphs for which either of these heuristics can find an optimal solution remains NP-hard.

Keywords: Computational complexity; completeness; minimum vertex cover heuristics; approximation; parallel access to NP.

1 Introduction

The minimum vertex cover problem is the problem of finding in a given graph a smallest possible set of vertices that covers at least one vertex of each edge. The decision version of the minimum vertex cover problem, VC, is one of the standard NP-complete problems [GJ79]. To cope with the intractability that appears to be inherent to this problem, various heuristics for finding minimum vertex covers have been proposed. Two of the most prominent such heuristics are the *edge deletion heuristic* and the *maximum-degree greedy heuristic*, see, e.g., [PS82,Pap94]. These algorithms run in linear time and, depending on the structure of the given input graph, may find a minimum vertex cover, or may provide a good approximation of the optimal solution.

It is common to evaluate heuristics for optimization problems by analyzing their worst-case ratio for approximating the optimal solution. In this regard, the

* This work was supported in part by grant NSF-INT-9815095/DAAD-315-PPP-güab. The second author was supported in part by a Heisenberg Fellowship of the Deutsche Forschungsgemeinschaft.

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 258–269, 2002.

two heuristics considered behave quite differently: the edge deletion heuristic always approximates the size of a minimum vertex cover within a factor of 2 and thus achieves the best approximation ratio known, whereas the maximum-degree greedy heuristic, in the worst case, can have an approximation ratio as bad as logarithmic in the input size. The latter result follows from the early analysis of the approximation behavior of the greedy algorithm for the minimum set cover problem that was done by Johnson [Joh74], Lovász [Lov75], and Chvátal [Chv79] (who studied the weighted version of minimum set cover). Note that the vertex cover problem is the special case of the set cover problem, restricted so that each element occurs in at most two sets. More recently, building on the work of Lund and Yannakakis [LY94], Feige [Fei98] showed that, unless NP has slightly superpolynomial-time algorithms, the set cover problem cannot be approximated within $(1-\epsilon)\ln n$, where $\epsilon > 0$ and ln denotes the natural logarithm.

In this paper, we study the problem of recognizing those input graphs for which either of the two heuristics can approximate the size of a minimum vertex cover within a constant factor of r, where $r \geq 1$ is a fixed rational number. Let $\mathcal{S}_r^{\mathrm{ED}}$ and $\mathcal{S}_r^{\mathrm{MDG}}$, respectively, denote this recognition problem for the edge deletion heuristic and for the maximum-degree greedy heuristic. Our main results are:

Theorem 3 For each rational number r with $1 \leq r < 2$, $\mathcal{S}_r^{\mathrm{ED}}$ is $\mathrm{P}_{\|}^{\mathrm{NP}}$-complete.
Theorem 6 For each rational number $r \geq 1$, $\mathcal{S}_r^{\mathrm{MDG}}$ is $\mathrm{P}_{\|}^{\mathrm{NP}}$-complete.

Here, $\mathrm{P}_{\|}^{\mathrm{NP}}$ denotes the class of problems that can be decided in polynomial time by parallel (i.e., truth-table) access to NP. Papadimitriou and Zachos [PZ83] introduced this class under the name $\mathrm{P}^{\mathrm{NP}[\mathcal{O}(\log n)]}$, where "$[\mathcal{O}(\log n)]$" denotes that at most logarithmically many Turing queries are made to the NP oracle. Hemaspaandra [Hem89] proved that $\mathrm{P}^{\mathrm{NP}[\mathcal{O}(\log n)]} = \mathrm{P}_{\|}^{\mathrm{NP}}$, and in fact many more characterizations of $\mathrm{P}_{\|}^{\mathrm{NP}}$ are known [KSW87,Wag90]. Other natural $\mathrm{P}_{\|}^{\mathrm{NP}}$-complete problems can be found in the papers by Krentel [Kre88], Wagner [Wag87], and Hemaspaandra et al. [HHR97,HR98].

The type of recognition problem studied in this paper was investigated for other problems and other heuristics as well. Bodlaender, Thilikos, and Yamazaki [BTY97] defined and studied the analogous problem for the independent set problem and the minimum-degree greedy heuristic, which they denoted by $\mathcal{S}_r$. They proved that $\mathcal{S}_r$ is coNP-hard and belongs to P^{NP}. Closing the gap between these lower and upper bounds, Hemaspaandra and Rothe [HR98] proved that $\mathcal{S}_r$ is $\mathrm{P}_{\|}^{\mathrm{NP}}$-complete. As in [HR98], we obtain $\mathrm{P}_{\|}^{\mathrm{NP}}$-hardness by reducing from a problem (namely, $\mathtt{VC}_{\mathtt{geq}}$, see Section 2) that can be shown to be $\mathrm{P}_{\|}^{\mathrm{NP}}$-complete using the techniques of Wagner [Wag87]. Also, we show that the vertex cover problem, restricted to those input graphs for which the heuristics considered can find an optimal solution, remains NP-hard. We then lift this NP-hardness lower bound to $\mathrm{P}_{\|}^{\mathrm{NP}}$-hardness, which proves our main results. This lifting requires a padding technique such that the given approximation ratio r is precisely met. In particular, to achieve $\mathrm{P}_{\|}^{\mathrm{NP}}$-hardness of $\mathcal{S}_r^{\mathrm{MDG}}$ for each rational number $r \geq 1$, we modify a construction by Papadimitriou and Steiglitz [PS82] that they use to

analyze the worst-case approximation behavior of the maximum-degree greedy heuristic.

2 Two Heuristics for the Vertex Cover Problem

We use the following notation. Fix the two-letter alphabet $\Sigma = \{0,1\}$. Σ^* is the set of all strings over Σ. Let $\langle \cdot, \cdot \rangle : \Sigma^* \times \Sigma^* \to \Sigma^*$ be a standard pairing function. For any set L, let $|L|$ denote the number of elements of L.

All graphs considered in this paper are undirected nonempty, finite graphs without multiple or reflexive edges. For any graph G, let $V(G)$ denote the set of vertices of G, and let $E(G)$ denote the set of edges of G. For any vertex $v \in V(G)$, the *degree of* v (denoted by $deg_G(v)$) is the number of vertices adjacent to v in G; if G is clear from the context, we omit the subscript and simply write $deg(v)$. Let $max\text{-}deg(G) = \max_{v \in V(G)} deg(v)$ denote the maximum degree of the vertices of graph G. Let G and H be two disjoint graphs. The *disjoint union of* G *and* H is defined to be the graph $U = G \cup H$ with vertex set $V(U) = V(G) \cup V(H)$ and edge set $E(U) = E(G) \cup E(H)$. The *join of* G *and* H is defined to be the graph $J = G \bowtie H$ with vertex set $V(J) = V(G) \cup V(H)$ and edge set $E(J) = E(G) \cup E(H) \cup \{\{x, y\} \mid x \in V(G) \ \wedge \ y \in V(H)\}$.

For any graph G, a subset $C \subseteq V(G)$ is a *vertex cover of* G if for all edges $\{v, w\} \in E(G)$, $\{v, w\} \cap C \neq \emptyset$. A vertex cover is said to be a *minimum vertex cover of* G if it is of minimum size. For any graph G, let $mvc(G)$ denote the size of a minimum vertex cover of G. The vertex cover problem (`VC`, for short; see [GJ79]) is defined to be the set of all pairs $\langle G, k \rangle$ such that G is a graph, k a positive integer, and $mvc(G) \leq k$.

All hardness and completeness results in this paper are with respect to the polynomial-time many-one reducibility, denoted $\leq_{\mathrm{m}}^{\mathrm{p}}$. For sets A and B, we say $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ if and only if there exists a polynomial-time computable function f such that for all inputs $x \in \Sigma^*$, $x \in A$ if and only if $f(x) \in B$.

We consider the following two heuristics (see, e.g., [PS82,Pap94]) for finding a minimum vertex cover of a given graph:

Edge Deletion Heuristic (ED): Given a graph G, the algorithm outputs a vertex cover C of G. Initially, C is the empty set. Nondeterministically choose an edge $\{u, v\} \in E(G)$, add both u and v to C, and delete u, v, and all edges incident to u and v from G. Repeat until there is no edge left in G.

Maximum-Degree Greedy Heuristic (MDG): Given a graph G, the algorithm outputs a vertex cover C of G. Initially, C is the empty set. Nondeterministically choose a vertex $v \in V(G)$ of maximum degree, add v to C, and delete v and all edges incident to v from G. Repeat until there is no edge left in G.

As mentioned in the introduction, these two heuristics have a quite different approximation behavior. While the worst-case ratio of the MDG algorithm is logarithmic in the input size [Pap94,Joh74], the ED algorithm always approximates the optimal solution within a factor of 2. Thus, despite its extreme simplicity, the

edge deletion heuristic achieves the best approximation ratio known for finding minimum vertex covers [Pap94].

The central question raised in this paper is: How hard is it to determine for which graphs G either of these two heuristics can approximate the minimum vertex cover of G within a factor of r, for a given rational number $r \geq 1$? Let *min-ed*(G) (respectively, *min-mdg*(G)) denote the minimum size of the output set of the ED algorithm (respectively, of the MDG algorithm) on input G, where the minimum is taken over all possible sequences of nondeterministic choices the algorithms can make. For any fixed rational $r \geq 1$, $\mathcal{S}_r^{\mathrm{ED}}$ (respectively, $\mathcal{S}_r^{\mathrm{MDG}}$) is the class of graphs for which ED (respectively, MDG) can output a vertex cover of size at most r times the size of a minimum vertex cover. Formally,

$$\mathcal{S}_r^{\mathrm{ED}} = \{G \mid G \text{ is a graph and } \mathit{min\text{-}ed}(G) \leq r \cdot \mathit{mvc}(G)\};$$
$$\mathcal{S}_r^{\mathrm{MDG}} = \{G \mid G \text{ is a graph and } \mathit{min\text{-}mdg}(G) \leq r \cdot \mathit{mvc}(G)\}.$$

We will prove that for each fixed rational number r with $1 \leq r < 2$, $\mathcal{S}_r^{\mathrm{ED}}$ is $\mathrm{P}_{\|}^{\mathrm{NP}}$-complete, and that for each fixed rational number $r \geq 1$, $\mathcal{S}_r^{\mathrm{MDG}}$ is $\mathrm{P}_{\|}^{\mathrm{NP}}$-complete. To this end, we give reductions from the problem $\mathtt{VC}_{\mathtt{geq}}$, which is defined by

$$\mathtt{VC}_{\mathtt{geq}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are graphs such that } \mathit{mvc}(G) \geq \mathit{mvc}(H)\}.$$

It is known that $\mathtt{VC}_{\mathtt{geq}}$ is $\mathrm{P}_{\|}^{\mathrm{NP}}$-complete, cf. Wagner [Wag87]. A reduction from any problem in $\mathrm{P}_{\|}^{\mathrm{NP}}$ to $\mathtt{VC}_{\mathtt{geq}}$ that in addition has some useful properties (see Lemma 1 below) can easily be obtained using the techniques of Wagner [Wag87]; see [SV00, Thm. 12] for an explicit proof of Lemma 1.

Lemma 1. (cf. [Wag87,SV00]) *For any set $X \in \mathrm{P}_{\|}^{\mathrm{NP}}$, there exists a polynomial-time computable function f that reduces X to $\mathtt{VC}_{\mathtt{geq}}$ in such a way that for each $x \in \Sigma^*$, $f(x) = \langle G, H\rangle$ is an instance of $\mathtt{VC}_{\mathtt{geq}}$ and*

$$x \in X \implies \mathit{mvc}(G) = \mathit{mvc}(H);$$
$$x \notin X \implies \mathit{mvc}(G) < \mathit{mvc}(H).$$

3 Hardness of Recognizing When the Edge Deletion Heuristic Can Approximate Minimum Vertex Covers

Lemma 2 below states that the vertex cover problem restricted to graphs in $\mathcal{S}_1^{\mathrm{ED}}$ is NP-hard. The proof of Lemma 2 can be found in the full version [HRS01] of this paper. The reduction g from Lemma 2 will be used in the proof of the main result of this section, Theorem 3. Define the problem

$$\mathtt{VC}\text{-}\mathcal{S}_1^{\mathrm{ED}} = \{\langle G, k\rangle \mid G \in \mathcal{S}_1^{\mathrm{ED}} \text{ and } k \in \mathbb{N}^+ \text{ and } \mathit{mvc}(G) \leq k\}.$$

Lemma 2. *There is a polynomial-time many-one reduction g from $\mathtt{VC}$ to $\mathtt{VC}\text{-}\mathcal{S}_1^{\mathrm{ED}}$ transforming any given graph G into a graph $H \in \mathcal{S}_1^{\mathrm{ED}}$ such that*

$$\mathit{mvc}(H) = 2(\mathit{mvc}(G) + |V(G)|). \tag{1}$$

Hence, $\mathtt{VC}\text{-}\mathcal{S}_1^{\mathrm{ED}}$ is NP*-hard.*

Theorem 3. *For each rational number r with $1 \leq r < 2$, $\mathcal{S}_r^{\mathrm{ED}}$ is $\mathrm{P}_{\|}^{\mathrm{NP}}$-complete.*

Proof. It is easy to see that $\mathcal{S}_r^{\mathrm{ED}}$ is in $\mathrm{P}_{\|}^{\mathrm{NP}}$. To prove $\mathrm{P}_{\|}^{\mathrm{NP}}$-hardness, let X be an arbitrary set in $\mathrm{P}_{\|}^{\mathrm{NP}}$, and let f be the reduction from X to $\mathtt{VC}_{\mathtt{geq}}$ stated in Lemma 1. Fix any rational number r with $1 \leq r < 2$, and let ℓ and m be integers such that $r = \frac{\ell}{m}$. Note that $1 \leq m \leq \ell < 2m$.

For any string $x \in \Sigma^*$, let $f(x) = \langle G_1, G_2 \rangle$. Since we can add isolated vertices to any graph G without altering $mvc(G)$, we may without loss of generality assume that $|V(G_1)| = |V(G_2)|$. Let g be the reduction from Lemma 2 that transforms any given graph G into a graph $H \in \mathcal{S}_1^{\mathrm{ED}}$ such that Equation (1) holds. Let $H_1 = g(G_1)$ and $H_2 = g(G_2)$. Thus, both H_1 and H_2 are in $\mathcal{S}_1^{\mathrm{ED}}$, and for $i \in \{1, 2\}$, we have $mvc(H_i) = 2(mvc(G_i) + |V(G_i)|)$.

We will define a graph $\widehat{H}$ and an integer $k \geq 0$ such that:

$$min\text{-}ed(\widehat{H}) = r(m \cdot mvc(H_2) + 2km); \tag{2}$$

$$mvc(\widehat{H}) = m \cdot mvc(H_1) + 2km. \tag{3}$$

The reduction mapping any given string x (via the pair $\langle G_1, G_2 \rangle$ obtained according to Lemma 1 and via the pair $\langle H_1, H_2 \rangle$ obtained according to Lemma 2) to the graph $\widehat{H}$ such that Equations (2) and (3) are satisfied will establish that $X \leq_{\mathrm{m}}^{\mathrm{p}} \mathcal{S}_r^{\mathrm{ED}}$. In particular, from these equations, we have that:

- $mvc(H_2) = mvc(H_1)$ implies $min\text{-}ed(\widehat{H}) = r \cdot mvc(\widehat{H})$, and
- $mvc(H_2) > mvc(H_1)$ implies $min\text{-}ed(\widehat{H}) > r \cdot mvc(\widehat{H})$.

Note that, due to Lemma 1, $mvc(H_2) \geq mvc(H_1)$.

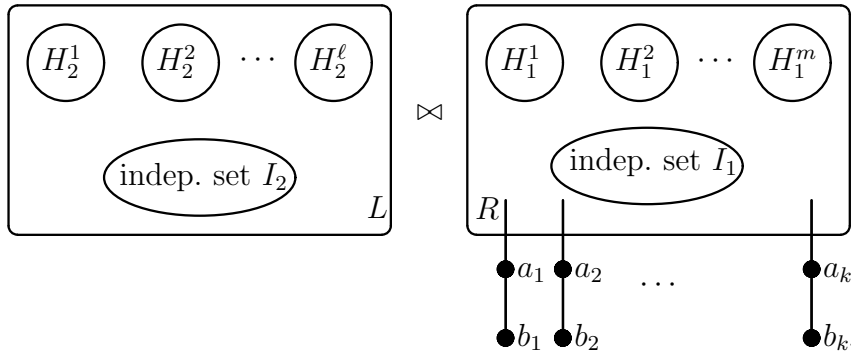


Fig. 1. The graph $\widehat{H}$ constructed from H_1 and H_2.

Look at Figure 1 for the construction of $\widehat{H}$ from H_1 and H_2. The graph $\widehat{H}$ consists of two subgraphs, L and R, that are joined by the join operation, plus some additional vertices and edges that are connected to R. Formally, let

$H_1^1, H_1^2, \ldots, H_1^m$ be m pairwise disjoint copies of H_1, and let $H_2^1, H_2^2, \ldots, H_2^\ell$ be ℓ pairwise disjoint copies of H_2. Let $k = \ell|V(H_2)| + m|V(H_1)|$. Let I_1 and I_2 be independent sets such that L contains exactly $k(2m-\ell)$ vertices and R exactly $k\ell$ vertices. (This is possible, because $k(2m-\ell) - \ell|V(H_2)|$ is not negative, since $2m - \ell \geq 1$, and $k\ell - m|V(H_1)|$ is not negative, since $\ell \geq 1$.) Let $e_i = \{a_i, b_i\}$ $(1 \leq i \leq k\ell)$ be additional edges. Every vertex a_i is adjacent to exactly one vertex in R, and each vertex in R is adjacent to exactly one vertex a_i. a_i and b_i are not adjacent to any other vertices.

1. We first determine $\mathit{min\text{-}ed}(\widehat{H})$. Let $\widehat{E}$ be a fixed minimum-size output set of the ED algorithm on input $\widehat{H}$, i.e., $\mathit{min\text{-}ed}(\widehat{H}) = |\widehat{E}|$. Since $\widehat{E}$ is a vertex cover of $\widehat{H}$, $\widehat{E}$ must contain a_i or b_i for each $i \in \{1, \ldots, k\ell\}$. Since the ED-algorithm can delete only edges, and $\widehat{E}$ is a minimum-size output set, it follows that $\widehat{E}$ contains all vertices a_i, all vertices from R, and no vertex b_i. Let C_L be a minimum-size output set of the ED-algorithm on input L. By construction of L, $|C_L| = \ell \cdot \mathit{min\text{-}ed}(H_2)$. Thus, since $H_2 \in \mathcal{S}_1^{\mathrm{ED}}$, $|C_L| = \ell \cdot \mathit{mvc}(H_2)$.
   Define $\widehat{E}' = V(R) \cup C_L \cup \bigcup_{i=1}^{k\ell} \{a_i\}$. It is easy to see that $\widehat{E}'$ is a minimum-size output set of the ED algorithm on input $\widehat{H}$. Hence,

$$\begin{aligned} \mathit{min\text{-}ed}(\widehat{H}) &= 2k\ell + \ell \cdot \mathit{mvc}(H_2) \\ &= r(2km + m \cdot \mathit{mvc}(H_2)). \end{aligned}$$

   This proves Eq. (2).
2. We now determine $\mathit{mvc}(\widehat{H})$. Let $\widehat{C}$ be a fixed minimum vertex cover of $\widehat{H}$, i.e., $\mathit{mvc}(\widehat{H}) = |\widehat{C}|$. Distinguish the following two cases.

   Case 1: $V(R) \subseteq \widehat{C}$. In this case, $\widehat{C}$ contains all vertices from R, at least one of a_i or b_i for each i, $1 \leq i \leq k\ell$, and a minimum vertex cover of L. Hence,

$$\mathit{mvc}(\widehat{H}) = 2k\ell + \ell \cdot \mathit{mvc}(H_2).$$

   Case 2: $V(L) \subseteq \widehat{C}$. In this case, $\widehat{C}$ contains all vertices from L, each vertex a_i, $1 \leq i \leq k\ell$, and a minimum vertex cover of R. Hence,

$$\begin{aligned} \mathit{mvc}(\widehat{H}) &= k(2m-\ell) + k\ell + m \cdot \mathit{mvc}(H_1) \\ &= 2km + m \cdot \mathit{mvc}(H_1). \end{aligned}$$

   Since $\mathit{mvc}(H_1) \leq \mathit{mvc}(H_2)$, $m \leq \ell$, and $2km \leq 2k\ell$, it follows that

$$\mathit{mvc}(\widehat{H}) = 2km + m \cdot \mathit{mvc}(H_1).$$

   This proves Eq. (3). □

4 Hardness of Recognizing When the Maximum-Degree Greedy Heuristic Can Approximate Minimum Vertex Covers

Lemma 4 below states that the vertex cover problem restricted to graphs in $\mathcal{S}_1^{\mathrm{MDG}}$ is NP-hard. The proof of Lemma 4 can be found in the full version [HRS01] of this paper; it is reminiscent of a proof by Bodlaender et al. [BTY97, Thm. 4], who show that the independent set problem restricted to graphs for which the minimum-degree greedy heuristic can find an optimal solution is NP-hard. The reduction g from Lemma 4 will be used in the proof of the main result of this section, Theorem 6. Define the problem

$$\texttt{VC-}\mathcal{S}_1^{\mathrm{MDG}} = \{\langle G, k\rangle \mid G \in \mathcal{S}_1^{\mathrm{MDG}} \text{ and } k \in \mathbb{N}^+ \text{ and } mvc(G) \leq k\}.$$

Lemma 4. *There is a polynomial-time many-one reduction g from* VC *to* $\texttt{VC-}\mathcal{S}_1^{\mathrm{MDG}}$ *transforming any given graph G into a graph $H \in \mathcal{S}_1^{\mathrm{MDG}}$ such that*

$$mvc(H) = mvc(G) + |E(G)|(max\text{-}deg(G) + 1). \tag{4}$$

Hence, $\texttt{VC-}\mathcal{S}_1^{\mathrm{MDG}}$ *is* NP*-hard.*

Lemma 5 below will be used in the proof of Theorem 6. The proof of Lemma 5 can be found in the full version [HRS01] of this paper. The construction of the graph G in this lemma is a modification of a construction given by Papadimitriou and Steiglitz [PS82, p. 408, Fig. 17-3], which shows that the worst-case approximation ratio of the MDG heuristic can be as bad as logarithmic in the input size, and so grows unboundedly. Similar constructions for achieving the worst-case approximation behavior of the greedy heuristic solving the more general minimum set cover problem were given by Johnson [Joh74], Lovász [Lov75], and Chvátal [Chv79].

Lemma 5. *For all positive integers n_1, n_2, δ, and μ satisfying*

$$\mu(\ln \mu - 2\ln(\delta + 2) - 1) \geq n_1 + n_2, \tag{5}$$

there exists a bipartite graph G with the following properties:

1. *$V(G) = V \cup \tilde{V}$ such that $V \cap \tilde{V} = \emptyset$ and both V and $\tilde{V}$ are independent sets, where*
   - *$V = \{u_1, u_2, \ldots, u_{n_1}, w_1, w_2, \ldots, w_\mu, z_1, z_2, \ldots z_{n_2}\}$ and*
   - *$\tilde{V} = \{\tilde{u}_1, \tilde{u}_2, \ldots, \tilde{u}_{n_1}, \tilde{w}_1, \tilde{w}_2, \ldots, \tilde{w}_\mu\}$.*
2. *$\{\{u_i, \tilde{u}_i\} \mid 1 \leq i \leq n_1\} \cup \{\{w_i, \tilde{w}_i\} \mid 1 \leq i \leq \mu\} \subseteq E(G)$.*
3. *Every vertex $\tilde{u}_i$, where $1 \leq i \leq n_1$, has degree 1.*
4. *For each induced subgraph S of G that can be obtained by deleting vertices from V such that $V \cap V(S) \neq \emptyset$, it holds that $\max_{v \in V \cap V(S)} deg_S(v) > \max_{v \in \tilde{V}} deg_S(v) + \delta$.*

Theorem 6. *For each rational number $r \geq 1$, $\mathcal{S}_r^{\mathrm{MDG}}$ is $\mathrm{P}_{\|}^{\mathrm{NP}}$-complete.*

Proof. It is easy to see that $\mathcal{S}_r^{\mathrm{MDG}}$ is in $\mathrm{P}_{\|}^{\mathrm{NP}}$. To prove $\mathrm{P}_{\|}^{\mathrm{NP}}$-hardness of $\mathcal{S}_r^{\mathrm{MDG}}$, let X be an arbitrary set in $\mathrm{P}_{\|}^{\mathrm{NP}}$, and let f be the reduction from X to $\mathtt{VC}_{\mathtt{geq}}$ stated in Lemma 1. For any string $x \in \Sigma^*$, let $f(x) = \langle G_1, G_2 \rangle$.

It is convenient to consider the special case of $r = 1$ and the case of $r > 1$ separately in the proof of Theorem 6. We start by proving that $\mathcal{S}_1^{\mathrm{MDG}}$ is $\mathrm{P}_{\|}^{\mathrm{NP}}$-complete. We will define a graph $\widehat{G}$ and an integer $q \geq 0$ such that:

$$\mathit{min\text{-}mdg}(\widehat{G}) = \mathit{mvc}(G_2) + q; \tag{6}$$

$$\mathit{mvc}(\widehat{G}) = \mathit{mvc}(G_1) + q. \tag{7}$$

The reduction mapping any given string x (via the pair $\langle G_1, G_2 \rangle$ obtained according to Lemma 1) to the graph $\widehat{G}$ such that Equations (6) and (7) are satisfied will establish that $X \leq_{\mathrm{m}}^{\mathrm{p}} \mathcal{S}_1^{\mathrm{MDG}}$. In particular, from these equations, we have that:

- $\mathit{mvc}(G_2) = \mathit{mvc}(G_1)$ implies $\mathit{min\text{-}mdg}(\widehat{G}) = \mathit{mvc}(\widehat{G})$, and
- $\mathit{mvc}(G_2) > \mathit{mvc}(G_1)$ implies $\mathit{min\text{-}mdg}(\widehat{G}) > \mathit{mvc}(\widehat{G})$.

Note that, due to Lemma 1, $\mathit{mvc}(G_2) \geq \mathit{mvc}(G_1)$.

We now describe the construction of $\widehat{G}$. Let g be the reduction from Lemma 4 and let $H_2 = g(G_2)$. Thus, H_2 is in $\mathcal{S}_1^{\mathrm{MDG}}$ and, by Equation (4),

$$\mathit{mvc}(H_2) = \mathit{mvc}(G_2) + |E(G_2)|(\mathit{max\text{-}deg}(G_2) + 1). \tag{8}$$

Since one can add isolated vertices to any graph G without affecting the values of $\mathit{mvc}(G)$ or $\mathit{min\text{-}mdg}(G)$, we may without loss of generality assume that

$$|V(H_2)| = |V(G_1)| + |E(G_2)|(\mathit{max\text{-}deg}(G_2) + 1). \tag{9}$$

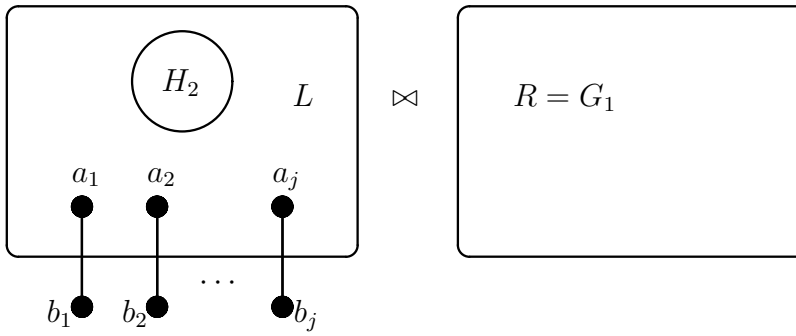


Fig. 2. The graph $\widehat{G}$ constructed from G_1 and H_2.

Look at Figure 2 for the construction of $\widehat{G}$ from G_1 and H_2. The graph $\widehat{G}$ consists of two subgraphs, L and R, that are joined by the join operation, plus

some additional vertices and edges that are connected to L. Formally, choose $2j$ new vertices a_i and b_i, $1 \leq i \leq j$, where j is a fixed integer large enough such that the degree of each vertex in R is larger than the maximum degree of the vertices in L. Note that the degree of each vertex in R must remain larger than the degree of any vertex in L even after some vertices have been removed from R.

Let B be the bipartite matching with the vertex set $V(B) = \{a_i \mid 1 \leq i \leq j\} \cup \{b_i \mid 1 \leq i \leq j\}$ and the edge set $E(B) = \{\{a_i, b_i\} \mid 1 \leq i \leq j\}$. Let $R = G_1$, and let L be the graph with the vertex set $V(L) = \{a_i \mid 1 \leq i \leq j\} \cup V(H_2)$ and the edge set $E(L) = E(H_2)$. The graph $\widehat{G}$ is defined by forming the join $L \bowtie R$, i.e., there are edges connecting each vertex of L with each vertex of R, plus attaching the vertices b_i, $1 \leq i \leq j$, to L by adding the j edges from $E(B)$.

We first consider $\mathit{min\text{-}mdg}(\widehat{G})$. By our choice of j, each vertex in R has a degree larger than the degree of any vertex not in R. Hence, on input $\widehat{G}$, the MDG algorithm first deletes all vertices from R. Subsequently, it can find a minimum vertex cover of H_2, which has size $\mathit{mvc}(G_2)+|E(G_2)|(\mathit{max\text{-}deg}(G_2)+1)$ by Equation (8), and eventually it can choose, say, the vertices a_i, $1 \leq i \leq j$, to cover the edges of B. Hence,

$$\begin{aligned} \mathit{min\text{-}mdg}(\widehat{G}) &= |V(G_1)| + \mathit{mvc}(G_2) + |E(G_2)|(\mathit{max\text{-}deg}(G_2) + 1) + j \\ &\overset{(9)}{=} \mathit{mvc}(G_2) + |V(H_2)| + j. \end{aligned}$$

We now consider $\mathit{mvc}(\widehat{G})$. Since every vertex cover of $\widehat{G}$ must contain all vertices of L or all vertices of R to cover the edges connecting L and R, it follows from Equations (8) and (9) that:

$$\begin{aligned} \mathit{mvc}(\widehat{G}) &= \min\{|V(G_1)| + \mathit{mvc}(H_2) + j,\ |V(H_2)| + j + \mathit{mvc}(G_1)\} \\ &= \min\{\mathit{mvc}(G_2) + |V(H_2)| + j,\ \mathit{mvc}(G_1) + |V(H_2)| + j\}. \end{aligned}$$

Since $\mathit{mvc}(G_2) \geq \mathit{mvc}(G_1)$, it follows that

$$\mathit{mvc}(\widehat{G}) = \mathit{mvc}(G_1) + |V(H_2)| + j.$$

Hence, setting $q = |V(H_2)| + j$, Equations (6) and (7) are satisfied, which completes the proof that $\mathcal{S}_1^{\mathrm{MDG}}$ is $\mathrm{P}_{\|}^{\mathrm{NP}}$-complete.

We now turn to the proof that $\mathcal{S}_r^{\mathrm{MDG}}$ is $\mathrm{P}_{\|}^{\mathrm{NP}}$-complete for $r > 1$. Fix any rational number $r = \frac{\ell}{m}$, where ℓ and m are integers with $1 \leq m < \ell$. Without loss of generality, we may assume that $\gcd(\ell - m, m) = 1$, where $\gcd(a, b)$ denotes the greatest common divisor of the integers a and b. Recall that the pair $\langle G_1, G_2 \rangle = f(x)$ of graphs is obtained using the reduction f from X to $\mathtt{VC}_{\mathtt{geq}}$ according to Lemma 1; hence, $\mathit{mvc}(G_2) \geq \mathit{mvc}(G_1)$.

We will define a graph $\widehat{G}_r$ and integers $p, q \geq 0$ such that:

$$\mathit{min\text{-}mdg}(\widehat{G}_r) = r(p \cdot \mathit{mvc}(G_2) + q); \tag{10}$$

$$\mathit{mvc}(\widehat{G}_r) = p \cdot \mathit{mvc}(G_1) + q. \tag{11}$$

The reduction mapping any given string x (via the pair $\langle G_1, G_2\rangle$ obtained according to Lemma 1) to the graph $\widehat{G}_r$ such that Equations (10) and (11) are satisfied will establish that $X \leq_{\mathrm{m}}^{\mathrm{p}} \mathcal{S}_r^{\mathrm{MDG}}$. In particular, from these equations, we have that:

- $mvc(G_2) = mvc(G_1)$ implies $min\text{-}mdg(\widehat{G}_r) = r \cdot mvc(\widehat{G}_r)$, and
- $mvc(G_2) > mvc(G_1)$ implies $min\text{-}mdg(\widehat{G}_r) > r \cdot mvc(\widehat{G}_r)$.

We now describe the construction of $\widehat{G}_r$:

- Let g be the reduction from Lemma 4 and let $H_2 = g(G_2)$. Thus, $H_2 \in \mathcal{S}_1^{\mathrm{MDG}}$ and Equation (8) holds:

$$mvc(H_2) = mvc(G_2) + |E(G_2)|(max\text{-}deg(G_2) + 1).$$

- Let $G_1^1, G_1^2, \ldots, G_1^m$ be m pairwise disjoint copies of G_1, and let $H_2^1, H_2^2, \ldots, H_2^\ell$ be ℓ pairwise disjoint copies of H_2.
- Let $\tilde{U} = \bigcup_{i=1}^{\ell} H_2^i$ be the disjoint union of these copies of H_2, and rename the vertices of $\tilde{U}$ by $V(\tilde{U}) = \{\tilde{u}_1, \tilde{u}_2, \ldots, \tilde{u}_{\ell \cdot |V(H_2)|}\}$.
- Let $Z = \bigcup_{i=1}^{m} G_1^i$ be the disjoint union of these copies of G_1, and rename the vertices of Z by $V(Z) = \{z_1, z_2, \ldots, z_{m \cdot |V(G_1)|}\}$.
- To apply Lemma 5, choose $n_1 = \ell \cdot |V(H_2)|$, $n_2 \geq m \cdot |V(G_1)|$, and $\delta = max\text{-}deg(H_2) + 1$, where the exact value of n_2 will be specified below. Choose the constant μ so as to satisfy Equation (5):

$$\mu(\ln \mu - 2\ln(\delta + 2) - 1) \geq n_2 + n_1.$$

- Given the constants n_1, n_2, δ, and μ, define $\widehat{G}_r$ to be the bipartite graph G from Lemma 5 extended by the edges between the $\tilde{u}_i$ vertices that were added above to represent the structure of the copies of H_2, and extended by the edges between the z_j vertices that were added above to represent the structure of the copies of G_1. That is, unlike G, the graph $\widehat{G}_r$ is no longer a bipartite graph. Formally, the vertex set of $\widehat{G}_r$ is given by

$$\begin{aligned} V(\widehat{G}_r) = V(G) &= V \cup \tilde{V}, \quad \text{where} \\ V &= \{u_1, u_2, \ldots, u_{n_1}, w_1, w_2, \ldots, w_\mu, z_1, z_2, \ldots z_{n_2}\} \quad \text{and} \\ \tilde{V} &= \{\tilde{u}_1, \tilde{u}_2, \ldots, \tilde{u}_{n_1}, \tilde{w}_1, \tilde{w}_2, \ldots, \tilde{w}_\mu\}, \end{aligned}$$

  and the edge set of $\widehat{G}_r$ is given by $E(\widehat{G}_r) = E(G) \cup E(\tilde{U}) \cup E(Z)$, where $E(G)$ is constructed as in the proof of Lemma 5.

This completes the construction of $\widehat{G}_r$. We now prove Equations (10) and (11).

1. We first consider $min\text{-}mdg(\widehat{G}_r)$. By construction, for each vertex v in $\tilde{V}$, we have

$$deg_{\widehat{G}_r}(v) \leq deg_G(v) + max\text{-}deg(H_2) < deg_G(v) + \delta. \tag{12}$$

Let S be any induced subgraph of $\widehat{G}_r$ that can be obtained by deleting vertices from V such that $V \cap V(S) \neq \emptyset$. Property 4 of Lemma 5 and Equation (12) imply that

$$\max_{v \in V \cap V(S)} deg_S(v) > \max_{v \in \tilde{V}} deg_S(v).$$

Hence, on input $\widehat{G}_r$, the MDG algorithm starts by choosing the $n_1 + \mu + n_2$ vertices from V, which isolates each vertex $\tilde{w}_i \in \tilde{V}$ and leaves ℓ isolated copies of H_2. Subsequently, since $H_2 \in \mathcal{S}_1^{\mathrm{MDG}}$, the MDG algorithm can choose a minimum vertex cover in each of these ℓ copies of H_2. By Equation (8),

$$mvc(H_2) = mvc(G_2) + |E(G_2)|(max\text{-}deg(G_2) + 1),$$

and hence,

$$min\text{-}mdg(\widehat{G}_r) = n_1 + \mu + n_2 + \ell(mvc(G_2) + |E(G_2)|(max\text{-}deg(G_2) + 1)).$$

2. We now consider $mvc(\widehat{G}_r)$. Define the set $C = \tilde{V} \cup D$, where D with $|D| = m \cdot mvc(G_1)$ is a minimum vertex cover of Z. It is obvious from the construction of $\widehat{G}_r$ that C is a minimum vertex cover of $\widehat{G}_r$. Hence,

$$mvc(\widehat{G}_r) = n_1 + \mu + m \cdot mvc(G_1).$$

To complete the proof, we have to choose $n_2 \geq m \cdot |V(G_1)|$ such that Equations (10) and (11) are satisfied for suitable integers p and q. Setting $p = m$ and $q = n_1 + \mu$ and requiring

$$n_1 + n_2 + \mu + \ell \cdot |E(G_2)|(max\text{-}deg(G_2) + 1) = r(n_1 + \mu) \tag{13}$$

or, equivalently,

$$m \cdot n_2 + m \cdot \ell \cdot |E(G_2)|(max\text{-}deg(G_2) + 1)) = (\ell - m)n_1 + (\ell - m)\mu \tag{14}$$

satisfies Equations (10) and (11). Our assumption that $\gcd(\ell - m, m) = 1$ implies that Equation (14) has integer solutions in the variables n_2 and μ. It is easy to see that one such solution, say (n_2, μ), simultaneously (a) satisfies Equation (5), (b) satisfies that both n_2 and μ are polynomially bounded in the size of the input of the reduction being described, and (c) can be computed efficiently [CF89]. This completes the proof of the theorem. □

Acknowledgments

We thank Dieter Kratsch and Andreas Brandstädt for interesting discussions on graph theory and graph-theoretical notation.

References

BTY97. H. Bodlaender, D. Thilikos, and K. Yamazaki. It is hard to know when greedy is good for finding independent sets. *Information Processing Letters*, 61:101–106, 1997.

CF89. M. Clausen and A. Fortenbacher. Efficient solution of linear diophantine equations. *Journal of Symbolic Computation*, 8(1/2):201–216, 1989.

Chv79. V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

Fei98. U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, July 1998.

GJ79. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

Hem89. L. Hemachandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39(3):299–322, 1989.

HHR97. E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Exact analysis of Dodgson elections: Lewis Carroll's 1876 voting system is complete for parallel access to NP. *Journal of the ACM*, 44(6):806–825, 1997.

HR98. E. Hemaspaandra and J. Rothe. Recognizing when greed can approximate maximum independent sets is complete for parallel access to NP. *Information Processing Letters*, 65(3):151–156, February 1998.

HRS01. E. Hemaspaandra, J. Rothe, and H. Spakowski. Recognizing when heuristics can approximate minimum vertex covers is complete for parallel access to NP. Technical Report cs.CC/0110025, Computing Research Repository (CoRR), October 2001. 16 pages. Available on-line at http://xxx.lanl.gov/abs/cs.CC/0110025.

Joh74. D. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974.

Kre88. M. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36:490–509, 1988.

KSW87. J. Köbler, U. Schöning, and K. Wagner. The difference and truth-table hierarchies for NP. *R.A.I.R.O. Informatique théorique et Applications*, 21:419–435, 1987.

Lov75. L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

LY94. C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, September 1994.

Pap94. C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

PS82. C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

PZ83. C. Papadimitriou and S. Zachos. Two remarks on the power of counting. In *Proceedings of the 6th GI Conference on Theoretical Computer Science*, pages 269–276. Springer-Verlag *Lecture Notes in Computer Science #145*, 1983.

SV00. H. Spakowski and J. Vogel. Θ_2^p-completeness: A classical approach for new results. In *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 348–360. Springer-Verlag *Lecture Notes in Computer Science #1974*, December 2000.

Wag87. K. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theoretical Computer Science*, 51:53–80, 1987.

Wag90. K. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19(5):833–846, 1990.

Complexity of Some Infinite Games Played on Finite Graphs

Hajime Ishihara[1] and Bakhadyr Khoussainov[2]

[1] Japan Advanced Institute of Science and Technology, Japan
ishihara@jaist.ac.jp
[2] Computer Science Department, The University of Auckland, New Zealand
bmk@cs.auckland.ac.nz

1 Introduction and Basic Concepts

Games played on finite graphs were first introduced by McNaughton in [6]. McNaughton, using the ideas of the paper by Gurevich and Harrington [3], proved that winners in his games have finite state winning strategies. Later based on McNaughton games, Nerode, Remmel and Yakhnis in a series of papers (see [7] and [8], for example) developed foundations of concurrent programming by identifying distributed concurrent programs with finite state strategies and studied complexities of finding winners in McNaughton games. Dinneen and Khoussainov use McNaughton games for modelling and studying structural and complexity-theoretical properties of update networks (see [1]). Later in [2] Bodlaender, Dinneen and Khoussainov generalize the study of update networks by introducing the concept of relaxed update network. They proved that it is possible to detect in polynomial time whether or not a given game represents a relaxed update network. In this paper we continue the line of research of the above mentioned work and begin with the following definition from [6]:

Definition 1. *A* **game** Γ *is a tuple* $(S \bigcup A, E, W, \Omega)$*, where:*

1. *The sets* S *and* A *are disjoint and finite, where* S *is the set of positions for Survivor and* A *is the set of positions for Adversary,*
2. *The set* E *of edges is such that* $E \subseteq A \times S \bigcup S \times A$ *and for all* $s \in S$ *and* $a \in A$ *there are* $a' \in A$ *and* $s' \in S$ *for which* $(s, a'), (a, s') \in E$*,*
3. *The set* W *is a subset of* $S \bigcup A$ *and* $\Omega \subseteq 2^W$*.*

The graph $\mathcal{G} = (V, E)$*, where* $V = S \cup A$*, is called the* **system** *or the* **graph of the game***, the pair* (W, Ω) *is the* **specification***, and each set* $U \in \Omega$ *is a* **winning set***.*

In game Γ, a **play (from** p_0**)** is an infinite sequence $\pi = p_0, p_1, \ldots$ such that $(p_i, p_{i+1}) \in E$, $i \in \omega$. Consider the set $Inf(\pi)$ consisting of those positions $p \in W$ that appear infinitely often in play π. Survivor **wins** the play if $Inf(\pi) \in \Omega$; otherwise, Adversary wins. The **histories** of the play π are its finite prefixes. The set $H(S)$ consists of all histories of all plays whose last positions are in S. The set $H(A)$ is defined similarly. A **strategy** for *Survivor* is a function $f : H(S) \to A$

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 270–281, 2002.

such that $(q_n, f(u)) \in E$ for all $u = q_0 \dots q_n \in H(S)$. A **strategy** for *Adversary* is defined similarly.

Let f be a strategy for a player and p be a position. Consider all the plays that begin from p which are played when the player follows the strategy f. We call these **plays consistent with f from p.**

Definition 2. *The strategy f of a player is a* **winning strategy** *from position p if all plays consistent with f from p are won by the player. In this case we say that the player* **wins the game** *from p.* **To decide game** *Γ means to find all the positions q in the game from which Survivor wins. We denote this set of position by $Win(S)$. The set $Win(A)$ is defined similarly*[1]*.*

In [6] McNaughton proved that there is an algorithm that decides any given McNaughton game. McNaughton's algorithm is quite inefficient, however. In [7] Nerode, Remmel and Yakhnis improved McNaughton algorithm by showing that it takes $O(|W|!2^{|W|}|W||E|)$ time to decide a given game Γ. Thus, a natural question arises as to under which conditions, put either on the specifications or the systems, the games can be decided more efficiently. Here is a list of some results and definitions related to this question.

A natural specification is to require Survivor to update every node of the system infinitely often. This is formalized as follows.

Definition 3. *A game Γ is an* **update game** *if $W = V$ and $\Omega = \{V\}$. If Survivor wins this game, then we call this game an* **update network**.

Update games can be decided in polynomial time as shown in [1]:

Theorem 1. *There exists an algorithm that given a game decides in $O(|V||E|)$ time whether or not the game is an update network.*

We use this theorem (most of the time without a reference to it) in obtaining several results of this paper. The proof of this theorem is based on finding certain structural properties of update networks. In [2] Theorem 1 has been generalized.

Definition 4. *A game Γ is a* **relaxed update game** *if $U \bigcap V = \emptyset$ for all distinct $U, V \in \Omega$. If Survivor wins this game, then we call Γ a* **relaxed update network**.

In [2], similar to the notion of rank defined in Gurevich-Harrington [3], certain natural concepts (such as forcing) are introduced for the investigation of relaxed update games, and the following theorem is proved:

Theorem 2. *There exists an algorithm that given a game decides in $O(|V|^2|E|)$-time whether or not the game is a relaxed update network.*

[1] Any McNaughton game Γ is a Borel game. Hence, by the known result of Martin, Γ is determined (see [5]). Therefore $Win(S) \bigcup Win(A) = S \bigcup A$.

In this paper our goal is three fold. Firstly, we generalize the theorems above by greater exploiting the ideas of the proofs of Theorem 1 and Theorem 2. Secondly, we give other types of natural specifications when the winners can be determined in polynomial time with a parameter. Finally, we study the interactions between efficent winning strategies, complexity of finding such strategies, and the structural properties of the underlying graphs for games with an emphasis to update games. We also briefly consider the relationship between McNaughton games and temporal logic.

2 Preliminary Results

Given a game Γ and a subset $X \subseteq V$, a node v is in the set $\mathrm{REACH}(S, X)$ if *Survivor* can force every play starting at v into X after a finite number of steps.

Lemma 1. *[2] The set REACH(S, X) can be computed in $O(|V| + |E|)$ time.*

Proof. We build a set R, that will eventually be $\mathrm{REACH}(S, X)$. Initially, $R = X$. If a node $x \in S$ has an edge to a node in R, then x is added to R. If a node $x \in A$ has only edges to nodes in R, then x is added to R. From every node in R *Survivor* can force plays to go to a node in X. When no nodes can be added to R anymore, then $\mathrm{REACH}(S, X)$=R. *Adversary* has a strategy to stay inside $V \backslash \mathrm{REACH}(S, X)$ when game begins in a node from $V \backslash \mathrm{REACH}(S, X)$. The procedure of constructing $\mathrm{REACH}(S, X)$ can be implemented in $O(|V|+|E|)$ time, by giving each node not in X a counter, that is initially 1 for nodes owned by *Survivor* and its outdegree for nodes owned by *Adversary*. Whenever we add a node v to R, we subtract 1 from the counters of each node with an edge to v; when a counter becomes 0 then the node is also added to R. □

Let $v \notin \mathrm{REACH}(S, X)$. Iteratively define the set $\mathrm{AVOID}(v, A, X)$: initially, $\mathrm{AVOID}(v, A, X)$=$\{v\}$. For $x \in A \cap \mathrm{AVOID}(v, A, X)$ we add a neighbor y of x into $\mathrm{AVOID}(v, A, X)$ if $(x, y) \in E$ and $y \notin \mathrm{REACH}(S, X)$. For $x \in S \cap \mathrm{AVOID}(v, A, X)$ we add all neighbors of x into $\mathrm{AVOID}(v, A, X)$. From Lemma 1 we obtain the following:

Lemma 2. *The set AVOID(v, A, X) has the following properties:*

1. *AVOID(v, A, X) can be constructed in $O(|V| + |E|)$ time.*
2. *AVOID$(v, A, X) \cap$ REACH$(S, X) = \emptyset$.*
3. Adversary *has a strategy such that when a play visits a node in AVOID(v, A, X) then all nodes visited afterwards are in AVOID(v, A, X).*
4. *For all s in AVOID$(v, A, X) \cap S$ and all $a \in A$ if $(s, a) \in E$ then a is in AVOID(v, A, X).*

Note that the sets $\mathrm{REACH}(A, X)$ and $\mathrm{AVOID}(v, S, X)$ can be defined in a similar matter. We will now need some notations that will be used later. The strategy for a player P to force the plays into X from a node v will be denoted by $Force_X^{P,v}$. Similarly, the strategy that keeps out all the plays from v to enter the set X will be denoted $Avoid_X^{P,v}$.

3 Games with Separable Winning Sets

Given a system we would not like the system to enter useless states during a computation. This naturally suggests that the specification (W, Ω) of the game to be such that $W = S \bigcup A$. Another natural assumption on the specification is that each winning condition U to be distinguishable from all other winning condition U' in Ω. We formalize this as follows:

Definition 5. *Game Γ is* **fully separated** *if $W = S \bigcup A$ and for each $U \in \Omega$ there is a s_U, called* **separator***, such that $s_U \in U$ but $s_U \notin U'$ for all $U'\Omega$ distinct from U.*

Thus, the separator s_U for $U \in \Omega$ can be thought as a certificate of the winning set U. Now our goal is to provide a polynomial time algorithm that decides fully separted games. We use techniques developed in [1] and [2].

Definition 6. *[2] A winning set $U \in \Omega$ is* **S-closed** *if it satisfies the following conditions:*

1. *For each $s \in U \cap S$ there is an $a \in U \cap A$ such that $(s, a) \in E$.*
2. *For each $a \in U \cap A$ and all s such that $(a, s) \in E$ we have $s \in U$.*

Thus, this definition informally tells us the following. If a play arrives to a node v in an S-closed set U then Survivor is able to always keep all the plays after v inside U no matter what the oppenent does. Here is a lemma which is true for any McNaughton game and therefore is of independent interest:

Lemma 3. *Let Γ be a McNaughton game. Then if Survivor wins Γ from a position p then one of the winning sets U in Ω must be S-closed.*

Proof. In order to prove the lemma, we assume the opposite and then construct a winning strategy for Adversary thus contradicting the assumption of the lemma.

Since each $U \in \Omega$ is not S-closed, there is a $p_U \in U$ that satisfies one of the following conditions:

1. $p_U \in S$ and for all $a \in A$ if $(p_U, a) \in E$ then $a \notin U$.
2. $p_U \in A$ and there is an s such that $(p_U, s) \in E$ and $s \notin U$.

We call p_U a **witness**. Here is a strategy for Adversray. Let $p_0, \ldots, p_n$ be a finite play. If p_n is not a witness and $p_n \in A$ then Adversray moves to any node s such that $(p_n, s) \in E$. Assume that p_n is a witness. Let $U_0, \ldots, U_{k-1}$ be *all* winning sets in Ω for which p_n is a witness. Note that if $p_n \in S$ then for every a such that $(p_n, a) \in E$ we have $a \notin U_0 \cup \ldots \cup U_{k-1}$. Assume now that $p_n \in A$. Let i be the number of times p_n appears in the finite play $p_0, \ldots, p_n$. Then Adversray moves to any s that does not belong to $U_{i+1(mod(k))}$. The basic idea is that, once p_n is reached, Adversary leaves the sets $U_0, \ldots, U_{k-1}$ turn by turn making in a cyclic manner.

We claim that the strategy described is a winning strategy for Adversary. Indeed let $\pi = p_0, p_1, \ldots$ be a play consistent with the strategy. Assume that

$In(\pi) = U$ and $U \in \Omega$. Let n be the first position after which no nodes outside of U appear in π. Then since a witness p_U appeares in $In(\pi)$ infinitely many times it must be the case that p_U appears in π after position n. Hence, by the definition of the strategy we described, there is a position $j > n$ such that $p_j \notin U$. This contradicts with the choice of n. The lemma is proved. □

We now need the following lemma that characterizes update networks (see Definition 3) in terms of the sets REACH(S, X).

Lemma 4. *Survivor wins an update game Γ if and only if $x \in REACH(S, \{y\})$ for all $x, y \in V$.*

Proof. Assume that $x \in \text{REACH}(\{y\})$ for all $x, y \in V$. List all the nodes $v_0, \ldots, v_n$. Survivor cycles by forcing the plays to visit v_0, then v_1, etc. This shows that Survivor wins the game Γ.

Assume that there exists x, y such that $x \notin \text{REACH}(S, \{y\})$. Then Adversray uses the strategy $Avoid_y^{A,x}$ as soon as a play is at the node x. By by Part 3 of Lemma 2 Adversary wins the game. The lemma is proved. □

The next lemma gives another sufficient condition for Adversary to win a fully separated game.

Lemma 5. *Let Γ be a fully separated game Γ such that every $U \in \Omega$ satisfies one of the following properties:*

1. *U is not S-closed,*
2. *U is S-closed and Adversary wins the update game $(U, \{U\})$.*

Then Adversray wins the game Γ.

Proof. By the lemma above, if $U \in \Omega$ is an S-closed set and $(U, \{U\})$ is not an update network then there exists a pair x_U, y_U such that $x_U \notin \text{REACH}(S, \{y_U\})$. If U is not S-closed then we take a witness p_U for U as in the proof of Lemma 3. Now we construct the following strategy g for Adversray. Let $p_0, \ldots, p_n$ be a finite play such that $p_n \in A$. Let p_i be the last separator seen in the finite play and $p_i = s_U$ for some $U \in \Omega$. There are three cases to consider.

Case 1. The set $\{p_i, \ldots, p_n\}$ is not a subset of U then Adversary chooses any p_{n+1} for which $(p_n, p_{n+1}) \in E$.

Case 2. The set $\{p_i, \ldots, p_n\}$ is a subset of U and U is S-closed. In this case Adversary follows $Avoid_{y_U}^{A,x_U}$ strategy.

Case 3. The set $\{p_i, \ldots, p_n\}$ is a subset of U and U is not S-closed. Then if $p_n = p_U$ then Adversray chooses p_{n+1} such that $p_{n+1} \notin U$ and $(p_n, p_{n+1}) \in E$.

Let $\pi = p_0, p_1, \ldots$ be a play consistent with g. Assume that $Inf(\pi) = U$ and $U \in \Omega$. Let n be the first position after which no nodes outside of U appear in π. Then since the separator s_U appeares infinitely many times in $In(\pi)$ it must be the case that s_U appears in π after position n. Hence, either *Case 2* or *Case 3* is applied. In *Case 2*, Adversary wins as the node y_U will not be visited infinitely

often, and hence $Inf(\pi) \neq U$ which is a contradiction. In *Case 3*, we will have a contradiction with the choice of position n. The lemma is proved. □

We are now ready to prove the following theorem whose proof uses the lemmas proved above as well as Theorem 1.

Theorem 3. *There exists an algorithm that decides any given fully separated game Γ in $O(|V|^2|E|)$ running time.*

Proof. Let p be the node from which all the plays begin. We describe the following algorithm $Procedure(\Gamma, p)$:

1. For each $U \in \Omega$, find whether or not $U \in \Omega$ is S-closed. If each $U \in \Omega$ is not S-closed then Adversary wins the game from p.
2. For each S-closed $U \in \Omega$ find whether or not the game $(U, \{U\})$ is an update network.
3. Let X be the union of all $U \in \Omega$ so that $(U, \{U\})$ is an update network and U is S-closed. If $X = \emptyset$ then Adversary wins the game from p.
4. If $p \in$ REACH (S, X) then Survivor wins the game from p.
5. If $p \notin$ REACH (S, X) then construct the game $\Gamma_1 = (V_1, E_1, W_1, \Omega_1)$ as follows. The set V_1 of nodes is AVOID(p, A, X), the set E_1 is the restriction of E to V_1, the set W_1 is V_1, and Ω_1 consists of all $U \in \Omega$ such that $U \subset V_1$. Note that $|V_1| < |V|$. Run $Procedure(\Gamma_1, p)$.

It is not hard to see that the algorithm runs in $O(|V|^2|E|)$ time. This proves the theorem. □

4 Games with Linear Winning Conditions

Let Γ be a game. The winning conditions set Ω is a partially ordered set in which the partial order is defined by means of the set-theoretic inclusion. We call it the **partial order associated with** the game Γ. The previous section deals with those winning conditions whose associated partial orders form antichains, that is $U \not\subseteq V$ for all distinct $U, V \in \Omega$. In this section, we investigate a dual case by considering games whose associated partial orders are linearly ordered.

Definition 7. *A game Γ is a* **linear game** *if the set Ω forms a linear order $U_1 \subset U_2 \subset \ldots \subset U_n$ and $W = V$. We call n the* **length of the winning conditions**.

Our goal is to show that linear games can be decided in polynomial time if the length of the winning conditions is fixed. We begin with an example.

Example 1. Consider the game $\Gamma = (S \cup A, E, W, \Omega)$, where:

1. $S = \{s_1, s_2, \ldots, s_n\}$, $A = \{a_1, a_2 \ldots, a_n\}$, $W = S \cup A$,
2. $E = \{(s_i, a_i) \mid 1 \leq i \leq n\} \cup \{(a_i, s_{i+1}) \mid 1 \leq i \leq n-1\} \cup \{(a_i, s_1) \mid 1 \leq i \leq n\}$,
3. $\Omega = \{U_i \mid 1 \leq i \leq n\}$, where $U_i = \{s_1, a_1, \ldots, s_i, a_i\}$.

In this game Survivor wins the whole game. It is worth to note that Adversary wins each of the game whose winning conditions set is a proper subset of Ω. The basic reason for this is that Survivor has no choice at any given node $s_i \in S$ but to move to a_i.

We need some notations, definitions, and lemmas for our next theorem. Let X, Y be a subset of nodes in a given game. Then REACH(S, X, Y) is the set of all nodes v such that *Survivor* can force every play starting at v into X after a finite number of steps and staying inside Y. As in Lemma 1 it can be shown that the set REACH(S, X, Y) can be computed in $O(|V| + |E|)$ time.

Let $\Gamma = (V, E, \Omega)$ be a game with the winning conditions $\Omega = \{U_1 \subset U_2 \subset \ldots U_n\}$. Assume that $n > 1$. For each $i < n$ consider the linear game $\Gamma_i = (V, E, \Omega_i)$, where $\Omega_i = \{U_1 \subset U_2 \subset \ldots U_i\}$. Here is a simple lemma whose proof is left to the reader.

Lemma 6. *1. Survivor wins the game Γ_1 from position p if and only if $p \in$ REACH(S, U_1), the set U_1 is S-closed, and Survivor wins the update game $(U_1, \{U_1\})$.*
2. If Survivor wins the game Γ_i from p then Survivor wins games Γ_j from position p for all $j \geq i$. □

For each $i < n$ consider the set X_i consisting of all positions p from which Survivor wins the game Γ_i. By the second part of the lemma above we have the sequence $X_1 \subseteq X_2 \subseteq \ldots \subseteq X_{n-1}$. Assume that $X_{n-1} \neq \emptyset$. For every position $p \notin$ REACH(S, X_{n-1}), consider the set AVOID(p, A, X_{n-1}). Now note that since $U_1 \subseteq U_2 \subseteq \ldots \subseteq U_n$ and $X_{n-1} \neq \emptyset$ it must be the case that $U_1 \subseteq X_{n-1}$. Therefore $Inf(\pi) \notin \Omega$ for any play π inside AVOID(p, A, X_{n-1}). We conclude that the following lemme is true:

Lemma 7. *Assume that $X_{n-1} \neq \emptyset$. Then Survivor wins the game Γ from position p if and only if $p \in$ REACH(S, X_{n-1}).* □

An important point of this lemma is that the original game can be reduced to a smaller linear game in case $X_{n-1} \neq \emptyset$.

Next we consider the case when $X_{n-1} = \emptyset$.

Lemma 8. *Assume that $X_{n-1} = \emptyset$. Then if Survivor wins the game Γ from position p then $p \in$ REACH(S, U_n) and U_n is S-closed.*

Proof. Clearly $p \in$ REACH(S, U_n). Assume that U_n is not S-closed. There is a $x \in U_n$ that satisfies one of the following conditions:

1. $x \in S$ and for all $a \in A$ if $(x, a) \in E$ then $a \notin U$.
2. $x \in A$ and there is an s such that $(x, s) \in E$ and $s \notin U$.

We describe a strategy for Adversary. Let $h = p_0 \ldots p_n$ with $p_0 = p$ be a finite play with $p_n \in A$. We consider several cases.

Case 1. $p_n = x$. Then Adversary moves outside of U_n.

Case 2. $p_i \neq x$ for all $i = 1, \ldots, n$. In this case Adversary plays his winning strategy in game Γ_{n-1} from p_0.

Case 3. Let $p_i = x$ and $x \notin \{p_{i+1}, \ldots, p_n\}$. In this case Adversary plays his winning strategy in game Γ_{n-1} from p_{i+1}.

Assume that π is a play consistent with the strategy described above. If x occurs in π finitely many times then there is a position p_i in the play after which the play becomes consistent with Adversray's winning strategy in game Γ_{n-1} from position p_i. Hence $Inf(\pi) \notin \Omega$ since $x \notin Inf(\pi)$ and $X_{n-1} = \emptyset$. If x occurs infinitely often then $Inf(\pi)$ has an element outside of U_n. Thus, the strategy is a winning strategy for Adversray. □

Assume that $X_{n-1} = \emptyset$. Let $x, y \in U_n$ be such that $x \notin \text{REACH}(S, \{y\}, U_n)$. Consider the set $\text{AVOID}(x, A, \{y\})$. We can define a new game denoted by $\Gamma(x, y)$ such that the graph of the game is $\text{AVOID}(x, A, \{y\})$, and the set $\Omega(x, y)$ of winning conditions are those $U \in \Omega$ which are subsets of $\text{AVOID}(x, A, \{y\})$. Note that if $\Omega(x, y)$ is the empty set then x is a winning position of Adversary in the original game. Also, $\Gamma(x, y)$ is a linear game and the length of its winning conditions is strictly less than n. Here is our next lemma.

Lemma 9. *Assume that $X_{n-1} = \emptyset$. Survivor wins the linear game Γ from position p if and only if p belongs to REACH(S, U_n), U_n is S-closed and one of the following two conditions is satisfied:*

1. *Survivor wins the update game $(U_n, \{U_n\})$.*
2. *For any pair $x, y \in U_n$ of nodes if $x \notin$ REACH($S, \{y\}, U_n$) then Survivor wins the game $\Gamma(x, y)$ from p while staying inside U_n.*

Proof. Assume that Survivor wins the game Γ from position p. Clearly it must be the case that $p \in \text{REACH}(S, U_n)$. Now assume that none of the conditions is true. We need to describe a winning strategy for Adversary. By the assumption, there must exist $x_0, y_0 \in U_n$ such that x_0 does not belong to the set $\text{REACH}((S, \{y_0\})$ and Adversray wins the game $\Gamma(x_0, y_0)$ while staying inside U_n. We fix x_0 and y_0. Here is now a strategy for Adversary. Let $h = p_0, \ldots, p_m$ be a finite play from p.

Case 1. $p_i = x_0$ for some $i \leq m$ and all nodes in h after p_i are in U_n. In this case Adversary follows his winning strategy (from position p_i) in game $\Gamma(x_0, y_0)$.

Case 2. Suppose that *Case 1* does not hold and all nodes in h are in U_n. In this case Adversary follows his winning strategy (from position p) in game $(V, W, \{U_1, \ldots, U_{n-1}\})$.

Case 3. $p_i \notin U_n$ for some $i \leq m$ and no node in h after p_i is x_0. In this case Adversary follows his winning strategy (from position p_i) in game $(V, W, \{U_1, \ldots, U_{n-1}\})$.

We need to show that this described strategy is a winning strategy for Adversary. Let $\pi = p_0, p_1, p_2, \ldots$ be a play, where $p = p_0$, consistent with the strategy. Assume that $p_i = x_0$ for some i so that $p_j \in U_n$ for all $j > i$. Then Adversary follows his winning strategy in $\Gamma(x_0, y_0)$. Note that $y_0 \notin Inf(\pi)$. We conclude that

$Inf(\pi) \notin \Omega$. Assume that $p_i \neq x_0$ for all p_i after some $p_j \notin U_n$. Then the play is consistent with the Adversary's winning strategy in game $(V, W, \{U_1, \ldots, U_{n-1})$ after p_j. Hence $Inf(\pi) \notin \{U_1, \ldots, U_{n-1}\}$. Since $x_0 \notin Inf(\pi)$ we conclude that $Inf(\pi) \neq U_n$. Assume that x_0 and y_0 appears infinitely often in π. This means π contains infinitely many nodes outside of U_n. Hence $Inf(\pi) \notin \Omega$. Thus, the strategy is winning strategy for Adversary. This is a contradiction.

Now assume that one of the two conditions is satisfied and $p \in$ REACH (S, U_n). Clearly if the first conditions is true then Survivor wins the game. Assume that the first condition is not satisfied. Let $p = x_0, x_1, \ldots, x_k$ be the list of all nodes in U_n. Survivor's strategy is as follows. Initially $i = 0$ and the current position is p. If the current position q of a play is in REACH$((S, \{x_{i+1(mod(k))}, U_n)$ then Survivor forces the play into $x_{i+1(mod(k))}$. As soon as $x_{i+1(mod(k))}$ is reached i is set to $i + 1(mod(k))$ and the current position is $x_{i+1(mod(k))}$. If the current position q is not in REACH$((S, \{x_{i+1(mod(k))}, U_n\})$ then Survivor plays his winning strategy inside $\Gamma(x_i, x_{i+1(mod(k))})$ while staying in U_n. We need to show that this is a winning strategy for Survivor. Let $\pi = p_0 p_1 p_2 \ldots$ (with $p_0 = p$) be a play consistent with the strategy. Let $U = Inf(\pi)$ be the infinity set of the play. Assume that $x_{i+1(mod(k))} \notin U$ for some i. This means that there is a position n in the play π such that all x_j with $j > n$ belong to AVOID$(x_i, A, x_{i+1(mod(k))})$. Since Survivor plays his winning strategy inside the game $\Gamma(x_i, x_{i+1(mod(k))})$ the set U must belong to Ω. □

From the lemmas above and Theorem 1 we now can derive the following result about complexity of deciding linear games.

Theorem 4. *There exists an algorithm that decides any linear game $\mathcal{G}$ with winning conditions $\{U_1, \ldots, U_n\}$ in $O(|V|^{2 \cdot n-1}|E|)$ running time. In particular, if n is fixed then deciding linear games with n winning conditions can be done in a polynomial time.*

Proof. We analyze the case when $n = 2$. We use the lemmas above. Constructing X_1 and checking if $p \in$ REACH(S, X_1) takes at most $O(|V||E|)$-time.

Assume that $X_1 = \emptyset$. Cheking that $p \in$ REACH(S, U_2) and $(U_2, \{U_2\})$ is an update network takes at most $(|V||E|)$-time.

Let us now compute the time needed to check the second condition in the lemma above. It is not hard to see that for each $y \in U_2$ the set AVOID$(A, \{y\})=$ $V \setminus$ REACH$(S, \{y\})$ can be constructed in $O(|E| + |V|)$-time (see Lemma 2). For each $x \in$ AVOID$(A, \{y\})$ checking if Survivor wins $\Gamma(x, y)$ inside U_2 takes at most $O(|$AVOID$(x, A, \{y\})||E|)$-time. Therefore, by varying x, y we see that the total time does not exceed $O(|V|^2 \times |V||E|)$. This proves the theorem for $n = 2$.

The rest can be done by using recursion and the use of the previous lemmas. The theorem is proved. □

5 No-memory Strategies, Complexity, and Structure

The goal in this section is twofold. On the one hand we show that finding efficient strategies in McNaughton games, even in a simple case such as games with one

winning set, is an untractible problem. On the other hand, we show how efficient winning strategies can be used to extract structural properties of the underlying graphs. In this section we consider games Γ in which $\Omega = \{U\}$ and $U = S$.

Argueably the most simple strategies are the ones that depend on the current node of a play and not any other part of its history. We single out such strategies in the following definition.

Definition 8. *A strategy for Survivor is a* **no-memory strategy** *if it is induced by a function $f : S \to A$ such that $(s, f(s)) \in E$ for all $s \in S$.*

Thus if f is a no-memory strategy and h is a finite play whose last symbol $last(h)$ is in S then Survivor's next move is $f(last(h))$. The next definition gives us a tool to analyze the structure of graph games.

Definition 9. *A cycle $a_0, s_0 \ldots, a_n, s_n$ in game graph $\mathcal{G}$ is a* **forced cycle** *if $(a_i, s) \in E$ implies $s = s_i$ for all $0 = 1, \ldots, n$.*

Here is our theorem that shows the interaction between no-memory winning strategies in update games and the structural properties of the underlying graphs.

Theorem 5. *Let $\Gamma = (V, E, W, \{S\})$ be a game whose graph is $\mathcal{G}$ and $W = S$. Then Survivor has a no-memory winning strategy if and only if all the vertices of S belong to a forced cycle.*

Proof. Assume that the graph S is in a forced cycle $a_0, s_0 \ldots, a_n, s_n$. Then the mapping $s_i \to a_{i+1(mod(n+1))}$ establishes a no-memory winning strategy for Survivor.

Assume that in game Γ Survivor has a no-memory winning strategy f. Consider a play $\pi = s_0, a_0, s_1, a_1, \ldots$ consistent with f. Thus $f(s_i) = a_i$ for all i. Since f is a no-memory winning strategy we have $Inf(\pi) = S$. In this play there exist positions i and $i + m$ such that $s_i = s_{i+m}$, $m > 0$, and no two Survivors nodes between positions i and $i + m$ coincide. It is not hard to see that $s_i, a_i, \ldots, a_{i+m}$ contains *all* the nodes from S as otherwise f would not be a winning strategy. Moreover, in this list if $k \neq t$ then $a_k \neq a_t$. Indeed, say $k < t$ and $a_k = a_t$. Then Adversary by always moving from a_k into s_{k+1} would win against strategy f. This would contradict the assumption that f is a winning strategy. Thus, $s_i, a_i, \ldots, a_{i+m-1}, s_{i+m}$ is in fact a forced cycle. The theorem is proved.

The ideas of the proof of the theorem can now be used to show the following hardness result:

Corollary 1. *The problem of finding whether or not Survivor has a no-memory winning strategy in a given game is NP-hard.*

Proof. We reduce the problem of finding a Hamiltonian path in a directed graph to the problem of interest. Let $\mathcal{G} = (V', E')$ be a directed graph. Construct the following game $\Gamma(\mathcal{G}) = (S \cup A, E, W, \Omega)$:

1. $S = V'$, $A = \{a_{(v,w)} \mid (v,w) \in E'\}$.
2. $E = \{(s, a_{(s,w)}) \mid s \in S, a_{(s,w)} \in A\} \cup \{(a_{(v,s)}, s) \mid a_{(v,s)} \in A, s \in S\}$.
3. $W = V'$ and $\Omega = \{V'\}$.

Basically, we subdevide each edge (v, w) of the original graph $\mathcal{G}$ by introducing new Adversary's node $a_{(v,w)}$ that is connected to v and w. Clearly, the construction of $\Gamma(\mathcal{G})$ is linear on the size of the graph $\mathcal{G}$. Note that the winning here is defined as follows. Survivor wins a play if the set of all the nodes of W in the play that appear infinitely often coincide with S. Moreover, it is easy to see that $\mathcal{G}$ has a Hamiltonian cycle if and only if Survivor has a non-memory winning strategy in $\Gamma(\mathcal{G})$. The corollary is proved. □

6 Games and Temporal Logic

We now say a few words in relation to connections with temporal logic. There are several ways to think about these games using the language of temporal logic. For example, one way to think about the specifications (W, Ω) is to identify them with classes of formulas of temporal logic. Formally, this can be established as follows as it is done in [4]. Given a system $(S \bigcup A, E)$, form propositions of temporal logic by identifying each $p \in S \cup A$ as an atomic proposition. In the inductive step, if ϕ and ψ are propositions then their Boolean combinations and the expressions $G\phi$ and $F\phi$ are also propositions. Semantics for these propsitions are the runs of the system $(S \bigcup A, E)$. Let $\pi = p_0, p_1, p_2, \ldots$ be a run and ϕ be formula. Let π^i be the sequence $p_i, p_{i+1}, \ldots$. One now can define what it means ϕ to be true on π, denoted by $\pi \models \phi$, by induction as follows. If $p_0 = p$ then $\pi \models p$. The case for Boolean connectives is defined naturally. For $\phi = F\psi$, $\pi \models \phi$ if $\pi^i \models \psi$ for some i. For $\phi = G\psi$, $\pi \models \phi$ if $\pi^i \models \psi$ for all i. Thinking of G as "globally" and of F as "future", we can represent specifications (W, Ω) in the language of temporal logic. Thus, given a system $(S \bigcup A, E)$ and a specification ϕ we can now ask whether or not the system satisfies ϕ. The satisfaction can be expressed in terms of winning. Namely, the system **satisfies** ϕ if Survivor has a strategy so that in every play π consistent with the strategy the formula ϕ is true. For example, for the system $(S \bigcup A, E)$ with nodes $\{p_0, \ldots, p_{n-1}\}$, the specification $(p_0 \vee \ldots \vee p_n)\&(\&_{0 \leq i \leq n-1} G(p_i \rightarrow F p_{i+1(mod\ n)}))$ tells us that Survivor must visit every node infinitely often. This is in fact a specification of update networks in terms of temporal logic. Thus, one can study the following natural questions:

1. (Model checking complexity) Given a system, what is the time complexity of finding whether or not a given temporal formula is staisfied in the system?
2. (Implementation complexity) Given a formula ϕ, what is the complexity of finding whether or not a given system satisfies ϕ?
3. (Combined complexity) What is the complexity of finding, given a formula ϕ and the system $\mathcal{A}$, that $\mathcal{A}$ satisfies ϕ?

For details on research on the relationship between verification and specifications of systems, temporal logic and games see Vardi [9].

7 Conclusion

The results of this paper can be generalized. For example, in fully separated games or in linear games the condition $W = S \bigcup A$ can be removed. Techniques for such a generalization can be found in [2]. We expect that it is possible to decide linear games more efficiently than the time bound presented in Theorem 4. We think that the methods and techniques developed in this paper, papers [1] and [2] give sufficient tools for a deep study of games from computational, algebraic and logical points of view. One can hope to provide fast algorithms for deciding those games in which the winning configurations can be decided efficiently, e.g. when the number of winning conditions is fixed. Section 5 shows that interesting results can be obtained in relation to implementing winning strategies by finite automata. This is a topic of our future papers. Note that apart from Corolloary 1 neither in this nor in any of the previous papers [1] [2] the topic on complexity of extracting winning strategies has been discussed. To our knowledge the only paper that deals with this issue explicetly is one by Nerode, Remmel, and Yakhnis [7]. A fruitful direction is related to the study of connections with temporal logic briefly described in the last section. One can also study the effect of the topology of the systems on finding the winners and winning strategies. As it is seen, much more is needed to be done.

References

1. M. J. Dinneen and B. Khoussainov. Update networks and their routing strategies. In *Proceedings of the 26th International Workshop on Graph-Theoretic Concepts in Computer Science, WG2000*, volume 1928 of *Lecture Notes on Computer Science*, pages 127–136. Springer-Verlag, June 2000.
2. H.L. Bodlaender, M.J. Dinneen and B. Khoussainov. On Game-Theoretic Models of Networks, in Algorithms and Computation (ISAAC 2001 proceedings), LNCS 2223, P. Eades and T. Takaoka (Eds.), p. 550-561, Springer-Verlag Berlin Heidelberg 2001.
3. Y. Gurevich and L. Harrington. Trees, Automata, and Games, STOCS, 1982, pages 60–65.
4. J. F. Knight and B. Luense. Control Theory, Modal Logic, and Games, In *Hybrid Systems IV*. Panos J. Antsaklis, Wolf Kohn, Anil Nerode, Shankar Sastry (Eds.), volume 1273 of *Lecture Notes in Computer Science*, pages 160–173. Springer, 1997.
5. D. Martin. Borel Determinacy. Ann. Math. Vol 102, 363-375, 1975.
6. R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65:149–184, 1993.
7. A. Nerode, J. Remmel, and A. Yakhnis. McNaughton games and extracting strategies for concurrent programs. *Annals of Pure and Applied Logic*, 78:203–242, 1996.
8. A. Nerode, A. Yakhnis, V. Yakhnis. Distributed concurrent programs as strategies in games. Logical methods (Ithaca, NY, 1992), pages 624–653, *Progr. Comput. Sci. Appl. Logic*, 12, Birkhauser Boston, Boston, MA, 1993.
9. M. Vardi. An automata-theoretic approach to linear temporal logic. Proceedings of the VIII Banff Higher Order Workshop. Springer Workshops in Computing Series, Banff, 1994.

New Algorithms for k–Face Cover, k–Feedback Vertex Set, and k–Disjoint Cycles on Plane and Planar Graphs

Ton Kloks*, C.M. Lee[1,**], and Jiping Liu[2,***]

[1] Department of Computer Science and Information Engineering
Chung Cheng University
Ming–Shiung, Chiayi 621, Taiwan
cmlee@cs.ccu.edu.tw
[2] Department of Mathematics and Computer Science
The University of Lethbridge
Alberta, T1K 3M4, Canada
liu@cs.uleth.ca

Abstract. We present new fixed parameter algorithms for the FACE COVER problem on plane graphs. We show that if a plane graph has a face cover with at most k faces then its treewidth is bounded by $O(\sqrt{k})$. An approximate tree decomposition can be obtained in linear time, and this is used to find an algorithm computing the face cover number in time $O(c^{\sqrt{k}}n)$ for some constant c. Next we show that the problem is in linear time reducible to a problem kernel of $O(k^2)$ vertices, and this kernel can be used to obtain an algorithm that runs in time $O(c^{\sqrt{k}}+n)$ for some other constant c. For the k–DISJOINT CYCLES problem and the k-FEEDBACK VERTEX SET problem on planar graphs we obtain algorithms that run in time $O(c^{\sqrt{k}\log k}n)$ for some constant c. For the k-FEEDBACK VERTEX SET problem we can further reduce the problem to a problem kernel of size $O(k^3)$ and obtain an algorithm that runs in time $O(c^{\sqrt{k}\log k}+n)$ for some constant c [1].

1 Introduction

Many problems of practical interest tend to be NP-hard when viewed from classical computation complexity theory. People are unlikely to find polynomial-time to solve them. There are several ways to coping with this apparent difficulty. Approximation algorithms are frequently chosen to deal with computational intractability. In other words, if a problem was once classified as NP-hard, people often try to design and analyze an approximation algorithm for it. In the recent ten years, *parameterized complexity* has opened a new road of attack against computational intractability.

* Supported by an EPSRC grant. Kaleidoscopic address: ton@cs.rhul.ac.uk.
** Corresponding author.
*** Supported by NSERC of Canada.
[1] Throughout we use the same "c" to stand for *some* constant.

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 282–295, 2002.

Definition 1. *Consider an algorithm for an instance of a parameterized problem (I, k) where I is the problem instance and k is the parameter. The algorithm is called* uniformly polynomial *if it runs in time $O(f(k)|I|^c)$ where $|I|$ is the size of I, $f(k)$ is an arbitrary function, and c is a constant. A parameterized problem is* fixed parameter tractable, (FPT)*, if it admits a uniformly polynomial algorithm.*

Parameterized complexity was introduced some ten years ago [13]. There are by now numerous algorithms known for fixed parameter tractable problems. For PLANAR DOMINATION, part of the stumbling history is reflected in [14,13,2]. A stunning breakthrough with a catalytic impact was made with the discovery of sub-exponential solutions (i.e., $O(c^{\sqrt{k}}n)$) for various domination problems on planar graphs [7,1,3,4,10]. It is shown [8] that for all MAX SNP-hard problems (such as DOMINATING SET) finding exact solutions in sub-exponential time is not possible unless W[1]=FPT. PLANAR DOMINATING SET (and various other problems) has no EPTAS running in time $O(2^{o(\sqrt{\frac{1}{\epsilon}})}p(n))$ unless W[1]=FPT [9]. In this paper we focus on FACE COVER, FEEDBACK VERTEX SET, and DISJOINT CYCLE problems. We start by considering the FACE COVER problem. Consider a *plane* graph $G = (V, E)$. That is, a specific plane drawing of a planar graph in the plane. The different regions inferred by this drawing as well as the subsets of vertices which can be coupled by curves fully contained within these regions are called the *faces* of the graph.

Definition 2. *Consider a* plane *graph G. A* face cover *is a set $\mathcal{F}$ of faces such that every vertex x is incident with at least one element of $\mathcal{F}$. The number of faces in a minimum face cover is called the* face cover number *and this is denoted by $fc(G)$.*

It is not hard to see that the FACE COVER problem is NP-complete by a reduction from VERTEX COVER. In [17] it is shown that the problem remains NP-complete for plane triangulations. In this paper we present new algorithms for the parameterized FACE COVER problem when restricted to plane graphs. A solution for the k–FACE COVER problem is given in [13] which runs in time $O(12^k n)$. Subsequently, a new algorithm for this problem was described in [1]. Their solution could be implemented to run in time $O(c^{\sqrt{k}}n + n^2)$ for some constant c. In Section 2 we further improve upon these results by giving first a simple algorithm solving the problem in time $O(c^{\sqrt{k}}n)$, where c is of similar size as in the algorithm of [1]. Then, in Section 3 we show that it is possible to reduce a problem instance I to a problem kernel I', i.e., an *equivalent* instance which is bounded in size by some function of the parameter k. In this case we show that it is possible to reduce the problem in linear time to a problem kernel of size $O(k^2)$. This then, together with the previously mentioned algorithm immediately implies the existence of an algorithm that runs in time $O(c^{\sqrt{k}} + n)$ for some constant c. In Section 4 we turn to the FEEDBACK VERTEX SET problem.

Definition 3. *A* feedback *vertex set, (*FVS*) of a graph G is a set X of vertices such that every cycle of G passes through at least one vertex of X. For a graph G we let $cc(G)$ denote the cardinality of a minimum feedback vertex set.*

Computing the cardinality of a minimum FVS is NP-complete for planar graphs. This is easy to see by a reduction from VERTEX COVER. Notice that "being acyclic within k vertices" is a minor closed property, (since being acyclic is minor closed [16]). Hence FVS is fixed parameter tractable for graphs in general. A concrete algorithm to solve the k-FEEDBACK VERTEX SET problem in $O((2k+1)^k n^2)$ time is described in [13]. It is remarked that the problem can be solved alternatively in time $O((17k^4)!(n+m))$. For descriptions of algorithms with these timebounds see [5,13].
In this paper we restrict ourselves to planar graphs and we show that for that case noticeably faster algorithms exist. Specifically, in Section 4, we show the existence of an algorithm that runs in $O(c^{\sqrt{k}\log k}n)$ time. Furthermore, in Section 5 we show that it is possible to reduce the problem to a problem kernel of size $O(k^3)$ in linear time. This then implies the existence of an algorithm that runs in time $O(c^{\sqrt{k}\log k}+n)$ for some constant c.
In Section 6 we turn our attention to the DISJOINT CYCLES problem on planar graphs.

Definition 4. *A* cycle packing *in a graph G is a set of vertex disjoint cycles. For a graph G we let $cp(G)$ stand for the cardinality of a maximum cycle packing.*

Notice that the k-DISJOINT CYCLES problem is NP-complete (also for planar graphs) because it contains PARTITION INTO TRIANGLES as a special case. The problem was shown to be fixed parameter tractable in [5,13]. In Section 6 we show that the k-disjoint cycles problem can be solved in time $O(c^{\sqrt{k}\log k}n)$ when restricted to planar graphs. Unfortunately, for this problem we were thus far unable to find a reduction to a problem kernel.

2 A Treewidth Algorithm for the Face Cover Number

Definition 5. *A* dominating set *D in a graph G is a set of vertices such that every vertex not in D has at least one neighbor in D.*

Definition 6. *A* tree decomposition *$(\mathcal{S}, T)$ for an undirected graph $G = (V, E)$ is a pair where T is a tree and $\mathcal{S}$ is a set of subsets of vertices, called* bags*. $\mathcal{S}$ is in 1-1 correspondence with the nodes of the tree T such that the following conditions are satisfied:*

1. *Every vertex is contained in at least one bag,*
2. *Both end vertices of every edge are contained in at least one bag,*
3. *For every vertex x of the graph, if x appears in bags S_i and S_j then it appears in every bag corresponding to the vertices which lie on the path in the tree T between the nodes i and j.*

The *width* of a tree decomposition $(\mathcal{S}, T)$ is the maximum cardinality of a bag S_i minus one, over all nodes in the tree T. The *treewidth* of the graph is the minimum width over all possible tree decompositions of the graph. For a graph G, let $tw(G)$ denote the treewidth of G.
Computing treewidth for graphs in general is NP-complete, however the problem is FPT. Moreover there exists a linear time algorithm to check if a graph has a bounded treewidth [7,19].
We use the result of [1]. (For general information on minors and treewidth we refer to [12].)

Theorem 1. *If a planar graph G has a dominating set of size at most k then the treewidth of G is at most $\delta\sqrt{k}$ for some constant δ. There is a linear time algorithm that finds a tree decomposition of this width.*

Remark 1. The best upperbound known to us at present is $\delta < 6\sqrt{34}$ [1].

Lemma 1. *If $fc(G) \leq k$ then $tw(G) = O(\sqrt{k})$.*

Proof. Consider a set of at most k faces in a plane embedding of G such that every vertex lies on at least one of these faces. Put a new vertex in each of these faces and take edges from these new vertices to all the vertices of the boundary. The new graph H has now a dominating set consisting of the k new vertices. Hence the treewidth of H is $O(\sqrt{k})$. But H is a supergraph of G and hence also the treewidth of G can be only $O(\sqrt{k})$. □

The k–FACE COVER problem for G can be solved in time $O(c^{\sqrt{k}}n)$ as follows. It is easiest to start with the construction of an auxiliary plane graph H. Place new, say *blue*, vertices in every face of the plane graph G. Add edges from every blue vertex to the vertices on the boundary of the face containing it.
We first show that the treewidth of H is also $O(\sqrt{k})$.

Theorem 2. $tw(H) \leq 4tw(G) + 1$ [2].

Proof. Consider a tree decomposition $D = (\mathcal{S}, T)$ for G with width $\ell - 1$ ($\ell \geq 1$). (So every bag of $\mathcal{S}$ has at most ℓ vertices.)
Consider an isolated vertex x of G. We may assume that x has a private bag in D containing only x. We add the blue vertex of the face that contains x to this bag. Consider an edge e of G. Add the (at most 2) blue vertices that are adjacent to both endpoints of e to all the bags of $\mathcal{S}$ that contain both endpoints of e. If we relocate the nodes in T of which the bags contain isolated vertices properly, D changes subsequently into a tree decomposition for H (we prove this below). Furthermore, we claim that every bag has now at most $4\ell - 2 = 4tw(G) + 2$ vertices.
We prove this claim by induction on the number of nodes of T. Assume first that T has only one node. Then G has ℓ vertices. Since G is planar, the number of faces

[2] *Conjecture 1.* $tw(H) \leq tw(G) + 4$.

is at most $3\ell-2$ (by Euler's formula: $\phi = 1+\eta+m-n \leq 1+\ell+3(\ell-1)-\ell = 3\ell-2$, where ϕ, η, m, and n are the number of faces, components, edges, and vertices of G, respectively). Hence, adding all the blue vertex to the one bag of D increases the cardinality with at most $3\ell - 2$ vertices.

Suppose that T has at least two nodes and consider any edge (x_1, x_2) of T. This edge divides T into two new trees, say T_1 and T_2. We let $D_i = (\mathcal{S}_i, T_i)$ be the decomposition induced by T_i. Let V_i be the set of vertices that appear in bags of $\mathcal{S}_i$ and let $G_i = G[V_i]$ be the subgraph of G induced by V_i, $i = 1, 2$. We may assume that each V_i is a proper subset of V: Assume that each vertex of V occurs in some node of T_1 but some edge (x, y) is *not* contained in any bag of $\mathcal{S}_1$. Then the edge must appear in some bag in T_2 and hence both x and y appear in the bags of x_1 and x_2, hence (x, y) is contained in a bag of $\mathcal{S}_1$ which is a contradiction.

If some face $\mathbb{F}$ of G has some but not all vertices in V_1 we consider the graph obtained from G by "short–cutting" $\mathbb{F}$: If α and β are two non-adjacent vertices of $\mathbb{F}$ in V_1 and if the clockwise path between α and β contains no other vertices in V_1 then we add an edge (α, β) to G_1 *and also* to G_2. (Notice that α and β must also be vertices of G_2 since they are both incident with at least one edge which is not in any bag of D_1. Hence, both α and β are elements of the bags at x_1 and x_2.) We draw the new edge in the plane embedding inside $\mathbb{F}$. Let G_1' and G_2' be the graphs obtained from G_1 and G_2 by adding all these short–cuts. Notice that each G_i' is a plane graph.

We claim that D_i is a tree decomposition for G_i' for $i = 1, 2$. Indeed, if (α, β) is a short–cut of $\mathbb{F}$ then α and β must both be in the bags of x_1 and x_2 since they are elements of V_1 and both are incident with at least one edge which is not in any bag of D_1.

Add blue vertices to all faces of G_i' and let H_i be the new graph ($i = 1, 2$). By induction we know that D_i changes into a tree decomposition for H_i with at most $4\ell + 2$ vertices per bag when we add the blue vertices of H_i to every bag that contains an adjacent edge of G_i', since D_i is a tree decomposition for G_i'. Finally, notice that since all short–cuts appear in x_1 and x_2, we may *identify* separate blue vertices belonging to the same face $\mathbb{F}$ in G. Like this we obtain a tree decomposition for H of width at most $4tw(G) + 1$. □

For actually *constructing* a tree decomposition for H the same method as indicated in [1] can be used.

One day, time and space will permit us to describe an algorithm that computes a minimum cardinality set of blue vertices dominating all other vertices. For DOMINATING SET it is well known that, given a tree decomposition with width ℓ, this problem can be solved in time $O(3^\ell n)$ (see, e.g., [18]) [3]. We obtain:

Theorem 3. *There exists an algorithm which runs in time $O(c^{\sqrt{k}}n)$ which decides either that $fc(G) > k$ or determines the exact value of $fc(G)$.*

[3] We can make our graph bipartite by removing all original edges between red vertices. This can be used to reduce the time complexity to $O(2^\ell n)$ (see [1]).

3 Reduction of Face Cover to a Problem Kernel

We work again with the plane graph H obtained from G by placing new, blue, vertices inside the faces of G and by creating edges from these blue vertices to all vertices on the boundary of the face. Call the old vertices of G red. We search for a "blue dominating set", i.e., a set of blue vertices dominating all red vertices.
If the plane graph G has a pendant vertex then the face that contains this vertex is fated to be in the face cover. Hence, in H we can put the blue vertex in the face cover and remove all red neighbors from the graph. The target blue domination number of the remaining graph decreases by one. Henceforth, we assume that G has no pendant vertices.
Consider a vertex x of degree 2. In general, this vertex belongs to 2 faces (or otherwise it is a cut-vertex that belongs to only 1 face). At least one of the faces incident with x should be in the cover. Notice that we may assume that every induced path contains *at most one* subdivision vertex. Let H^* be the graph obtained from H by replacing subsets of red subdivision vertices on the same "carrier–edge" by 1 representative. Let G^* be the graph obtained from G by removing pendant vertices and "dissolving" *all* vertices of degree 2 (i.e., removing the vertex and putting an edge between its neighbors if there was none). Hence H^* is a subdivision of G^* with at most one subdivision vertex per edge in G^*.

Lemma 2. *Suppose that G consists of two disjoint components. Then consider the face between the two components. In each component choose two adjacent vertices (a vertex if a component is a singleton) and link the two components by two edges between these vertices. Choose the new links such that the new face has at most four edges. In case there is a cut-vertex or a bridge, add an extra edge in a similar way. These operations do not change the face cover number of G.*

Proof. Adding an edge in the plane graph cannot decrease the face cover number. Observe that, if the face between the two components is chosen in an optimal face cover, then in the new graph this face without the square or triangle will serve equally well. □

A graph in which every vertex has degree at least 3 will be called *rich*. Hence, we may assume that the graph G^* is *biconnected and rich* and H^* is a subdivision of G^* with at most one subdivision vertex per edge in G^*.

Lemma 3. *Let $\mathbb{F}$ be a face of the biconnected and rich graph G^* with more than $4k$ vertices. Then $\mathbb{F}$ must be in any face cover with k faces.*

Proof. Assume G^* has a face cover with at most k faces and assume that $\mathbb{F}$ takes no part in it. Since every vertex of $\mathbb{F}$ has degree at least 3, each of these has at least one neighbor outside $\mathbb{F}$.
Consider a component C of $G^* - \mathbb{F}$ with at least 2 vertices. First let $|N(C)| = 2$. If C has at least 2 vertices with 2 neighbors in $\mathbb{F}$ then C has a vertex which is

not on the outerface of $G^*[C \cup \mathbb{F}]$. Otherwise C contains a cycle since G is rich. In any case, it follows that $G^*[C \cup \mathbb{F}]$ contains at least one vertex that is not on the outerface and hence C accounts for at least *one* finite face which must be in the cover.
Moreover, if a component C has at least 3 neighbors in $\mathbb{F}$ (which is the case if C has only one vertex), it must have at least $\lceil \frac{|N(C)|}{2} - 1 \rceil$ faces in the cover, since every second vertex of $\mathbb{F}$ of this neighborhood which is not on the outside, accounts for a face in the cover. (This can be easily seen by contracting the component C to a single vertex.) It follows that $\mathbb{F}$ can be split up in sets of 4 vertices each adding at least one face to the cover, the "worst case" being that every component has 4 neighbors in $\mathbb{F}$. □

Corollary 1. *If H^* has a face $\mathbb{F}$ with more than $8k$ vertices then $\mathbb{F}$ must be in any face cover with k faces.*

Proof. Assume $\mathbb{F}$ is a face of H^* with more than $8k$ vertices. Since every edge of G^* has at most one subdivision vertex, it follows that $\mathbb{F}$ grounds a face with more than $4k$ vertices in G^*. If H^* would have a face cover without $\mathbb{F}$ as a member, than a similar statement would hold true for G^*. □

Consider now the following process: As long as there is a face in H^* with more than $8k$ vertices, then put the blue face-vertex in the face cover and remove all its red neighbors from the graph. In the new graph we now look for a face cover with $k-1$ faces. Subsequently reduce the graph as described above (getting rid of red pendants, dissolving vertices of degree 2). Assume this process stops at the i^{th} instance. Then in the remaining graph every face has at most $8(k-i)$ vertices, and in this graph we look for a face cover with at most $k-i$ faces. Hence if we have more than $8(k-i)^2$ vertices, there must be a vertex which is not covered by any element in the face cover. Like this, we come to rest with a kernel of at most $8k^2$ vertices, or we have to conclude that no face cover with k elements can exists.
It's quite easy to see (but rather laborious to describe) that the reduction to the problem kernel can be performed in linear time. If we subsequently apply the algorithm of Section 2 we obtain:

Theorem 4. *There exists an algorithm that runs in time $O(c^{\sqrt{k}} + n)$ which solves the k–*FACE COVER *problem.*

4 A Treewidth Algorithm for Feedback Vertex Set

Recall the following result:

Lemma 4. *Every rich planar graph has a face of length at most 5.*

Proof. Assume every face has length at least 6. Let ϕ be the number of faces. Counting the number of faces and incident edges in two ways we obtain: $2m \geq 6\phi$. According to Euler's theorem $m - n + 2 = \phi \leq \frac{m}{3}$. Hence $2m \leq 3n - 6$. On the other hand, since every vertex has degree at least 3, $2m \geq 3n$. □

This result implies the existence of a *bounded search tree* algorithm (see [13]) for planar FVS (ideally running in time $O(5^k n)$): Reduce the graph until every vertex has degree at least 3, and repeatedly find faces of length at most 5, building a bounded search tree. In this section we improve this to $O(c^{\sqrt{k}\log k} n)$.

Theorem 5. *Let G be a planar graph with $cc(G) \leq k$. Then $tw(G) = O(\sqrt{k})$.*

Proof. Assume that G has a FVS W of cardinality k. We show that there is a planar supergraph H of G such that W is a dominating set in H.
Consider a plane embedding of G. Since every face is a cycle, it must contain at least one vertex of W. For every face, fix one such vertex in W and add edges between every other vertex of the face and this vertex of W. Call this graph H. Observe that the set W is a dominating set in the planar graph H. By Theorem 1 H has treewidth $O(\sqrt{k})$, and since G is a subgraph of H, also G has treewidth $O(\sqrt{k})$. □

We describe a fairly standard algorithm computing a FVS using a tree decomposition. Let $\ell - 1$ be the *width* of this tree decomposition (i.e., every bag of this decomposition has at most ℓ vertices). Let the pair $(T, \mathcal{S})$ designate the tree decomposition of G where T is a rooted tree T and $\mathcal{S} = \{S_i \mid i \in V(T)\}$ a collection of subsets of $V(G)$ called *bags*, each corresponding uniquely with a node in T.
Let G_i be the subgraph of G induced by the union of bags corresponding with nodes in the subtree of $i \in V(T)$. Let $X_i = V(G_i) - S_i$. Consider a FVS W in G_i. Let $W^\circ = W \cap X_i$. Removal of W° from G_i turns $G_i[X_i]$ into a forest. We characterize this forest and its joints to S_i by a *pattern* defined by the following rules (*pattern dicta*):

1. Contract every component of the forest to a single vertex thus converting it into an independent set. Disregard loops that turn up in the process. However, the contractions might create multiple edges between vertices of the independent set and vertices of S_i.
2. If a vertex of S_i is incident with some multiple edge then create a loop incident with this vertex and remove the multiple edge.
3. Remove vertices of the independent set which are adjacent to at most one vertex in S_i.
4. Replace multiple copies of vertices in the independent set that are adjacent to the same pair of vertices in S_i by *two* copies.

Definition 7. *Call the obtained independent set I and let F be the union of the set of obtained edges and loops between vertices of I and S_i and the edges of $G[S_i]$. The graph $P = (I \cup S_i, F)$ is called a* pattern.

Lemma 5. *The number of different patterns is $O(c^{\ell \log \ell})$ for some constant c.*

Proof. Consider a pattern $P = (I \cup S_i, F)$. Since the class of planar graphs is closed under taking minors the graph P is planar. Call two vertices of I equivalent if they have the same neighbors in S_i. Let H be the of vertices of I with at least three neighbors in S_i and let $L = I - H$. Notice that $|H| \leq 2\ell$ since the number of edges between H and S_i is at least $3|H|$ but at most $2(|H|+\ell)$. (If the number of vertices in a planar graph without triangles is at least 3 then the number of edges is even bounded by $2n - 4$.)
"Dissolving" a vertex of L is the removal of the vertex and creating an edge between its neighbors if these were non-adjacent. Now $|L| \leq 2(3\ell - 6)$ since dissolving the vertices of L in $P[I \cup S_i]$ gives a planar graph with at most ℓ vertices which has at most $3\ell - 6$ edges and each equivalence class in L has at most two elements.
It follows that the number of different pattern is bounded by the number of different bipartite planar graphs with $O(\ell)$ vertices in each colour class. The bound follows since a planar graph has only a linear number of edges. □

We may assume that the tree decomposition $(T, \mathcal{S})$ of width $\ell - 1$ is in *standard form*. That is, each node of T is one of 4 different types. A node p is a START NODE if p is a leaf of the tree different from the root. Node p is an INTRODUCE NODE if it has exactly one child q and the bag at node p has one more vertex x than the bag at node q, i.e., $S_p = S_q + x$. Node p is a FORGET NODE if if it has exactly one child q and the bag S_p contains one vertex x *less* than S_q, i.e., $S_p = S_q - x$. Finally, a node p is a JOIN NODE if it has two children q and r and $S_p = S_q = S_r$.
We proceed by describing how to obtain a table of patterns for each node in the tree decomposition from the tables of patterns at the children of that node. With each pattern we store the *age* of it. The age $A(P)$ of a pattern P is the minimum cardinality of a feedback vertex set W° in X_i giving rise to this pattern. A pattern is *feasible* if its age is at most k. We store the different feasible patterns with their ages.

START NODES. Let p be a start node. In this case there is only one pattern $P_p = G[S_p]$. The age of this pattern is $A(P_p) = 0$.

JOIN NODES. Let p be a join node with children q and r. Consider pairs of patterns P_q and P_r with $A(P_q) + A(P_r) \leq k$. Reduce the union of the two patterns according to pattern dictum 4. Update the age of this pattern by taking the minimum of the currently stored ages (if at all) and $A(P_q)+A(P_r)$.

FORGET NODES. Let p be a forget node with child q and let $x \in S_q - S_p$. Consider a feasible pattern $P_q = (I_q \cup S_q, E_q)$ for node q. We consider two evolutions. First consider the case where x becomes part of the feedback vertex set. Then x disappears from the pattern P_q and the remaining pattern is updated according to the pattern dicta 3 and 4. The age of the resultant pattern is updated by taking the minimum of its current age (if at all) and $A(P_q) + 1$. Consider the scenario in which x is *not* added to the feedback vertex set. This can lead to a feasible pattern for node p if and only if x is not incident with a loop of P_q. Apply the pattern dicta in order to obtain the new pattern.

The age of the resulting pattern is updated by taking the minimum of its current age and $A(P_q)$.

INTRODUCE NODES. Let p be an introduce node with child q and let $x \in S_p - S_q$. Every pattern for q translates straightaway into a pattern for p. The two patterns have the same age.

Correctness of the procedure described above is evident, and we obtain:

Theorem 6. *There exists an algorithm which runs in time $O(c^{\sqrt{k}\log k}n)$ for some constant c that solves the k-*FEEDBACK VERTEX SET *problem.*

5 Crystallization of Feedback Vertex Set

Consider a connected planar graph $G = (V, E)$. The *reduction algorithm* is defined as follows: DELETE pendant vertices from G and DISSOLVE vertices of degree 2; i.e., delete such a vertex but add an edge between its two neighbors if these were not connected already.

Proposition 1. *If H is obtained from a connected planar graph G by the reduction algorithm, then H is a planar rich graph and $cc(H) = cc(G)$.*

Lemma 6. *Let G be a* biconnected, plane, and rich *graph. Let F be a* FVS. *If some vertex x has degree s then either $x \in F$ or there are at least $\frac{s}{2}$ vertices of F on faces around x or contained in the interior of some of these faces.*

Proof. Let $x \notin F$ and let x have neighbors $x_0, \ldots, x_{s-1}$ in some clockwise ordering. Let $\mathbb{F}_i$ be the face incident with x, x_i, x_{i+1} (indices taken modulo s). Notice that since G is biconnected, all these faces are different. If every vertex of F is incident with at most 2 faces we are done. So assume some vertex $\alpha \in F$ is incident with at least 3 faces.

Assume first that α is incident with *every* face $\mathbb{F}_i$. (This takes care of the base case $s = 3$ as well.) Since every vertex has degree at least three, each x_i must have some neighbor leading to some cycle inside $\mathbb{F}_i$ or $\mathbb{F}_{i-1}$. Hence, either $x_i \in F$ (which also takes care of the case that $x_i = \alpha$ for some i) or there is some element of F in $\mathbb{F}_i$ or in the interior of $\mathbb{F}_i$ (or similar with $\mathbb{F}_{i-1}$), connected by a path with x_i. A well–chosen path like that avoids $x, \alpha, x_{i\pm1}$ and cannot be extended to $x_{i\pm1}$ since it would "cut" $\mathbb{F}_i$ (or $\mathbb{F}_{i-1}$) into two faces. It follows that in this case at least s vertices of F are incident with faces around x or contained in the interior of some $\mathbb{F}_i$.

Assume that α misses at least one $\mathbb{F}_i$ and let $s > 3$ (the degree of α is at least 4). For simplicity picture x, x_0, x_{s-1} on the outerface and assume that α is incident with $\mathbb{F}_0$ and $\mathbb{F}_i$ but is not incident with any face $\mathbb{F}_{i+1}, \ldots, \mathbb{F}_{s-1}$. Let $\mathbb{O}$ be the outerface of $\mathbb{F}_0 \cup \mathbb{F}_i$ and consider the subgraph H induced by $\mathbb{O}$ and all vertices of G placed in the interior of $\mathbb{O}$. If some vertex of $\mathbb{O}$ has degree 2 in H then dissolve it.

Remark 2. This dissolving process could destroy one face (when all vertices of $\mathbb{O}\backslash\{x,\alpha\}$ have degree 2 in H) since we don't allow multiple edges. If this is the case, we consider the outerface $\mathbb{O}'$ of $\mathbb{F}_0 \cup \mathbb{F}_{i-1}$ plus the interior of $\mathbb{F}_i$ which is connected to $\mathbb{O}'$ instead. In essence this doesn't change the vindication. For the argument's sake, assume that the degree of x remains $i+2$ in H.

By induction, we have at least $\frac{i+2}{2}$ vertices of F left in H (including α) since x has degree $i+2$ in H.
Now consider the subgraph H' induced by $\mathbb{O}$ and all vertices that lie outside $\mathbb{O}$. Without loss of generality, assume that x has degree $s-i$ in H'. Since only α is counted twice, we find at least $\frac{i+2}{2} + \frac{s-i}{2} - 1 = \frac{s}{2}$ vertices of F. □

Corollary 2. *If G is a biconnected, planar, and rich graph and if some vertex x of G has degree at least $2k+1$ then x is contained in any* FVS *of cardinality at most k.*

We now proceed to show how to deal with the general case. Assume G is a connected planar graph and apply the reduction algorithm. Call the new graph again G. Let x be a cut-vertex of G. If x is incident with some biconnected component B of G such that x has degree at least $2k+1$ in B then Corollary 2 says that x must be in any feedback vertex set of cardinality at most k. Hence, in that case we can remove x from the graph and search for a feedback vertex set of cardinality at most $k-1$ in $G-x$.

Assume that x is a cut-vertex but x has degree at most $2k$ in any biconnected component of G. If x is incident with at least $k+1$ biconnected components with at least 3 vertices, then again we know that x must be in any feedback vertex set of cardinality k since every such biconnected component adds at least one to the number of cycles having only x in common.

Finally assume that x is a cut-vertex such that the degree of x at most $2k$ in any biconnected component, and x incident with at most k biconnected components with at least 3 vertices. If x is incident with some bridge, then we can remove this bridge from G (without its end-vertices) and proceed with the two components. Hence we may assume that x is a cut-vertex which is incident only with biconnected components with at least 3 vertices. In that case the degree of x is at most $k(2k) = 2k^2$.
The following lemma of H. J. Voss [20] will prove to be quite handy.

Lemma 7. *If H is a rich graph, then for any feedback vertex set F of H, $|V(H)| \leq |F|(\Delta+1) - 2$, where $\Delta = \Delta(H)$ is the maximal degree in H.*

Let G' be the graph obtained from G after the previous reductions. Applying Voss' Lemma to G' yields $V(G') = O(k^3)$ since the degree of G' is $O(k^2)$.
Applying the algorithm of Section 4 to the kernel G' we now obtain our main result:

Theorem 7. *There exists an $O(c^{\sqrt{k}\log k} + n)$ algorithm for the k-*FEEDBACK VERTEX SET *problem on planar graphs for some constant c.*

6 Disjoint Cycles

Theorem 8. *Let G be a planar graph. Then $cc(G) \leq 5\ cp(G)$.*

Proof. Start with an empty set F and repeatedly take the following steps until not a single one is apt.

FIRSTLY, repeatedly, remove pendant vertices from G until G has no more vertices of degree ≤ 1.
SECONDLY, repeatedly, remove vertices of degree 2 that have neighbors which are not adjacent and connect its two neighbors by an edge.
THIRDLY, assume G has a vertex of degree 2 and the two neighbors are connected by an edge. Then put the triangle in F and take it out of G.
FOURTHLY, and finally, assume that G has no more vertices of degree ≤ 2. Then G has a face with length at most 5 by Lemma 4. Choose such a face $\mathbb{F}$ and put all its vertices in F. Take $\mathbb{F}$ out of G.

Let C be a chordless cycle of G. As long as C has length at least 4 and contains degree 2 vertices these are removed in the second step. If C ends up with 3 vertices and one of them has degree 2, then the third step ensures that C contains a vertex which is put in the feedback vertex set. If C has only vertices of degree at least 3 then none of its vertices is removed from the graph without it being put in the feedback vertex set (in the third or fourth phase). This shows that F is a feedback vertex set in G.
Notice that whenever vertices are put in the set F all the vertices of a cycle of length at most 5 are removed from the graph. For each cycle that is removed at most 5 vertices are put in F. This shows that F contains at most 5 times the maximum number of vertex disjoint cycles of G. □

The next theorem is now an immediate consequence of Theorem 8 and Theorem 5.

Theorem 9. *Let G be a planar graph with $cp(G) \leq k$. Then $tw(G) = O(\sqrt{k})$.*

With a similar technique as for k-FEEDBACK VERTEX SET we can now obtain:

Theorem 10. *There exists an algorithm running in time $O(c^{\sqrt{k}\log k}n)$ which checks if a planar graph G has k vertex disjoint cycles.*

Clearly, for any graph G, $cc(G) \geq cp(G)$. In general it was shown in [15,20] that there exist family of graphs G with $cc(G) = \Theta(cp(G)\log cp(G))$. We feel fairly confident that there is a much tighter link for planar graphs

Conjecture 2. [4] For planar graphs G, $cc(G) \leq 2\ cp(G)$.

Notice that the factor 2 would be optimal as shown by the wheel graphs. We show that the conjecture is true for outerplanar graphs

[4] Jones' conjecture.

Theorem 11. *Let G be an outerplanar graph. Then $cc(G) \leq 2\ cp(G)$.*

Proof. Without loss of universality we may assume that G is 2-edge connected. Recall that G can be thought of as a *tree of cycles*, i.e., G can be constructed by the following process.

1. Start with an arbitrary cycle.
2. Repeatedly, take a new cycle C and identify either a vertex or an *edge* of C with a vertex or an edge in the existing graph. The vertex or edge which is being identified with a vertex or edge in the existing graph is called the *attachment* of C_i. The cycle C_1 has an empty attachment.

Let $[C_1, \ldots, C_\ell]$ be the sequence of cycles in this construction of G. Consider the following algorithm constructing two sets $\mathbb{CC}$ and $\mathbb{CP}$.

FIRSTLY, set $\mathbb{CC} = \mathbb{CP} = \emptyset$.
SECONDLY, starting with $i = \ell$ working backwards down to $i = 2$:
  IF no vertex of C_i is in $\mathbb{CC}$ then we put C_i in the set $\mathbb{CP}$ and the vertices of the attachment in $\mathbb{CC}$.
  REMOVE all vertices of C_i from the graph except those of its attachment.
FINALLY, IF C_1 has no vertex in $\mathbb{CC}$ then put one arbitrary vertex of C_1 in $\mathbb{CC}$ and add C_1 to $\mathbb{CP}$.

It's a flimsy gain that $|\mathbb{CC}| \leq 2\ |\mathbb{CP}|$, since every attachment consists of at most 2 vertices. One moment's reflection shows that the cycles in $\mathbb{CP}$ are vertex disjoint since each of those has at least one vertex in $\mathbb{CC}$. Hence $|\mathbb{CP}| \leq cp(G)$. We show that $\mathbb{CC}$ hits all cycles in G. Assertion of this claim proves the theorem since $cc(G) \leq |\mathbb{CC}| \leq 2\ |\mathbb{CP}| \leq 2\ cp(G)$. Let X be any cycle in G. Notice that the vertex set of X must be the union of vertex sets of some constituents cycles in the tree of cycles. Hence X must contain *all* vertices of some constituent cycle. All constituent cycles receive at least one vertex in $\mathbb{CC}$ during the computation of $\mathbb{CC}$. □

Define $cc(k) = \max\{cc(G) \mid cp(G) \leq k\}$. It was shown by Erdös and Pósa that $cc(k) = \Theta(k \log k)$. Bollobás proved $cc(1) = 3$ ([11], see also [20]). Other unpublished proofs were given by Pósa and by Sachs (see [15,20]). Voss showed that $cc(2) = 6$ and $9 \leq cc(3) \leq 12$ [20,21].

Acknowledgments

We thank Ladislav Stacho and Edgar Martínez-Moro for nice discussions about Jones' conjecture. We also thank the people of the math department of Charles University in Prague for pointing out some errors in the original version of the paper.

References

1. Alber, J., H. Bodlaender, H. Fernau, and R. Niedermeier, Faster algorithms for domination of planar graphs, *Proc. of the* 17^{th} *SWAT'00*, Springer–Verlag LNCS 1851 (2000), pp. 97–110.
2. Alber, J., H. Fan, M. Fellows, H. Fernau, R. Niedermeier, F. Rosamond, U. Stege, Refined search tree techniques for the planar dominating set problem, Proceedings 26^{th} MFCS 2001.
3. Alber, Jochen, Henning Fernau, and Rolf Niedermeier, Parameterized complexity: Exponential speed-up for planar graph problems, Proceedings 28^{th} ICALP, 2001.
4. Alber, J., H. Fernau, R. Niedermeier, Graph separators: A parameterized view, Proceedings 7^{th} COCOON 2001.
5. Bodlaender, Hans, L., On disjoint cycles, *International Journal of Foundation of Computer Science* **5**, (1994), pp. 59–68.
6. Bodlaender, H., A linear time algorithm for finding tree decompositions of small treewidth, *SIAM J. Comput.* **25**, (1996), pp. 1305–1317.
7. Bodlaender, Hans L., Ton Kloks, Jochen Alber, Henning Fernau, and Rolf Niedermeier, Faster algorithms for planar dominating set and related problems, 40. Workshop Komplexitätstheorie, Datenstrukturen und Effiziente Algorithmen, Technische Universität Ilmenau, 2000.
8. Cai, Liming and D. W. Juedes, Subexponential parameterized algorithms collapse the W-hierarchy, Proceedings 28^{th} ICALP, 2001.
9. Cai, Liming, Michael Fellows, David Juedes, and Frances Rosamond, Efficient polynomial-time approximation schemes for problems on planar structures: upper and lowerbounds. Manuscript 2001.
10. Chang M.-S, T. Kloks, and C. M. Lee, Maximum clique transversals, Proceedings 27^{th} WG 2001, Springer-Verlag LNCS 2204 (2001), pp. 32–43.
11. Bollobás, B., Graphs without two independent circuits, (Hungarian), *K. Mat. Lapok* **15**, (1964), pp. 311–321.
12. Diestel, R., *Graph Theory*, Springer–Verlag, New York, 1997.
13. Downey, Rod G. and Mike R. Fellows, *Parameterized complexity*, Monographs in Computer Science, Springer–Verlag, 1999.
14. Downey, R. G. and M. R. Fellows, Parameterized computational feasibility, *Proceedings of feasible mathematics II* (eds. P. Clote and J. B. Remmel), Birkhauser (1995), pp. 219–244.
15. Erdös, P. and L. Pósa, On independent circuits contained in a graph, *Canad. J. Math.* **17**, (1964), pp. 347–352.
16. Fellows, M. R. and M. A. Langston, Nonconstructive tools for proving polynomial time decidability, *J. ACM* **35**, (1988), pp. 727–739.
17. Mohar, Bojan, Face covers and the genus problem for apex graphs, *Journal of Combinatorial Theory, Series B* **82**, (2001), pp. 102–117.
18. Telle,Jan Arne and Andrezj Proskurowski, Efficient sets in partial k-trees, *Discrete Applied Math.* **44**, (1993), pp. 109–117.
19. Kloks, T., *Treewidth-Computations and approxmations*, Springer-Verlag, LNCS 842, 1994.
20. Voss, H. J., Some properties of graphs containing k independent circuits, *Proc. Colloq. Tihany*, Academic Press, New York, 1968, pp. 321–334.
21. Voss, H. J., Über die Taillenweite in Graphen, die maximal unabhängige Kreise enthalten, und über die Anzahl der Knotenpunkte, die alle Kreise repräsentieren, *X. Internat. Wiss. Koll. TH Ilmenau 1965 Heft* **11**, Mathematische Probleme der Ökonomie und Rechentechnik, Vortragsreihe, 23–27.

A Multi-scale Algorithm for the Linear Arrangement Problem

Yehuda Koren and David Harel

Dept. of Computer Science and Applied Mathematics
The Weizmann Institute of Science, Rehovot, Israel
{yehuda,harel}@wisdom.weizmann.ac.il

Abstract. Finding a linear ordering of the vertices of a graph is a common problem arising in diverse applications. In this paper we present a linear-time algorithm for this problem, based on the multi-scale paradigm. Experimental results are similar to those of the best known approaches, while the running time is significantly better, enabling it to deal with much larger graphs. The paper contains a general multi-scale construction, which may be used for a broader range of ordering problems.

1 Introduction

Many computational tasks involve the combinatorial problem of ordering the vertices of a graph so as to optimize certain objective functions. The reader is referred to [3] for a detailed survey. We concentrate here on a common variant of the problem, known as *the minimum linear arrangement problem (MinLA)*, which consists of placing the n vertices of the graph at positions $1 \ldots n$ on a line, so as to minimize the sum of the edge lengths. Put differently, given n pins, some of which are connected by wires, we want to fix the pins in a linear sequence of holes so as to minimize the total wire length. This variant arises in VLSI design, graph drawing, modeling nervous activity in the cortex and job scheduling; see [3]. As is the case with many ordering problems, MinLA is NP-hard and the corresponding decision problem is NP-complete [4].

In this paper, we devise a linear-time heuristic algorithm for the MinLA problem. Our experiments with it show its attractiveness in terms of output quality and actual running time. The algorithm embodies a novel multi-scale scheme for the MinLA problem, which we have constructed to fit many linear ordering tasks. Hence, it could have applications beyond the MinLA problem.

2 The Minimum Linear Arrangement Problem

Let $G(V, E)$ be a graph, where $V = \{1 \ldots n\}$ is the set of n vertices, and E is the set of weighted edges, with w_{ij} being the weight of edge $\langle i, j \rangle \in E$. Throughout the paper we assume, without loss of generality, that G is connected, otherwise the problem we deal with can be solved independently for each connected component.

Definition 21 *A* linear arrangement *of G is a bijection $\pi : V \longrightarrow \{1, \ldots, n\}$. Equivalently, π is simply a permutation of V. We call $\pi(i)$ the* location *of vertex i.*

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 296–309, 2002.

Definition 22 *The* cost *of the arrangement π is defined to be:*

$$LA_\pi(G) \stackrel{def}{=} \sum_{\langle i,j\rangle \in E} w_{ij} \cdot |\pi(i) - \pi(j)|$$

The objective of the MinLA problem is to find a permutation π that minimizes $LA_\pi(G)$.

2.1 Algorithms

For certain classes of "well-structured" graphs, such as trees, grids and hyper-cubes, there exist polynomial time algorithms for solving the MinLA problem. However, the general problem is NP-hard. Thus most research involves approximation and heuristic algorithms. Here we briefly describe some relevant algorithms.

Dynamic Programming. Using dynamic programming, we can find the exact minimum linear arrangement in time $O(2^n \cdot |E|)$ and space $O(2^n)$, which is better than the naive $\Theta(n!)$ algorithm that scans all possible permutations. The complexity of this algorithm makes it useless for all, but very small graphs. However, this algorithm will serve us later on for reordering small subgraphs. The algorithm is based on the fact that the minimum linear arrangement problem possesses a "locality property", to the effect that the best ordering of a set of vertices S that placed to the right of the vertices in the set L and to the left of those in R, is independent of the internal ordering of L and R. This algorithm is described (along with a proof of the locality property) in the full version of this paper [10].

Spectral Sequencing. Spectral information has been successfully applied to vertex ordering problems, see e.g., [1,6]. Sequencing the vertices is done by sorting them according to their components in the Fiedler vector, which is the second smallest eigenvalue of the Laplacian matrix associated with the graph. Fortunately, computation of the Fiedler vector can be carried out very rapidly using a multi-scale methods of [2,9].

Simulated Annealing. Simulated annealing [8] is a powerful, general (if slow) optimization technique that is appropriate for the MinLA problem. It repeatedly modifies the ordering as follows: Given the current candidate ordering π, a new candidate $\hat{\pi}$ is generated that is close to π. If $LA_{\hat{\pi}} \leqslant LA_\pi$, the new ordering $\hat{\pi}$ is adopted. Otherwise, $\hat{\pi}$ may still be adopted, but with a probability that decreases as the process proceeds.

Petit [11] has conducted extensive tests of many algorithms for the MinLA problem, on several classes of graphs. We quote his conclusion:

> "...the best heuristic to approximate the MinLA problem on sparse graphs is Simulated Annealing when measuring the quality of the solutions. However, this heuristic is extremely slow whereas Spectral Sequencing gives results not too far from those obtained by Simulated Annealing, but in much less time. In the case that a good approximation suffices, Spectral Sequencing would clearly be the method of choice." [11]

Our algorithm, described in the following sections, produces results whose quality is similar to that of simulated annealing, but its running time is much better.

3 The Median Iteration

We now describe the *median iteration*, a rapid randomized algorithm for decreasing the cost of a linear arrangement. The heart of the method is a continuous relaxation of the MinLA problem, where we allow vertices to share the same place, or to be placed on non-integral points. Given a graph $G(V,E)$, a *linear placement* is a function $p : V \longrightarrow \mathbb{R}$, and its *cost* is:

$$LP_p(G) = \sum_{\langle i,j\rangle \in E} w_{ij} \cdot |p(i) - p(j)|$$

Notice that a linear arrangement, which was defined as a bijection $V \longrightarrow \{1,\ldots,n\}$, is a special case of a linear placement.

It is obvious that $LP_p(G)$ is minimal when all the vertices are placed at the same location. Hence, the minimal linear placement problem is not interesting. What will be useful for us is a certain process of minimizing $LP_p(G)$, which we shall use to improve linear arrangements. We begin by defining the median of the places of i's neighbors.

Definition 31 *Let $G(V,E)$ be a graph and p a linear placement. Given some $i \in V$, the* median *of i's neighborhood is denoted by $med_p^G(i)$.*
Since $med_p^G(i)$ is a usual median, it satisfies:

$$\sum_{j\in N(i), p(j)\leqslant med_p^G(i)} w_{ij} \geqslant \sum_{j\in N(i), p(j)> med_p^G(i)} w_{ij}$$

$$\sum_{j\in N(i), p(j)\geqslant med_p^G(i)} w_{ij} \geqslant \sum_{j\in N(i), p(j)< med_p^G(i)} w_{ij}$$

The main observation is the following:

Proposition 1. *Let $G(V,E)$ be a graph and p a linear placement. Fix the places of all vertices except a single $i \in V$. A location of i that minimizes the linear placement cost is $med_p^G(i)$.*

Proof. See the full version of this paper [10].

We can now define an iterative process for reducing the cost of a linear placement, based on minimizing the cost associated with each vertex separately:

Function Median_Iteration ($G(V,E)$, p, k)
% Parameters:
% $G(V,E)$ - a graph, p - a linear placement, k - no. of sweeps
  repeat k times:
    for every $i \in V$ **do**
      $p(i) \leftarrow med_p^G(i)$

  % When a valid linear arrangement is needed:
  Sort nodes according to respected entries in p
  for every $i \in V$ **do**
    $p(i) \leftarrow \langle$sorted place of $i\rangle$

Suppose that the initial value of p is a valid linear arrangement. When k is overly large, a significant fraction of V will be placed at a single point, leaving us very little information. But, for relatively small k, we will get many small clusters of vertices placed together. This provides important information about the global structure of the linear arrangement. Hence, we can construct a new valid linear arrangement by sorting the nodes according to their values in the linear placement p. The decision about the internal order of vertices that are placed at the same point is random.

We call this method *median iteration*. Since the median can be computed in linear time, the time complexity is $O(k \cdot |E|)$ for the k sweeps, plus $O(n \log n)$ for sorting the nodes by their place. Since we take k to be fixed (a typical value would be $k = 50$), the total time complexity is $O(|E| + n \log n)$. The median iteration is a simple way to reduce the cost of linear arrangement. It addresses the global structure of the ordering, while making local decisions randomly.

4 The Multi-scale Algorithm

The *multi-scale* (or, *multi-level*) paradigm is a powerful general technique that allows fast exploration of properties related to the 'global structure' of complex objects, that depend on many elements within. Multi-scale algorithms have proved to be successful in a variety of areas in physics, chemistry and engineering; see, e.g., [12]. Multi-scale techniques transform a high-dimensional problem in an iterative fashion into ones of increasingly lower dimensions,via a process called *coarsening*. On the coarsest scale the problem is solved exactly, following which a *refinement* process starts, whereby the solution is progressively projected back into higher and higher dimensions, updated appropriately at each scale, until the original problem is reproduced. A multi-scale algorithm must be tailored for the particular problem at hand, so that the coarsening process keeps the essence of the problem unchanged.

In terms of its use for dealing with graph-theoretic optimization problems, the multi-scale approach is widely used for graph-partitioning, see, e.g., [7]. More recently, Walshaw [13] has used this approach for the TSP and vertex-coloring problems. We have used it for the related problem of drawing graphs aesthetically [5,9].

4.1 Segment Graphs

One of the most prominent requirements of a good multi-scale algorithm is that it keep the inherent structure of the problem unchanged during coarsening. In our case, the form of the MinLA problem, as described in Section 2, will not be preserved during reasonable notions of coarsening, but we can define a more general problem that is preserved during coarsening, and it will contain the original problem as a special case. In fact, this general problem emerges naturally from trying to find an arrangement for a more general entity, that we shall call an *s-graph*, where the *s* stands for *segment*.

Definition 41 *An* s-graph *is a graph $G(V, E)$, whose vertices can be thought of as line segments; a vertex i is associated with a nonnegative real number l_i, denoting its length. Each edge $\langle i, j \rangle$ is associated with two coordinates, ${}_P\langle i, j \rangle$ and $\langle i, j \rangle_P$, denoting the positions of its endpoints inside vertices i and j, respectively. Clearly we have, $0 \leqslant_P \langle i, j \rangle \leqslant l_i$ and $0 \leqslant \langle i, j \rangle_P \leqslant l_j$.*

Fig. 1 shows an example of an s-graph. This is a 3-clique (a triangle), so the vertex set is $\{1,2,3\}$. The vertex lengths are $l_1 = 3$, $l_2 = 4$, $l_3 = 2$. Weights of the edges are $w_{13} = 2$, $w_{12} = 4$, $w_{23} = 3$. Finally, the coordinates of the edges are ${}_P\langle 1,3\rangle = 1$, $\langle 1,3\rangle_P = 0$, ${}_P\langle 1,2\rangle = 2$, $\langle 1,2\rangle_P = 1$, ${}_P\langle 2,3\rangle = 2$, $\langle 2,3\rangle_P = 1$.

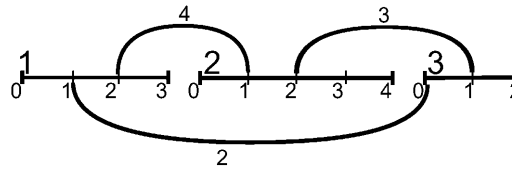


Fig. 1. An s-graph.

In a linear arrangement of an s-graph, we place the vertices on a line, in such a way that no two of them intersect. Since we seek an arrangement with short edges, we can safely rule out unused gaps between consecutive vertices. (Though, in the figures we have placed small gaps between consecutive vertices, to make the drawings clearer.) Hence, we define a linear arrangement of an s-graph as follows:

Definition 42 *Let G be an s-graph. A linear arrangement of G is a bijection $\pi : V \longrightarrow \{1,\ldots,n\}$. The location of vertex i is denoted by $p^\pi_G(i)$, and it satisfies $p^\pi_G(i) = \sum_{j,\pi(j)<\pi(i)} l_j$. We will often omit the G in $p^\pi_G(i)$.*

In Fig. 1 we are showing a linear arrangement for which the order of vertices is determined by the bijection: $\pi(1) = 1$, $\pi(2) = 2$, $\pi(3) = 3$. The exact locations of the vertices are: $p^\pi(1) = 0$, $p^\pi(2) = 3$, $p^\pi(3) = 7$.

Definition 43 *Given an s-graph G and a linear arrangement π,* the length of edge *$\langle i,j\rangle$, w.r.t. π, is defined to be:*

$$len^\pi_G(\langle i,j\rangle) \stackrel{def}{=} |p^\pi(j) + \langle i,j\rangle_P - p^\pi(i) -_P \langle i,j\rangle|$$

We will often omit the G in $len^\pi_G(\langle i,j\rangle)$.

Thus, in Fig. 1, $len^\pi(\langle 1,3\rangle) = 6$, $len^\pi(\langle 1,2\rangle) = 2$, $len^\pi(\langle 2,3\rangle) = 3$.

We now generalize the notion of the cost of a linear arrangement (see Def. 22), to handle s-graphs:

Definition 44 *Given an s-graph G, the* cost *of the linear arrangement π is defined to be :*

$$LA_\pi(G) \stackrel{def}{=} \sum_{\langle i,j\rangle \in E} w_{ij} \cdot len^\pi(\langle i,j\rangle)$$

Given a regular graph G (not an s-graph), if we set the length of each vertex to be 1 and for every $\langle i,j\rangle$ we take ${}_P\langle i,j\rangle = \langle i,j\rangle_P = 1$, the definition of the generalized cost of a linear arrangement coincide with the regular case.

4.2 The Coarsening Process

Let G be an s-graph with n nodes. A single coarsening step would be to replace G with a smaller s-graph G^R, containing only $m < n$ nodes (typically, $m \approx \frac{1}{2}n$). Of course, the structure of G^R should be strongly linked to that of G, so that in some crucial way both will describe approximately the same entity, but on different scales. Moreover, a good linear arrangement of G^R should correspond to a good arrangement of G.

Our attitude to coarsening is to place restrictions on the possible arrangements. These will reduce the search space, so that the restricted problem will be of a "lower dimension", and will involve a smaller graph. We will require that certain pairs of vertices be placed adjacently, in the hope that they will end up being fairly close in the minimal arrangement of G. In this optimistic situation there is a solution in the restricted subspace that is quite close to the optimal solution, up to some local refinement that does not change the global structure of the arrangement but affects only local portions thereof.

Restricting the Search Space. We would like to be able to restrict the search space in a way that satisfies two conflicting goals: First, the degrees of freedom should be significantly reduced, enabling representation by a much smaller graph. Second, the minimal arrangement should be affected only locally, while preserving its global structure.

We have found a way that attempts to achieve these two seemingly competing goals. Consider an initial arrangement π, constructed by some fast algorithm like spectral sequencing or median iteration. For simplicity, assume that the number of vertices n is even. For every pair of vertices v_1, v_2 such that $\pi(v_1) = 2i-1$ and $\pi(v_2) = 2i$ for some $i \geqslant 1$, introduce a restriction that requires any feasible linear arrangement φ to satisfy $\varphi(v_2) = \varphi(v_1) + 1$. The restricted problem is to arrange $n/2$ *restricted vertex pairs*. Hence, the size of the *restricted search space* is $(n/2)!$ while the size of the original search space was $n!$. If n is odd, choose at random some odd k, with $1 \leqslant k \leqslant n$, and restrict consecutive pairs in the series $\pi^{-1}(1), \ldots, \pi^{-1}(k-1), \pi^{-1}(k+1), \ldots, \pi^{-1}(n)$. Vertex $\pi^{-1}(k)$ is not restricted. If the initial arrangement π was not too far from the minimal arrangement, the restrictions just introduced affect the solution only locally, as desired.

Throughout this section, we illustrate the coarsening process using the s-graph G, shown in Fig. 2. All its edges have weight 1. We arbitrarily order the vertices by their names and denote this linear arrangement by π. As the reader can verify, its cost is $LA_\pi(G) = 43$. We should remark that, in general, smarter orderings, such as that produced by the median iteration, work much better than an arbitrary initialization.

As a consequence of the initial linear arrangement π, the set of restricted vertex pairs in the example is $R = \{(1,2),(3,4),(5,6),(7,8)\}$. This restricts every linear arrangement to place vertex 2 immediately after vertex 1, vertex 4 immediately after vertex 3, and so on.

We now show how to formulate the restricted problem as a linear arrangement problem on a much smaller graph.

The Coarsening Step. Given an s-graph $G(V,E)$ and a set $R \subset V \times V$ of restricted vertex pairs, we construct a coarser graph $G^R(V^R, E^R)$, such that there is a 1-1 correspondence between linear arrangements of G^R and linear arrangements in the restricted search space of G. G^R is produced by contracting pairs of restricted vertices. Vertex lengths and edge

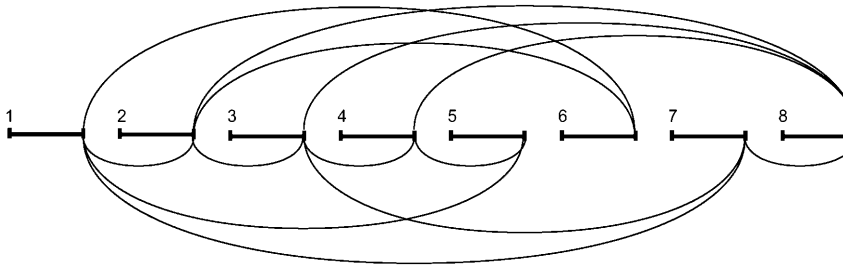


Fig. 2. Linear arrangement of the graph G. Every vertex is of length 1.

weights are preserved, by accumulating them. Positions of the new edges are calculated by averaging those of old edges.

Here is the formal construction of G^R. To ease the technicalities, we will be assuming that if $\langle i,j\rangle \notin E$, then $w_{ij} = 0$ and ${}_P\langle i,j\rangle = \langle i,j\rangle_P = 0$. Also, for simplicity, we assume that n is even, so that every $v \in V$ appears in a single restricted pair.

- The coarse vertex set V^R is simply the set of restricted vertex pairs, R.
- The length of a coarse vertex (v_1, v_2) is $l_{(v_1,v_2)} = l_{v_1} + l_{v_2}$
- The coarse edge set is defined as follows:

$$E^R = \left\{ \langle (v_1,v_2),(v_3,v_4)\rangle \;\middle|\; \begin{array}{l} (v_1,v_2),(v_3,v_4) \in V^R, \\ (v_1,v_2) \neq (v_3,v_4), \\ \langle v_1,v_3\rangle \in E \text{ or } \langle v_1,v_4\rangle \in E \\ \text{ or } \langle v_2,v_3\rangle \in E \text{ or } \langle v_1,v_4\rangle \in E \end{array} \right\}$$

- The weight of a coarse edge $\langle (v_1,v_2),(v_3,v_4)\rangle$, denoted as usual by $w_{(v_1,v_2)(v_3,v_4)}$, is $w_{v_1v_3} + w_{v_1v_4} + w_{v_2v_3} + w_{v_2v_4}$.
- The positions of the endpoints of the coarse edge $\langle (v_1,v_2),(v_3,v_4)\rangle$ are the weighted averages of the endpoints of the corresponding fine edges:

$${}_P\langle (v_1,v_2),(v_3,v_4)\rangle = \frac{\left(\sum_{i\in\{v_1,v_2\},j\in\{v_3,v_4\}} w_{ij}\cdot {}_P\langle i,j\rangle\right)+l_{v_1}\cdot(w_{v_2v_3}+w_{v_2v_4})}{w_{v_1v_3}+w_{v_1v_4}+w_{v_2v_3}+w_{v_2v_4}}$$

$$\langle (v_1,v_2),(v_3,v_4)\rangle_P = \frac{\left(\sum_{i\in\{v_1,v_2\},j\in\{v_3,v_4\}} w_{ij}\cdot \langle i,j\rangle_P\right)+l_{v_3}\cdot(w_{v_1v_4}+w_{v_2v_4})}{w_{v_1v_3}+w_{v_1v_4}+w_{v_2v_3}+w_{v_2v_4}}$$

When n is odd, there exists a single unrestricted vertex, v, in which case we add a vertex (v,v) to V^R, where $l_{(v,v)} = l_v$. Definitions regarding the edges adjacent to (v,v) should be slightly modified.

Hence, for our example graph G, as arranged in Fig. 2, we will get the coarse graph G^R shown in Fig 3. We denote the linear arrangement of G^R by φ. The reader can verify that its cost is $LA_\varphi(G^R) = 40$.

There is a simple 1-1 correspondence between the restricted linear arrangements of the fine graph G and the linear arrangements of the coarse graph G^R:

Definition 45 *Let G be a graph, R a set of restricted vertex pairs, and π a restricted linear arrangement of G. Now let φ be a linear arrangement of the respective coarse*

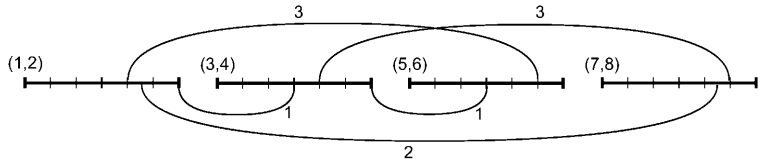


Fig. 3. The graph G^R that is obtained by coarsening the graph of Fig. 3. The length of each vertex is 2, and weights and coordinates of edges were calculated so as to preserve the information of the original graph.

graph G^R. The two arrangements are corresponding *if for every $(v_1, v_2) \in R$, we have:*

$$\pi(v_1) = 1 + \sum_{\varphi((i,j)) < \varphi((v_1,v_2))} |(i,j)|$$

where

$$|(i,j)| \stackrel{def}{=} \begin{cases} 2, & i \neq j \\ 1, & i = j \end{cases}$$

Equivalently, in the inverse direction, for every $(v_1, v_2) \in R$:

$$\varphi((v_1,v_2)) = \sum_{(i,j)\in R,\, \pi(i) \leqslant \pi(v_1)} 1$$

Notice that when n is even $(i,j) \in R$ implies that $i \neq j$, so we can simply write the condition for correspondence as follows: for every $(v_1, v_2) \in R$:

$$\pi(v_1) = 2 * \varphi((v_1,v_2)) - 1$$

Using the close relationship between lengths of vertices in G and G^R, we observe that for two corresponding linear arrangements π and φ, the actual locations of the vertices are related, for each $(v_1, v_2) \in R$, by:

$$p^{\pi}_{G}(v_1) = p^{\varphi}_{G^R}((v_1,v_2)) \tag{1}$$

It is straightforward to verify in our example that π and φ, are corresponding linear arrangements. Notice that: $p^{\pi}_{G}(1) = p^{\varphi}_{G^R}((1,2)) = 0$, $p^{\pi}_{G}(3) = p^{\varphi}_{G^R}((3,4)) = 2$, $p^{\pi}_{G}(5) = p^{\varphi}_{G^R}((5,6)) = 4$, $p^{\pi}_{G}(7) = p^{\varphi}_{G^R}((7,8)) = 6$.

During coarsening, several edges become self-loops, and are canceled. We can calculate the cost related to these lost edges:

Definition 46 *Given a graph $G(V,E)$ and a set of restricted vertex pairs R, define* the internal cost of *R to be:*

$$C(R) = \sum_{\langle i,j\rangle \in E, (i,j)\in R} w_{ij} \cdot (\langle i,j\rangle_P + l_i -_P \langle i,j\rangle)$$

The most important fact is that the costs of two corresponding linear arrangements are identical up to adding $C(R)$, which is independent of the particular arrangements.

Lemma 1. *Let G be a graph and R a set of restricted vertex pairs. Let π and φ be corresponding linear arrangements of G and G^R, respectively. Then, $LA_\pi(G) = LA_\varphi(G^R) + C(R)$.*

The proof is rather technical and is given in the full version of this paper [10].

In the example graph three edges lie within a restricted pair, and they are $\langle 1,2\rangle$, $\langle 3,4\rangle$, $\langle 7,8\rangle$. Thus, the internal cost is $C(R) = 3$. In agreement with Lemma 1, $LA_\pi(G) = LA_\varphi(G^R) + C(R) = 40 + 3$.

As can be seen in Fig. 3, the visual complexity of G^R is much lower than that of G. In fact, it is not too hard to compute the minimum linear arrangement φ^* of G^R, shown in Fig. 4(a). Its cost is $LA_{\varphi^*}(G^R) = 24$. After solving the problem for the coarse graph, we interpolate the resulting arrangement to the original graph, obtaining the corresponding linear arrangement π^* of G, as shown in Fig. 4(b). In accordance with Lemma 1, $LA_{\pi^*}(G) = LA_{\varphi^*}(G^R) + C(R) = 24 + 3$, a significant decrease in the arrangement cost, which may also be appreciated visually.

We should remark that in the more general case the coarse graph would still be too large to facilitate finding its optimal linear arrangement. Hence, as we shall explain, our strategy is to continue recursively with the coarsening process until we reach a reasonably small graph.

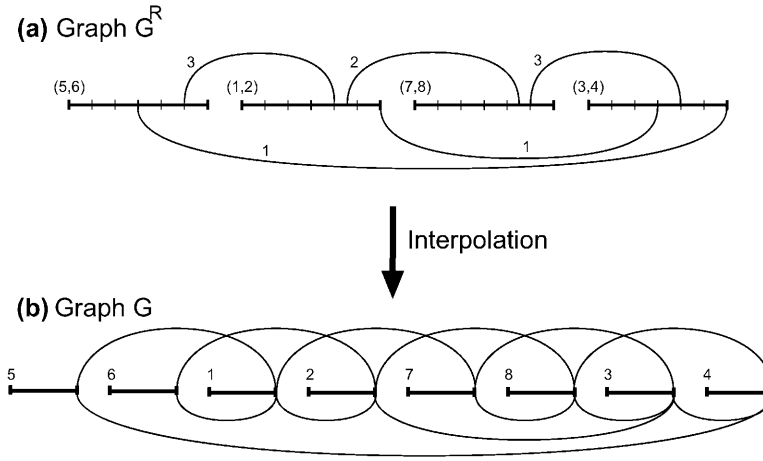


Fig. 4. Optimizing the linear arrangement of G^R facilitates the construction of a better arrangement for G.

We conclude that given a set R of restricted vertex pairs, we can construct a coarser graph G^R with $\lceil \frac{n}{2} \rceil$ vertices. Linear arrangements of G^R correspond to linear arrangements in the restricted search space of G, and have the same costs (up to adding a uniform constant). For reasonable restrictions, the restricted search space contains arrangements that have structure similar to those of the minimum cost arrangements. Hence, we may hope that a good arrangement of G^R corresponds to a reasonably good arrangement of G, and this arrangement can become a truly good one by the local refinement process we now describe.

4.3 Local Refinement

Given a linear arrangement π of a graph G, we seek a method that improves the quality of π. Now, in our approach π will typically possess a satisfactory global structure, as a result of minimizing the problem on the restricted search space. Consequently, we have constructed a local refinement process that specializes in locally optimizing arrangements. The idea is to take each k consecutive vertices in π and to rearrange them internally such that the global arrangement cost is minimized. Such an optimal arrangement is achieved using the dynamic programming algorithm mentioned in Subsection 2.1, in a time $O(2^k)$. To improve the results the process can be repeated several times.

The local refinement process is depicted in Figure 5. Taking k to be constant (in our experiments we usually set $k = 6$), the time complexity is $O(|E|)$, and the space requirements are also linear in the size of the input.

Function Local_Refine ($G(V = \{1, \ldots, n\}, E), \pi$)
% Parameters:
% $G(V, E)$ – an s-graph , π – a linear arrangement of V
% Constants:
% $interval[= 6]$ – number of consecutive vertices to optimize together
% $repetitions[= 5]$ – number of iterations of the algorithm
% Auxiliary function:
% MinLA_local($G(V, E)$, π, j, $j + k - 1$) – finds best internal
% ordering of the k vertices placed in $\pi[j], \ldots, \pi[j + k - 1]$.
for $i = 1$ to $repetitions$
for $j = 1$ to $n - interval + 1$
MinLA_local($G(V, E)$, π, j, $j + interval - 1$)
end for
end for

Fig. 5. The local refinement process.

4.4 Putting It All Together

We can now put all this together to obtain the multi scale algorithm.

Preprocessing Stage. Prior to running the algorithm we want to obtain, rapidly, a reasonable linear arrangement. To that end, we first order the vertices using spectral sequencing (see Section 2) and then improve the result by applying the median iteration.

The V-cycle. Our basic multi-scale tool is the *V-cycle*, which starts by refining the arrangement locally. The intention of the refinement is not only to minimize the arrangement cost, but to also improve the quality of the coarsening step that follows it. The next step is to coarsen the graph based on restricting consecutive vertex pairs of the current arrangement. The problem is then solved in the restricted solution space, by running the V-cycle recursively on the coarse graph. Once we have found a good solution in the restricted solution space, we refine it locally (in the full solution space).

In the most optimistic case, the optimal solution on each scale is reachable from the best restricted solution by local optimization, so that we are guaranteed to find it. Our hope is that in a realistic case a pretty good solution is reachable in this fashion, which, of course, depends on the quality of the refinement process and on the restrictions we impose.

Figure 6(a) shows the V-cycle algorithm. It uses several functions. The first of these, 'coarsen(G, π, G^R, π^R)', receives an s-graph G and a linear arrangement of its vertices, π, and produces a coarse graph G^R and a linear arrangement its vertices, π^R. Here, π and π^R are corresponding linear arrangements as per Def. 45. The way to construct G^R and π^R is described in Subsection 4.2. The function 'interpolate(G^R, π^R, π)', receives a coarse graph G^R and a linear arrangement π^R, and produces the corresponding linear arrangement of the fine graph, π.

The recursion in the V-cycle continues as long as G is 'large enough', and a good termination condition would be when the graph is small enough to facilitate finding the optimum linear arrangement directly. In all the graphs that we have tested, local refinement was useless after more than five levels of recursion, due to the good global structure of the initial arrangement.

All the functions in the V-cycle run in linear time and space, and the recursive call is to a problem of approximately half the size. Thus, the time and space complexity of the V-cycle algorithm is $O(|E|)$.

(a)

Function V-cycle ($G(V, E)$, π)
% Parameters:
% $G(V, E)$ – an s-graph,
π – a linear arrangement of V

Local_Refine(G, π)
if G is 'large enough' **then**
coarsen(G, π, G^R, π^R)
V-cycle(G^R, π^R)
interpolate(G^R, π^R, π)
Local_Refine(G, π)
end if

(b)

Function MS_MinLA ($G(V, E)$, π, k)
% Parameters:
% $G(V, E)$ – an s-graph
% π – a linear arrangement of V
% k – no. of V-cycle iterations

Spectral_Sequencing(G, π)
Median_Iteration(G, π, 40)
for $i = 1$ to k
Median_Iteration(G, π, 10)
V-cycle(G, π)
end for

Fig. 6. (a) The V-cycle (multi-scale) algorithm **(b)** The full algorithm.

Iterating the V-cycle. The V-cycle can benefit from starting out with a better arrangement. Thus, repeating the V-cycle can improve its results. In fact, this kind of iteration is common in multi-scale algorithms; see [12]. Iterating the V-cycle can only decrease the cost of the arrangement. In fact, our experience with the algorithm is that after a few iterations the process seems to converge, after which improvement is very slow, if at all. We can obtain better results by carrying out several median iterations (e.g., 10), before entering each new V-cycle. This perturbs the arrangement and provides the next V-cycle with a different starting point, thus overcoming premature convergence. In practice we

gradually reduce the number of sweeps of the median iteration, lessening its effect; see [10]. The full algorithm appears in Figure 6(b).

5 Experiments

A Case Study. We want to demonstrate the benefit of embedding the local refinement process inside the multi-scale scheme. To that end, we have chosen the 10-dimensional hyper-cube, which consist of 1024 vertices and 5120 edges. What we have observed for this example is typical of many graphs we have considered. The optimal linear arrangement of this graph, π^*, can be computed efficiently, and its cost is 523776; see [11].

As the first stage we applied spectral sequencing to the hyper-cube, resulting in an arrangement with cost 659490. We then improved the arrangement using 40 median iterations. The cost of the resulting arrangement, π^0, is 599958. We should note that this is the random part of the experiment, and other runs provide quite different arrangements (while the median iteration consistently improves the initial arrangement rather significantly).

We then tried to improve π^0 in two ways. First, we applied 100 iterations of the Local_Refine function (setting the local constant $repetitions$ to 100), and obtained an arrangement with cost 593120. More iterations of Local_Refine did not decrease the cost any further. Second, we applied a single V-cycle to π^0. During this, the Local_Refine function performed only 5 iterations in each of its executions, thus the running time was much faster. The result of this was the optimal linear arrangement π^*, a clear improvement.

This illustrates the effect of embedding the local refinement process in the V-cycle multi-scale scheme,which improves both running time and output quality. In general, embedding a local refinement process in a multi-scale scheme adds global considerations to the refinement process, because local moves on a coarse graph express more global ones on the original graph. Thus, in some sense, the “wisdom" of a multi-scale algorithm can be divided into two parts: One is related to local optimization and it resides in the local optimization process, and the other deals with the global properties of the problem and it resides in the coarsening construction.

Petit's Test Suite. We have implemented the algorithm using C++, and use our ACE algorithm [9] for spectral sequencing. The program runs on a 700MHZ Pentium III, under Windows NT. We have tested it on a test suite of graphs compiled by Petit [11], chosen so as to represent several graph families.

Petit computed linear arrangements of these graphs using many algorithms. The best results were obtained by first running spectral sequencing, and then carrying out a refinement using simulated annealing (SA). The SA process was run for $C \cdot n^2$ iterations ($C > 1$ is a constant related to the rate of temperature decrease in SA). We cannot compare his running times with our directly, since the platforms are different. But to get an impression, the running time of SA for the largest graph in Petit's set, the whitaker3, was over 11 hours. We provide the costs computed by SA in Table 1.

For each graph in this set, we ran our multi-scale algorithm (that of Figure 6(b)) first with a single V-cycle and then with ten (i.e., with $k = 1$ and $k = 10$). The results appear

Table 1. The test suit of Petit [11]. For each graph the table shows the cost of the linear arrangement and the running time in seconds (separated by a slash), for spectral sequencing, median iteration, multi-scale with a single V-cycle and multi-scale with 10 iterated V-cycles. The times for multi-scale include spectral sequencing, median iteration and V-cycles. The rightmost column shows the cost computed by Petit using simulated annealing (SA).

Graph Name	Size |V|	Size |E|	Spectral Sequencing	Median Iteration	Multi-Scale 1 Iteration	Multi-Scale 10 Iterations	Cost using SA
randomA1	1000	4974	1156890/.08	1020028/.03	938168/2.39	909126/22.94	900992
randomA2	000	24738	7377237/.27	7284497/.42	6755035/7.02	6606174/63.94	6584658
randomA3	1000	49820	15279645/.38	16543660/.96	14731040/12.12	14457452/109.17	14310861
randomA4	1000	8177	2167121/.11	1955837/.05	1807038/3.15	1765217/30.04	1753265
randomG4	1000	8173	195054/.06	175879/.06	154990/2.11	149513/18.97	150940
hc10	1024	5120	580910/.02	542476/.03	523776/1.9	523776/18.51	548352
mesh33x33	1089	2112	35750/.04	34118/.01	32486/1.04	31729/9.91	34515
bintree10	1023	1022	52992/.09	6114/0	4246/.88	3950/7.96	4069
3elt	4720	13722	429086/.43	394238/.11	385572/6.55	373464/61.24	375387
airfoil1	4253	12289	352897/.37	312387/.11	305191/6.53	291794/61.02	288977
whitaker3	9800	28989	1259607/.92	1238557/.28	1226902/13.86	1205919/127.79	1199777
c1y	828	1749	103224/.02	71359/.01	66836/.92	64934/8.88	63858
c2y	980	2102	95346/.03	84259/.01	82070/1.12	80148/10.81	79500
c3y	1327	2844	175700/.04	145332/.02	137131/1.58	127315/15.3	124708
c4y	1366	2915	133044/.05	124576/.02	121460/1.62	118437/15.57	117254
c5y	1202	2557	144603/.04	115239/0.02	109280/1.39	104076/13.33	102769
gd95c	62	144	599/.02	595/0	509/.08	509/.61	509
gd96a	1076	1676	170700/.04	122567/.01	115525/1.24	106668/11.94	104698
gd96b	111	193	1836/0	1825/.01	1435/.11	1434/.98	1416
gd96c	65	125	701/0	601/0	522/.06	519/.6	519
gd96d	180	228	3691/0	2807/.01	2438/.16	2420/1.53	2393

in Table 1, which also provides the results of spectral sequencing and median iteration — the first stage of the algorithm. The quality of our results after 10 V-cycle iterations is comparable to that of Petit's SA, but our running time is significantly better. The results may be further improved by executing more iterations and/or by increasing the value of the constant $interval$ in the function Local_Refine. However, the rate of improvement would be slow.

6 Discussion

We have presented a multi-scale algorithm for the minimum linear arrangement problem. It produces arrangements on a par with the best known algorithms, but requires significantly less time, allowing it to deal with much larger graphs (as shown in [10]).

The heart of the algorithm is a novel construction of a hierarchy of coarse graphs, corresponding to increasingly restricted parts of the original problem. A local refinement scheme becomes considerably improved when embedded in the multi-scale construction. In fact, the multi-scale construction is independent of the specific refinement heuristic, and we believe that other heuristics could benefit from being embedded in such a multi-scale scheme. Also, variants of the proposed multi-scale construction can be used for other vertex ordering problems, such as bandwidth or cutwidth minimization [3].

An interesting property of our multi-scale construction has to do with the way we build coarse graphs. The common approach to coarsening is based solely on the graph 's structure. Most frequently such a coarsening is performed by contracting a set of edges, such as those that participate in a maximal matching (see, e.g. [7]). In contrast, our approach to coarsening relies on a specific solution of the problem, on which we impose

several restrictions. When we have a globally good approximate solution, our approach has the advantage of utilizing the wisdom of this solution for peforming a coarsening that forces reasonable restrictions. On the other hand, coarsening based on graph structure is most often very local in nature, and may impose globally bad restrictions, like identifying vertices that should be very distant in the optimal solution. See also [13].

The median iteration process we have proposed seems to have value in its own right. It is actually an extremely fast method for decreasing the cost of an arrangement, using a continuous relaxation of the original problem. In [10] it was applied successfully to graphs with millions of edges.

Acknowledgement

We would like to thank Chris Walshaw for his useful comments and corrections.

References

1. J. E. Atkins, E. G. Boman and B. Hendrickson, “A Spectral Algorithm for Seriation and the Consecutive Ones Problem", *SIAM Journal on Computing* **28** (1998), 297–310.
2. S. T. Barnard and H. D. Simon, “A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems", *Concurrency: Practice & Experience* **6** (1994), 101–117.
3. J. Diaz, J. Petit and M. Serna, “A Survey on Graph Layout Problems", Technical report LSI-00-61-R, Universitat Politècnica de Catalunya, Departament de Llenguatges i Sistemes Informàtics, 2000.
4. M. R. Garey and D. S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.
5. D. Harel and Y. Koren, “A Fast Multi-Scale Method for Drawing Graphs Nicely”, *Proceedings of Graph Drawing’00* , Lecture Notes in Computer Science, Vol. 1984, Springer Verlag, pp. 183–196, 2000.
6. M. Juvan and B. Mohar, “Optimal Linear Labelings and Eigenvalues of Graphs", *Discrete Applied Math.* **36** (1992), 153–168.
7. G. Karypis and V. Kumar, “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs", *SIAM Journal on Scientific Computing* **20** (1999), 359–392.
8. S. Kirkpatrick J. Gelatt and M.P. Vecchi, “Optimization by Simulated Annealing", *Science* **220** (1983), 671-680.
9. Y. Koren, L. Carmel and D. Harel “ACE: A Fast Multiscale Eigenvectors Computation for Drawing Huge Graphs", Technical Report MCS01-17, Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, 2001. To appear in Proc. IEEE Information Visualization 2002.
10. Y. Koren and D. Harel “Multi-Scale Algorithm for the Linear Arrangement Problem", Technical Report MCS02-04, Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, 2002. Available at: `www.wisdom.weizmann.ac.il/reports.html`
11. J. Petit, “Experiments on the Minimum Linear Arrangement Problem", Technical report LSI-01-7-R, Universitat Politècnica de Catalunya, Departament de Llenguatges i Sistemes Informàtics, 2001.
12. S. H. Teng, “Coarsening, Sampling, and Smoothing: Elements of the Multilevel Method", *Algorithms for Parallel Processing*, IMA Volumes in Mathematics and its Applications, Vol. 105, Springer Verlag, pp. 247–276, 1999.
13. C. Walshaw, “Multilevel Refinement for Combinatorial Optimisation Problems", Technical Report 01/IM/73, Comp. Math. Sci., Univ. Greenwich, London, UK, 2001.

On the b-Chromatic Number of Graphs

Jan Kratochvíl[1,⋆], Zsolt Tuza[2,⋆⋆], and Margit Voigt[3]

[1] Department of Applied Mathematics and
Institute for Theoretical Computer Science
Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic
honza@kam.ms.mff.cuni.cz
[2] Computer and Automation Institute, Hungarian Academy of Sciences
Kende ú 13-17, H-1111 Budapest; and
Department of Computer Science, University of Veszprém, Hungary
tuza@sztaki.hu
[3] Insitut für Mathematik, Technische Universität Ilmenau
D-98684 Ilmenau, Germany
voigt@mathematik.tu-ilmenau.de

Abstract. The b-chromatic number $b(G)$ of a graph $G = (V, E)$ is the largest integer k such that G admits a vertex partition into k independent sets X_i $(i = 1, \ldots, k)$ such that each X_i contains a vertex x_i adjacent to at least one vertex of each X_j, $j \neq i$. We discuss on the tightness of some bounds on $b(G)$ and on the complexity of determining $b(G)$. We also determine the asymptotic behavior of $b(G_{n,p})$ for the random graph, within the accuracy of a multiplicative factor $2 + o(1)$ as $n \to \infty$.

1 Introduction

A *b-coloring* of a graph G by k colors is a proper coloring of the vertices of G such that in each color class there exists a vertex having neighbors in all the other $k-1$ color classes. Such a vertex will be called a *color-dominating vertex*. The *b-chromatic number* $b(G)$ of a graph G is the maximum k for which G has a b-coloring by k colors.

The b-chromatic number was introduced in [6]. The motivation, similarly as for the previously studied achromatic number (cf. e.g., [1,2,3,5,8]), comes from algorithmic graph theory. Suppose one colors a given graph properly, but in an arbitrary way. After all vertices are colored, one would wish to perform some simple operations to reduce the number of colors. The simplest operation at hand is recoloring all vertices in one color class with another color, i.e., unifying two color classes. In an achromatic coloring there is an edge between any two color classes, and hence such recoloring is impossible. The achromatic number of a graph is thus the worst case number of colors that may be needed to color the graph under the above described heuristics.

⋆ Supported by the Ministery of Education of the Czech Republic as project LN00A056.

⋆⋆ Research supported in part by the Hungarian Research Fund under grant OTKA T–032969.

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 310–320, 2002.

A slightly more involved operation would take one color class and recolor its vertices, but not necessarily each with the same color. Obviously, such recoloring is impossible if each color class contains a color-dominating vertex. Hence the b-chromatic number of the graph serves as the tight upper bound for the number of colors used by this more sophisticated coloring heuristics.

From this point of view, both complexity results and tight bounds for the b-chromatic number are interesting. We strengthen some complexity results of [7] and present several new bounds on the b-chromatic number, including a tight (upto a small multiplicative constant) bound on the b-chromatic number of the random graph. To state the results we need to recall the upper bound presented already in [6].

Assume that the vertices $v_1, \ldots, v_n$ of G are ordered such that $d(v_1) \geq d(v_2) \geq \ldots \geq d(v_n)$, where $d(x)$ denotes the degree of x. Let

$$t(G) := \max\{i \mid d(v_i) \geq i - 1\}$$

be the maximum number i such that G contains at least i vertices of degree $\geq i - 1$. Obviously, $t(G) \leq \Delta(G) + 1$ where $\Delta(G)$ denotes the maximum degree of G.

Proposition 1. ([6,7]) *For every graph G, $\chi(G) \leq b(G) \leq t(G)$.*

This upper bound is tight. For example, consider a split graph G (i.e., a graph $G = (V, E)$ which has a partition $V = I \cup C$ $(I \cap C = \emptyset)$ such that I is an independent set in G and C builds a clique in G. It is easy to see that $b(G) = t(G) = \chi(G)$. On the other hand, $b(G)$ may be arbitrarily far from $t(G)$, shown e.g., by the complete bipartite graph $K_{n,n}$ [6].

Our paper is organized as follows. In the next section we present some results showing that under certain sufficient conditions the upper bound $t(G)$ for the b-chromatic number is met. The proofs are based on the idea of precoloring some vertices which should then play the role of the color-dominating vertices, and extending the precoloring to the entire graph by making use of results on list colorings and graph choosability.

In Sections 3 and 4 we present our complexity results. First we show a characterization of graphs having b-chromatic number 2 (note that such graphs are necessarily bipartite). This characterization yields a polynomial time algorithm to decide $b(G) = 2$. Thus two-b-colorability, similarly to ordinary bi-colorability, is an easy problem. However, b-colorbility defines a problem more difficult than ordinary colorability. We prove that the problem to decide whether there is a b-coloring by $t(G)$ colors is NP-complete even for connected bipartite graphs and $t(G) = \Delta(G)+1$, thus strengthening NP-completeness results of [7,10]. This result shows that already for bipartite graphs, b-colorability becomes a difficult problem.

Finally, in Section 5, we study the behaviour of 'typical' graphs and we succeed in taming the b-chromatic number with respect to ordinary chromatic number. We prove that for every constant edge probability p $(0 < p < 1)$ the

b-chromatic number of the random graph $G_{n,p}$ asymptotically is not larger than twice its chromatic number.

2 Precoloring of Color-Dominating Vertices

For a given graph G denote by $dist(v, w)$ the number of edges in a shortest path connecting v and w. If v and w belong to different components of a disconnected graph G then $dist(v, w) = \infty$. For the sake of brevity, we write $\Delta = \Delta(G)$ if it is clear what graph G we have in mind.

Theorem 1. *Let G be a graph containing vertices $v_1, \ldots, v_{\Delta+1}$ such that $d(v_i) = \Delta$ for all i and $dist(v_i, v_j) \geq 4$ for all $i \neq j$. Then $b(G) = \Delta + 1$.*

Proof. We shall construct a b-coloring of G by $\Delta+1$ colors. For all $i = 1, \ldots, \Delta+1$, color vertex v_i by color i and its neighborhood by colors $\{1, \ldots, \Delta + 1\} \setminus \{i\}$ such that all Δ colors are used in the neighborhood. So far this coloring is a proper coloring, because the distance constraint implies that there are no adjacent vertices with the same color, and each of the vertices v_i is color-dominating. Furthermore the precoloring can easily be extended to a coloring of the entire graph by the greedy algorithm. Note that every vertex has degree at most Δ and we have $\Delta + 1$ available colors. Thus $b(G) = \Delta + 1$.

An easy example for the graphs considered in Theorem 1 is the disconnected graph G consisting of $t + 1$ stars $K_{1,t}$. The idea can also be used to prove the following result on d-regular graphs.

Corollary 1. *Let G be a d-regular graph ($d \geq 2$) with at least d^4 vertices. Then $b(G) = d + 1$.*

Proof. We intend to find $d+1$ vertices $v_1, \ldots, v_{d+1}$ where $dist(v_i, v_j) \geq 4$ for all $i \neq j$. Take an arbitrary vertex v_1 and remove it together with its first, second and third neighborhood from the graph. The number of removed vertices is at most $1 + d + d(d-1) + d(d-1)^2 = d^3 - d^2 + d + 1 < d^3$. Continue in this way removing vertices $v_2, \ldots, v_d$ and the corresponding neighborhoods. The number of removed vertices altogether is smaller than d^4, and there remains at least one vertex v_{d+1}. Furthermore $dist(v_i, v_j) \geq 4$ for all $i \neq j$. Applying Theorem 1 we obtain $b(G) = \Delta + 1 = d + 1$.

For $d = 2$ it is easy to see that only the graphs consisting of one or two 4-cycles have b-chromatic number 2 whereas all other 2-regular graphs have b-chromatic number 3. So far there was no problem to extend the precoloring because we had $\Delta + 1$ available colors. If we would like to find a b-coloring by fewer colors the situation becomes more complicated. An example of a result describing such situation is below. Denote by $girth(G)$ the girth of G that is the length of a shortest cycle in G.

Theorem 2. *Let G be a planar graph of girth at least 5, and $t \geq 4$ an integer. If G contains t vertices $v_1, \ldots, v_t$ such that $d(v_i) \geq t-1$ for all $i = 1, \ldots, t$ and $dist(v_i, v_j) \geq 5$ for all $i \neq j$, then $b(G) \geq t$.*

Proof. Again we try to find a b-coloring by t colors precoloring the vertices $v_1, \ldots, v_t$ and the neighborhoods such that v_i is colored by i and all the other colors occur in its neighborhood. Such a precoloring is possible in a proper way because we have $t \geq 4$ available colors, G is planar and $dist(v_i, v_j) \geq 5$.

Now we claim that every non-colored vertex has at most one colored neighbor. This is true because of the girth- and the distance constraints. Thus for every non-colored vertex there are at least $t - 1 \geq 3$ available colors. Thomassen [11] proved that every planar graph of girth at least 5 is 3-choosable. Thus it is possible to extend the precoloring to the entire graph.

3 Bipartite Graphs

The main result of this section is a characterization of bipartite graphs for which the lower bound on the b-chromatic number is attained.

Lemma 1. *Let G be a connected bipartite graph with bipartition classes A and B. If G has a b-coloring with $k \geq 3$ colors, then one bipartition class contains vertices of all k colors and the other class contains vertices of at least $k - 1$ colors.*

Proof. For $k \geq 3$, at least 2 color-dominating vertices belong to one bipartition class, say A. Thus all the k colors have to occur in B. If there is a color-dominating vertex in B then it has neighbors of $k - 1$ different colors in A, otherwise all color-dominating vertices belong to A, that is A contains vertices of all the k colors.

Now we are in a position to characterize all bipartite graphs with $b(G) = 2$ and $b(G) > 2$, respectively. Let $\overline{N}(v) = \{w \in V(G) \mid vw \notin E(G), v \neq w\}$ denote the non-neighborhood of the vertex v.

Theorem 3. *Let G be bipartite and $G_1, \ldots, G_r$ its connected components of sizes at least 3. Then $b(G) > 2$ if and only if*

1. *$r = 1$ and $A \subseteq \bigcup_{v \in B} \overline{N}(v)$ or $B \subseteq \bigcup_{v \in A} \overline{N}(v)$ where A and B are the bipartition classes of G_1, or*
2. *$r = 2$ and at least one of G_1, G_2 is not complete bipartite, or*
3. *$r \geq 3$.*

Proof. Note that components with at most two vertices do not play a role for a b-coloring by at least 3 colors because they cannot contain color-dominating vertices.

First, let $r = 1$. Assume $B \subseteq \bigcup_{v \in A} \overline{N}(v)$. We shall show that there is a b-coloring by at least 3 colors. Consider an arbitrary vertex w_1 of B. By assumption, there is a vertex $v_1 \in A$ such that v_1 is not adjacent to w_1. Color v_1 and $\overline{N}(v_1) \subseteq B$ by 1. There has to be at least one uncolored vertex w_2 in B, otherwise G_1 is disconnected. Color w_2, a vertex $v_2 \in A \setminus v_1$ which is not adjacent to w_2 and the non-neighborhood of v_2 by 2. There is at least one uncolored vertex in G_1 otherwise we have a second partition of $V(G_1)$ into two independent

subsets, which is not possible for a connected graph. Continue the coloring in the same way until all vertices of B are colored. If there are uncolored vertices in A, color all of them by an additional color. Thus, the number k of colors used is at least 3. At least $k-1$ color classes contain vertices of both A and B. If the coloring is a b-coloring we are done. Otherwise there is a color class without a color-dominating vertex. Recolor the vertices of this color class obtaining a proper coloring with $k-1$ colors. Note that again at least $k-2$ color classes have vertices in both classes A and B. Continue in this way until a b-coloring is obtained. Note that it is not possible to obtain a coloring by 2 colors because in this case G_1 is disconnected, a contradiction to the assumption. Hence the result is a b-coloring of G_1 with at least 3 colors and it can be easily extended to a b-coloring of the entire G.

Now, assume there is a b-coloring of G, and hence of G_1, which uses $k \geq 3$ colors. Then by Lemma 1 at least one of the classes A and B contains vertices of all k colors. Without loss of generality, let $v_i \in A$ be a vertex of color i for $i = 1, \ldots, k$. Obviously, we have $B \subseteq \bigcup_{i=1}^{k} \overline{N}(v_i)$.

Now, let $r = 2$. Assume G_1 has color classes A and B, and is not complete bipartite. Let $v \in A$ and $w \in B$ be non-adjacent vertices in G_1. Color v and w by color 1, all vertices of $A \setminus \{v\}$ by color 2 and all vertices of $B \setminus \{w\}$ by color 3. Because of the connectivity of G_1 there is a color-dominating vertex of color 2 in A and a color-dominating vertex of color 3 in B. Furthermore color one of the classes of G_2 by 1, and the other class by 2 and 3 in such a way that there is a color-dominating vertex of color 1 in G_2.

Now assume G_1 and G_2 are complete bipartite and there is a b-coloring by at least 3 colors. Thus at least one of them contains at least two dominating vertices, say G_1 contains dominating vertices of colors 1 and 2. If they belong to the same independent set of G_1 then colors 1 and 2 have to occur also in the other set, thus the coloring is not a proper one. If they belong to distinct vertex classes then color 3 has to occur in both classes, a contradiction. Thus each G_i $i = 1, 2$ contains at most one dominating vertex and $b(G) = 2$.

For $r \geq 3$ a b-coloring by 3 colors is obvious.

Kouider and Mahéo [7] noticed that there are graphs which have a b-coloring with k colors, no b-coloring with $k+1$ but again a b-coloring with $k+2$ colors. In fact, the gap between the integers for which a b-coloring exists may be arbitrarily large.

Proposition 2. *For every n there is a graph G having a b-coloring by k colors if and only if $k = 2$ or $k = n$.*

Proof. Using Lemma 1, it is easy to see that the graph G obtained from the complete bipartite graph $K_{n,n}$ by removing a perfect matching has the required property.

It is also true that the gap between the b-chromatic number of a graph and an induced subgraph can be arbitrarily large (and interestingly, the subgraph having the larger b-chromatic number).

Proposition 3. *For every n, there is a graph G and an induced subgraph H of G such that $b(H) - b(G) > n$.*

Proof. The graph G obtained from the complete bipartite graph $K_{n+4,n+4}$ by removing a matching of $n+3$ edges has b-chromatic number 2 because of Theorem 3. On the other hand G contains subgraphs H_k obtained from the complete bipartite graph $K_{k,k}$ by removing a perfect matching for all $k = 3, \ldots, n+3$ where $b(H_k) = k$ as mentioned in the proof of Proposition 2. Thus $b(H_{n+3}) - b(G) = n + 3 - 2 = n + 1$

By the infinite analogues of the latter construction we obtain:

Proposition 4. *For every infinite cardinal κ there exists a graph G with $b(G) = 2$, from which the removal of two vertices yields an induced subgraph H such that $b(H) = \kappa$.*

4 Algorithmic Aspects

In this section we deal with the complexity of deciding whether a b-coloring by a given number of colors exists, or whether the b-chromatic number is at least a given number. The main result of this section is the NP-completeness of the problem to decide whether there is a b-coloring by $t(G)$ colors even for connected bipartite graphs and $t(G) = \Delta(G) + 1$. Thus it is not possible to find a "nice" characterization for bipartite graphs with $b(G) = \Delta(G) + 1$.

We consider the following problems:

b-Coloring
Instance: Graph $G = (V, E)$, integer k
Question: Is there a b-coloring of G by k colors?

b-Chromatic Number
Instance: Graph $G = (V, E)$, integer k
Question: Is $b(G) \geq k$?

Note that b-Coloring $\propto$ b-Chromatic Number , but it is not obvious at first sight how knowing $b(G)$ would help to decide if G has a b-coloring using exactly k colors. Irving and Manlove [6] proved that b-Chromatic Number is NP-complete in general. In fact they proved that it is NP-complete to decide whether there is a b-coloring by $t(G)$ colors. However, their construction does not answer what happens in the special case $t(G) = \Delta(G) + 1$. We answer this question, even for bipartite graphs.

Theorem 4. b-Coloring *is NP-complete for $k = t(G)$ even for connected bipartite graphs and $t(G) = \Delta(G) + 1$.*

Proof. Obviously, b-Coloring belongs to NP. To prove its NP-completeness we present a reduction from the following problem: Given a 3-regular bipartite

graph $G = (V, E)$ with $V = A \cup B$, is there a coloring of B by 3 colors such that every vertex of A has a neighbor of each color?. NP-completeness of this problem can be easily derived from 3-edge-colorability of 3-regular graphs, which is known to be NP-complete [4].

Take a 3-regular bipartite graph with $k-3$ vertices in A. Construct a new bipartite graph H in the following way. Let $v_1, v_2, \ldots, v_{k-3}$ be vertices of A and $N(v_1) = \{w_1, w_2, w_3\} \subseteq B$ the neighborhood of v_1. Add independent vertex sets V_1, V_2, V_3 each of cardinality at least $k-4$ to A and join w_i with all vertices of V_i, $i = 1, 2, 3$. Furthermore add an independent vertex subset $W = \{w_4, \ldots, w_k\}$ to B and join v_i with all vertices of $W \setminus \{w_{i+3}\}$ for all $i = 1, \ldots, k-3$. If k is assumed to be $\Delta + 1$ then $|V_i| = k-4$ for $i = 1, 2, 3$.

We claim that H has a b-coloring by k colors if and only if G allows a feasible 3-coloring of the bipartition class B.

Assume G has such a 3-coloring and, without loss of generality, $c(w_i) = i$ for $i = 1, 2, 3$. Color V_1, V_2, V_3 such that w_i is a color-dominating vertex of color i for $i = 1, 2, 3$. Furthermore $c(v_i) = c(w_{i+3}) = i + 3$ for $i = 1, \ldots, k-3$, and $c(v) = k$ for all remaining vertices of A. Consequently v_i is a color-dominating vertex of color $i + 3$ and so we have a b-coloring of H by k-colors.

Now assume that H has a b-coloring by k colors. Then $w_1, w_2, w_3, v_4, \ldots, v_k$ are the dominating vertices because all the other vertices have degree smaller than $k-1$. Without loss of generality, let $c(w_i) = i$ for $i = 1, 2, 3$ and $c(v_i) = i+3$ for $i = 1, \ldots, k-3$. Note that $d(v_i) = k-1$ for $i = 1, \ldots, k-3$. It follows that $W = \{w_4, \ldots, w_k\}$ must be colored by $k-3$ different colors. Assume there is a vertex of W colored by 1, say $c(w_4) = 1$. Then no vertex of W is colored by 4 and there has to be at least one vertex of $(B \setminus W) \setminus \{w_1, w_2, w_3\}$ colored by 4. Recolor the graph setting $c'(w_4) = 4$, $c'(w) = 1$ for $w \in B$ with $c(w) = 4$ and $c'(v) = c(v)$ for all the other vertices. The new coloring is again a b-coloring with k color-dominating vertices because every v_i, $i = 2, \ldots, k-3$, is adjacent to w_4 and has a neighbor colored by 4 in the original coloring. Moreover, in the neighborhood of v_1 nothing changes. If there are vertices in W colored by 2 or 3, change the coloring in an analogous way. Thus the vertices of $B \setminus W$ are colored by $1, 2, 3$ such that each vertex of $v_1, \ldots, v_{k-3}$ has a neighbor of each color in W giving a feasible 3-coloring of G.

On the other hand, some problems concerning B-COLORING and B-CHROMATIC NUMBER can be decided in polynomial time. The following assertions follow immediately from Theorem 3 and its proof :

Corollary 2. *For $k = 3$,* B-CHROMATIC NUMBER *is polynomial for bipartite graphs.*

Corollary 3. B-COLORING *is polynomial for bipartite graphs G with $t(G) \leq 3$. A corresponding b-coloring can also be found in polynomial time.*

The following lemma gives a sufficient condition for the existence of a b-coloring by 3 colors.

Lemma 2. *Let G be a connected bipartite graph and $V = A \cup B$ the partition of V into independent subsets. If there is a pair of vertices v_1, v_2 belonging to the same subset A or B without a common neighbor, then G has a b-coloring by 3 colors.*

Proof. Assume $v_1, v_2 \in A$ do not have a common neighbor. Color v_1 by 1, v_2 by 2, all vertices of B not adjacent to v_1 by 1 and all remaining vertices of B by 2. The remaining vertices of A are colored by 3. Obviously this coloring is a proper coloring. Because of the connectedness of G there has to be a dominating vertex in each color.

From the above lemma and a result of Mohar, Tuza and Woeginger (cf. [9]), a further result on planar bipartite graphs can be deduced.

Lemma 3. *For k=3,* b-Coloring *is polynomial for connected planar bipartite graphs.*

Proof. We use Lemma 2 to construct a coloring. If the assumption of the lemma is fulfilled then we use the coloring given in the proof of that lemma.

If the assumption of Lemma 2 is not fulfilled then every pair of vertices belonging to the same class A or B has at least one common neighbor.

We claim that in a b-coloring of G by 3 colors there has to be a cyclically colored 6-cycle (that is the consecutive colors on the cycle are $1, 2, 3, 1, 2, 3$). At least 2 dominating vertices belong to the same class, say $v_1, v_2 \in A$ where v_i is the dominating vertex of color i $(i = 1, 2)$. Then they have a common neighbor colored by 3. Furthermore, v_1 has a neighbor w_2 colored by 2 and v_2 has a neighbor w_1 colored by 1. These two vertices w_1 and w_2 have a common neighbor, colored by 3. Thus, these 6 vertices build a cyclically colored 6-cycle.

Consequently, we have to check whether G contains a 6-cycle and for every 6-cycle we have to decide whether a cyclical coloring of it can be extended to the entire graph. For a given 6-cycle it is possible to decide in linear time whether such an extension exists [9]. If there is no 6-cycle or there is no 6-cycle with a possible coloring extension of a cyclical coloring then no b-coloring by 3 colors exists.

Maybe one can generalize this method to find a b-colorings by 3 colors not only in this case. The problem is to extend the precoloring of the 6-cycle.

5 Random Graphs

Let p be a real number, $0 < p < 1$. We denote by $G_{n,p}$ the random graph with n vertices and edge probability p. Let $q = 1 - p$.

Theorem 5. *For every fixed edge probability p $(0 < p < 1)$,*

$$(\tfrac{1}{2} - o(1))\frac{n \log \frac{1}{q}}{\log n} \leq b(G_{n,p}) \leq (1 + o(1))\frac{n \log \frac{1}{q}}{\log n}$$

with probability $1 - o(1)$ as $n \to \infty$.

Proof. The lower bound immediately follows from the basic fact that $G_{n,p}$ contains no independent sets of size larger than $(2\log n)/\log\frac{1}{q}$, almost surely. (The counting argument is practically the same as the one for $p=\frac{1}{2}$ given in the classical proof of Erdős.)

To prove the upper bound, let k be arbitrary. We estimate the probability P of the event that $G_{n,p}$ admits a b-coloring by k colors. We will prove that $P\to 0$ holds whenever k exceeds the bound given in the theorem.

Assume that $V=V_1\cup\cdots\cup V_k$ is a b-coloring by k colors. For every pair i,j $(1\le i<j\le k)$ consider dominating vertices $v_i\in V_i$ and $v_j\in V_j$. Domination implies that either the edge v_iv_j is present or v_iv_j is not an edge and there is an edge from v_i to $V_j\setminus\{v_j\}$ and from v_j to $V_i\setminus\{v_i\}$. Note that the first case applies whenever $\min(|V_i|,|V_j|)=1$. An upper bound on P is

$$P\le \sum_{\substack{|V_1|=n_1,\ldots,|V_k|=n_k\\ n_1+\ldots+n_k=n}} \underbrace{\prod_{i=1}^{k} q^{\binom{n_i}{2}}}_{A}\underbrace{\prod_{i=1}^{k} n_i}_{B}\prod_{i<j}\underbrace{(p+q(1-q^{n_i-1})(1-q^{n_j-1}))}_{C}$$

Here A is the probability that each V_i is independent, B is the number of ways to select the dominating vertices $v_i\in V_i$, and C is the probability of domination between V_i and V_j.

We estimate the sum above in the following way. There are at most $\frac{k^n}{k!}$ color partitions of the n vertices into at most k classes, that is less than

$$e^{n\log k-k\log k+k}$$

by the Stirling formula. It is also clear that

$$A=e^{-\frac{1}{2}\sum_{i=1}^{k} n_i(n_i-1)\log\frac{1}{q}}\ ,\qquad B=e^{\sum_{i=1}^{k}\log n_i}$$

Moreover, the terms in C can be written as

$$p+q-q^{n_i}-q^{n_j}+q^{n_1+n_2-1}=(1-q^{n_i})(1-q^{n_j})+\tfrac{p}{q}q^{n_1+n_2}$$

In order to obtain an upper bound on C, we take the estimate

$$\sqrt{(1-q^{n_i})(1-q^{n_j})}$$

for all pairs i,j. This is an upper bound indeed, that can be verified by analyzing the behavior of the function

$$f(x,y)=(1-x)(1-y)+\tfrac{p}{q}xy-\sqrt{(1-x)(1-y)}$$

over the domain $[0,q]\times[0,q]$. It is a matter of routine to check that fixing any one of the variables x and y, the function becomes a *convex* real function. Consequently, $f(x,y)\le 0$ on the entire domain because $f(0,0)=f(q,q)=0$ and $f(0,q)=f(q,0)=p-\sqrt{p}\le 0$.

We make some further simplification on the terms of C as follows:

$$\prod_{i<j} C \leq \prod_{i=1}^{k} (1-q^{n_i})^{\frac{k-1}{2}} < \prod_{i=1}^{k} e^{-\frac{k-1}{2} q^{n_i}} = e^{-\frac{k-1}{2} \sum q^{n_i}}$$

(Here we applied the fact that $1 - x < e^{-x}$ for $x > 0$.)

Combining the estimates above, we obtain

$$P \leq \exp\,(n \log k - k \log k + k \underbrace{-\frac{1}{2} \sum_{i=1}^{k} n_i(n_i - 1) \log \tfrac{1}{q}}_{A'} \underbrace{+ \sum_{i=1}^{k} \log n_i}_{B'} \underbrace{- \frac{k-1}{2} \sum q^{n_i}}_{C'})$$

Since the corresponding functions are convex and concave, respectively, we further obtain

(A') $\frac{\log \frac{1}{q}}{2} \sum_{i=1}^{k} n_i(n_i - 1) \geq \frac{\log \frac{1}{q}}{2} n(\frac{n}{k} - 1)$

(B') $\sum_{i=1}^{k} \log n_i \leq \sum_{i=1}^{k} \log \frac{n}{k} = k(\log n - \log k)$

(C') $\sum q^{n_i} \geq k q^{n/k}$, thus $C' \leq -\frac{k(k-1)}{2} q^{n/k}$

Substituting the above estimates yields

$$P \leq \exp\,(n \log k - k \log k + k \underbrace{-\frac{\log \frac{1}{q}}{2} n(\frac{n}{k} - 1)}_{A''} \underbrace{+ k(\log n - \log k)}_{B''} \underbrace{- \frac{k(k-1)}{2} q^{\frac{n}{k}}}_{C''})$$

In the case that $k = cn$, the significant part is the last one; namely C'' gives that P tends to 0 as n gets large. On the other hand, assuming $k = o(n)$ we have that both k and $k \log n$ are $o(n \log k)$. Therefore, it will suffice to compare $n \log k$ and $\frac{k^2}{2} q^{\frac{n}{k}}$. Writing k in the form $k = \frac{c\, n \log \frac{1}{q}}{\log n}$ we have

$$q^{\frac{n}{k}} = e^{-\frac{n}{k} \log \frac{1}{q}} = n^{-\frac{1}{c}}$$

Thus, if $c > 1$ then $k^2 q^{\frac{n}{k}}$ grows at least as fast as $n^{1+c'}$ for some constant $c' > 0$. It follows that $P = o(1)$ as $n \to \infty$, unless $k \leq (1 + o(1)) \frac{n \log \frac{1}{q}}{\log n}$. This implies the validity of the theorem.

6 Conclusion

From the complexity point of view, we have mainly presented results on the b-chromatic number of bipartite graphs, giving a full characterization (yielding a polynomial time decision algorithm) of bipartite graphs of $b(G) = 2$. We have further shown that deciding $b(G) = \Delta + 1$ is NP-complete even for bipartite graphs, thus strengthening the results of [7,10].

Note that we can decide in polynomial time if $b(G) \geq 3$ for a bipartite graph G. However, the following is open:

Question 1: Can one decide in polynomial time whether a bipartite graph allows a b-coloring using exactly 3 colors?

Question 2: Is the more general b-Coloring problem polynomial for planar bipartite graphs?

We have seen examples of graphs of arbitrarily large gaps in the set of k's for which there exists a b-coloring using exactly k colors. We find the following question quite interesting:

Question 2: For which sets of integers S do there exist graphs $G = G(S)$ such that G admits a b-coloring by k colors if and only if $k \in S$?

Finally, we have determined the b-chromatic number of the random graph, almost surely. This result shows that despite the anomality examples and hardness results, for typical graphs the b-chromatic number is not much larger than the ordinary chromatic number. In this sense it would be interesting to determine the b-chromatic number (or bounds on it) for random bipartite graphs.

References

1. H.L. Bodlaender: Achromatic number is NP-complete for cographs and interval graphs, Inf. Process. Lett. 31 (1989) 135-138
2. F. Harary, S. Hedetniemi: The achromatic number of a graph, J. Combin. Th. 8 (1970) 154-161
3. P. Hell, D.J. Miller: Graphs with given achromatic number, Discrete Math. 16 (1976) 195-207
4. I. Holyer: The NP-completeness of edge-coloring, SIAM J. Comput. 10 (1981) 718-720
5. F. Hughes, G. MacGilliway: The achromatic number of graphs: A survey and some new results, Bull. Inst. Comb. Appl. 19 (1997) 27-56
6. R. W. Irving, D. F. Manlove, The b-chromatic number of a graph, Discrete Applied Math., 91 (1999), 127–141.
7. M. Kouider, M. Mahéo, The b-chromatic number of a graph, manuscript, 2000.
8. C. McDiarmid: Achromatic numbers of random graphs, Math. Proc. Camb. Philos. Soc. 92 (1982) 21-28
9. J. Kratochvíl, Zs. Tuza, M. Voigt: New trends in the theory of graph colorings: Choosability and list coloring, In: Contemporary Trends in Discrete Mathematics (from DIMACS and DIMATIA to the future) (eds. R.L.Graham, J.Kratochvíl, J.Nešetřil, F.S.Roberts), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 49, American Mathematical Society, Providence, RI, 1999, pp. 183-197
10. D. Manlove: Minimaximal and maximinimal optimization problems: a partial order - based approach, PhD. thesis, University of Glasgow, Dept. of Computing Science, June 1998
11. C. Thomassen: 3-list-coloring planar graphs of girth 5, J. Combin. Theory Ser B 64 (1995) 101-107

Budgeted Maximum Graph Coverage

Sven Oliver Krumke[2,⋆], Madhav V. Marathe[3], Diana Poensgen[2], S.S. Ravi[4,⋆⋆], and Hans-Christoph Wirth[1]

[1] University of Würzburg
Department of Computer Science, Am Hubland, 97074 Würzburg, Germany
wirth@informatik.uni-wuerzburg.de
[2] Konrad-Zuse-Zentrum für Informationstechnik, Berlin (ZIB)
Takustraße 7, 14195 Berlin-Dahlem, Germany
{krumke,poensgen}@zib.de
[3] Los Alamos National Laboratory
P.O.Box 1663, MS 997, Los Alamos, NM 87545, USA
marathe@lanl.gov
[4] University at Albany – SUNY
Department of Computer Science, Albany, NY 12222, USA
ravi@cs.albany.edu

Abstract. An instance of the *maximum coverage* problem is given by a set of weighted ground elements and a cost weighted family of subsets of the ground element set. The goal is to select a subfamily of total cost of at most that of a given budget maximizing the weight of the covered elements.
We formulate the problem on graphs: In this situation the set of ground elements is specified by the nodes of a graph, while the family of covering sets is restricted to connected subgraphs. We show that on general graphs the problem is polynomial time solvable if restricted to sets of size at most 2, but becomes NP-hard if sets of size 3 are permitted. On trees, we prove polynomial time solvability if each node appears in a fixed number of sets. In contrast, if vertices are allowed to appear an unbounded number of times, the problem is NP-hard even on stars. We finally give a polynomial time algorithm for the special case where a star is covered by paths.

Keywords: budgeted maximum coverage, approximation algorithm, maximum weight matching, treewidth, integer linear programming

1 Introduction, Preliminaries, and Related Work

The *budgeted maximum coverage* problem is defined as follows: An instance specifies a set $X = \{x_1, \ldots, x_n\}$ of ground elements with weight function $w\colon X \to \mathbb{R}_0^+$ and a family $F \subseteq 2^X$ of *covering sets* with associated costs $c\colon F \to \mathbb{R}_0^+$. The goal is to select a subfamily $F' \subseteq F$ which does not exceed a given constraint on the total cost and maximizes the weight of the covered elements.

⋆ Research supported by the German Science Foundation (DFG, grant GR 883/10)
⋆⋆ Research supported by NSF Grant CCR-97-34936

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 321–332, 2002.

The unit cost variant of the problem (where $c \equiv 1$) is known as the *maximum coverage* problem (see e. g. [6] for a survey). A straightforward reduction from the VERTEX COVER problem shows that the unweighted maximum coverage problem is NP-hard even if each ground element appears in no more than two sets.

The problem with general cost function $c \not\equiv 1$ has been investigated by Khuller et al. [8]. The authors give an approximation algorithm with performance $(1 - 1/e) \approx 0.63$ and show that this is best possible unless NP $\subseteq$ DTIME($N^{O(\log \log N)}$).

There is an alternative definition of the budgeted maximum coverage problem used by Ageev et al. [1, 2]: "Given ground elements I, a family $F \subseteq 2^I$ with weights $w\colon F \to \mathbb{R}_0^+$, and an integer $p \in \mathbb{N}$, find a subset $X \subseteq I$ of the ground elements with $|X| = p$ which maximizes the total weight of the sets from F intersecting X." Comparing the two definitions, is appears that the role of ground elements and sets is interchanged. Notice that a set of size k in the definition of Ageev et al. transforms into a ground element appearing in k sets in the definition employed by Khuller et al. We will stick to the notation used by Khuller et al. [8] throughout the paper.

Definition 1 (Budgeted Maximum Graph Cover problem)
An instance of BUDGETED MAXIMUM GRAPH COVER *(GC for short) is given by an undirected simple graph $G = (V, E)$ with node weight function $w\colon V \to \mathbb{R}$, a weighted family $F = \{S_1, \ldots, S_{|F|}\}$ of connected subgraphs S_i with cost function $c\colon F \to \mathbb{R}_0^+$, and a budget value $B \in \mathbb{N}$. The goal is to find a subcollection $F' \subseteq F$ of subgraphs of total cost*

$$c(F') := \sum_{S \in F'} c(S) \leq B\,,$$

such that the total weight $w(F')$ covered by the subcollection, defined by

$$w(F') := \sum_{v \in \bigcup_{S \in F'} S} w(v)\,,$$

is maximized.

We assume without loss of generality that each of the subgraphs does not violate the budget constraint, i. e., $c(S) \leq B$ for all $S \in F$. By GC^{unit} we denote the set of instances where all sets have cost 1. For any natural number k, we use "k-GC" to denote the fact that every member of the family F has at most k nodes. Also, for a graph class Γ, we use "Γ-GC" to denote that each member of the family belongs to the graph class Γ. Further, the notion "GC on Γ" means that the input graph G is restricted to graph class Γ. A covering set $S \in F$ with $|S| = 1$ is called a *singleton*.

2 Maximum Graph Coverage on General Graphs

This section considers problem GC^{unit} on general graphs. We first address the case of 2-GC^{unit} (where each set has cardinality at most 2) and give a polyno-

mial time algorithm based on matching techniques. We then show hardness of k-GC$^{\text{unit}}$ for $k \geq 3$.

2.1 GC$^{\text{unit}}$ with Covering Sets of Size 2

Given an instance of problem 2-GC$^{\text{unit}}$, we claim that we can assume without loss of generality that each set has cardinality exactly 2. This can be verified easily: for each singleton $S = \{v\} \in F$ we can insert a dummy node v' with zero weight into the graph, add a dummy edge joining v and v' and replace S by $S' := \{v, v'\}$. We thus obtain an equivalent instance of 2-GC$^{\text{unit}}$ whose optimal value equals the optimal value of our original instance. Thus, for the remainder of the section we will assume that we are only given sets of size 2. Due to this observation, each set $S \in F$ corresponds to an edge in G. We will also assume in the sequel that G does not contain "useless edges", that is, edges which are not sets from F. The following lemma shows that the graph cover problem can be reduced to a matching problem with a cardinality constraint:

Lemma 2 (Budget constrained covering)
Problem 2-GC$^{\text{unit}}$ with input graph $G = (V, E)$ and budget constraint B is equivalent to the problem of finding a maximum weight matching in a graph with $2|V|$ vertices and $|E|+|V|$ edges subject to the constraint that the matching contains exactly B edges.

Proof. Consider the graph H consisting of all vertices and edges in G and, in addition, for each vertex $v \in V$ a new vertex v', called the "mate" of v, which is joined to v by an edge of weight $w(v)$ (hence the only neighbor of a mate note v' is v itself). The weight of an edge (u, v) in H which originates from the edge $(u, v) \in E$ is set to $w(u) + w(v)$.

Let $S = \{S_1, \ldots, S_B\}$ denote an optimal solution for the given instance of 2-GC on graph G, which covers a total weight of W^*. Observe that without loss of generality we can assume that the edge set S decomposes into node disjoint stars, i. e., there is no path of length 3 formed by the edges from S.

We construct a matching in H of weight at least W^* which uses exactly B edges. View the sets $\{S_1, \ldots, S_B\}$ as edges in H and denote by H_S the subgraph of H containing all vertices of H and the edges in S. Obviously, if each vertex in H_S has degree at most one, then the edges in S form a matching. In this case, since no two edges in S share a common endpoint, the weight covered by the sets in S equals that of the corresponding matching.

It remains to handle the case where there are vertices of degree greater than 1 in H_S. Let $v \in H$ be such a vertex, u be one of its neighbors in H_S. Then the degree of u equals 1, otherwise there would be a path of length 3. Hence we can replace edge (v, u) by edge (u, u') thus decreasing the degree of v by one. By repeating this replacement procedure, we end up with a matching in graph H. It is easy to observe that the total weight of the edges in this matching equals the weight of the nodes covered by S.

Conversely, let H_S be an arbitrary matching in H of total weight W, consisting of B edges. Obviously, the total weight of the nodes from G incident on edges

from H_S equals W. To construct a valid covering, replace each edge (v, v') of H_S incident on a mate node v' by an arbitrary edge incident on v. This is possible, since otherwise v would be an isolated node in G. Obviously, this replacement operation does not shrink the set of covered nodes from G. □

The following statement can be found as an exercise in [10, Problem 11.5c].

Lemma 3 (Cardinality constrained matching)
The problem of finding a maximum weight matching containing exactly k edges can be solved in polynomial time. □

From Lemma 2 and Lemma 3 we can immediately conclude:

Theorem 4 (Solving 2-GC$^{\text{unit}}$)
Problem 2-GC^{unit} can be solved in polynomial time. □

2.2 GC$^{\text{unit}}$ with Covering Sets of Larger Size

The results of the previous section for 2-GC$^{\text{unit}}$ are now complemented by a hardness result which shows that the problem is NP-hard when the bound on the size of the sets increases.

Theorem 5 (Hardness of k-GC$^{\text{unit}}$)
Problem k-GC^{unit} is NP-hard for any fixed $k \geq 3$, even when restricted to k-path-GC^{unit} on bipartite graphs.

Proof. We use a reduction from 3-Dimensional Matching which is well known to be NP-complete (see [4, Problem SP1]). The reduction shows the hardness of 3-path-GC on bipartite graphs and easily extends to any fixed $k \geq 3$.

An instance of 3-Dimensional Matching is given by a set $M \subseteq W \times X \times Y$, where W, X, and Y are disjoint sets having the same number q of elements. The question posed is, whether M contains a matching of size q, i.e., a subset $M' \subseteq M$ such that $|M'| = q$ and no two elements of M' agree in any coordinate.

We construct the natural bipartite graph G with vertex set $W \cup X \cup Y$ and edge set $W \times X \cup X \times Y$. For any triple $(w, x, y) \in M$ the collection F contains a the set $S = \{w, x, y\}$ which forms a path in G. The budget is set to $B := q$. It is easy to see that M contains a matching if and only if there is a cover in G with q sets from F which covers the whole graph. □

3 Maximum Graph Coverage on Paths and Cycles

In view of the results of Section 2 we consider restricted families of host graphs. Probably the simplest case is the restriction of the input graph to being a path. By a straightforward reduction from Knapsack one can show that the problem is still NP-hard. On the other hand, the unit cost variant GC$^{\text{unit}}$ on a path is solvable in polynomial time using a dynamic programming approach. Moreover, applying a scaling technique yields a fully polynomial approximation scheme for GC on a path [9].

4 Maximum Graph Coverage on Trees

The following result shows that GC on trees is intractable as long as the cover sets of the instance are allowed to be (very simple shaped) trees themselves.

Theorem 6 (Hardness of GC$^{\text{unit}}$ on trees)
GC^{unit} is NP-hard, even when the underlying graph is a star and all vertices have weight 1.

Proof. The claim follows from a straightforward reduction from Exact Cover by 3-Sets (X3C, see [4, Problem SP2]). Use the set X of ground elements of the X3C instance as the node set of the graph, and augment the node set by a new center node. Construct a star by connecting each node by an edge to the center. Each covering set defines in the star graph a sub-star with 3 rays in an obvious manner. The budget is set to $B := |X|/3$. □

Due to this result, we will restrict the investigation to path shaped covering sets in the following section.

4.1 Path-GC$^{\text{unit}}$ on Stars

In this section we consider problem path-GC$^{\text{unit}}$ on stars, which can be equivalently formulated as problem 3-path-GC$^{\text{unit}}$ on stars. We derive a polynomial time algorithm based on the algorithm for 2-GC$^{\text{unit}}$ given in Section 2.

Lemma 7
On stars, any instance of path-GC^{unit} can be solved by solving at most $\lfloor |F|/2+1 \rfloor$ instances of 2-GC^{unit}, where $|F|$ is the number of cover sets.

Proof. Let $(G = (V, E), F)$ be the input graph and covering family. Denote by v_0 the center node of the star shaped graph G. First assume that F contains no singletons. Obviously, each set contains the center node v_0. Moreover, we can assume that each set has cardinality 3: this can be achieved by adding a dummy node of weight zero to the star and augmenting all sets of cardinality 2 by that dummy node.

Since node v_0 is contained in each set and thus covered anyway, we can remove v_0 from each set of the family F. This yields an instance of 2-GC$^{\text{unit}}$ which is solvable in polynomial time according to Theorem 4 (see Figure 1 for an illustration). The solution for (G, F) is obtained afterwards by adding the center node and incrementing the total weight by $w(v_0)$.

We now handle the case where family F contains singletons. Then, any optimal solution either consists solely of singletons (which can be chosen by a simple greedy algorithm) or the center node v_0 is covered by at least one non-singleton. If we know in advance that some non-singleton S_0 is part of the optimal solution, we can reset the weight of the nodes from set S_0 to zero, decrease budget B by 1, augment all sets including the singletons to sets of cardinality 3 and solve the remaining instance as described above. In order to determine such a set S_0, we

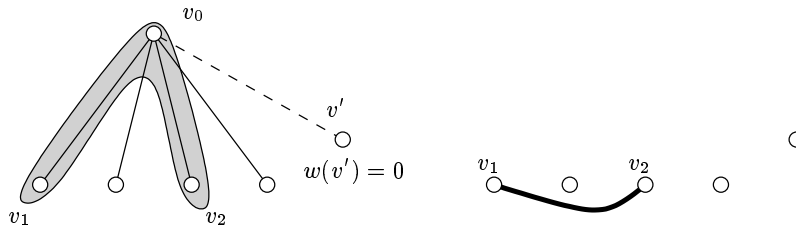


Fig. 1. Transformation of GC$^{\text{unit}}$ on a star (left) to 2-GC$^{\text{unit}}$ on a general graph (right). An arbitrary covering set $S = \{v_1, v_2, v_0\}$ is transformed to edge (v_1, v_2). Node v' is an additional dummy node of weight zero.

can perform the test for all of the at most $|F| - 1$ non-singleton sets and take the best solution.

The following observation shows how to reduce the number of tests. Assume that $S_0 \in F$ is a non-singleton set of maximal weight, i. e., $w(S_0) \geq w(S)$ for all $S \in F$. Let $S_0 = \{v_0, x_0, y_0\}$. Let $F^* \subseteq F$ be an arbitrary solution. We claim that we can transform F^* into a solution F' which covers both nodes x_0 and y_0 and satisfies $w(F') \geq w(F^*)$: Assume that node x_0 is not covered by F^*. If node y_0 is covered by some set $S_1 \in F^*$, then $w(S_1) \leq w(S_0)$, and replacing S_1 by S_0 yields solution F' without decreasing the total weight. Otherwise, the same argument holds even for an arbitrarily chosen set S_1. This proves the claim.

To this end, let κ be the number of non-singleton sets of maximal weight, let $S_1, \ldots, S_\kappa$ be the collection of that sets, where $S_i = \{v_0, x_{2i-1}, x_{2i}\}$. Denote by $n(x_i)$ the number of sets from family F which contain node x_i. The number of sets from F which share all of its points with $\{v_0, x_1, \ldots, x_{2\kappa}\}$ is bounded from above by $\min\{2\kappa^2, F\}$. An averaging argument shows that there is a foot $x^* \in \{x_1, \ldots, x_{2\kappa}\}$ which is covered by no more than

$$\frac{1}{2\kappa} \sum_{i=1}^{2\kappa} n(x_i) \leq \frac{1}{2\kappa}(|F| + \min\{2\kappa^2, |F|\}) \leq \min\left(\frac{|F|}{2\kappa} + k, \frac{|F|}{k}\right)$$

sets from F. Observe that this number attains its maximum value $|F|/2 + 1$ for $k = 1$. Since we can assume that foot x^* is contained in an optimal solution, it suffices to perform the test on at most $\lfloor |F|/2 + 1 \rfloor$ instances as described above. □

Corollary 8
Problem path-GC^{unit} on stars is solvable in polynomial time. □

4.2 GC$^{\text{unit}}$ on Trees with Elements of Bounded Frequency

Recall Theorem 6 which shows that GC$^{\text{unit}}$ on trees is hard to solve. Observe that the instance used in that reduction contains elements which appear in many

(or all) covering sets. We show that bounding the frequency of elements changes the complexity of the problem significantly.

Let $I = (G, F)$ be an instance of GC^{unit}, where $G = (V, E)$ is a tree. For arbitrary node $v \in V$, denote by

$$\phi_v := |\{ S \in F : v \in S \}|$$

the *frequency* of v, i. e., the number of covering sets containing v. By $\phi(F) := \max_{v \in V} \phi_v$ we denote the *maximum frequency* of the family F. The remainder of this section is devoted to proving the following result:

Theorem 9 (Solving GC^{unit} on trees)
For any fixed $b \in \mathbb{N}$, problem tree-GC^{unit} on trees restricted to instances with bounded frequency $\phi(F) \leq b$ and polynomially bounded weight function w (i. e., $w(x) \in O(poly|X|)$ for all ground elements $x \in X$) can be solved in polynomial time.

We recall the notion of a tree-decomposition (see e.g. [3]):

Definition 10 (Tree-Decomposition, Treewidth)
A tree-decomposition *of a graph $G = (V, E)$ is a pair $D = (\mathcal{S}, T)$ where $\mathcal{S} = \{ X_i : i \in I \}$ is a collection of subsets of V and T is a tree with node set isomorphic to $\mathcal{S}$, such that the following three conditions are satisfied:*

(i) $\bigcup_{i \in V(T)} X_i = V$,
(ii) for all edges $(u, v) \in E$, there exists a subset $X_i \in \mathcal{S}$ containing both vertices u and v,
(iii) for each vertex $v \in V$, the set of nodes $\{ i : v \in X_i \}$ forms a subtree of T.

The width *of the tree-decomposition D is defined to be $\max_{i \in I} |X_i| - 1$. The* treewidth $\text{tw}(G)$ *of a graph G is the minimum width of a tree-decomposition of G.*

In the following, we describe GC^{unit} on trees as an *integer linear program*. Then we define the *interaction graph* on that program, which describes the correspondence between variables, and we use a result from [5] to show that this interaction graph is of bounded treewidth. This allows us to apply a result from [11] providing a polynomial time algorithm for (a class of) integer linear programs whose interaction graphs are of bounded treewidth.

To this end, let Z be a set of variables and $\mathcal{C}$ be a set of constraints on Z. The *bipartite graph $BP(Z, \mathcal{C})$ associated with* $(Z, \mathcal{C})$ (also called the *constraint graph*) is the bipartite graph with color classes Z and $\mathcal{C}$, where $z \in Z$ is adjacent to $C \in \mathcal{C}$ if and only if variable z appears in constraint C. The *interaction graph* $\text{IG}(Z, \mathcal{C})$ for $(Z, \mathcal{C})$ is defined to be the graph with vertex set Z, where two vertices are adjacent if and only if they have a common neighbor in the constraint graph.

We consider the following integer linear program formulation of $\mathrm{GC}^{\mathrm{unit}}$:

$$
\begin{aligned}
\text{(IP)} \qquad \text{maximize} \quad & \sum_{v \in V} w(v) \cdot y_v & & \\
\text{subject to} \quad & y_v \leq \sum_{S \in F : v \in S} x_S & \text{for all } v \in V & \quad (1) \\
& \sum_{S \in F} x_S \leq B & & \quad (2) \\
& y_v \in \{0,1\} & \text{for all } v \in V & \\
& x_S \in \{0,1\} & \text{for all } S \in F &
\end{aligned}
$$

By (IP') we denote the program which is derived from (IP) by omitting the budget constraint (2).

The following two theorems show that in order to prove that program (IP) is solvable in polynomial time, it suffices to show that it can be re-formulated in an equivalent form which has an associated bipartite graph of bounded treewidth.

Theorem 11 ([11])
Let p be a polynomial. Let Z be a set of variables taking values from the domain $\{0, \ldots, K\}$, where $K \in \mathcal{O}(p(|Z|))$, and let $\mathcal{C}$ be a set of constraints on Z. Then, for any fixed $k \in \mathbb{N}$ and any nonnegative vector $c = (c_z)_{z \in Z} \in \{0, \ldots, p(|Z|)\}^Z$, the integer program of maximizing $\sum_{z \in Z} c_z \cdot z$ subject to the constraints $\mathcal{C}$, restricted to those instances where the interaction graph for $(Z, \mathcal{C})$ has bounded treewidth at most k, can be solved in time $K^{\mathcal{O}(k)}$. □

Theorem 12 ([5])
Let Z be a set of variables and $\mathcal{C}$ be a set of constraints on Z. Suppose that each constraint contains at most k variables. Let $BP(Z, \mathcal{C})$ be the bipartite graph associated with $(Z, \mathcal{C})$ and $IG(Z, \mathcal{C})$ be the interaction graph. Then,

$$\mathrm{tw}(IG(Z, \mathcal{C})) \in \mathcal{O}(k \cdot \mathrm{tw}(BP(Z, \mathcal{C})))$$ □

As a first step, we consider the bipartite graph of (IP').

Lemma 13
The bipartite graph of (IP') with node set introduced by variables x and y and constraints (1) has treewidth of at most $\phi(F) + 1$.

Proof. We construct a tree decomposition $(\mathcal{S}, T)$ of the bipartite graph of (IP') as follows (see Figure 2 for an illustration). The tree T in the decomposition is a copy of the tree G in the given instance $I = (G, F)$ of $\mathrm{GC}^{\mathrm{unit}}$ on trees. For each vertex $v \in T$ we define the set $X_v := \{y_v, x_{S_1}, \ldots, x_{S_{\phi_v}}, C_v\}$, where $S_1, \ldots, S_{\phi_v}$ are the subtrees in F containing v and C_v is the constraint from (1) for vertex v. The collection $\mathcal{S}$ is defined by $\mathcal{S} := \{ X_v \mid v \in V \}$.

We now argue that $(\mathcal{S}, T)$ is in fact a tree-decomposition. It is obvious that conditions (i) and (ii) of Definition 10 are satisfied. Also, condition (iii) clearly

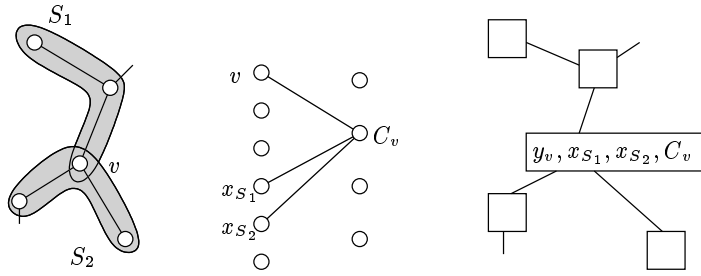


Fig. 2. Original graph G (left), constraint graph BP (center), tree decomposition of BP (right).

holds for the y_v and C_v vertices, each of which appear in exactly one set. To see that condition (iii) also holds for the x_S-vertices in the bipartite graph, observe that x_S appears exactly in those sets X_v with $v \in S$, that is, on the sets attached to those vertices v which form the subtree S.

Since the maximum cardinality of a set X_v is at most $\phi(F) + 2$, the width of decomposition $(\mathcal{S}, T)$ is bounded by $\phi(F) + 1$. □

The previous construction shows that if we restrict GC^{unit} on trees to those instances where $\phi(F)$ is fixed, the constraints (1) will result in a constraint graph with bounded treewidth. However, the budget constraint (2) in its original formulation has $|F|$ variables and a straightforward use will cause the treewidth of the constraint graph of (IP) to become unbounded. Our goal is now to replace constraint (2) by an equivalent set of new constraints and variables such that a (slightly relaxed) bound on the treewidth will be preserved.

To this end, we first transform the tree-decomposition $(\mathcal{S}, T)$ of (IP') presented in Lemma 13 into a new tree-decomposition $(\mathcal{S}', T')$ with the following properties:

1. Tree T' is a rooted binary tree.
2. For each variable x_S, there is a leaf $\{x_S\}$ in T'.
3. The width of $(\mathcal{S}', T')$ is not larger than that of $(\mathcal{S}, T)$.

The second property is achieved by connecting a new node $\{x_S\}$ to an arbitrary node which already contains x_S. The root can be chosen arbitrarily. If the tree is non-binary, then split each inner node with $d > 2$ children into a chain of $d-1$ nodes and connect each of the former sons to one of the new nodes (see e.g. [3]). Notice that this construction preserves treewidth. Moreover, the number of nodes in the tree remaines linear in the number of nodes of the underlying graph, hence the construction can be carried out in polynomial time.

Let r be the root of T'. The *depth* of a node w in T' is defined to be the number of edges on the unique path from r to w. For two vertices u and w in T' their *lowest common ancestor* $\text{lca}(u, w)$ is defined to be the node of largest depth which is both an ancestor of u and w in T'.

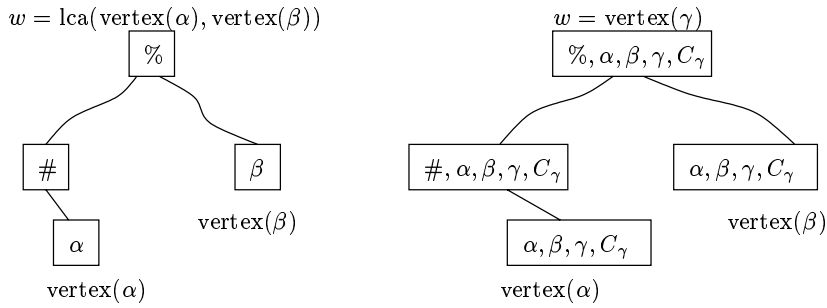


Fig. 3. Modification of tree decomposition. The width increases by at most 4.

We now define a set A of new variables and a set $\mathcal{C}$ of new constraints guided by the tree T' as follows (see Figure 3). Initially, both A and $\mathcal{C}$ are empty. We maintain a set L of pairs $(\alpha, \text{vertex}(\alpha))$ where α is a variable and $\text{vertex}(\alpha)$ is a vertex in T'. Initially, $L := \{ (x_S, \text{leaf}(x_S)) : S \in F \}$, where $\text{leaf}(x_S)$ is an arbitrary leaf in T' containing solely variable x_S.

As long as $|L| > 1$, we choose two variables α and β from L with the property that the lowest common ancestor w of $\text{vertex}(\alpha)$ and $\text{vertex}(\beta)$ has maximum depth among all possible pairs of variables represented by L. Let γ be a new variable. We add γ to A. We remove $(\alpha, \text{vertex}(\alpha))$ and $(\beta, \text{vertex}(\beta))$ from L, and add (γ, w) to L. We add the new constraint $C_\gamma \colon \alpha + \beta = \gamma$ to $\mathcal{C}$.

Let P be the unique path between $\text{vertex}(\alpha)$ and $\text{vertex}(\beta)$ in T' (which passes through their lowest common ancestor w). For all nodes $u \in P$ we add α, β, γ, and C_γ to the set X_u from the tree decomposition. Then the procedure is iterated. Notice that the size of L decrease by one in each iteration.

If $|L| = 1$, i. e., L contains only one single element, say $L = \{(\delta, \text{vertex}(\delta)\}$, we finally add the constraint $C_\delta \colon \delta \leq B$ to $\mathcal{C}$. We also add C_δ to $X_{\text{vertex}(\delta)}$.

Let $(\mathcal{S}'', T')$ denote the tree T' together with the modified sets X_u at termination of the above procedure. The following statement follows immediately from the construction described above:

Lemma 14 (Tree decomposition)
The pair $(\mathcal{S}'', T')$ is a valid tree-decomposition for the bipartite graph associated with the original variables x_S and y_v, the new variables A and the constraints (1) together with the new constraints $\mathcal{C}$. □

Lemma 15 (Width of tree decomposition)
The width of the tree-decomposition $(\mathcal{S}'', T')$ is at most $\phi(F) + 9$.

Proof. We show that each vertex in T' participates in at most two paths where labels are added. Since each step adds at most four new labels, and the treewidth of the initial graph was bounded by $\phi(F) + 1$ (Lemma 13), this shows the claim.

Let w be an arbitrary node in T'. Denote by P_i $(i = 1, 2, \ldots)$ the paths through w, by $\text{vertex}(\alpha_i)$ and $\text{vertex}(\beta_i)$ their endpoints, and by w_i their topmost

Table 1. Overview of the complexity of problem GC. Results in parentheses follow from the explicit result stated in that column.

	unit cost			general cost
	on paths	on trees	on general graphs	
1-GC	(P)	(P)	(P)	NP-hard (even on paths) [*9*]
2-GC	(P)	P [*Theorem 4*]	P [*Theorem 4*]	
3-GC	(P)	(unknown)	NP-hard [*Theorem 5*]	
path-GC	P [*9*]	(unknown)	(NP-hard)	FPAS on paths [*9*]
tree-GC	(not defined)	P (with restricted frequency) [*Theorem 9*] NP-hard (even on stars) [*Theorem 6*]	(NP-hard)	

node, i. e., $w_i := \text{lca}(\text{vertex}(\alpha_i), \text{vertex}(\beta_i))$. If w participates in three or more paths, then either there is a pair $i \neq j$ such that $w = w_i = w_j$ or such that $w \neq w_i$ and $w \neq w_j$.

Assume that $w = w_i = w_j$. Let $\text{vertex}(\alpha_i), \text{vertex}(\alpha_j)$ be the nodes in the left subtree of w. Then, the lowest common ancestor of these two nodes is a strict descendant of w, a contradiction.

Consider the case $w \neq w_i$ and $w \neq w_j$. Each of the paths P_i, P_j starts in a node in the subtree below w, say in $\text{vertex}(\alpha_i)$ and $\text{vertex}(\alpha_j)$. Hence, the lowest common ancestor of these nodes is w or a descendant of w. This contradicts the fact that both w_1 and w_2 are (strict) ancestors of w. □

Lemma 16 (Constraints)
The set $\mathcal{C}$ of new constraints is equivalent to the single constraint (2).

Proof. The claim can be shown by an easy induction on the number of iterations. Observe the invariant that at each time the sum of all variables represented by the set L equals the bound B. □

Proof (of Theorem 9). From Lemma 14 and Lemma 16 we can conclude that $(\mathcal{S}'', T')$ is a tree-decomposition of the bipartite graph of an equivalent formulation of problem (IP). Lemma 15 shows that this decomposition is of bounded treewidth. Theorem 9 now follows from Theorem 11 and Theorem 12. □

5 Conclusion and Open Problems

Table 1 gives an overview of the variants of the maximum graph coverage problem investigated in this paper. It turns out that the unit-cost problem is easy to solve on paths, while on trees we can expect polynomial time algorithms only for the case of bounded frequency. The most interesting open questions which remain are to settle the complexities of the problems path-GC^{unit} and k-path-GC^{unit} for fixed $k \in \mathbb{N}$ on trees (without any restriction on the frequency).

References

1. A. A. Ageev and M. I. Sviridenko · *Approximation algorithms for maximum coverage and max cut with given sizes of parts* · Proceedings of 7th Conference on Integer Programming and Combinatorial Optimization (IPCO'99), Lecture Notes in Computer Science, vol. 1610, 1999, pp. 17–30.
2. A. A. Ageev and M. I. Sviridenko · *Pipage rounding: a new method of constructing algorithms with proven performance guarantee* · Preliminary version appeared as [1], to appear, 2002.
3. H. L. Bodlaender · *A tourist guide through treewidth* · Tech. Report RUU-CS-92-12, Department of Computer Science, Utrecht University, Utrecht, The Netherlands, 1992.
4. 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.
5. H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns · *Parallel approximation schemes for a class of planar and near planar combinatorial problems* · Information and Computation (2002).
6. D. Hochbaum · *Approximation covering and packing problems* · in [7], 1997.
7. D. S. Hochbaum (ed.) · *Approximation algorithms for NP-hard problems* · PWS Publishing Company, Boston, 1997.
8. S. Khuller, A. Moss, and J. Naor · *The budgeted maximum coverage problem* · Information Processing Letters **70** (1999), 39–45.
9. S. O. Krumke, H. Noltemeier, H.-C. Wirth, and A. Ziegler · *Budgeted maximum coverage on paths* · in preparation, 2002.
10. C. H. Papadimitriou and K. Steiglitz · *Combinatorial optimization* · Prentice-Hall, Inc., 1982.
11. R. E. Stearns and H. B. Hunt III · *An algebraic model for combinatorial problems* · SIAM Journal on Computing **25** (1996), no. 2, 448–476.

Online Call Admission in Optical Networks with Larger Demands

Sven Oliver Krumke* and Diana Poensgen

Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustr. 7, 14195 Berlin, Germany
{krumke,poensgen}@zib.de

Abstract. In the problem of *Online Call Admission in Optical Networks*, briefly called OCA, we are given a graph $G = (V, E)$ together with a set of wavelengths W and a finite sequence $\sigma = r_1, r_2, \ldots$ of calls which arrive in an online fashion. Each call r_j specifies a pair of nodes to be connected and an integral demand indicating the number of required lightpaths. A lightpath is a path in G together with a wavelength $\lambda \in W$. Upon arrival of a call, an online algorithm must decide immediately and irrevocably whether to accept or to reject the call without any knowledge of calls which appear later in the sequence. If the call is accepted, the algorithm must provide the requested number of lightpaths to connect the specified nodes. The essential restriction is the wavelength conflict constraint: each wavelength is available only once per edge, which implies that two lightpaths sharing an edge must have different wavelengths. Each accepted call contributes a benefit equal to its demand to the overall profit. The objective in OCA is to maximize the overall profit.
Competitive algorithms for OCA have been known for the special case where every call requests just a single lightpath. In this paper we present the first competitive online algorithms for the more general case in which the demand of a call may be as large as $|W|$.

1 Introduction

In current telecommunication networks, the *wavelength division multiplexing* technique (WDM) enables the provider to send several optical signals in parallel over the same glass fiber cable by assigning different wavelengths to them. However, the optical signals are converted back into electronic form at intermediate nodes in order to switch them. This so-called "o-e-o-conversion" limits the speed of the connections. In next generation's fully optical networks, optical signals are no longer converted back into electronic form at intermediate nodes but switched optically. This requires a change in the underlying mathematical model, because the wavelength on which a signal enters the network remains unchanged until the signal reaches its destination.

A connection in a fully optical network is modeled as a *lightpath*, that is, a path together with a wavelength. Since each wavelength is available only once

* Research supported by the German Science Foundation (DFG, grant GR 883/10)

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 333–344, 2002.

per fiber, simultaneously routed lightpaths which use the same fiber must have different wavelengths. This crucial restriction is called the *wavelength conflict constraint*.

1.1 Problem Definition

An instance of the *Online Call Admission Problem in Optical Networks* (OCA) consists of an undirected graph $G = (V, E)$ together with a set of χ eligible wavelengths $W = \{\lambda_1, \ldots, \lambda_\chi\}$ and a finite request sequence $\sigma = r_1, r_2, \ldots, r_m$ of calls. Each of the wavelengths in W is available once per edge. A *lightpath* is a pair (P, λ), where P is a path in G and λ is one of the wavelengths in W. In the sequel, we will use the terms wavelength and color interchangeably.

A *call* $r_j = (s_j, t_j, b_j)$ specifies the nodes $s_j \in V$ and $t_j \in V$ to be connected as well as the required number $b_j \in \mathbb{N}$ of lightpaths, that is, its *demand*. Upon arrival of a new request $r_j = (s_j, t_j, b_j)$, an algorithm for OCA must decide whether to route or to reject r_j. If the call is accepted, the algorithm must provide the requested number b_j of lightpaths, thereby obeying the wavelength conflict constraint. Once accepted, a call can not be preempted: the lightpaths used for the call can not be changed or removed anymore. Each accepted call r_j contributes a profit equal to its demand b_j to the total profit obtained by an algorithm. The overall goal of OCA is to maximize the overall profit, that is, the total accepted demand.

An *online algorithm* for OCA must base its decision for call r_j without knowledge of calls r_i with $i > j$. A standard tool to measure the quality of an online algorithm ALG is *competitive analysis*, where one compares for each input sequence σ the profit $\text{ALG}(\sigma)$ obtained by ALG to the optimal profit achievable on that sequence, denoted by $\text{OPT}(\sigma)$.

Definition 1.1 (Competitive Deterministic Algorithm). *A deterministic online algorithm* ALG *for* OCA *is* c-competitive *if for any request sequence* σ *the inequality* $\text{ALG}(\sigma) \geq \frac{1}{c} \cdot \text{OPT}(\sigma)$ *holds.*

A randomized online algorithm is a probability distribution over a set of deterministic online algorithms. Thus, the objective value produced by a randomized algorithm is a random variable. In this paper we analyze the performance of randomized online algorithms against an *oblivious adversary*. An oblivious adversary knows the online algorithm's probability distribution, but can not see the outcomes of the random choices made by the online algorithm and therefore has to generate a request sequence in advance. We refer to [4] for details on the various adversary models.

Definition 1.2 (Competitive Randomized Algorithm). *A randomized online algorithm* RALG *for* OCA *is defined to be* c-competitive against an oblivious adversary *if for any request sequence* σ *the inequality* $\mathbb{E}\left[\text{RALG}(\sigma)\right] \geq \frac{1}{c} \cdot \text{OPT}(\sigma)$ *holds.*

The *competitive ratio* of an algorithm is defined to be the infimum over all c such that the algorithm is c-competitive.

1.2 Previous Work

If the set of eligible wavelengths W contains only a single wavelength, the problem of providing lightpaths reduces to the problem of finding edge disjoint paths in the given graph, which we will refer to as *Online Edge Disjoint Path Allocation* (OEDPA). Competitive algorithms for OEDPA are known for special graphs like lines, trees, and meshes. The currently best competitive ratios of randomized algorithms against an oblivious adversary for these topologies are $\lceil \log n \rceil$ for the line with n nodes [2, 1], $2 \log n$ and $\mathcal{O}(\log D)$ for a tree with n nodes and with diameter D, respectively, [2, 1, 6] and $\mathcal{O}(\log n)$ for the $n \times n$-mesh [5, 6].

So far, OCA with $\chi = |W| > 1$ wavelengths has been investigated only for the special case in which each call has demand one, i.e., $b_j = 1$ for all j. Awerbuch et al. ([1]) developed the competitive algorithm FFC (*First-Fit-Coloring*), which is based on a "virtual" online algorithm for OEDPA.

Theorem 1.3 (Awerbuch et al. [1]). *Let* SLAVE *be a c-competitive algorithm for* OEDPA. *Then there is a* $(c + 1)$*-competitive algorithm* FFC *for the special case of* OCA *where each call requires one lightpath.*

Note that the competitive ratio of FFC does not depend on the number of eligible wavelengths in the network and differs from that of the subroutine used for OEDPA only by an additive constant of 1.

1.3 Our Contribution

We present the first competitive algorithms for the general case of OCA in which the demand of a call may be greater than 1. We assume, however, that no call asks for more than χ lightpaths. This assumption is reasonable since accepting a call of demand higher than χ (if at all possible) would plug up the network immediately. In particular, on trees this assumption means no restriction. Notice that a call of demand b is different from b calls each of demand 1, since the online algorithm must either accept the whole call and provide the requested b lightpaths or reject the call; it is not allowed to route a call partially.

The first of our algorithms, *Copy-Coloring* (CC), is deterministic and works for OCA in general graphs. As the FFC algorithm CC uses an algorithm for Online Edge Disjoint Path Allocation as a subroutine. However, as demands may now be as high as χ, CC's competitive ratio contains χ as a linear factor. Notice that the known lower bound of $n - 1$ for deterministic OEDPA algorithms can be generalized to a $\chi(n - 1)$ lower bound for deterministic algorithms for OCA. This shows that any deterministic algorithm has to put up with χ as a linear factor is in its competitive ratio. In particular, on trees, CC is optimal in the class of deterministic algorithms. An improved ratio in which χ is incurred only logarithmically, is achieved by our randomized algorithm *First-fit-coloring-scaled* (FFCS) for the case that the underlying topology is a tree. The analysis of FFCS relies on the uniqueness of the path which connects two nodes of a tree. Note that all results carry over to the case of a directed network, with a minor change of the constant in the ratio of FFCS.

Table 1. Results in Online Call Admission in Optical Networks. The parameter Γ_G is defined as $\max\{\Gamma_{s,t} \mid s,t \in V\}$, where $\Gamma_{s,t}$ denotes the maximum number of edge disjoint paths connecting s and t.

Topology	competitive ratio using generic c-competitive algorithm for OEDPA	competitive ratio using best known competitive algorithm for OEDPA	known lower bounds for OEDPA (OCA with $\chi = 1$)
arbitrary network with n nodes, χ wavelengths	$c \cdot \chi \cdot \Gamma_G$ (Theorem 2.1)	$\mathcal{O}(\chi \log n)$ on $n \times n$ meshes	deterministic: $n-1$ randomized: $n^{1-\log_4 3}$ [3]
tree with n nodes, χ wavelengths	$12(c+1)(\lceil \log \chi \rceil + 1)$ (Theorem 3.2)	$(24 \log n + 2) \cdot (\lceil \log \chi \rceil + 1)$	deterministic: $n-1$ randomized: $\lfloor \log \frac{n}{2} \rfloor$ [1]
line with n nodes, χ wavelengths	$8(c+1)(\lceil \log \chi \rceil + 1)$ (Corollary 3.7)	$(8\lceil \log n \rceil + 8) \cdot (\lceil \log \chi \rceil + 1)$	deterministic: $n-1$ randomized: $\lfloor \log \frac{n}{2} \rfloor$ [1]

Table 1 gives an overview of our results together with the known lower bounds for OEDPA from the literature.

2 A Deterministic Algorithm for General Graphs

Let the graph $G = (V, E)$ together with the set of eligible wavelengths $W = \{\lambda_1, \ldots, \lambda_\chi\}$ be given in an arbitrary instance of OCA. Remember that the OEDPA problem can be considered to be a special case of OCA in which there is only one eligible wavelength (and calls have demand 1). The deterministic online algorithm *Copy Coloring* (CC) uses an algorithm for OEDPA as a subroutine. This algorithm, called SLAVE in the sequel, works on the instance of OEDPA given by the graph G and a single wavelength.

Algorithm Copy Coloring
Let SLAVE be an online algorithm for OEDPA. Upon arrival of a call $r_j = (s_j, t_j, b_j)$, hand the "sized down" call $\tilde{r}_j = (s_j, t_j, 1)$ to SLAVE.
If SLAVE rejects $\tilde{r}_j$, reject r_j. If SLAVE accepts $\tilde{r}_j$ and routes it on path P, then accept r_j and route its demand using the lightpaths $(P, \lambda_1), \ldots, (P, \lambda_{b_j})$.

It is easy to see that CC yields a valid solution. Recall that each call has demand at most χ. If we view the graph G together with its set of χ eligible colors as χ copies $G_1, \ldots, G_\chi$ of G, each in a different color, then SLAVE creates a feasible routing for the accepted calls from the modified sequence in G_1. By construction, CC routes an accepted call with demand b_j along the same path P which SLAVE chooses, using its first b_j wavelengths in order to obtain the required number of lightpaths. Therefore, in each G_i only a subset of the paths routed in G_1 is established. Consequently, we have a feasible routing in each color, i.e., the wavelength conflict constraint is satisfied.

The weakness of CC is obvious: even if a call could easily be routed by using one of the wavelengths of higher index, it might be rejected because it can not be routed in G_1 anymore. In the worst-case, the sequence contains only calls with demand 1, and the algorithm never uses any of the wavelengths $\lambda_2, \ldots, \lambda_\chi$. However, if we modify the algorithm such that it may use other wavelengths if λ_1 is not available anymore, the analysis becomes intractable. Let $\Gamma_G := \max\{\Gamma_{s,t} \mid s,t \in V\}$, where $\Gamma_{s,t}$ denotes the maximum number of edge disjoint paths connecting s and t. nodes in the given network.

Theorem 2.1. *Let G be a graph with χ eligible wavelengths, and let $\Gamma_G := \max\{\Gamma_{s,t} \mid s,t \in V\}$, where $\Gamma_{s,t}$ denotes the maximum number of edge disjoint paths connecting s and t. If* SLAVE *is a c-competitive algorithm for* OEDPA*, then* CC *is $(c \cdot \chi \cdot \Gamma_G)$-competitive on G.*

Proof. By ALG_k we denote the algorithm ALG which has only the first k colors of W at its disposal (and can therefore only handle sequences of calls with demand at most k). In particular, OPT_1 is the optimal offline algorithm for OEDPA on the given graph G and OPT_χ is the optimal offline algorithm for OCA with χ eligible wavelengths. Let $\sigma^{(q)}$ be the sequence obtained from σ by changing the demand of each call to q. Note that the maximum number of edge disjoint paths connecting any two nodes in G is bounded by Γ_G.

Given a sequence σ of calls, consider the maximum number of calls in $\sigma^{(1)}$ that can be routed simultaneously in one color (i.e., by edge disjoint paths). By definition this number equals $\text{OPT}_1(\sigma^{(1)})$. Since SLAVE is c-competitive for OEDPA, we have $\text{SLAVE}(\sigma^{(1)}) \geq \frac{1}{c} \cdot \text{OPT}_1(\sigma^{(1)})$.

Let r_j be a call which is routed by the optimal offline algorithm OPT_χ on lightpaths $(P_1, \lambda_{i_1}), \ldots, (P_{b_j}, \lambda_{i_{b_j}})$. Each of the lightpaths will be referred to as a *fragment* of r_j. Let $\text{OPT}_\chi(i, \sigma)$ denote the share of profit that $\text{OPT} = \text{OPT}_\chi$ gains by call fragments routed in color i. Clearly, this number is bounded from above by $\Gamma_G \cdot \text{OPT}_1(\sigma^{(1)})$. As a consequence, we have that

$$\text{OPT}_\chi(\sigma) = \sum_{i=1}^{\chi} \text{OPT}(i, \sigma) \leq \chi \cdot \Gamma_G \cdot \text{OPT}_1(\sigma^{(1)}) \leq \chi \cdot \Gamma_G \cdot c \cdot \text{SLAVE}(\sigma^{(1)}),$$

where the last inequality follows from the competitiveness of SLAVE for OEDPA. Clearly, CC makes as least as much profit on σ as the SLAVE algorithm it uses makes on $\sigma^{(1)}$. Therefore,

$$\text{OPT}_\chi(\sigma) \leq \chi \cdot \Gamma_G \cdot c \cdot \text{CC}(\sigma),$$

which shows the claim of the theorem.

At first glance, the competitive ratio of CC does not seem to be very good. However, the following theorem shows that without restrictions the bound achieved by CC is essentially the best which we can expect for deterministic algorithms.

Theorem 2.2. *On a line with n nodes, no deterministic algorithm for* OCA *can be c-competitive with $c < \chi(n-1)$.*

Proof. The worst case sequence is a straightforward generalization of the known lower bound construction from [3] for OEDPA on the line with n nodes. Let the nodes be numbered by $v_1, v_2, \ldots, v_n$ from left to right. The adversary first issues a request $r_1 = (v_1, v_n, 1)$. It is straightforward to see that any deterministic algorithm which achieves a finite competitive ratio must accept r_1. The adversary then presents the $n-1$ requests $(v_1, v_2, \chi), (v_2, v_3, \chi), \ldots, (v_{n-1}, v_n, \chi)$, none of which the deterministic online algorithm can accept.

3 An Improved Randomized Algorithm for Trees

We now present the randomized algorithm *First-Fit-Coloring-Scaled (*FFCS*)* and analyze its performance on trees and the line. For these graph classes FFCS achieves an exponential improvement in the competitive ratio compared to the deterministic algorithm CC from the previous section.

We derive FFCS as a probability distribution over a set of $\lceil \log \chi \rceil + 1$ deterministic algorithms which we denote by FFCS^i, $i = 0, \ldots, \lceil \log \chi \rceil$. Recall that $\sigma^{(q)}$ is the sequence obtained from sequence σ by changing the demand of each call to q, and that ALG_k is the algorithm ALG working on a graph having k wavelengths at its disposal. If the subscript is omitted, we always refer to the original problem in which we are given the graph G together with χ eligible wavelengths.

Algorithm First-Fit-Coloring-Scaled
Partition the set of possible calls into $\lceil \log \chi \rceil + 1$ classes as follows: Class K_0 contains all calls with demand 1. For $i = 1, \ldots, \lceil \log \chi \rceil$, class K_i contains those calls whose demand is in $(2^{i-1}, 2^i]$.
Choose $i \in \{0, \ldots, \lceil \log \chi \rceil\}$ uniformly at random and from this point on, use the deterministic algorithm FFCS^i.

FFCS^i If call $r_j = (s_j, t_j, b_j)$ does not belong to class K_i, reject r_j. Otherwise size the demand of r_j down to 1 and hand the modified call $\tilde{r}_j = (s_j, t_j, 1)$ over to $\text{FFC}_{\lfloor \chi/2^i \rfloor}$, that is the version of FFC which works on G but has only $\lfloor \chi/2^i \rfloor$ wavelengths $\{w_1, \ldots, w_{\lfloor \chi/2^i \rfloor}\}$ at its disposal.
If $\text{FFC}_{\lfloor \chi/2^i \rfloor}$ rejects modified call $\tilde{r}_j$, then reject r_j. If $\text{FFC}_{\lfloor \chi/2^i \rfloor}$ accepts the modified call $\tilde{r}_j$ and routes in on path P in wavelength w_k, accept the original call r_j and route it on the lightpaths $(P, \lambda_{(k-1)2^i+1}), (P, \lambda_{(k-1)2^i+2}), \ldots, (P, \lambda_{(k-1)2^i+b_j})$.

Proposition 3.1. FFCS *produces a valid routing for the calls.*

Proof. Let i be the value of the random choice by FFCS. If a call $r_j = (s_j, t_j, b_j)$ is accepted by FFCS, it must belong the class class K_i, implying that $b_j \leq 2^i$, and the sized-down call $\tilde{r}_j = (s_j, t_j, 1)$ must be accepted by $\text{FFC}_{\lfloor \chi/2^i \rfloor}$.

The algorithm $\text{FFC}_{\lfloor \chi/2^i \rfloor}$ produces a valid routing in the graph G for the set of accepted calls from $\sigma^{(1)}$. Since each wavelength w_k of $\text{FFC}_{\lfloor \chi/2^i \rfloor}$ corresponds to a set $\lambda_{(k-1)\cdot 2^i+1}, \ldots, \lambda_{k\cdot 2^i}$ of 2^i wavelengths in the original graph, it follows that all accepted calls (which, as mentioned, have all demand at most 2^i) can in fact be routed as specified without violating the wavelength conflict constraint.

Theorem 3.2. *Suppose that the following two conditions are satisfied:*

(i) Each algorithm $\text{FFC}_{\lfloor \chi/2^i \rfloor}$ *uses a c-competitive algorithm for the* OEDPA *problem as a subroutine.*

(ii) For any input sequence π *with the property that* $\frac{b}{2} < b_j \leq b$ *for all requests* $r_j = (s_j, t_j, b_j) \in \pi$ *the estimate*

$$\text{OPT}_\chi(\pi) \leq c' \cdot b \cdot \text{OPT}_{\lfloor \chi/b \rfloor}(\pi^{(1)})$$

holds. Then, the randomized algorithm FFCS *(with* χ *wavelengths at its disposal) achieves a competitive ratio of*

$$2c'(c+1)(\lceil \log(\chi) \rceil + 1)).$$

Proof. Let σ be an arbitrary call sequence. We have to show that the expected profit of FFCS satisfies

$$\mathbb{E}\left[\text{FFCS}(\sigma)\right] \geq \frac{1}{2c'(c+1)(\lceil \log(\chi) \rceil + 1)} \cdot \text{OPT}(\sigma). \tag{1}$$

Since FFCS_χ chooses $i \in \{0, \ldots, \lceil \log \chi \rceil\}$ uniformly at random and then uses the deterministic algorithm FFCS^i we can rewrite the left hand side of (1) as

$$\mathbb{E}\left[\text{FFCS}(\sigma)\right] = \frac{1}{\lceil \log(\chi) \rceil + 1} \cdot \sum_{i=0}^{\lceil \log \chi \rceil} \text{FFCS}^i(\sigma). \tag{2}$$

Recall that the deterministic algorithm FFCS^i rejects all calls which do not belong to class K_i. Thus, it only gains profit on calls from K_i, and we have that $\text{FFCS}^i(\sigma) = \text{FFCS}^i(\sigma|_{K_i})$, where $\sigma|_{K_i}$ is the subsequence of σ which consists of calls belonging to K_i. Using this equality gives us

$$\mathbb{E}\left[\text{FFCS}(\sigma)\right] = \frac{1}{\lceil \log(\chi) \rceil + 1} \cdot \sum_{i=0}^{\lceil \log \chi \rceil} \text{FFCS}^i(\sigma|_{K_i}). \tag{3}$$

Let $P_i^*(\sigma)$ denote that share of the total optimal profit which is gained with calls in K_i, that is,

$$P_i^*(\sigma) = \sum_{\substack{r \in \sigma|_{K_i} \\ r \text{ is accepted by OPT} \\ \text{when given input } \sigma}} b_j.$$

Therefore, $\text{OPT}_\chi(\sigma) = \sum_{i=0}^{\lceil \log \chi \rceil} P_i^*(\sigma)$. We now compare $\text{FFCS}^i(\sigma|_{K_i})$ to $P_i^*(\sigma)$.

How big can the share of OPT's profit gained from K_i be? $P_i^*(\sigma)$ gets largest if OPT uses all its resources for calls from K_i, which would be the optimal profit gained if only the sequence $\sigma|_{K_i}$ was given. Therefore, $P_i^*(\sigma) \leq \text{OPT}(\sigma|_{K_i})$, and this yields

$$\text{OPT}(\sigma) = \sum_{i=0}^{\lceil \log \chi \rceil} P_i^*(\sigma) \leq \sum_{i=0}^{\lceil \log \chi \rceil} \text{OPT}(\sigma|_{K_i}). \tag{4}$$

Hence, it suffices to upper bound the profit $\text{OPT}(\sigma|_{K_i})$ in terms of $\text{FFCS}^i(\sigma|_{K_i})$. To this end we estimate the profit gained by the deterministic algorithm FFCS^i on the sequence $\sigma|_{K_i}$. By construction, FFCS^i accepts those calls whose modified version is accepted by $\text{FFC}_{\lfloor \chi/2^i \rfloor}$. As $\text{FFC}_{\lfloor \chi/2^i \rfloor}$ gets profit 1 for each accepted call (it was given calls whose demand was sized down to 1), the number of accepted calls equals $\text{FFC}_{\lfloor \chi/2^i \rfloor}((\sigma|_{K_i})^{(1)})$. Since FFCS^i gets profit $b_j \geq 2^{i-1}$ for each accepted call, we obtain that

$$\text{FFCS}^i(\sigma|_{K_i}) \geq 2^{i-1} \cdot \text{FFC}_{\lfloor \frac{\chi}{2^i} \rfloor}((\sigma|_{K_i})^{(1)}).$$

We now apply Theorem 1.3 about the competitiveness of FFC to $\text{FFC}_{\lfloor \frac{\chi}{2^i} \rfloor}$. This results in

$$\text{FFCS}^i(\sigma|_{K_i}) \geq 2^{i-1} \cdot \frac{1}{c+1} \cdot \text{OPT}_{\lfloor \frac{\chi}{2^i} \rfloor}((\sigma|_{K_i})^{(1)}). \tag{5}$$

Observe that all the demands in $\sigma|_{K_i}$ are within a factor of two. Hence, we can use assumption (ii) with $\pi = \sigma|_{K_i}$ and $b = 2^i$ to obtain:

$$\text{OPT}_{\lfloor \frac{\chi}{2^i} \rfloor}((\sigma|_{K_i})^{(1)}) \geq \frac{1}{c' 2^i} \cdot \text{OPT}_\chi(\sigma|_{K_i}). \tag{6}$$

Plugging (6) into (5) and using this result in (3) gives

$$\mathbb{E}\left[\text{FFCS}(\sigma)\right] \geq \frac{1}{\lceil \log(\chi) \rceil + 1} \sum_{i=0}^{\lceil \log \chi \rceil} \frac{1}{2c'(c+1)} \cdot \text{OPT}_\chi(\sigma|_{K_i}). \tag{7}$$

The claim of the theorem now follows from (4).

Theorem 3.2 bounds the competitive ratio of FFCS in terms of (i) the competitive ratio c of a virtual online algorithm for OEDPA, and (ii) the ratio c' between the optimal offline profit $\text{OPT}_\chi(\pi)$ and b times the optimal offline profit on a scaled sequence $\pi^{(1)}$ with fewer wavelengths $\lfloor \chi/b \rfloor$.

In the sequel we address the existence of the second ratio c' for the case of trees and, as a special case, for the line.

3.1 Call Coloring on Trees

Note that on trees, the problem of finding a feasible routing for a given set of calls reduces to the problem of path coloring, since each call uniquely determines the path to be used. Therefore, we will also speak of *call coloring* or *path coloring*.

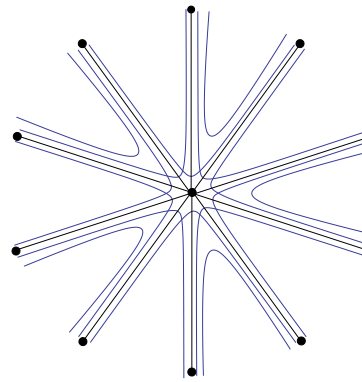


Fig. 1. An example in which χ/b colors do not suffice to color all given paths ($\chi = 6, b = 2$).

Lemma 3.3. *Let G be a tree. Let π be a call sequence such that the demand b_j of each call in π satisfies $\frac{b}{2} < b_j \leq b$. Then*

$$\text{OPT}_\chi(\pi) \leq 6 \cdot b \cdot \text{OPT}_{\lfloor \frac{\chi}{b} \rfloor}(\pi^{(1)}).$$

Proof. We prove the claim in three steps. We first consider the special case in which (a) $b_j = b$ for all j, and (b) b divides χ. For this special case we show that

$$\text{OPT}_\chi(\pi) \leq \frac{3}{2} \cdot b \cdot \text{OPT}_{\lfloor \frac{\chi}{b} \rfloor}(\pi^{(1)}). \tag{8}$$

We then show that we can drop the two assumptions one by one, losing factor 2 in each of the two steps.

Assume that conditions (a) and (b) hold. Let S be the set of calls from sequence π by which the optimal profit on the left hand side in (8) is achieved. Then $\text{OPT}_\chi(\pi) = |S| \cdot b$, since by assumption (a), the demand of each call equals b. If we were able to show that the calls in $S^{(1)}$, i.e., the paths corresponding to the calls in $S^{(1)}$, can be colored with $\frac{\chi}{b}$ colors, this would imply that the set $S^{(1)} \subset \pi^{(1)}$ could be accepted and routed feasibly by any algorithm which has $\frac{\chi}{b}$ colors at its disposal, yielding profit $|S| = |S^{(1)}|$. Since the optimal offline algorithm can do only better, we would obtain $\text{OPT}_{\frac{\chi}{b}}(\pi^{(1)}) \geq |S| = \frac{1}{b} \cdot \text{OPT}_\chi(\pi)$.

Unfortunately, it is in general *not* possible to color all the calls in $S^{(1)}$ by $\frac{\chi}{b}$ colors. This is illustrated by Figure 1. It shows a star T and a set of 13 calls (i.e.paths) on T. It is possible to assign $b = 2$ different colors to each of the given paths such that two intersecting paths have disjoint color sets and only $\chi = 6$ colors are used overall. Alas, it is impossible to assign 1 color to each of the paths using only $3 = \chi/b$ colors such that intersecting paths have different colors.

However, we will show that at least two third of the calls in $S^{(1)}$ can be colored using χ/b colors. To this end, we define the *maximum (unweighted) load* of a set of paths in a graph to be the maximum number of paths which have an edge in common.

Theorem 3.4 ([7]). *There is a feasible routing of requests of maximum load L per link of undirected trees using no more than $\frac{3}{2}L$ wavelengths.* □

We seek to apply Theorem 3.4. We know that there is a feasible routing for the calls in S, that is, each call in S is assigned b lightpaths (the uniquely determined path together with b different colors), using at most χ different colors in total. On each edge, the maximum load of the whole set of lightpaths established to route S is hence bounded from above by χ. Therefore, the maximum load of paths (calls) in $S^{(1)}$ is bounded from above by $\frac{\chi}{b}$, as each call in $S^{(1)}$ corresponds to one path whereas a call in S corresponds to b lightpaths.

Applying Theorem 3.4 yields that there is a coloring of the calls in $S^{(1)}$ using $\frac{3}{2} \cdot \frac{\chi}{b}$ colors. If we consider those $\frac{\chi}{b}$ among the $\frac{3}{2} \cdot \frac{\chi}{b}$ colors from the coloring which accommodate the most calls, a simple averaging argument gives us that they accommodate a set $M^{(1)} \subset S^{(1)}$ of calls of cardinality at least $\frac{2}{3} \cdot |S^{(1)}|$. The optimal algorithm on χ/b colors might even accept a larger set of calls from π^1, so since $|S| = |S^{(1)}|$ and b divides χ, we can conclude that

$$\textsc{opt}_{\lfloor \frac{\chi}{b} \rfloor}(\pi^{(1)}) = \textsc{opt}_{\frac{\chi}{b}}(\pi^{(1)}) \geq |M^{(1)}| \geq \frac{2}{3} \cdot |S| = \frac{2}{3b} \cdot \textsc{opt}_{\chi}(\pi),$$

which is exactly what we claimed in (8).

Now assume that only condition (b) holds, that is, b divides χ, but the demand b_j of a call may take any integral value between $\frac{b}{2}$ and b, i.e., $\frac{b}{2} < b_j \leq b$. Consider the routing defined by $\textsc{opt}_{\chi}(\pi)$. If we provide χ additional colors, we could duplicate this routing, i.e., provide $2b_j$ lightpaths for each accepted call r_j. In particular, since $b - b_j < b_j$, we can accommodate $b - b_j$ additional lightpaths for each of the accepted calls if we provide χ additional colors. We thus get a valid solution for the sequence $\pi^{(b)}$ in the graph with 2χ wavelengths. This lets us conclude that $\textsc{opt}_{\chi}(\pi) \leq \textsc{opt}_{2\chi}(\pi^{(b)})$. As b divides χ, it also divides 2χ, and since the demand of each call in π^b equals b, we can apply inequality (8). We obtain

$$\begin{aligned} \textsc{opt}_{\chi}(\pi) \leq \textsc{opt}_{2\chi}(\pi^b) &\overset{(8)}{\leq} \frac{3}{2} \cdot b \cdot \textsc{opt}_{\frac{2\chi}{b}}(\pi^{(1)}) \leq \frac{3}{2} \cdot b \cdot 2 \cdot \textsc{opt}_{\frac{\chi}{b}}(\pi^{(1)}) \\ &= 3 \cdot b \cdot \textsc{opt}_{\frac{\chi}{b}}(\pi^{(1)}), \end{aligned}$$

where the last inequality holds as $\textsc{opt}_{\chi/b}$ can accept at least those calls from $\pi^{(1)}$ which were accepted by $\textsc{opt}_{2\chi/b}$ and routed in the "fuller" half of the $\frac{2\chi}{b}$ colors used.

We finally show that we can drop the condition (b) 'b divides χ' using the part we just proved for the second inequality in the following chain:

$$\textsc{opt}_{\chi}(\pi) \leq \textsc{opt}_{2 \cdot \lfloor \frac{\chi}{b} \rfloor \cdot b}(\pi) \leq 3 \cdot b \cdot \textsc{opt}_{2 \cdot \lfloor \frac{\chi}{b} \rfloor}(\pi^{(1)}) \leq 2 \cdot 3 \cdot b \cdot \textsc{opt}_{\lfloor \frac{\chi}{b} \rfloor}(\pi^{(1)}).$$

The first inequality simply holds because $\chi \leq 2 \cdot \lfloor \frac{\chi}{b} \rfloor \cdot b$, the last by the same reasoning as before: the "fuller" half of the $2 \cdot \lfloor \frac{\chi}{b} \rfloor$ colors accommodates at least half of the calls.

This completes the proof of the lemma.

Corollary 3.5. *If there is a c-competitive algorithm for* OEDPA *on trees, then* FFCS *achieves a competitive ratio of* $12(c+1)(\lceil \log \chi \rceil + 1)$ *for routing in* χ *wavelengths.* □

Using one of the $2 \log n$-competitive algorithms for OEDPA on trees with n vertices from [2, 1, 6], this results in a $(24 \log n + 2)(\lceil \log \chi \rceil + 1)$-competitive algorithm for OCA on the same graph class.

3.2 Call Coloring on Paths

In case that the underlying graph is a simple path, a slightly better result than the one in Lemma 3.3 can be achieved.

Lemma 3.6. *Let G be a path. Let π be a call sequence such that the demand b_j of each call in π satisfies $\frac{b}{2} < b_j \leq b$. Then*

$$\text{OPT}_\chi(\pi) \leq 4 \cdot b \cdot \text{OPT}_{\lfloor \frac{\chi}{b} \rfloor}(\pi^{(1)}).$$

Proof. As before, we first consider the case that all demands satisfy $b_j = b$ for all j and that b divides χ. For this case, we show that

$$\text{OPT}_\chi(\pi) = b \cdot \text{OPT}_{\lfloor \frac{\chi}{b} \rfloor}(\pi^{(1)}). \tag{9}$$

As shown in the proof of Lemma 3.3, dropping the two assumptions above costs us a factor of 2 each. This yields the desired result. It remains to prove (9). It is straightforward to see that $\text{OPT}_\chi(\pi) \geq b \cdot \text{OPT}_{\lfloor \frac{\chi}{b} \rfloor}(\pi^{(1)})$. We will now show that the inequality also holds the other way around, namely that $\text{OPT}_\chi(\pi) \leq b \cdot \text{OPT}_{\lfloor \frac{\chi}{b} \rfloor}(\pi^{(1)})$.

Let $A \subseteq \pi$ be the set of calls accepted by OPT_χ on input π. Consider the following auxiliary graph H with vertex set corresponding to the calls in A: we insert one (interval-) vertex for each of the b call fragments of each accepted call. Two vertices in H are adjacent if the corresponding call fragments share an edge. Clearly, H is an interval graph with maximum clique size $\omega(H) \leq \chi$.

We consider the subgraph H' obtained from H by removing for each call $r \in A$ all but one of the b vertices representing its call fragments and their adjacent edges. Hence, the vertices in H' correspond to a subset of the calls in $\pi^{(1)}$. Clearly, $\omega(H') = \omega(H)/b \leq \chi/b$. Since interval graphs are perfect, we can color the vertices in H' using at most $\omega(H') = \chi/b$ colors. Thus, the coloring of the calls in A translates into a solution for the $\pi^{(1)}$-instance in which as many calls (of profit 1) are accepted by $\text{OPT}_{\frac{\chi}{b}}$ on $\pi^{(1)}$ as calls (with profit b) are accepted by OPT_χ on π.

Corollary 3.7. *If there is a c-competitive algorithm for* OEDPA *on the line, then* FFCS *achieves a competitive ratio of* $8(c+1)(\lceil \log \chi \rceil + 1)$ *for routing in* χ *wavelengths.* □

References

1. B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Rosén, *On-line competitive algorithms for call admission in optical networks*, Proceedings of the 4th Annual European Symposium on Algorithms, vol. 1136, 1996, pp. 431–444.
2. B. Awerbuch, Y.Bartal, A. Fiat, and A. Rosén, *Competitive, non-preemptive call control*, Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, 1994, pp. 312–320.
3. Y. Bartal, A. Fiat, and S. Leonardi, *Lower bounds for on-line graph problems with applications to on-line curcuit and optimal routing*, Proceedings of the 28th Annual ACM Symposium on the Theory of Computing, 1996, pp. 531–540.
4. A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*, Cambridge University Press, 1998.
5. J. Kleinberg and E. Tardos, *Disjoint paths in densely embedded graphs*, Proceedings of the 36th Annual IEEE Symposium on the Foundations of Computer Science, 1995, pp. 531–540.
6. S. Leonardi, A. Marchetti-Spaccamela, A. Presciutti, and A. Rosèn, *On-line randomized call-control revisited*, Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, 1998, pp. 323–332.
7. P. Raghavan and E. Upfal, *Efficient routing in all-optical networks*, Proceedings of the 26th Annual ACM Symposium on the Theory of Computing, 1994, pp. 134–143.

The Forest Wrapping Problem on Outerplanar Graphs

Isabella Lari[1], Federica Ricca[1], and Andrea Scozzari[2]

[1] Dip. Statistica, Probabilità e Statistiche Applicate
Università di Roma "La Sapienza"
{isabella.lari,federica.ricca}@uniroma1.it
[2] Dip. Matematica per le Decisioni Economiche, Finanziarie ed Assicurative
Università di Roma "La Sapienza"
andrea.scozzari@uniroma1.it

Abstract. In this paper we study the Forest Wrapping Problem (FWP) which can be stated as follows: given a connected graph $G = (V, E)$, with $|V| = n$, let π_0 be a partition of G into K (not necessarily connected) components, find a connected partition π^* of G that wraps π_0 and has maximum number of components.
The Forest Wrapping problem is NP-complete on grid graphs while is solvable in $O(n \log n)$ time on ladder graphs. We provide a two-phase $O(n^2)$ time algorithm for solving FWP on outerplanar graphs.

Keywords: Outerplanar graphs, Connected partition, Steiner Forest, Maximum Split Clustering.

1 Introduction

Given a connected graph $G = (V, E)$, with $|V| = n$, a partition of G into K components is a partition of the set V of its vertices into K subsets. A partition is connected if each subset induces a connected subgraph in G. Moreover, a partition π of G *wraps* a partition π' if each component of π' is included in some component of π [4].

In this paper we study the Forest Wrapping Problem (FWP) which can be stated as follows: let π_0 be a partition of G into K (not necessarily connected) components, find a connected partition π^* of G that wraps π_0 and has maximum number of components, that is, the minimum number of unions between components must be performed. Notice that, for each connected partition π there exists a spanning forest F of G such that each tree in F corresponds to a component of π and viceversa. Therefore, each solution of FWP is a spanning forest of G. The FWP is strictly related to the Steiner Forest Problem [1]: given a connected and undirected graph $G = (V, E)$, a cost function on the edges $c : E \rightarrow Z^+$ and a collection of disjoint subsets of V, $T_1, \ldots, T_K$, find a minimum cost forest in which each pair of vertices belonging to the same T_k is connected. Indeed, if the edge costs are all equal, the Steiner Forest Problem is the same as the FWP.

The FWP arose in the literature connected to the Maximum Split Problem with connectivity constraints on graphs [2,3]. In [3] the authors showed that the

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 345–354, 2002.

Maximum Split Problem on a graph G can be solved by solving a sequence of Forest Wrapping Problems. They also proved that the Forest Wrapping Problem is NP-complete on grid graphs, while in [4] it is proved that FWP is solvable in $O(n \log n)$ time on grid graphs with two rows and an arbitrary number of columns (ladder graphs). FWP finds a natural application in telecommunication networks design [6].

We provide a two-phase $O(n^2)$ time algorithm for solving FWP on outerplanar graphs. First, all pairs of components which must be necessarily joined in a connected partition are identified and joined (pre-processing phase). In a second phase a connected partition of G with maximum number of components is found.

The remainder of the paper is organized as follows. Section 2 provides some definitions and properties of outerplanar graphs and the procedure for the pre-processing phase. Section 3 is devoted to the FWP algorithm.

2 Definitions and Properties

An outerplanar graph $G(V, E)$, with $|V| = n$ and $|E| = m$, is a planar graph where all the vertices lie on the outer cycle *Cycle(G)*. Without loss of generality, we suppose that G is biconnected. Otherwise, the optimal solution of FWP in G can be found by solving a sequence of FWP on each biconnected component of G.

We assign a number i, $i = 1, ..., n$, to each vertex of G, such that each vertex is adjacent to at least one vertex j with $j > i$ and to at least one vertex j' with $j' < i$. The only exceptions are the vertices numbered 1 and n.

Since the dual graph of an outerplanar graph G is a tree T, it is always possible to assign a number h, $h = 1, ..., H$, to each face of G (i.e., each vertex of the dual graph) such that each face h is adjacent exactly to one face with a number grater than h except for face H which is the outer face of G. The chord separating face h and a face h', with $h' > h$, will be referred to as the *gate of face h* and its vertices will be called the *ends* of the gate. We denote by i_ℓ^h the left end of the gate, that is the vertex with the lower number, and by i_r^h the right end. Notice that $i_\ell^H = 1$ and $i_r^H = n$.

We root the dual tree in face H, and denote it by T_H. Each face h corresponds to a vertex of T_H and the gate of face h corresponds to the edge connecting h to its parent in T_H. We denote by T_h the subtree of T_H rooted in h. We will refer to the gate connecting T_h to $T_H \backslash T_h$ as the *gate of subtree* T_h. A subtree $T_{h'}$ *nested* in T_h is a subtree of T_h rooted in face h'.

In the following we denote by π_0 the *initial partition* of the FWP, and we define *feasible partition* a connected partition wrapping π_0.

For each component C of a partition π, let $I(C)$ and $T(C)$ be the smallest and largest numbered vertices of C, respectively. Since each vertex i belongs to only one component, we introduce a label $C(i)$ such that if i belongs to component C', then $C(i) = C'$.

We define C an *expanding component with respect to face h* if it has vertices both in T_h and in $T_H \backslash T_h$.

Given a graph G, let C' and C'' be two different components of the initial partition π_0. A partition π' *separates* C' and C'' if C' and C'' are included into two different components of π' [4].

We will now introduce some definitions and results which are useful to understand the pre-processing phase.

Definition 1. Two components C' and C'' of π_0 *overlap* in G if there is no feasible partition π' which separates C' and C''.

Given a component C, let P_C be a path in *Cycle(G)* which connects all the vertices which belong to C.

Definition 2. Given an outerplanar graph G, two components C' and C'' of π_0 *alternate* in *Cycle(G)* if two vertex-disjoint paths $P_{C'}$ and $P_{C''}$ do not exist.

Definition 3. A set of q components, $C_1, C_2, \ldots, C_q$, produces a chain of alternancies if there is a permutation $i_1, i_2, \ldots, i_q$, such that each pair of consecutive components C_{i_j} and $C_{i_{j+1}}$ alternate. Components C_{i_1} C_{i_q} are called the *extremities* of the chain $C_{i_1}, C_{i_2}, \ldots, C_{i_q}$ which, hereafter, will be denoted by (C_{i_1}, C_{i_q}).

Remark 1. Two components C' and C'' of π_0 overlap in *Cycle(G)* if and only if they alternate or are the extremities of a chain of alternancies.

Theorem 1. *Two components C' and C'' of π_0 overlap in G if and only if they overlap in Cycle(G).*

Proof. Since *Cycle(G)* is a connected subgraph of G, it follows immediately that two components which overlap in G necessarily overlap in *Cycle(G)*. On the other hand, assume that C' and C'' overlap in *Cycle(G)* and suppose by contradiction that they do not overlap in G. In such case the following is true:

a) C' and C'' alternate or are the extremities of a chain of alternancies, (C', C'') (by Remark 1)
b) there is a connected partition $\overline{\pi}$ of G in which C' and C'' are separated (by definition).

Since a) holds, then there are two cases:

If C' and C'' alternate, then since b) holds, there must be two chords, (s, t) and (u, v), which connect the elements of C' and of C'' in $\overline{\pi}$, respectively. By hypothesis, C' and C'' overlap in *Cycle(G)*, therefore vertices s, t, u, v and *Cycle(G)* form a subgraph of G homomorphic to k_4. This is impossible because G is an outerplanar graph [5].

If C' and C'' are the extremities of a chain of alternancies, then there is at least a pair of alternating components C_{i_k}, $C_{i_{k+1}}$ in (C', C'') which are separated in $\overline{\pi}$. Using the same reasoning as above, G contains a subgraph homomorphic to k_4, which implies that G cannot be outerplanar. □

2.1 The Pre-processing Phase

To solve the FWP we need a pre-processing phase which merges every pair of overlapping components. Pre-processing can be performed in $O(n \log n)$ time by using a procedure similar to the one provided in [4]. Notice that, according to Remark 1 and Theorem 1, the search for overlapping components in G can be restricted to the search of all the alternancies in *Cycle(G)*.

Let us consider *Cycle(G)* and the $n-1$ paths which can be obtained by removing each one of its edges.

Remark 2. If two components C' and C'' alternate in *Cycle(G)*, then they alternate in each of the $n-1$ possible paths generated by removing an edge in *Cycle(G)*.

In particular, Remark 2 holds if edge (1,n) is removed, hence we denote by $P(G)$ the resulting path. Notice that two components C' and C'' alternate in $P(G)$ if $I(C') < I(C'')$ and there exists at least one vertex i belonging to C' such that $I(C'') < i < T(C'')$. The pre-processing algorithm is briefly described below.

algorithm PRE-PROCESSING
input: A cycle with n vertices; a partition π_0 in K components;
$I(C)$ and $T(C)$ for each component C
output: A partition π_1 with no overlapping components
begin
  S:=$\emptyset$
  for $i = 1$ **to** n **do**
    if $(T(C(i)) > I(C(i)))$ **then**
      if $(T(C(i)) = i)$ **then**
        remove $C(i)$ from S
      else
        if $C(i) \notin S$ **then** put $C(i)$ in S
        find C' such that
        $T(C') = min_{(C \in S)} T(C)$
        if $(T(C(i)) > T(C'))$ **then**
          merge components $C(i)$ and C'
  end for
end

Theorem 2. *PRE-PROCESSING algorithm finds and merges all the overlapping components of Cycle(G) in $O(n \log n)$ time.*

Proof. The algorithm scans the vertices of *Cycle(G)* from 1 to n and for each i, updates the set S containing the labels of the components C' such that $I(C') < i$ and $T(C') > i$. The algorithm checks if $C(i)$ alternates or not with another component in S by comparing $T(C(i))$ with $\min_{(C \in S)} T(C)$. By using an heap,

this can be done for all i in $O(n \log n)$ time. Merging all the components can be done in $O(n \log n)$ time if each time we merge two components we move the elements of the smaller component into the bigger one. □

By running the pre-processing phase on *Cycle(G)* we obtain a partition π_1 - not necessarily connected - without overlapping components which wraps π_0.

3 The Algorithm for the FWP on Outerplanar Graphs

In this section we describe the algorithm for the FWP which starts after the partition π_1 is obtained from the pre-processing phase.

Definition 4. *A* feasible solution $\pi(h)$ *of the subtree* T_h *satisfies the following two properties:*

1) all non expanding components are connected;
2) all the vertices of an expanding component belong to component $C(i^h_\ell)$ *or to component* $C(i^h_r)$*. It also may happen that* $C(i^h_\ell) = C(i^h_r)$*.*

Given $\pi(h)$, we denote by $q(\pi(h))$ the minimum number of unions to reach a connected partition of G starting from $\pi(h)$.

An *optimal solution* of the subtree T_h is a feasible solution of T_h obtained with the minimum number of unions. Notice that an optimal solution of T_H is also an optimal solution for FWP in G.

Remark 3. Let $\pi(h)$ be a feasible solution of T_h such that the ends of the gate of face h belong to the same component and let $\pi'(h)$ be a feasible solution of the same tree where the ends of the gate belong to different components, then $q(\pi(h)) \leq q(\pi'(h))$.

Proposition 1. *Given two feasible solutions* $\pi'(h)$ *and* $\pi''(h)$ *of* T_h*, then*

$$|q(\pi'(h)) - q(\pi''(h))| \leq 1.$$

Proof. Suppose that $q(\pi''(h)) > q(\pi'(h))$. By Remark 3, this occurs only if in $\pi''(h)$ the ends of the gate of face h belong to different components. Let $\pi(h)$ be the solution obtained from $\pi''(h)$ by merging these two components. By Remark 3, $q(\pi(h)) \leq q(\pi'(h))$. Since $q(\pi''(h))$ is the minimum number of unions to reach a connected partition of G starting from $\pi''(h)$, $q(\pi(h)) + 1 \geq q(\pi''(h))$. Then, $q(\pi''(h)) \leq q(\pi'(h)) + 1$. □

After having identified all the faces of G, the idea of the algorithm is to visit T_H bottom-up minimizing the number of unions in each T_h, $h = 1, \ldots, H$. Proposition 1 will be useful to prove that minimizing the number of unions in each subtree T_h leads to an optimal solution of G.

In each subtree T_h we have three cases: the optimal solution is unique; there are multiple optimal solutions among which at least one with the ends of the gate

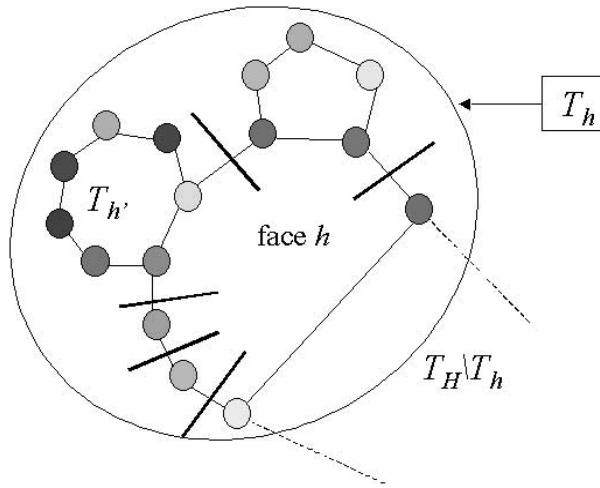


Fig. 1. The removed edges in face h.

of face h belonging to the same component; there are multiple optimal solutions and all of them have the ends of the gate belonging to different components. In the first two cases we solve the subtree T_h definitively, by considering either the unique solution or a solution with the ends of the gate belonging to the same component. Then, we prune the subtree T_h from T_H. In the third case T_h will be visited again when solving a subtree $T_{h'}$ in which T_h is nested.

Remark 4. There exists an optimal solution of T_h in which at least one edge of face h is removed, that is, it does not belong to the corresponding optimal spanning forest in T_h (see Fig.1). Notice that, if the removed edge is the gate of a subtree $T_{h'}$ nested in T_h, and $T_{h'}$ has not been pruned, then, in T_h we are removing both this edge and at least one edge in $T_{h'}$ (see Fig.2).

Following Remark 4, we solve a subtree T_h by removing one edge of *Cycle(G)* at a time from i_ℓ^h to i_r^h. More precisely, according to the numeration of the vertices of G ($i = 1, ..., n$), in face h we remove all the edges ($i-1$,i) of *Cycle(G)*, while in a subtree $T_{h'}$ nested in T_h we remove only those edges which correspond to an optimal solution of $T_{h'}$. Once an edge has been removed, we count the number of unions necessary to connect all the components in T_h from i to i_r^h and from $i-1$ to i_ℓ^h, separately.

In the following we outline the pseudo-code of the FW-ALGORITHM. In order to implement the algorithm we introduce a circular list necessary to scan the graph. For each vertex i, a pair of pointers, $next(i)$ and $previous(i)$, is defined. At the beginning of the algorithm $next(n) = 1$ and $previous(1) = n$, while $next(i) = i + 1$, $i = 1, ..., n-1$ and $previous(i) = i - 1$, $i = 2, ..., n$. Each time a face h is visited, $next(i)$ and $previous(i)$ update as follows: according to the numeration of the vertices in G, $next(i)$ points to the first vertex $j > i$ in T_h which belongs to an expanding component $C(j) \neq C(i)$, if no such vertex exists, $next(i)$ points to i_r^h. On the other hand, $previous(i)$ refers to the first vertex

$j < i$ in T_h which belongs to an expanding component $C(j) \neq C(i)$, if no such vertex exists, $previous(i)$ points to i_ℓ^h.

K more circular lists are necessary to visit the sequence of vertices belonging to each component and to efficiently perform the merging operation between two components.

In the algorithm U^* denotes the minimum number of unions to obtain a connected partition π_2 which wraps π_1.

algorithm `FW-ALGORITHM`
input: A biconnected outerplanar graph G with n vertices,
the tree T_H and a partition π_1 with no overlapping components
output: A connected partition π_2 which wraps π_1
with the minimum number of unions U^*
begin
  starting from the leaves and proceeding bottom-up in T_H,
  for each subtree T_h **do**
    let $i = i_\ell^h$
    while $(i < i_r^h)$ **do**
      if $(i = i_\ell^{h'})$ for some nested subtree $T_{h'}$ **then**
        while $(i < i_r^{h'})$ **do**
          according to the numeration of the vertices of G
          remove the last edge of the path from i to $next(i)$
          find a feasible partition of T_h and the corresponding
          number of unions U_i (see Remark 4)
          update $i = next(i)$
        end
      else
        remove edge $(i, next(i))$
        find a feasible partition of T_h and the corresponding
        number of unions U_i (see Remark 4)
        update $i = next(i)$
      end
    end
    let $U_h = \min_i\{U_i\}$
    let S_h be the set of the removed edges providing optimal solutions of T_h
    if $(|S_h| = 1$ **or** there is a solution in S_h with the ends of the gate
    belonging to the same component) **then**
      prune T_h and perform the unions of the optimal solution of T_h
    end
    update the circular lists for the vertices in T_h
  end
  let $U^* = \sum_h U_h$
  choose an element in S_H and perform the unions
end

The algorithm searches for a connected partition of T_h by iteratively removing an edge. In particular, two cases are possible: the removed edge is in face h or

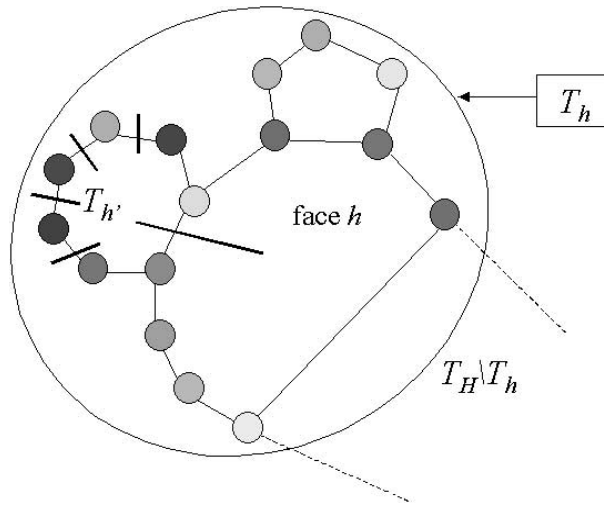


Fig. 2. The removed edges in a subtree $T_{h'}$.

it is in a nested subtree $T_{h'}$. In both cases two paths formed by only edges of face h, are determined, one lying on the right and the other on the left of the removed edge. Denote them by P_{right} and P_{left}, respectively. When the algorithm removes edge $(i, next(i))$ in face h, P_{right} is the path from $next(i)$ to i_r^h and P_{left} is from i to i_ℓ^h. On the other end, if the removed edge is in a nested subtree $T_{h'}$, P_{right} is the path from $i_r^{h'}$ to i_r^h and P_{left} is from $i_\ell^{h'}$ to i_ℓ^h (see Remark 4). Consider path P_{right}, since the components do not overlap, it is possible to find a sequence of vertex-disjoint subpaths $[a, b]$ of P_{right} in which all the vertices must be necessarily connected. For each subpath the procedure counts (without performing) the number of unions to connect the components in all the subpaths $[a, b]$ of P_{right}. It must be noticed that the procedure counts the number of unions between components from vertex a to vertex b, except for those unions which have been already counted in some subtree $T_{h'}$ with $a \leq i_\ell^{h'} < i_r^{h'} \leq b$. The procedure works in a similar way when P_{left} is considered. In order to scan P_{right} the algorithm uses the circular list $next(.)$, while to scan P_{left} it uses the circular list $previous(.)$. Finally, U_i is the sum of all the unions performed on P_{right} and P_{left}.

After T_h has been visited, S_h is the set of the removed edges which provide optimal solutions of T_h and U_h is the minimum number of unions. Notice that, each removed edge in S_h corresponds to a sequence of subpaths from i_ℓ^h to i_r^h which can be efficiently stored by exploiting the pointers $next(.)$. In fact, when $i = i_\ell^{h'}$, according to the definition, $next(i)$ allows to easily identify the sequence of the edges which correspond to optimal solutions of $T_{h'}$. On the other hand, if T_h is pruned $next(i_\ell^h) = i_r^h$, $previous(i_r^h) = i_\ell^h$ and the procedure merges all the components in each subpath from i_ℓ^h to i_r^h. Therefore, before visiting the next subtree, pointers $next(i)$ and $previous(i)$ need to be updated for each $i_\ell^h \leq i \leq i_r^h$.

At the end, once T_H has been visited, S_H corresponds to the set of the optimal solutions of FWP on G. The optimal partition π_2 can be found by selecting an element of S_H and performing the corresponding unions.

Theorem 3. *FW-ALGORITHM always finds a connected partition π_2 which wraps π_1 by performing the minimum number of unions.*

Proof. Suppose that for each $h = 1, ..., H$ there is a unique optimal solution $\pi_2(h)$ in T_h. Since $\pi_2(h)$ gives the minimum number of unions, any other feasible solution $\pi'(h)$ performs at least one more union. By Proposition 1, the minimum number of unions needed to obtain a feasible solution for T_H starting from $\pi_2(h)$ is greater than the minimum number of unions for T_H starting from $\pi'(h)$ of at most one. Therefore, the optimal solution of T_H can be obtained by a sequence of optimal solutions in T_h, $h = 1, ..., H$.

Suppose that in T_h there is a set of optimal solutions. By Proposition 1, there is at least one which will be used to minimize the number of unions in T_H. If one of them has $C(i_\ell^h) = C(i_r^h)$, then by Remark 3, it can be chosen as the (unique) optimal solution of T_h. Otherwise, all the optimal solutions will be reconsidered and the choice among them will depend on the optimal solutions of the ancestors of face h in T_H. Starting from the leaves of T_H, FW-ALGORITHM finds an optimal solution of T_h, for each $h = 1, ..., H$. According to Remark 4, FW-ALGORITHM finds an optimal solution of T_h by enumerating all the feasible solutions obtained when an edge of T_h is removed. If the removed edge is the gate of a nested subtree $T_{h'}$, and in $T_{h'}$ there are multiple optimal solutions, the algorithm visits $T_{h'}$ and removes only those edges which correspond to optimal solutions of $T_{h'}$. Hence, an optimal solution of T_h, $h = 1, ..., H$, is obtained by starting from optimal solutions of the subtrees nested in T_h. □

Theorem 4. *FW-ALGORITHM finds an optimal solution in time $O(n^2)$.*

Proof. Let $t_h \leq n$ be the number of vertices of G in subtree T_h and let n_h be the number of vertices of face h. When solving T_h, in order to find the sequence of subpaths $[a, b]$ of P_{right} and P_{left}, the algorithm visits at most n_h vertices for each removed edge. Since in T_h at most t_h edges can be removed, the time complexity to find an optimal solution of T_h is $t_h n_h$. Then, the time complexity to find an optimal solution in T_H is $\sum_{h=1}^{H} t_h n_h \leq n \sum_{h=1}^{H} n_h = O(n^2)$.

Merging component C with L vertices and a component C' with $L' < L$ vertices requires the following two operations: update the lists which scan the sequence of vertices belonging to components C and C', and update the label $C(i) = C'$ to $C(i) = C$ for all vertices i belonging to C'. The first operation requires $O(1)$ time, since C and C' do not overlap, while the latter can be done by following a strategy similar to the one used in the pre-processing phase. Hence, in T_H merging all the components of an optimal solution takes $O(n \log n)$ time.

The time needed to update *next(.)* and *previous(.)* in T_h is $O(t_h)$, since the procedure needs to visit *Cycle(G)* from i_r^h to i_ℓ^h and from i_ℓ^h to i_r^h, respectively. Therefore, in T_H the time complexity to update *next(.)* and *previous(.)* is $\sum_{h=1}^{H} t_h = O(n^2)$. Hence, the overall time complexity of FW-ALGORITHM is $O(n^2)$. □

References

1. M.X. Goemans, D.P. Williamson, "A General Approximation Technique for Constrained Forest Problems", *SIAM J. of Computing* **24**, 296-317 (1995).
2. P. Hansen, B. Jaumard, K. Musitu, "Weight Constrained Maximum Split Clustering", *J. of Classification* **7**, 217-240 (1990).
3. P. Hansen, B. Jaumard, B. Simeone and V. Doring, "Maximum Split Clustering Under Connectivity Constraints", *Les Cahiers du GERARD* report G-93-06, (1993).
4. I. Lari, "Connected Maximum Split Clustering of Ladder Graphs", *Proceedings of Gellschaft fur Klassifikation e. V. 24th Annual Conference* - Passau 2000.
5. J. Valdes, R.E. Tarjan, and E.L. Lawler "The recognition of Series Parallel Digraphs", *SIAM J. on Comp.* **11**, 298-313 (1982).
6. J.A. Wald, C.J. Colbourn, "Steiner Trees, Partial 2-Trees, and Minimum IFI Networks", *Networks* **13**, 159-167 (1983).

On the Recognition of P_4-Comparability Graphs

Stavros D. Nikolopoulos and Leonidas Palios

Department of Computer Science, University of Ioannina
GR-45110 Ioannina, Greece
{stavros,palios}@cs.uoi.gr

Abstract. We consider the problem of recognizing whether a simple undirected graph is a P_4-comparability graph. This problem has been considered by Hoàng and Reed who described an $O(n^4)$-time algorithm for its solution, where n is the number of vertices of the given graph. Faster algorithms have recently been presented by Raschle and Simon and by Nikolopoulos and Palios; the time complexity of both algorithms is $O(n + m^2)$, where m is the number of edges of the graph.
In this paper, we describe an $O(n\,m)$-time, $O(n+m)$-space algorithm for the recognition of P_4-comparability graphs. The algorithm computes the P_4s of the input graph G by means of the BFS-trees of the *complement* of G rooted at each of its vertices, without however explicitly computing the complement of G. Our algorithm is simple, uses simple data structures, and leads to an $O(n\,m)$-time algorithm for computing an acyclic P_4-transitive orientation of a P_4-comparability graph.

Keywords: Perfectly orderable graph, comparability graph, P_4-comparability graph, recognition, P_4-component, P_4-transitive orientation.

1 Introduction

We consider simple non-trivial undirected graphs. Let $G = (V, E)$ be such a graph. An *orientation* of G is an antisymmetric directed graph obtained from G by assigning a direction to each edge of G. An orientation (V, F) of G is called *transitive* if it satisfies the following condition: $\overrightarrow{ab} \in F$ and $\overrightarrow{bc} \in F$ imply $\overrightarrow{ac} \in F$, for all $a, b, c \in V$, where by $\overrightarrow{uv}$ or $\overleftarrow{vu}$ we denote an edge directed from u to v [8]. An orientation of a graph G is called P_4*-transitive* if it is transitive when restricted to any P_4 (chordless path on 4 vertices) of G; an orientation of such a path $abcd$ is transitive if and only if the path's edges are oriented in one of the following two ways: $\overrightarrow{ab}$, $\overleftarrow{bc}$ and $\overrightarrow{cd}$, or $\overleftarrow{ab}$, $\overrightarrow{bc}$ and $\overleftarrow{cd}$.

A graph which admits an acyclic transitive orientation is called a *comparability graph* [7,8,9]; Figure 1(a) depicts a comparability graph. A graph is a P_4-*comparability graph* if it admits an acyclic P_4-transitive orientation [11,12]. In light of these definitions, every comparability graph is a P_4-comparability graph. However, the converse is not always true; the graph depicted in Figure 1(b) is a P_4-comparability graph but it is not a comparability graph (it is often referred to as a pyramid). The graph shown in Figure 1(c) is not a P_4-comparability graph. The class of P_4-comparability graphs was introduced by Hoàng and Reed, along

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 355–366, 2002.

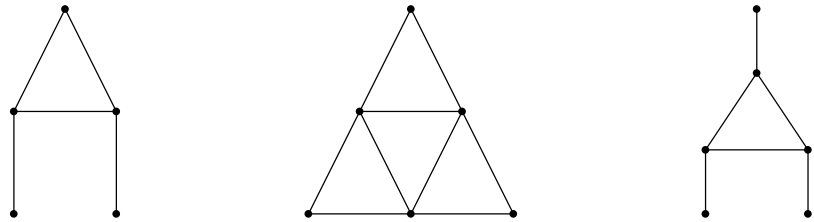


Fig. 1. (a) a comparability graph; (b) a P_4-comparability graph (which is not comparability); (c) a graph which is not P_4-comparability.

with the classes of the P_4-indifference, the P_4-simplicial, and the Raspail graphs, and all four classes were shown to be perfectly orderable [12].

The class of *perfectly orderable* graphs was introduced by Chvátal in the early 1980s [4]; it is a very important class of graphs since a number of problems which are NP-complete in general can be solved in polynomial time on its members [2,8,10]; unfortunately, it is NP-complete to decide whether a graph is perfectly orderable [15]. Chvátal showed that the class of perfectly orderable graphs contains the comparability and the triangulated graphs [4]. It also contains a number of other classes of perfect graphs which are characterized by important algorithmic and structural properties, such as, the classes of 2-threshold, brittle, co-chordal, weak bipolarizable, distance hereditary, Meyniel ∩ co-Meyniel, P_4-sparse, etc. [3,8]. Finally, since every perfectly orderable graph is strongly perfect [4], the class of perfectly orderable graphs is a subclass of the well-known class of perfect graphs.

Algorithms for many different problems on almost all the subclasses of perfectly orderable graphs are available in the literature. The comparability graphs in particular have been the focus of much research which culminated into efficient recognition and orientation algorithms [3,8,14]. On the other hand, the P_4-comparability graphs have not received as much attention, despite the fact that the definition of the P_4-comparability graphs is a direct extension of the definition of comparability graphs [6,11,12,17].

Our main objective in this paper is to study the recognition problem on the class of P_4-comparability graphs. This problem along with the problem of producing an acyclic P_4-transitive orientation have been addressed by Hoàng and Reed who described an $O(n^4)$- and an $O(n^5)$-time algorithm respectively for their solution [11,12], where n is the number of vertices of the input graph. Improved results on these problems were provided by Raschle and Simon [17]. Their algorithms work along the same lines, but focus on the P_4-components of the graph; both algorithms run in $O(n + m^2)$, where m is the number of edges of the input graph. Recently, Nikolopoulos and Palios described different $O(n+m^2)$-time algorithms for these problems [16]. Their approach relies on the construction of the P_4-components by means of BFS-trees of the input graph.

In this paper, we present an $O(n\,m)$-time recognition algorithm for P_4-comparability graphs, where n and m are the number of vertices and edges of the input graph. The algorithm computes the P_4s of the input graph G by means of the BFS-trees of the *complement* of G rooted at each of its vertices, without however explicitly computing the complement of G. Instrumental for the algorithm are the observations that the complement of a P_4 is also a P_4 and that for a graph G, the number of vertices in all the levels, but the 0th and the 1st, of the BFS-tree of the complement of G rooted at a vertex v does not exceed the degree of v in G. The proposed recognition algorithm is simple, uses simple data structures and requires $O(n + m)$ space. Along with the result in [16], it leads to an $O(n\,m)$-time algorithm for computing an acyclic P_4-transitive orientation of a P_4-comparability graph.

2 Theoretical Framework

We consider simple non-trivial undirected graphs. Let $G = (V, E)$ be such a graph. A *path* in G is a sequence of vertices $(v_0, v_1, \ldots, v_k)$ such that $v_{i-1}v_i \in E$ for $i = 1, 2, \ldots, k$; 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 simple path $(v_0, v_1, \ldots, v_k)$ is *chordless* if $v_iv_j \notin E$ for any two non-consecutive vertices v_i, v_j in the path. Throughout the paper, the chordless path on n vertices is denoted by P_n. In particular, a chordless path on 4 vertices is denoted by P_4.

Two P_4s are called *adjacent* if they have an edge in common. The transitive closure of the adjacency relation is an equivalence relation on the set of P_4s of a graph G; the subgraphs of G spanned by the edges of the P_4s in the equivalence classes are the P_4-*components* of G. With slight abuse of terminology, we consider that an edge which does not belong to any P_4 belongs to a P_4-component by itself; such a component is called *trivial*. A P_4-component which is not trivial is called *non-trivial*; clearly a non-trivial P_4-component contains at least one P_4.

The definition of a P_4-comparability graph requires that such a graph admits an acyclic P_4-transitive orientation. However, Hoàng and Reed [12] showed that in order to determine whether a graph is a P_4-comparability graph one can restrict one's attention to the P_4-components of the graph. What they proved ([12], Theorem 3.1) can be paraphrased in terms of the P_4-components as follows:

Lemma 1. [12] *Let G be a graph such that each of its P_4-components admits an acyclic P_4-transitive orientation. Then G is a P_4-comparability graph.*

Our recognition algorithm relies on the following important lemma in order to achieve its stated time complexity.

Lemma 2. *Let G be an undirected graph and let $T_{\overline{G}}(v)$ be the BFS-tree of the complement $\overline{G}$ of G rooted at a vertex v. Then, the number of vertices in all the levels of $T_{\overline{G}}(v)$, except for the 0th and the 1st, does not exceed the degree of v in G.*

Proof. Clearly true, since the vertices in all the levels of $T_{\overline{G}}(v)$, except for the 0th and the 1st, are vertices which are not adjacent to v in $\overline{G}$. ∎

3 Recognition of P_4-Comparability Graphs

The algorithm works by constructing and orienting the P_4-components of the given graph, say, G, and then by checking whether they are acyclic (Lemma 1). The P_4-components are constructed as follows: the algorithm considers initially m (partial) P_4-components, one for each edge of G; then, it locates the P_3s of all the P_4s of G, and whenever the edges of such a P_3 belong to different (partial) P_4-components it unions and appropriately orients these P_4-components. Since we are interested in a P_4-transitive orientation of each P_4-component, the edges of such a P_3 need to be oriented either towards their common endpoint or away from it.

As stated earlier, the P_4s of the graph G are computed by means of processing the BFS-trees of the complement $\overline{G}$ of G rooted at each of its vertices. It is important to observe that if *abcd* is a P_4 then its complement is the P_4 *bdac* and it belongs to the complement $\overline{G}$ of G. Let us consider the BFS-tree $T_{\overline{G}}(b)$ of $\overline{G}$ rooted at b. Since *bdac* is a P_4 of $\overline{G}$, the vertices b, d, and a have to belong to the 0th, 1st, and 2nd level of $T_{\overline{G}}(b)$ respectively; the vertex c belongs to the 2nd or 3rd level, but not to the 1st, since c is not adjacent to b in $\overline{G}$. These two cases are shown in Figure 2.

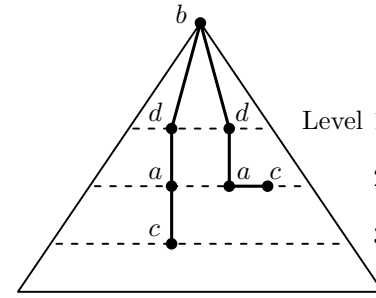


Fig. 2. The two positions of the P_4 *bdac* in the BFS-tree $T_{\overline{G}}(b)$.

The algorithm is described in more detail below. We consider that the input graph is connected; the case of disconnected graphs is addressed in Section 3.3. Additionally, we assume that initially each edge of G belongs to a P_4-component by itself and is assigned an arbitrary orientation.

P_4-comparability Graph Recognition Algorithm. Input: a connected graph G on n vertices and m edges. *Output:* yes, if G is a P_4-comparability graph; otherwise, no.

1. Initialize to 0 all the entries of an array $M[\,]$ which is of size n;
2. For each vertex v of the graph G, do
   2.1 compute the sets L_1, L_2, and L_3 of vertices in the 1st, 2nd, and 3rd level respectively of the BFS-tree of the complement $\overline{G}$ rooted at v;
   2.2 partition the set L_2 into subsets of vertices so that two vertices belong to the same subset iff they have (in $\overline{G}$) the same neighbors in L_1;
   2.3 for each vertex x in L_2, do

2.3.1 for each vertex w adjacent to x in G, do
$M[w] \leftarrow 1$; $\{mark\ in\ M[\,]\ the\ neighbors\ of\ x\ in\ G\}$

2.3.2 for each vertex y in L_3 do
if $M[y] = 0$
then $\{xvy\ is\ a\ P_3\ in\ a\ P_4\ of\ G\}$
If the edges xv and vy belong to the same P_4-component and do not both point towards v or away from it, then the P_4-component cannot admit a P_4-transitive orientation and we conclude that the graph G is not a P_4-comparability graph.
If the edges xv and vy belong to different P_4-components, then we union these components into a single component and if the edges do not both point towards v or away from it, we invert (during the unioning) the orientation of all the edges of the unioned P_4-component with the fewest edges.

2.3.3 for each vertex y in L_2 do
if $M[y] = 0$ and the vertices x and y belong to different partition sets of L_2 (see Step 2.2)
then $\{xvy\ is\ a\ P_3\ in\ a\ P_4\ of\ G\}$
process the edges xv and vy as in Step 2.3.2;

2.3.4 for each vertex w adjacent to x in G, do
$M[w] \leftarrow 0$; $\{clear\ M[\,]\}$

3. After all the vertices have been processed, we apply topological sorting on the directed graph spanned by the directed edges associated with each of the non-trivial P_4-components; if the topological sorting succeeds then the component is acyclic, otherwise there is a directed cycle. If any of the P_4-components contains a directed cycle, then the graph is not a P_4-comparability graph.

For each P_4-component, we maintain a linked list of the records of the edges in the component, and the total number of these edges. Each edge record contains a pointer to the header record of the component to which the edge belongs; in this way, we can determine in constant time the component to which an edge belongs and the component's size. Unioning two P_4-components is done by updating the edge records of the smallest component and by linking them to the edge list of the largest one, which implies that the union operation takes time linear in the size of the smallest component. As mentioned above, in the process of unioning, we may have to invert the orientation in the edge records that we link, if the current orientations are not compatible.

Correctness of the Recognition algorithm. The correctness of the algorithm follows (i) from the fact that in Steps 2.3.2 and 2.3.3 it processes precisely the P_3s participating in P_4s of the input graph G (Lemmata 3 and 4) and that it assigns correct orientations on the edges of these P_3s, (ii) from the correct construction of the P_4-components by unioning partial P_4-components whenever a P_3 is processed whose edges belong to more than one such partial components, and (iii) from Lemma 1 in conjunction with Step 2 of the algorithm.

Note that the initial assignment of 0 to all the entries of the array $M[\,]$ and the clearing of all the set entries at Step 2.3.4 of the algorithm guarantee that the only entries of the array which are set at any iteration are precisely those corresponding to the vertices adjacent in G to the vertex processed at Step 2.3.

Lemma 3. *Every P_3 in a P_4 of the input graph G is considered at Steps 2.3.2 or 2.3.3 of the recognition algorithm.*

Proof. Let $abcd$ be a P_4 of the graph G; we will show that the P_3 abc is considered at Step 2.3.2 or 2.3.3 of the recognition algorithm. Since the algorithm processes each vertex v of G in Step 2 and considers the BFS-tree of $\overline{G}$ rooted at v, it will process b, it will consider the BFS-tree $T_{\overline{G}}(b)$ of $\overline{G}$ rooted at b, and it will compute the sets L_1, L_2, and L_3 of vertices in the 1st, 2nd, and 3rd level of $T_{\overline{G}}(b)$ respectively. Let us consider the two cases of Figure 2. In the first case, the vertices a and c belong to the 2nd and 3rd level of $T_{\overline{G}}(b)$ respectively and they are adjacent in $\overline{G}$. Thus, $a \in L_2$ and $c \in L_3$. Moreover, since a and c are adjacent in $\overline{G}$, then a and c are not adjacent in G. Hence, $M[c] = 0$ when $x = a$ in Step 2.3. Therefore, the P_3 abc is considered in Step 2.3.2 when $x = a$ and $y = c$. In the second case of Figure 2, the vertices a and c belong to the 2nd level of $T_{\overline{G}}(b)$, they are adjacent in $\overline{G}$, and a is adjacent to $d \in L_1$ in $\overline{G}$ whereas c is not. Thus, $a \in L_2$, $c \in L_2$ and $M[c] = 0$ when $x = a$ in Step 2.3, and the vertices a and c belong to different sets in the partition of the vertices in L_2 depending on the vertices in L_1 to which they are adjacent in $\overline{G}$. Therefore, the P_3 abc is considered in Step 2.3.3 when $x = a$ and $y = c$. ∎

Lemma 4. *The sequence (x, v, y) of vertices considered at Steps 2.3.2 and 2.3.3 of the recognition algorithm is a P_3 in a P_4 of the input graph G.*

Proof. Let us first consider Step 2.3.2; in this case, the vertices x and y are in the 2nd and 3rd level of $T_{\overline{G}}(v)$ respectively. Then, the path vp_xxy is a P_4 in $\overline{G}$, where p_x is the parent of x in $T_{\overline{G}}(v)$. This implies that $xvyp_x$ is a P_4 in G and xvy is a P_3 in a P_4 of G. Let us now consider Step 2.3.3. Then, the vertices x and y are in the 2nd level of the BFS-tree $T_{\overline{G}}(v)$ of $\overline{G}$ rooted at v. Moreover, since $M[y] = 0$, then x and y are not adjacent in G, that is, they are adjacent in $\overline{G}$. Finally, the fact that x and y do not belong to the same partition set of L_2, implies that there is a vertex in the 1st level of $T_{\overline{G}}(v)$ which is adjacent to one of them in $\overline{G}$ and not to the other one. Suppose that this vertex is z and that it is adjacent to x; the case where z is adjacent to y and not to x is similar. Then, the path $vzxy$ is a P_4 in $\overline{G}$, which implies that $xvyz$ is a P_4 in G. Clearly, xvy is a P_3 in a P_4 of G. ∎

Before analyzing the complexity of the recognition algorithm, we explain in more detail how Steps 2.1 and 2.2 are carried out.

3.1 Computing the Vertex Sets L_1, L_2, and L_3

The computation of these sets can be done by means of the algorithms of Dahlhaus et al. [5] and Ito and Yokoyama [13] for computing the BFS-tree of the

complement of a graph in time linear in the size of the given graph. Both algorithms require the construction of a special representation of the graph. However, we will be using another algorithm which computes the vertices in each level of the BFS-tree of the complement of a graph (i.e., it effectively implements breadth-first search on the complement) in the above stated time complexity. The algorithm is very simple and uses the standard adjacency list representation of a graph. It works by constructing each level L_{i+1} from the previous one, L_i, based on the following lemma.

Lemma 5. *Let G be an undirected graph and let L_i be the set of vertices in the i-th level of a BFS-tree of $\overline{G}$. Consider a vertex w which does not appear in any of the levels from the 0th up to the k-th. Then w is a vertex of the $(k+1)$-st level if and only if there exists at least one vertex of L_k which is not adjacent to w in G.*

Proof. The vertex w is a vertex of the $(k+1)$-st level if and only if it is adjacent in $\overline{G}$ to at least one vertex in L_k. The lemma follows. ∎

We give below the description of the algorithm.

Algorithm for computing the BFS-tree of a vertex v in the complement of a given graph G.

1. Initialize to 0 all the entries of the array $Adj[\,]$ which is of size n;
2. Construct a list L_0 containing a single record associated with the vertex v and a list S containing a record for each of the vertices of G except for v;
3. $i \leftarrow 0$;
   While the list L_i is not empty, do
   3.1 initialize the list L_{i+1} to the empty list;
   3.2 for each vertex u in L_i do
   for each vertex w adjacent to u in G do
   increment $Adj[w]$ by 1;
   3.3 for each vertex s in S do
   if $Adj[s] < |L_i|$
   then remove s from S and add it to the list L_{i+1}
   else $Adj[s] \leftarrow 0$;
   3.4 increment i by 1;

The correctness of the algorithm follows from Lemma 5. Note that the set S contains the vertices which, until the current iteration, have not appeared in any of the computed levels. Moreover, because of Steps 1 and 3.3, the entries of the array $Adj[\,]$ corresponding to the vertices in S are equal to 0 at the beginning of each iteration of the while loop in Step 3. In this way, the test "$Adj[s] < |L_i|$" correctly tests the number of vertices of L_i which are adjacent to the vertex s in G against the size of L_i. Finally, it must be noted that when the while loop of Step 3 terminates, the list S may very well be non-empty; this happens when the graph $\overline{G}$ is disconnected.

Suppose that the input graph G has n vertices and m edges. Clearly, Steps 1 and 2 take $O(n)$ time. In each iteration of the while loop of Step 3, Steps 3.1 and 3.4 take constant time, while Step 3.2 takes $O(\sum_{u \in L_i} d_G(u))$ time, where $d_G(u)$ denotes the degree of the vertex u in G. Step 3.3 takes time linear in the current size of the list S; the elements of S can be partitioned into two sets: (i) the vertices which end up belonging to L_{i+1}, and (ii) the vertices for which the corresponding entries of the array $Adj[\,]$ are equal to $|L_i|$. The number of elements of S in the former set does not exceed $|L_{i+1}|$, while the number of elements in the latter set does not exceed the sum of the degrees (in G) of the vertices in L_i. Thus, Step 3.3 takes $O(|L_{i+1}| + \sum_{u \in L_i} d_G(u))$ time.

Therefore, the time taken by the algorithm is

$$\begin{aligned} & O(n) + \sum_i \Big(O(1) + O\Big(|L_{i+1}| + \sum_{u \in L_i} d_G(u) \Big) \Big) \\ &= O(n) + O\Big(\sum_i \Big(1 + |L_{i+1}| \Big) \Big) + O\Big(\sum_i \sum_{u \in L_i} d_G(u) \Big) \\ &= O(n) + O(n) + O(m). \end{aligned}$$

The inequalities $\sum_i |L_i| \leq n$ and $\sum_i \sum_{u \in L_i} d_G(u) \leq \sum_u d_G(u) = 2m$ hold because each vertex belongs to at most one level of the BFS-tree. Moreover, the space needed is $O(n + m)$. Consequently, we have:

Theorem 1. *Let G be an undirected graph on n vertices and m edges, and v be a vertex of G. Then, the above algorithm computes the vertices in the levels of the BFS-tree of the complement $\overline{G}$ of G rooted at v in $O(n + m)$ time and $O(n + m)$ space.*

3.2 Partitioning the Vertices in L_2

It is not difficult to see that the partition of the vertices in L_2 depending on their neighbors in $\overline{G}$ which are in L_1 is identical to the partition of the vertices in L_2 depending on their neighbors in G which are in L_1. This is indeed so, because the subset of vertices in L_1 which are adjacent (in G) to a vertex $x \in L_2$ is $L_1 - N_x$, where N_x is the subset of L_1 containing vertices which are adjacent (in $\overline{G}$) to x. But then, if for two vertices x and y the sets N_x and N_y are equal then so do the sets $L_1 - N_x$ and $L_1 - N_y$, whereas if $N_x \neq N_y$ then $L_1 - N_x \neq L_1 - N_y$. Therefore, in the algorithm we will be working with neighbors in G instead of neighbors in $\overline{G}$.

The algorithm initially considers a single set (list) which contains all the vertices of the set L_2. It then processes each vertex, say, u, of the set L_1 as follows: For each set of the current partition, we check if none, all, or only some of its elements are neighbors of u in G; in the first and second case, the set is not modified, in the third case, it is split into the subset of neighbors of u in G and the subset of non-neighbors of u in G. After all the vertices of L_1 have been processed, the resulting partition is the desired partition.

Algorithm for partitioning the set L_2 in terms of adjacency to elements of the set L_1.

1. Initialize to 0 the entries of the arrays $M[\,]$ and $size[\,]$ which are of size n; insert all the vertices in L_2 in the list $LSet[1]$ and set $size[1] \leftarrow |L_2|$; $k \leftarrow 1$; {*k holds the number of sets*}
2. for each vertex u in L_1 do
   2.1 for each vertex w adjacent to u in G do
   $M[w] \leftarrow 1$; {*mark in $M[\,]$ the neighbors of u in G*}
   2.2 $k_0 \leftarrow k$;
   for each list $LSet[i]$, $i = 1, 2, \ldots, k_0$, do
   2.2.1 traverse the list $LSet[i]$ and count the number of its vertices which are neighbors of u in G (use the array $M[\,]$); let ℓ be the number of these vertices;
   2.2.2 if $\ell > 0$ and $\ell < size[i]$
   then {*split $LSet[i]$; create a new set*}
   increment k by 1;
   traverse the list $LSet[i]$ and for each of its vertices w which is a neighbor of u in G (use $M[\,]$), delete w from $LSet[i]$ and insert it in $LSet[k]$;
   $size[k] \leftarrow \ell$;
   decrease $size[i]$ by ℓ;
   2.3 for each vertex w adjacent to u in G do
   $M[w] \leftarrow 0$; {*clear $M[\,]$*}
3. for each list $LSet[i]$, $i = 1, 2, \ldots, k$, do
   traverse the list $LSet[i]$ and for each of its vertices set the corresponding entry of the array $Set[\,]$ equal to i;

Note that thanks to the array $Set[\,]$, checking whether two vertices x and y belong to the same partition set of L_2 reduces to testing whether the entries $Set[x]$ and $Set[y]$ are equal.

The correctness of the algorithm follows from induction on the number of the processed vertices in L_1. At the basis step, when no vertices from the set L_1 have been processed, all the elements of the set L_2 belong to the same set, as desired. Suppose that after processing $i \geq 0$ vertices from L_1, the resulting partition of L_2 is correct with respect to the processed vertices. Let us consider the processing of the next vertex, say, u, from L_1: then, only the sets which contain at least one vertex which is adjacent (in G) to u and at least one vertex which is not adjacent (in G) to u should be split, and indeed these are the only ones that are split; the splitting produces a subset of neighbors of u in G and a subset of non-neighbors of u (Step 2.2.2). Note that because of Steps 1, 2.1, and 2.3, the array $M[\,]$ is clear at the beginning of each iteration of the for loop in Step 2, so that in Step 2.2 the marked entries are precisely those corresponding to the neighbors of the current vertex u in G.

Step 1 of the algorithm clearly takes $O(n)$ time. Steps 2.1 and 2.3 take $O(d_G(u))$ time, where $d_G(u)$ is equal to the degree of u in G. Step 2.2.1 takes

$O(|LSet(i)|)$ time and so does Step 2.2.2, since deleting an entry from and inserting an entry in a list can be done in constant time, and the remaining operations take constant time. Therefore, Step 2.2 takes time linear in the total size of the lists $LSet[i]$ which existed when u started being processed; since the lists contained the vertices in L_2 and none of these lists was empty, we conclude that Step 2.2 takes $O(|L_2|)$ time. Then, Step 2 can be executed in $O(\sum_u (d_G(u) + |L_2|)) = O(m + n|L_2|)$ time. Step 3 takes time linear in the total size of the final lists $LSet[i]$, i.e., $O(|L_2|)$ time. Thus the entire partitioning algorithm takes $O(m + n\,|L_2|)$ time. Since all the initialized lists $LSet[i]$ contain at least one vertex from L_2 and since these lists do not share vertices, then the space complexity is $O(n + |L_2|) = O(n)$.

The results of the paragraph are summarized in the following theorem.

Theorem 2. *Let G be an undirected graph on n vertices and m edges, and let L_1 and L_2 be two disjoint sets of vertices. Then, the above algorithm partitions the vertices in L_2 depending on their neighbors in $\overline{G}$ which belong to L_1 in $O(m + n\,|L_2|)$ time and $O(n)$ space.*

Time and Space Complexity of the Recognition algorithm. Clearly, Step 1 of the algorithm takes $O(n)$ time. In accordance with Theorems 1 and 2, Steps 2.1 and 2.2 take $O(n+m) = O(m)$ and $O(m+n\,|L_2|)$ time respectively in the processing of each one of the vertices of G, while Steps 2.3.1 and 2.3.4 take $O(n)$ time. If we ignore the cost of unioning P_4-components, then Steps 2.3.2 and 2.3.3 require $O(1)$ time per vertex in L_3 and L_2 respectively; recall that testing whether two vertices belong to the same partition set of L_2 takes constant time. If we take into account Lemma 2, we have that $|L_2| \leq d_G(v)$ and $|L_3| \leq d_G(v)$ where $d_G(u)$ denotes the degree of vertex u in G. Therefore, if we ignore P_4-component unioning, the time complexity of Step 2 of the algorithm is

$$T_2 \;=\; \sum_v \Big(O\Big(m + n\, d_G(v)\Big) + \sum_x O\Big(d_G(v) + d_G(x)\Big)\Big).$$

If we observe that x belongs to L_2, we conclude that x assumes at most $d_G(v)$ different values. Thus,

$$\begin{aligned} T_2 &= O\Big(\sum_v m + n\sum_v d_G(v)\Big) + O\Big(\sum_v \sum_x \Big(d_G(v) + d_G(x)\Big)\Big) \\ &= O(nm) + O(nm) + O\Big(\sum_v \Big(d_G^2(v) \;+\; \sum_x d_G(x)\Big)\Big) \\ &= O(nm) + O\Big(\sum_v \; d_G^2(v)\Big) + O\Big(\sum_v \sum_x d_G(x)\Big) \\ &= O(nm) \end{aligned}$$

since $\sum_v d_G^2(v) \leq n \sum_v d_G(v) = O(nm)$ and $\sum_v \sum_x d_G(x) \leq \sum_v 2m = 2nm$. Now, the time required for all the P_4-component union operations during the

processing of all the vertices is $O(m \log m)$ [1]; there cannot be more than $m-1$ such operations (we start with m P_4-components and we may end up with only one), and each one of them takes time linear in the size of the smallest of the two components that are unioned.

Finally, constructing the directed graph from the edges associated with a non-trivial P_4-component and checking whether it is acyclic takes $O(n + m_i)$, where m_i is the number of edges of the component. Thus, the total time taken by Step 2 is $O\big(\sum_i (n+m_i)\big) = O(nm)$, since there are at most m P_4-components and $\sum_i m_i = m$. Thus, the overall time complexity is $O(n + nm + m \log m + nm) = O(nm)$; note that $\log m \leq 2 \log n = O(n)$.

The space complexity is linear in the size of the graph G: the array $M[\,]$ takes linear space, both Steps 2.1 and 2.2 require linear space (Theorems 1 and 2), the set L_1 is represented as a list of $O(n)$ size, the sets L_2 and L_3 are represented as lists having $O(d_G(v))$ size each, and the handling of the P_4-components requires one record per edge and one record per component. Thus, the space required is $O(n + m)$.

Therefore, we have proven the following result:

Theorem 3. *It can be decided whether a connected undirected graph on n vertices and m edges is a P_4-comparability graph in $O(nm)$ time and $O(n + m)$ space.*

3.3 The Case of Disconnected Input Graphs

If the input graph is disconnected, we compute its connected components and work on each one of them as indicated above. In light of Theorem 3 and since the connected components of a graph can be computed in time and space linear in the size of the graph by means of depth-first search [1], we conclude that the overall time complexity is $O(n + m) + \sum_i O(n_i\, m_i) = O(n \sum m_i) = O(nm)$ and the space is $O(n + m) + \sum_i O(n_i + m_i) = O(n + m)$ since $\sum_i n_i = n$ and $\sum_i m_i = m$.

Theorem 4. *It can be decided whether an undirected graph on n vertices and m edges is a P_4-comparability graph in $O(n\,m)$ time and $O(n + m)$ space.*

4 Concluding Remarks

In this paper, we presented an $O(n\,m)$-time and linear space algorithm to recognize whether a graph on n vertices and m edges is a P_4-comparability graph. The algorithm exhibits the currently best time and space complexity to the best of our knowledge, and is simple enough to be easily used in practice. Along with the work of [16], it leads to an $O(n\,m)$-time algorithm for computing an acyclic P_4-transitive orientation of a P_4-comparability graph, thus improving the upper bound on the time complexity for this problem as well. We also described a simple algorithm to compute the levels of the BFS-tree of the complement $\overline{G}$ of a graph G in time and space linear in the size of G.

The obvious open question is whether the P_4-comparability graphs can be recognized and/or oriented in $o(n\,m)$ time. Moreover, it is worth investigating whether taking advantage of properties of the complement of the input graph can help establish improved algorithmic solutions for other problems as well; note that breadth-first and depth-first search on the complement of a graph can be executed in time linear in the size of the graph.

References

1. A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.
2. S.R. Arikati and U.N. Peled, A polynomial algorithm for the parity path problem on perfectly orderable graphs, *Discrete Appl. Math.* **65**, 5–20, 1996.
3. A. Brandstädt, V.B. Lê, and J.P. Spinrad, *Graph Classes: A Survey*, Monographs on Discrete Mathematics and Applications 3, SIAM, 1999.
4. V. Chvátal, Perfectly ordered graphs, *Annals of Discrete Math.* **21**, 63–65, 1984.
5. E. Dahlhaus, J. Gustedt, and R.M. McConnell, Efficient and practical modular decomposition, *Proc. 8th ACM-SIAM Symp. on Discrete Algorithms (SODA'97)*, 26–35, 1997.
6. C.M.H. de Figueiredo, J. Gimbel, C.P. Mello, and J.L. Szwarcfiter, Even and odd pairs in comparability and in P_4-comparability graphs, *Discrete Appl. Math.* **91**, 293–297, 1999.
7. P.C. Gilmore and A.J. Hoffman, A characterization of comparability graphs and of interval graphs, *Canad. J. Math.* **16**, 539–548, 1964.
8. M.C. Golumbic, *Algorithmic graph theory and perfect graphs*, Academic Press, Inc., New York, 1980.
9. M.C. Golumbic, D. Rotem, and J. Urrutia, Comparability graphs and intersection graphs, *Discrete Math.* **43**, 37–46, 1983.
10. C.T. Hoàng, Efficient algorithms for minimum weighted colouring of some classes of perfect graphs, *Discrete Appl. Math.* **55**, 133–143, 1994.
11. C.T. Hoàng and B.A. Reed, Some classes of perfectly orderable graphs, *J. Graph Theory* **13**, 445–463, 1989.
12. C.T. Hoàng and B.A. Reed, P_4-comparability graphs, *Discrete Math.* **74**, 173–200, 1989.
13. H. Ito and M. Yokoyama, Linear time algorithms for graph search and connectivity determination on complement graphs, *Inform. Process. Letters* **66**, 209–213, 1998.
14. R.M. McConnell and J. Spinrad, Linear-time transitive orientation, *Proc. 8th ACM-SIAM Symp. on Discrete Algorithms (SODA'97)*, 19–25, 1997.
15. M. Middendorf and F. Pfeiffer, On the complexity of recognizing perfectly orderable graphs, *Discrete Math.* **80**, 327–333, 1990.
16. S.D. Nikolopoulos and L. Palios, Recognition and orientation algorithms for P_4-comparability graphs, *Proc. 12th Symp. on Algorithms and Computation (ISAAC'01)*, 320–331, 2001.
17. T. Raschle and K. Simon, On the P_4-components of graphs, *Discrete Appl. Math.* **100**, 215–235, 2000.

Bend-Minimum Orthogonal Drawings of Plane 3-Graphs*
(Extended Abstract)

Md. Saidur Rahman and Takao Nishizeki

Graduate School of Information Sciences, Tohoku University
Aoba-yama 05, Sendai 980-8579, Japan
saidur@nishizeki.ecei.tohoku.ac.jp, nishi@ecei.tohoku.ac.jp

Abstract. In an orthogonal drawing of a plane graph, any edge is drawn as a sequence of line segments, each having either a horizontal or a vertical direction. A bend is a point where an edge changes its direction. A drawing is called a bend-minimum orthogonal drawing if the number of bends is minimum among all orthogonal drawings. This paper presents a linear-time algorithm to find a bend-minimum orthogonal drawing of any given plane 3-graph, that is, a plane graph of maximum degree three.

Keywords: Graph, Algorithm, Graph Drawing, Orthogonal Drawing, Bend.

1 Introduction

In this paper we deal with "orthogonal drawings" of "plane 3-graphs." A *plane graph* is a planar graph with a fixed planar embedding. For an integer i, an i-*graph* is a graph of the maximum degree i. Figure 1(a) depicts a plane 3-graph. An *orthogonal drawing* of a plane graph G is a drawing of G with the given embedding such that each vertex is mapped to a point, each edge is drawn as a sequence of alternate horizontal and vertical line segments, and any two edges do not cross except at their common end. A point where an edge changes its direction in a drawing is called a *bend* of the drawing. Figure 1(d) depicts an orthogonal drawing of the graph in Fig. 1(a) with four bends, which are indicated by dotted circles in Fig. 1(d). Orthogonal drawings have attracted much attention due to their numerous practical applications in circuit schematics, data flow diagrams, entity relationship diagrams, etc. [DETT99, S84, T87]. A plane 3-graph often appears in practical applications like circuit schematics, and it is desired to find an orthogonal drawing with a small number of bends, because a bend corresponds to a "via" or "through-hole," and increases the fabrication cost of VLSI [L90, S84]. We hence wish to find a *bend-minimum* orthogonal drawing, that is, an orthogonal drawing with the minimum number of bends. The drawing in Fig. 1(d) is indeed a bend-minimum one since there is no drawing with at most three bends.

* This work is supported by JSPS grants.

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 367–378, 2002.

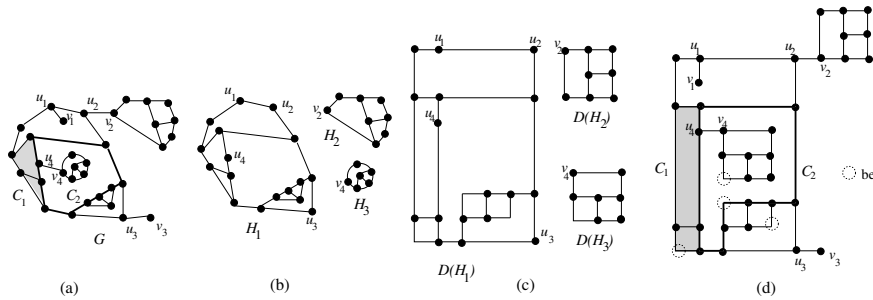


Fig. 1. (a) A plane 3-graph G, (b) three biconnected components H_1, H_2 and H_3, (c) orthogonal drawings of H_1, H_2 and H_3, and (d) a bend-minimum orthogonal drawing of G.

Garg and Tamassia [GT97] presented an algorithm to find a bend-minimum orthogonal drawing of any plane 4-graph G; if a plane graph has a vertex of degree 5 or more, then there is no orthogonal drawing. Their algorithm [GT97] takes time $O(n^{7/4}\sqrt{\log n})$ if G has n vertices. More efficient algorithms are known for restricted classes of plane 3-graphs. Rahman, Nakano and Nishizeki [RNN99] gave a linear-time algorithm for triconnected cubic plane graphs, and Nakano and Yoshikawa [NY01] extended the algorithm to a linear-time algorithm for biconnected cubic plane graphs. In a cubic graph, every vertex has degree 3. Hence, any cubic graph is a 3-graph. On the other hand, Rahman, Naznin and Nishizeki [RNN02] obtained a necessary and sufficient condition for a plane 3-graph to have an orthogonal drawing without bends, and gave a linear-time algorithm to find such a drawing if it exists.

In this paper we give a linear-time algorithm to find a bend-minimum orthogonal drawing of any plane 3-graph. The complexity of our algorithm is better than the complexity $O(n^{7/4}\sqrt{\log n})$ of the algorithm in [GT97] and is optimal within a constant factor although the algorithm in [GT97] works for plane 4-graphs. The class of plane 3-graphs is larger than the classes of triconnected or biconnected cubic plane graphs for the linear algorithms in [RNN99, NY01]. Our algorithm is also a generalization of the algorithm in [RNN02].

The outline of our algorithm is as follows. We first decompose a given plane 3-graph G to biconnected components as illustrated in Figs. 1(a) and (b). We then find "optimal" orthogonal drawings of all components as illustrated in Fig. 1(c). (An "optimal" drawing of a component is an orthogonal drawing of the component which satisfies a trivial necessary condition for an orthogonal drawing of G and uses the minimum number of bends, as defined in Section 2.) We finally combine the drawings of all components to a bend-minimum orthogonal drawing of G as illustrated in Fig. 1(d). The key idea is to reduce the problem of finding an "optimal" orthogonal drawing of a biconnected component to the problem of finding an orthogonal drawing without bends by inserting "dummy" vertices of degree 2 to appropriate edges.

The rest of the paper is organized as follows. Section 2 gives some definitions. Section 3 presents our main result on bend-minimum orthogonal drawings of

plane 3-graphs. Section 4 presents an algorithm to find an "optimal" orthogonal drawing of "bad cycles" in a biconnected component of a plane 3-graph G. Finally Section 5 is a conclusion.

2 Preliminaries

In this section we give some definitions.

Let $G = (V, E)$ be a connected simple graph of vertex set V and edge set E. Let n be the number of vertices in G. The *degree* of a vertex v is the number of neighbors of v in G. A vertex of degree 2 in G is called a 2-*vertex* of G. If every vertex of G has degree three, then G is called a *cubic graph*. The *connectivity* $\kappa(G)$ of a graph G is the minimum number of vertices whose removal results in a disconnected graph or a single-vertex graph K_1. We say that G is k-*connected* if $\kappa(G) \geq k$. We call a vertex of G a *cut vertex* in G if its removal results in a disconnected graph.

We call a maximal biconnected subgraph of G a *biconnected component* of G. We call an edge (u, v) a *bridge* of G if the deletion of (u, v) results in a disconnected graph. Any graph can be decomposed to biconnected components and bridges. The graph G in Fig. 1(a) has three biconnected components H_1, H_2 and H_3 depicted in Fig. 1(b), and has four bridges (u_1, v_1), (u_2, v_2), (u_3, v_3) and (u_4, v_4).

A graph is *planar* if it can be embedded in the plane so that no two edges intersect geometrically except at a vertex to which they are both incident. A *plane* graph is a planar graph with a fixed embedding. A plane graph divides the plane into connected regions called *faces*. We regard the *contour* of a face as a cycle formed by the edges on the boundary of the face. We denote the contour of the outer face of G by $C_o(G)$, or simply by C_o if there is no confusion.

For a simple cycle C in a plane graph G, we denote by $G(C)$ the plane subgraph of G inside C (including C). We say that cycles C and C' in a plane graph G are *independent* if $G(C)$ and $G(C')$ have no common vertex. An edge of G which is incident to exactly one vertex of a cycle C and located outside C is called a *leg* of the cycle C. The vertex of C to which a leg is incident is called a *leg-vertex* of C. Every leg-vertex in a plane 3-graph has degree 3. A cycle C in G is called a k-*legged cycle* of G if C has exactly k legs in G and there is no edge which joins two vertices on C and is located outside C. In Figs. 1(a) and 1(d) a 4-legged cycle C_1 is shaded, and a 9-legged cycle C_2 is drawn by thick lines.

Each cycle C in a plane graph G is drawn as a rectilinear polygon in any orthogonal drawing D of G. The polygon is denoted by $D(C)$. A (polygonal) vertex of the rectilinear polygon is called a *corner of the drawing* $D(C)$. Thus a corner is either a vertex of G or a bend of D, and has an interior angle 90° or 270°. A corner of an interior angle 90° is called a *convex corner* of $D(C)$, while a corner of an interior angle 270° is called a *concave corner*. A vertex v on C is called a *non-corner* of $D(C)$ if v is not a corner of $D(C)$. Thus any vertex on C is a convex corner, a concave corner, or a non-corner of $D(C)$.

Let C be a cycle in G, and let v be a vertex on C which is a cut vertex in G. We call v an *outcut vertex* for C in G if v is a leg-vertex of C in G, otherwise we

call v an *incut vertex* for C in G. Any outcut vertex for C is not a concave corner of $D(C)$ in any orthogonal drawing D of G; otherwise, the leg of C could not be drawn as a horizontal or vertical line segment. Similarly, any incut vertex for C is not a convex corner. Thus any orthogonal drawing D of G has the following property.

(p1) If v is an outcut vertex for a cycle C then v is either a convex corner or a non-corner of $D(C)$, and if v is an incut vertex for a cycle C then v is either a concave corner or a non-corner of $D(C)$.

Thus (p1) is a trivial necessary condition for an orthogonal drawing $D(G)$ of G. For any subgraph H of G, the drawing $D(H)$ of H in $D(G)$ has Property (p1). In the plane graph G in Fig. 1(a), vertices u_2 and u_3 are outcut vertices for the outer cycle $C_o(H_1)$ of a biconnected component H_1, and u_1 is an incut vertex for $C_o(H_1)$. Vertex u_4 is an outcut vertex for the shaded cycle C_1, but is an incut vertex for the cycle C_2 drawn by thick lines. The orthogonal drawing $D(H_1)$ of H_1 in Fig. 1(c) has Property (p1), because outcut vertices u_2 and u_3 for a cycle are convex corners of the drawing of the cycle, incut vertex u_1 for a cycle is a non-corner of the drawing of the cycle, and incut or outcut vertex u_4 for a cycle is a non-corner of the drawing of the cycle.

An orthogonal drawing $D(H)$ of a biconnected subgraph H is called an *optimal* one if $D(H)$ has Property (p1) and has the minimum number $b(H)$ of bends among all orthogonal drawings of H with Property (p1).

3 Bend-Minimum Orthogonal Drawing

In this section we present our main result on the bend-minimum orthogonal drawing of a plane 3-graph.

Our idea is to decompose the plane graph G into biconnected components and bridges as illustrated in Figs. 1(a) and (b), and then find an optimal orthogonal drawing $D(H)$ of each biconnected component H as illustrated in Fig. 1(c). A bridge is drawn as either a horizontal or a vertical line segment. Finally we obtain a bend-minimum orthogonal drawing of G by combining the drawings of all biconnected components and bridges without introducing new bends, as illustrated in Fig. 1(d). The difficulty is that we cannot directly use the algorithm for biconnected cubic plane graphs in [NY01] to find an optimal drawing of a biconnected component, because a biconnected component is not always cubic and a drawing obtained by the algorithm in [NY01] does not always have Property (p1). We reduce the problem of finding an optimal drawing of a biconnected component to the problem of finding an orthogonal drawing of the component without bends by inserting "dummy" vertices of degree 2 to appropriate edges.

Rahman *et al.* [RNN02] have obtained a necessary and sufficient condition for a biconnected plane subgraph H of a plane 3-graph G to have an orthogonal drawing which is optimal and has has no bend, i.e., $b(H) = 0$. We now present the condition. Let $C_o(H)$ contain four or more 2-vertices of H, and let x, y, z and w be any four of them which clockwise appear on $C_o(H)$ in this order. Then

an orthogonal drawing $D(H)$ of H is called an *optimal no-bend drawing* for x, y, z and w if $D(H)$ satisfies the following three conditions (i), (ii) and (iii).

(i) $D(H)$ has Property (p1);
(ii) $D(H)$ has no bend, and hence $b(H) = 0$; and
(iii) $D(H)$ intersects none of the four quadrants, the first quadrant with the origin at x, the fourth quadrant with the origin at y, the third quadrant with the origin at z, and the second quadrant with the origin at w, after rotating the drawing if necessary.

Figure 2 illustrates an optimal no-bend drawing, where the four quadrants are shaded. Clearly x, y, z, and w must be convex corners of the drawing $D(C_o(H))$ of the outer cycle $C_o(H)$ of H, and the drawing $D(H)$ should intersect neither the horizontal open halfline with left end at x nor the vertical halfline with the lower end at x, and so on for y, z, and w. Vertices x, y, z and w may be leg-vertices in G, and then the leg will be drawn as a straight line-segment on an open halfline with an end on them.

Let H be a biconnected plane subgraph of G, and let C be a cycle in H. We say that a vertex v on C is *good* for C if v is a 2-vertex of H and is not an incut vertex for C in G. Only a good vertex for C can be drawn as a convex corner of the rectilinear polygon $D(C)$ in an optimal orthogonal drawing $D(H)$.

We are now ready to present the condition in [RNN02].

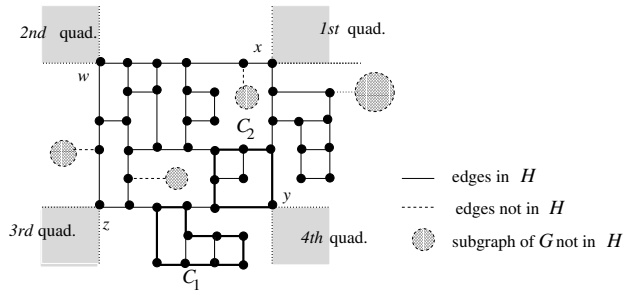


Fig. 2. An optimal no-bend drawing of H for x, y, z and w.

Lemma 1. *Let G be a connected plane 3-graph, let H be a biconnected plane subgraph of G, and let x, y, z and w be 2-vertices of H which clockwise appear on $C_o(H)$ in this order. Then H has an optimal no-bend drawing for x, y, z and w if and only if the following (a), (b) and (c) hold:*

(a) all the vertices x, y, z and w are good for the cycle $C_o(H)$ in G;
(b) there are at least two good vertices for every 2-legged cycle C in H; and
(c) there are at least one good vertex for every 3-legged cycle C in H.

Furthermore the drawing above can be found in linear time.

In this paper we call the linear algorithm in [RNN02] for finding an optimal no-bend drawing of a biconnected subgraph H of G **No-Bend-Draw**.

We call a cycle C in H a *bad cycle* if C does not satisfy (b) or (c) in Lemma 1. Thus a bad cycle has one of the following three types depending on the numbers of legs and good vertices.

(t1) A 2-legged cycle C for which there is no good vertex.
(t2) A 2-legged cycle C for which there is exactly one good vertex.
(t3) A 3-legged cycle C for which there is no good vertex.

The two leg-vertices divide a bad cycle C of Type (t1) into two paths; we call them the *contour paths* of C, and call C a *2-path cycle*. The two leg-vertices and the good vertex divide a cycle C of Type (t2) into three paths. Similarly the three leg-vertices divide a cycle C of Type (t3) into three paths. We call a cycle C of Type (t2) and (t3) a *3-path cycle*, and call each of the three paths on C a *contour path*.

For a bad cycle C in H, we now define a set $Des(C)$ of bad cycles and the hierarchical structure of bad cycles in $Des(C)$ as follows. We call a cycle C' a *descendant bad cycle* of C if $C \neq C'$, C' is a bad cycle, and C' is contained in the subgraph $H(C)$ of H inside C. There are two cases to consider.

Case 1: C is a 2-path cycle.

In this case, we choose any of the two leg-vertices of C as a reference vertex r for C, and let $Des(C)$ be the set of all descendant bad cycles of C not containing r. A cycle $C' \in Des(C)$ is called a *child-cycle* of C (with respect to r) if C' is not located inside any other bad cycle in $Des(C)$.

Case 2: C is a 3-path cycle.

In this case, let $Des(C)$ be the set of all descendant bad cycles of C. A cycle $C' \in Des(C)$ is called a *child-cycle* of C if C' is not located inside of any other bad cycle in $Des(C)$.

We now have the following lemma, a proof of which is omitted in this extended abstract.

Lemma 2. *If C is a bad cycle in H, then the child-cycles of C are independent of each other.*

A bad cycle C in H is defined to be a *maximal bad cycle of H* if C is not contained in $H(C')$ for any other bad cycle C' in H. We can prove that all the maximal bad cycles of H are independent of each other; the proof is similar to that of Lemma 2 and is omitted in this extended abstract. We regard all the maximal bad cycles as the child-cycles of the outer cycle $C_o(H)$ of H. We find child-cycles of each maximal bad cycle and then find child-cycles of each child-cycle recursively, and eventually we get a (hierarchical) tree structure of bad cycles in H. The structure is represented by a "genealogical tree" T_H; the root of T_H is $C_o(H)$, the children of root $C_o(H)$ are the maximal bad cycles of H, and so on. Because of the choices of a reference vertex r, T_H may have some variations. However, we choose an arbitrary (but) fixed one as T_H. Figure 3(b) illustrates T_H for H in Fig. 3(a). Based on the genealogical tree T_H we will give

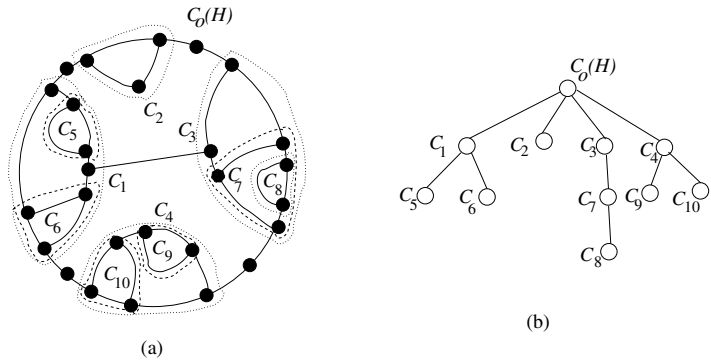


Fig. 3. (a) A biconnected component H, and (b) a genealogical tree T_H.

later in Section 4 **Algorithm Bad-Cycle-Draw** to find an optimal orthogonal drawings of $H(C)$ for a bad cycle C in T_H.

Using **No-Bend-Draw** and **Bad-Cycle-Draw**, we now give Algorithm **Bi-Comp-Draw** for finding an optimal orthogonal drawing $D(H)$ of a biconnected component H of G, that is, an orthogonal drawing $D(H)$ with Property (p1) and the minimum number $b(H)$ of bends. We assume for simplicity that there are at least four good vertices for $C_o(H)$; otherwise we shall carefully insert dummy vertices of degree 2 into edges on $C_o(H)$ as good vertices for $C_o(H)$. (Note that these dummy vertices will appear as bends in an optimal orthogonal drawing of H.) Then H satisfies (a) in Lemma 1 if one regards, as x, y, z and w, any four good vertices for $C_o(H)$.

Algorithm Bi-Comp-Draw(H)
begin
1 Select any four good vertices for $C_o(H)$ as x, y, z and w.
2 **if** H satisfies (b) and (c) in Lemma 1 **then**
3 find an optimal no-bend drawing of H by **No-Bend-Draw**
4 **else**
begin
5 Find the maximal bad cycles $C_1, \cdots, C_l$ of $C_o(H)$;
6 For each i, $1 \leq i \leq l$, contract cycle C_i to a single vertex v_i;
7 Let H^* be the resulting graph; (H^* satisfies (a)–(c) in Lemma 2.1.)
8 Find an optimal no-bend orthogonal drawing of H^*
by **No-Bend-Draw**;
9 Find an optimal orthogonal drawing of each $H(C_1), \cdots, H(C_l)$
by **Bad-Cycle-Draw**;
10 Patch the drawings $D(H(C_1)), \cdots, D(H(C_l))$ into $D(H^*)$;
11 The resulting drawing $D(H)$ is an optimal orthogonal drawing of H.
end
end.

Figure 4 illustrates Algorithm **Bi-Comp-Draw**. The biconnected component H in Fig. 4(a) has exactly one maximal bad cycle C drawn by thick line

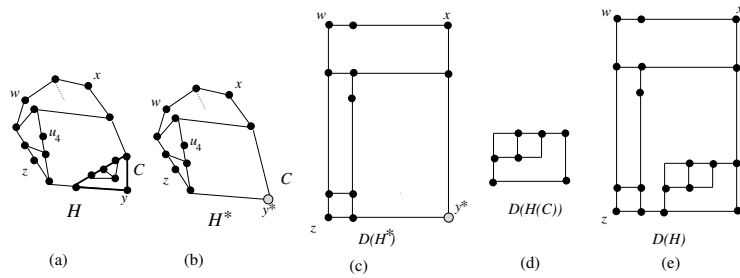


Fig. 4. Illustration for Algorithm **Bi-Comp-Draw**.

segments. The graph H^* in Fig. 4(b) is obtained from H by contracting $H(C)$ to a single vertex y^*. Figure 4(c) illustrates an optimal no-bend drawing $D(H^*)$ of H^* obtained by **No-Bend-Draw**. The drawing in Fig. 4(d) is an optimal orthogonal drawing $D(H(C))$ of the subgraph $H(C)$ obtained by **Bad-Cycle-Draw**. Figure 4(e) illustrates an orthogonal drawing $D(H)$ of H obtained by patching $D(H(C))$ into $D(H^*)$.

We now have following lemma on **Bi-Comp-Draw**.

Lemma 3. *Let G be a connected plane 3-graph, and let H be a biconnected component of G. Then Algorithm* **Bi-Com-Draw** *finds an optimal orthogonal drawing of H in linear time.*

We finally give **Algorithm Bend-Min-Draw** to find a bend-minimum orthogonal drawing of a connected plane 3-graph G, as follows. First, decompose G into biconnected components and bridges. Then find an optimal orthogonal drawing $D(H)$ of each biconnected component H of G by Algorithm **Bi-Comp-Draw**. Each bridge is drawn as either a horizontal or vertical line segment. Then construct a "block-cut-vertex tree" of G, and combine the drawings of all components without introducing new bends to a bend-minimum orthogonal drawing of G. We now have the following theorem, which is the main result of this paper.

Theorem 1. *Algorithm* **Bend-Min-Draw** *finds a bend-minimum orthogonal drawing of a connected plane 3-graph G in linear time.*

4 Optimal Orthogonal Drawings of Bad Cycles

In this section we give an algorithm **Bad-Cycle-Draw** to find an orthogonal drawing of $H(C)$ for a bad cycle C in a biconnected component H of G.

We have the following two facts for any orthogonal drawing of H with Property (p1).

Fact 4 *At least two bends must appear on a 2-path cycle in H.*

Fact 5 *At least one bend must appear on a 3-path cycle in H.*

Using Facts 4 and 5, in Section 4.1 we determine the number of bends which are necessary for an optimal orthogonal drawing of a subgraph $H(C)$ for a bad cycle C. In Section 4.2 we give an algorithm **Bad-Cycle-Draw** for finding an optimal orthogonal drawing of $H(C)$.

4.1 Classification and Assignment

We classify the contour paths of each bad cycle C in a genealogical tree T_H as a *blue path* or a *green path* or a *red path* in a recursive manner based on T_H. In addition, we assign an integer $bc(C)$, called the *bend count* of C, to each bad cycle C in T_H. We will show later that $H(C)$ has an optimal orthogonal drawing and that $b(H(C)) = bc(C)$. Furthermore we will show that

(i) for any blue path P of C, $H(C)$ has an optimal orthogonal drawing with two or more bends on P,

(ii) for any green path P of C, $H(C)$ has an optimal orthogonal drawing with one or more bends on P,

(iii) for any red path P of C, $H(C)$ has an optimal orthogonal drawing with no bend on P, and

(iv) C contains a blue path or a green path.

Let C be a bad cycle in T_H, and let $C_1, C_2, \cdots, C_l$ be the child-cycles of C. Assume that we have already defined the classification and the assignment for all child-cycles of C and are going to define them for C. We have the following four cases.

Case 1: C has no child-cycle.

If C is a 2-path cycle, then we classify the two contour paths of C as blue paths and set $bc(C) = 2$. If C is a 3-path cycle, then we classify the three contour paths of C as green paths and set $bc(C) = 1$.

Case 2: C is a 2-path cycle, and all child-cycles of C are 2-legged cycles.

Let x and y be the two leg-vertices of C, and let P_1 and P_2 be the clockwise contour paths from x to y and from y to x, respectively. We may assume that x is chosen as the reference vertex r.

Subcase 2a: any child-cycle of C has neither a green path on C nor a blue path on C.

In this case we classify both the contour paths of C as blue paths and set $bc(C) = 2 + \sum_{i=1}^{l} bc(C_i)$.

Subcase 2b: C has a child-cycle which has a blue path on C.

In this case we set $bc(C) = \sum_{i=1}^{l} bc(C_i)$. We classify the two contour paths P_1 and P_2 of C as follows. We classify P_i as a blue path if either P_i contains a blue path of a child cycle of C or P_i contains green paths of two or more child-cycles. We classify P_i as a green path if P_i contains a green path of exactly one child-cycle and does not contain a blue path of any child-cycle. We classify P_i as a red path if P_i contains none of the blue paths and the green paths of all child-cycles of C.

Subcase 2c: C has no child-cycle which has a blue path on C, but has a child-cycle which has a green path on C.

If P_1 contains green paths of two or more child-cycles, then we classify P_1 as a blue path and set $bc(C) = \sum_{i=1}^{l} bc(C_i)$. In such a case we classify P_2 as a blue path if it contains green paths of two or more child-cycles, as a green path if it contains a green path of exactly one child-cycle, and as a red path if it contains neither a green path nor a blue path of a child-cycle.

If P_2 contains green paths of two or more child cycles, then we classify the contour paths P_1 and P_2 similarly as above and set $bc(C) = \sum_{i=1}^{l} bc(C_i)$.

If each of P_1 and P_2 contains a green path of exactly one child cycle, then we classify both of P_1 and P_2 as green paths and set $bc(C) = \sum_{i=1}^{l} bc(C_i)$.

Otherwise, exactly one of the two contour paths P_1 and P_2 contains a green path of exactly one child-cycle. We classify it as a blue path and classify the other contour path as a green path. In this case we set $bc(C) = 1 + \sum_{i=1}^{l} bc(C_i)$.

Case 3: C is a 2-path cycle, and has a 3-legged child-cycle.

Omitted in this extended abstract.

Case 4: C is a 3-path cycle and has one or more child-cycles.

Omitted in this extended abstract.

Using a method similar to one in [RNN99], the classification and assignments above can be done in linear time. We can prove the following lemma by induction.

Lemma 6. *For any cycle C in T_H, $H(C)$ has a set $\mathcal{S}$ of vertex disjoint cycles consisting of l_2 2-path cycles and l_3 3-path cycles such that $bc(C) = 2l_2 + l_3$.*

We have the following lemma immediately from Facts 4, 5 and Lemma 6.

Lemma 7. *Any cycle C in T_H satisfies $b(H(C)) \geq bc(C)$.*

Conversely proving $b(H(C)) \leq bc(C)$, we have $b(H(C)) = bc(C)$ for any cycle C in T_H. Indeed we will prove a stronger claim later in Lemmas 8 and 9.

4.2 Bad-Cycle-Draw

Let C be a 2-path cycle in T_H, and let x and y be the two leg vertices of C. Since C is in T_H, C has a blue path or a green path. For a blue path P of C, an optimal orthogonal drawing of $H(C)$ is defined to be *2-bend optimal for P* if at least two bends appear on the blue path P and the drawing satisfies the condition (c1) given below. For a green path P of C, an optimal orthogonal drawing of $H(C)$ is defined to be *1-bend optimal for P* if at least one bend appears on the green path P and the drawing satisfies the condition (c2) given below.

(c1) The drawing $D(H(C))$ intersects neither the second quadrant with the origin at x nor the third quadrant with the origin at y, after rotating the drawing and renaming the leg-vertices if necessary. See Fig. 5(a). Note that C is not always drawn by a rectangle.

(c2) The drawing $D(H(C))$ intersects neither the first quadrant with the origin at x nor the third quadrant with the origin at y, after rotating the drawing and renaming the leg-vertices if necessary. See Fig. 5(b). Note that C is not always drawn by a rectangle.

Let C be a 3-path cycle in T_H, and let x, y and z be the three leg-vertices of C if C is a 3-legged cycle, otherwise let x and y be the two leg-vertices and let z be the non-incut vertex of degree 2. (We may change the name of vertices x, y, z if necessary.) One may assume that x, y and z appear clockwise on C.

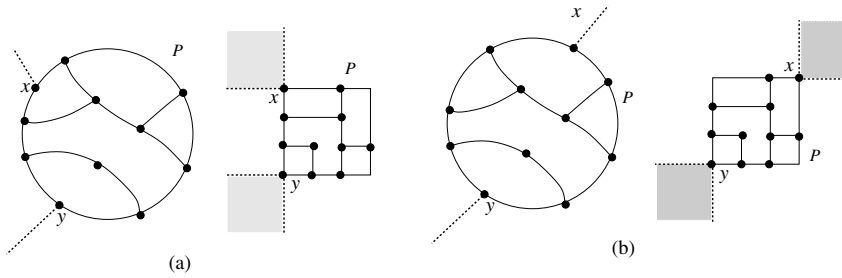


Fig. 5. (a) A 2-bend optimal drawing of a 2-path cycle, and (b) a 1-bend optimal drawing of a 2-path cycle.

Since C is in T_H, C has a blue or green path. For a blue path P on C with ends x and y, an optimal orthogonal drawing of $H(C)$ is defined to be *2-bend optimal for* P if at least two bends appear on the blue path P and the drawing satisfies the condition (c3) given below. For a green path P on C with ends x and y, an optimal orthogonal drawing of $H(C)$ is defined to be *1-bend optimal for* P if at least one bend appears on the green path P and the drawing satisfies the condition (c3) given below.

(c3) The drawing $D(H(C))$ intersects none of the following three quadrants: the first quadrant with origin at x, the third quadrant with origin at y, and the second quadrant with origin at z, after rotating the drawing and renaming the leg-vertices if necessary.

An orthogonal drawing of $H(C)$ for a bad cycle C is simply called an *optimal drawing* if it is 1-bend optimal or 2-bend optimal for a green or blue path. We now have the following lemma.

Lemma 8. *Let G be a connected plane 3-graph, and let H be a biconnected component of G. If a cycle C in T_H has a blue path P, then $H(C)$ has an orthogonal drawing which is 2-bend optimal for P.*

Proof. We give a recursive algorithm to find an orthogonal drawing of $H(C)$ which is 2-bend optimal for P. Our algorithm inserts dummy vertices on some edges of a subgraph of H and uses the algorithm **No-Bend-Draw** in each recursive step. The algorithm is similar to one in [RNN99, RNN02]. We omit the detail in this extended abstract. $\mathcal{Q}.\mathcal{E}.\mathcal{D}.$

Lemma 9. *Let G be a connected plane 3-graph, and let H be a biconnected component of G. If a cycle C in T_H has a green path P, then $H(C)$ has an orthogonal drawing which is 1-bend optimal for P.*
Proof. We give a recursive algorithm to find a 1-bend optimal drawing of $H(C)$ for the green path P. The algorithm is similar to that in the proof of Lemma 8. The details are omitted in this extended abstract. $\mathcal{Q}.\mathcal{E}.\mathcal{D}.$

The algorithm for finding an optimal orthogonal drawing of $H(C)$ described in the proofs of Lemmas 8 and 9 above is hereafter called **Bad-Cycle-Draw**. We now have the following lemma on **Bad-Cycle-Draw**.

Lemma 10. *Let G be a connected plane 3-graph and let H be a biconnected component of G. Then Algorithm* **Bad-Cycle-Draw** *finds an optimal orthogonal drawing of $H(C)$ for any bad cycle C in T_H in linear time.*

Proof. The proof is similar to the proof of Lemma 13 in [RNN99]. $\mathcal{Q.E.D.}$

5 Conclusions

In this paper we have presented a linear-time algorithm to find an orthogonal drawing of any given plane 3-graph with the minimum number of bends. It is remaining as a future work to find a linear-time algorithm for a plane 4-graph.

Our linear algorithm works for a plane 3-graph, that is, a planar 3-graph with a fixed planar embedding. Di Battista *et al.* [DLV98] gave an algorithm to find a bend-minimum orthogonal drawing of a planar 3-graph G in time $O(n^5 \log n)$, where the planar embedding of G is not fixed. In their algorithm they used a "min-cost flow" algorithm of time complexity $O(n^2 \log n)$ to find a bend-minimum drawing of a plane 3-graph. Using our linear algorithm, the time complexity of the algorithm in [DLV98] may be improved.

References

DETT99. G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall Inc., Upper Saddle River, New Jersey, 1999.

DLV98. G. Di Battista, G. Liotta, and F. Vargiu, *Spirality and optimal orthogonal drawings*, SIAM J. Comput. 27(6), pp. 1764-1811, 1998.

GT97. A. Garg and R. Tamassia, *A new minimum cost flow algorithm with applications to graph drawing*, Proc. of Graph Drawing'96, Lect. Notes in Computer Science, 1190, pp. 201-226, 1997.

L90. T. Lengauer *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons, Chichester, 1990.

NY01. S. Nakano and M. Yoshikawa, *A linear-time algorithm for bend-optimal orthogonal drawings of biconnected cubic plane graphs*, Proc. of Graph Drawing'00, Lect. Notes in Computer Science, 1984, pp. 296-307, 2001.

RNN99. M. S. Rahman, S. Nakano and T. Nishizeki, *A linear algorithm for bend-optimal orthogonal drawings of triconnected cubic plane graphs*, Journal of Graph Alg. and Appl., http://www.cs.brown.edu/publications/jgaa/, 3(4), pp. 31-62, 1999.

RNN02. M. S. Rahman, M. Naznin and T. Nishizeki, *Orthogonal drawings of plane graphs without bends*, Proc. of Graph Drawing'01, Lect. Notes in Computer Science, 2265, pp. 392-406, 2002.

S84. J. Storer, *On minimum node-cost planar embeddings*, Networks, 14, pp. 181-212, 1984.

T87. R. Tamassia, *On embedding a graph in the grid with the minimum number of bends*, SIAM J. Comput., 16, pp. 421-444, 1987.

Cluster Graph Modification Problems

Ron Shamir*, Roded Sharan**, and Dekel Tsur

School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel
{rshamir,roded,dekelts}@tau.ac.il

Abstract. In a clustering problem one has to partition a set of elements into homogeneous and well-separated subsets. From a graph theoretic point of view, a cluster graph is a vertex-disjoint union of cliques. The clustering problem is the task of making fewest changes to the edge set of an input graph so that it becomes a cluster graph. We study the complexity of three variants of the problem. In the Cluster Completion variant edges can only be added. In Cluster Deletion, edges can only be deleted. In Cluster Editing, both edge additions and edge deletions are allowed. We also study these variants when the desired solution must contain a prespecified number of clusters.
We show that Cluster Editing is NP-complete, Cluster Deletion is NP-hard to approximate to within some constant factor, and Cluster Completion is polynomial. When the desired solution must contain exactly p clusters, we show that Cluster Editing is NP-complete for every $p \geq 2$; Cluster Deletion is polynomial for $p = 2$ but NP-complete for $p > 2$; and Cluster Completion is polynomial for any p. We also give a constant factor approximation algorithm for Cluster Editing when $p = 2$.

1 Introduction

Problem Definition and Motivation: Clustering is a central optimization problem with applications in numerous fields including computational biology (cf. [15]), image processing (cf. [16]), VLSI design (cf. [7]), and many more. The input to the problem is typically a set of elements and pairwise similarity values between elements. The goal is to partition the elements into subsets, which are called *clusters*, so that two meta-criteria are satisfied: *Homogeneity* - elements inside a cluster are highly similar to each other; and *separation* - elements from different clusters have low similarity to each other. Concrete realizations of these criteria generate a variety of combinatorial optimization problems [8].

In the basic graph theoretic approach to clustering, one builds from the raw data a *similarity graph* whose vertices correspond to elements and there is an edge between two vertices if and only if the similarity of their corresponding elements exceeds a predefined threshold [9,8]. Ideally, the resulting graph would be a *cluster graph*, that is, a graph composed of vertex-disjoint cliques. In practice, it is only close to being such, since similarity data is experimental and, therefore, error-prone.

* Supported in part by the Israel Science Foundation (grant number 565/99).
** Supported by an Eshkol fellowship from the Ministry of Science, Israel.

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 379–390, 2002.

Following [2] we formalize the resulting problem as the task of changing (adding or deleting) fewest edges of an input graph so as to obtain a cluster graph. We call this problem *Cluster Editing*. In the related *Cluster Deletion* problem (respectively, *Cluster Completion*) one has to remove (respectively, add) fewest edges from an input graph so that it becomes a cluster graph. Completion (deletion) problems arise when the data contains only false negative (positive) errors. The above problems belong to the class of *edge modification problems* (cf. [13]), in which one has to minimally change the edge set of a graph so as to satisfy a certain property. Another variant of these problems arises when the solution is also required to consist of a prespecified number of clusters. This variant is motivated by many real-life applications in which a partition of elements into a known number of categories is desired (see, e.g., [6, 1]).

Previous Results: Edge modification problems were studied extensively in [13] where earlier studies are also reviewed. Most of these problems were shown to be NP-complete. Polynomial algorithms were given for bounded degree input graphs. In particular, a constant factor approximation algorithm was given for editing and deletion problems with respect to any property that can be characterized by a finite set of forbidden induced subgraphs. Since a graph is a cluster graph if and only if it is P_2-free (i.e., it does not contain an induced path of two edges), this result implies a $3d$-approximation algorithm for Cluster Editing and Cluster Deletion on input graphs with degree bounded by d.

The Cluster Editing problem was first studied by Ben-Dor et al. [2], who presented a polynomial algorithm that solves the problem with high probability under a stochastic data model. The complexity of the problem was left open. Cluster Deletion was shown to be NP-complete by Natanzon [12].

Contribution of This Paper: We prove that Cluster Editing is NP-complete, Cluster Deletion is NP-hard to approximate to within some constant factor, and Cluster Completion is polynomial. We also study the p-Cluster versions of these problems, in which the required graph must also be a vertex-disjoint union of p cliques. We show that p-Cluster Editing is NP-complete for every $p \geq 2$; p-Cluster Deletion is polynomial for $p = 2$ but NP-complete for $p > 2$; and p-Cluster Completion is polynomial for any p. We also give a 0.878-approximation algorithm for a weighted variant of 2-Cluster Editing.

For lack of space some proofs are only sketched or omitted.

2 Preliminaries

All graphs in this paper are simple, i.e., contain no parallel edges or self-loops. Let $G = (V, E)$ be a graph. We denote its set of edges by $E(G)$. For two disjoint subsets $A, B \subseteq V$, we denote by $E_{A,B}$ ($\overline{E}_{A,B}$) the set of all edges (non-edges) with one endpoint in A and the other in B. The *complement graph* of G is $\overline{G} = (V, \{(u, v) \in (V \times V) \setminus E : u \neq v\})$. See [3] for more definitions of graphs and hypergraphs.

A graph $G = (V, E)$ is called a *cluster graph* if every connected component of G is a complete graph. G is called a *p-cluster graph* if it is a cluster graph

with p connected components or, equivalently, if it is a vertex-disjoint union of p cliques. If G is any graph and $F \subset V \times V$ is such that $G' = (V, E \triangle F)$ is a cluster graph, then F is called a *cluster editing set* for G ($E \triangle F$ denotes the symmetric difference between E and F, i.e., $(E \setminus F) \cup (F \setminus E)$). If in addition $F \subseteq E$, then F is called a *cluster deletion set* for G. If $F \cap E = \phi$ then F is called a *cluster completion set* for G. *p-cluster editing set*, *p-cluster deletion set*, and *p-cluster completion set* are similarly defined. We denote by $P(F)$ the partition of V into disjoint subsets of vertices according to the connected components (cliques) of G'. For a partition $P = (V_1, \ldots, V_l)$ of V, we denote by N_P the size of the cluster editing set implied by P, that is,

$$N_P \equiv |\bigcup_{i=1}^{l} \{(u,v) \notin E : u, v \in V_i\} \cup \{(u,v) \in E : u \in V_i, v \in V_j, i \neq j\}| \ .$$

The problems we study in this paper are of two types:

Problem 1 (Cluster Editing/Completion/Deletion). Given a graph G and an integer k, determine if G has a cluster editing/completion/deletion set of size at most k.

Problem 2 (p-Cluster Editing/Completion/Deletion). Given a graph G and an integer k, determine if G has a p-cluster editing/completion/deletion set of size at most k.

3 Cluster Editing

We prove in this section that Cluster Editing is NP-complete by reduction from a restriction of exact cover by 3-sets:

Problem 3 (3-Exact 3-Cover (3X3C)). Given a collection C of triplets of elements from a set $U = \{u_1, \ldots, u_{3n}\}$, such that each element of U is a member of at most 3 triplets, determine if there is a sub-collection $I \subseteq C$ of size n which covers U.

The 3X3C problem is known to be NP-complete [4, Problem SP2].

Theorem 1. *Cluster Editing is NP-complete.*

Proof. Membership in NP is trivial. We prove NP-hardness by reduction from 3X3C. Let $m \equiv 30n$. Given an instance $\langle C, U \rangle$ of 3X3C we build a graph $G = (V, E)$ as follows:

$$\begin{aligned}
V &= \bigcup_{S \in C} \{v_{S,1}, \ldots, v_{S,m}\} \cup U \ , \\
E &= E_1 \cup E_2 \cup E_3 \ , \\
E_1 &= \{(v_{S,i}, u) : S \in C, 1 \leq i \leq m, u \in S\} \ , \\
E_2 &= \{(v_{S,i}, v_{S,j}) : S \in C, 1 \leq i < j \leq m\} \ , \\
E_3 &= \{(u, u') : \exists S \in C \text{ s.t. } u, u' \in S\} \ .
\end{aligned}$$

In words, we build a clique of size $m+3$ around each triplet S by fully connecting S and m additional vertices. For each triplet $S \in C$ we denote $V_S = \{v_{S,1}, \ldots, v_{S,m}\}$ and call the elements of V_S, *S-vertices*. Let $q = \sum_{S \in C} |S| = 3|C|$. Define $N \equiv m(q-3n)$ and $M \equiv |E_3| - 3n$. We prove that there is an exact cover of U iff there is a cluster editing set for G of size at most $N+M$:

($\Rightarrow$) Suppose that $I \subseteq C$ is an exact cover of U. Let $F_1 = \{(v_{S,i}, u) : S \notin I, 1 \leq i \leq m, u \in S\}$ and let $F_2 = \{(u,u') \in E_3 : \nexists S \in I \text{ s.t. } u,u' \in S\}$. It is easy to verify that $F = F_1 \cup F_2$ is a cluster editing set for G, whose size is $|F| = |F_1| + |F_2| = N + M$.

($\Leftarrow$) Let F be an editing set of G' of minimum size, and $|F| \leq N+M$. We shall prove that $|F| = N+M$ and one can derive from F an exact cover of U. Since each element of U occurs in at most 3 triplets, $q \leq 9n$. Thus, $|E_3| \leq q \leq 9n$ and $|F| \leq N+M \leq 6mn + 6n = 180n^2 + 6n < \frac{m}{2}(\frac{m}{2} - 2)$.

Let $G' = (V, E \triangle F)$ be the cluster graph obtained by editing G according to F. We shall prove that for every subset $S \in C$ there is a unique clique in G' which contains V_S. To this end, we first show that there is a clique K_S in G' such that $|K_S \cap V_S| \geq m/2 + 3$: Suppose that the vertices of V_S are partitioned among k cliques $X_1, \ldots, X_k$ in G'. Let $s(X_i) = |V_S \cap X_i|, i = 1, \ldots, k$. Suppose to the contrary that $s(X_i) \leq m/2 + 2$ for all i. Therefore,

$$|F| \geq \frac{1}{2}\sum_{i=1}^{k} s(X_i)(m - s(X_i)) \geq \frac{1}{2}\sum_{i=1}^{k} s(X_i)(\frac{m}{2} - 2) = \frac{m}{2}(\frac{m}{2} - 2) \ .$$

A contradiction follows.

Let K_S be the clique X_i for which $s(X_i)$ is maximum ($|K_S \cap V_S| \geq m/2+3$). We next prove that $V_S \subseteq K_S \subseteq V_S \cup S$. Let $x = |K_S \setminus (V_S \cup S)|$. Consider a new partition P' of V, which is obtained from $P(F)$ by splitting K_S into $K_S \cap (V_S \cup S)$ and $K_S \setminus (V_S \cup S)$. Clearly, $N_{P(F)} - N_{P'} \geq (m/2+3)x - 3x = xm/2$. But F is an optimum cluster editing set. Therefore, $x = 0$ and $K_S \subseteq V_S \cup S$. To see that $K_S \supseteq V_S$, suppose to the contrary that there is some index $1 \leq i \leq m$ such that $v_{S,i} \notin K_S$. Let K' be the clique in G' which contains $v_{S,i}$. Let P'' be a new partition of V, which is obtained from $P(F)$ by moving $v_{S,i}$ from K' to K_S. Then $N_{P(F)} - N_{P''} \geq m/2 + 3 - (m/2 - 4 + 3) = 4$, a contradiction. We conclude that for every $S \in C$ there is a unique clique in G' which contains V_S and is contained in $V_S \cup S$.

Let $F_1 = F \cap E_1$. Examine an element $u \in U$ which is a member of (at least) two subsets $S_1, S_2 \in C$. By the previous claim, V_{S_1} and V_{S_2} are subsets of distinct cliques in G'. Hence, either $E_{V_{S_1},\{u\}} \subseteq F$, or $E_{V_{S_2},\{u\}} \subseteq F$ (or both). Therefore, $|F_1| \geq N$. Moreover, since $|F_1| \leq N+M$ and $M \leq 6n$, each vertex $u \in U$ must be adjacent in G' to the S-vertices of exactly one set S where $u \in S$. Call this set the *S-set* of u.

Let $F_2 = F \setminus F_1$. For every two vertices $u, u' \in U$ such that $(u,u') \in E$, and the S-sets of u and u' differ, we must have $(u,u') \in F_2$. Since each subset in C contains 3 elements, every $u \in U$ has at most 2 neighbors in U. Therefore, $|F_2| \geq M$, with equality iff there is a partition of U into triplets of elements, such that the S-set of the elements in each triplet is the same. Since $|F| \leq N+M$,

we must have $|F| = N + M$, and the implied partition into triplets induces an exact cover of U. □

3.1 p-Cluster Editing

In this section we study the p-Cluster Editing problem. We first show that 2-Cluster Editing is NP-complete. We then conclude that p-Cluster Editing is NP-complete for every $p \geq 2$.

To prove the hardness of 2-Cluster Editing, we define the following problem:

Problem 4 (3-Uniform Hypergraph Balanced 2-Colorability). Given a 3-uniform hypergraph G, determine if there is a 2-coloring of G such that the number of vertices that are colored by each color is the same.

This problem can be shown to be NP-complete by a trivial reduction from Hypergraph 2-Colorability on 3-uniform hypergraphs, whose NP-hardness was proven by Lovasz [11].

Theorem 2. 2-*Cluster Editing is NP-complete.*

Proof. Membership in NP is trivial. We reduce from 3-Uniform Hypergraph Balanced 2-Colorability. Given a hypergraph $G = (V, E)$, we build an instance of 2-Cluster Editing $\langle G' = (V', E'), k\rangle$ as follows: Let n and m be the number of vertices and hyperedges, respectively, in G, and assume that $V = \{1, \ldots, n\}$. Let $M \equiv 2n^3$. We define $V' = \cup_{i=1}^{n} V_i$ where $V_i = \{v_{i,j} : j = 1, \ldots, M\}$. For a triplet of indices $1 \leq i < j < l \leq n$ define the set $E_{i,j,l} = \{(v_{i,r}, v_{j,r}), (v_{j,r+1}, v_{l,r}), (v_{l,r+1}, v_{i,r+1})\}$, where $r = 2(n^2 i + nj + l) - 1$. We add edges to G' by building a clique around each V_i, and for every triplet of indices $i < j < l$ such that $(i, j, l) \notin E$, we add the edges of $E_{i,j,l}$. Finally, we set $k \equiv 2\binom{n/2}{2}(M^2 - (n - 2)) + (\frac{n}{2})^2(n - 2) - m$.

The sets $V_1, \ldots, V_n$ will be called *clusters*. We say that a partition $(S, V' \setminus S)$ *splits* a cluster V_i if $V_i \cap S \neq \phi$ and $V_i \not\subseteq S$. For convenience we also define a graph G'' which is built like G' except that it contains the edges $E_{i,j,l}$ for every triplet $i < j < l$. We now prove the correctness of this reduction, namely that there is a balanced 2-coloring of G iff there is a 2-cluster editing set of G' of size at most k.

($\Rightarrow$) Suppose that $f: V \to \{0, 1\}$ is a balanced 2-coloring of G. Let $S = \cup_{i:f(i)=0} V_i$, and let F', F'' be the 2-cluster editing sets of G' and G'', respectively, that correspond to the partition $(S, V \setminus S)$. Since f is balanced, each side of the partition $(S, V \setminus S)$ consists of $n/2$ clusters. We first compute the size of F''. For two clusters V_i and V_j (with $i < j$), and for each $l \neq i, j$, one of $E_{i,j,l}$, $E_{i,l,j}$, or $E_{l,i,j}$ contains exactly one edge between V_i and V_j. Therefore, there are exactly $n - 2$ edges between every pair of clusters in G''. It follows that F'' consists of $2\binom{n/2}{2}(M^2 - (n - 2))$ edges that are not in E between clusters in the same set in the partition $(S, V \setminus S)$, and $(\frac{n}{2})^2(n - 2)$ edges in E between clusters in different sets in the partition. Thus, $|F''| = 2\binom{n/2}{2}(M^2 - (n - 2)) + (\frac{n}{2})^2(n - 2)$.

We now compute the size of F'. For each edge $(i,j,l) \in E$, the edges of $E_{i,j,l}$ in G'' contribute two edges to F'' (as the clusters V_i, V_j, and V_k are not all in the same set in the partition), while the non-existence of the edges of $E_{i,j,l}$ in G' contributes only one edge to F' (between the two clusters on the same side). It follows that $|F'| = |F''| - m = k$.

($\Leftarrow$) Suppose that F is a 2-cluster editing set of G' of minimum size, and $|F| \leq k$. Let $P(F) = (S, V' \setminus S)$. We first claim that $(S, V' \setminus S)$ splits no cluster. Suppose conversely that $(S, V' \setminus S)$ splits at least one cluster.

If $(S, V' \setminus S)$ splits more than one cluster, then let V_i be a cluster that is split by the partition such that $|V_i \cap S|$ is minimum, and let V_j be a cluster that is split by the partition such that $|V_j \cap S|$ is maximum and $j \neq i$. Denote $a = |V_i \cap S|$ and $b = |V_j \cap S|$. Select some vertex $u \in V_i \cap S$ and a vertex $w \in V_j \setminus S$. Let $S' = S \cup \{w\} \setminus \{u\}$, and let F' be the 2-cluster editing set that corresponds to $(S', V' \setminus S')$. We will show that $|F| - |F'| \geq 0$. Note that a vertex $v \in V_i$ has at most one neighbor outside V_i. If such a neighbor exists, denote it by n_v. The number of edges in F that are incident on u is at least $(M-a) + (|S| - a - 1)$ (the term -1 is due to the possibility that n_u exists and $n_u \in S$) and the number of edges in F that are incident on w is is at least $b + (nM - |S| - (M-b) - 1)$ (the term -1 is due to the possibility that n_w exists and $n_w \in V' \setminus S$). The total number of edges of these two kinds is $nM - 2a + 2b - 2$. Similarly, the number of edges in F' that are incident on u or w is at most $a + (nM - |S| - (M-a) - 1) + (M-b) + (|S| - b - 1) = nM + 2a - 2b - 2$. It follows that

$$|F| - |F'| \geq (nM - 2a + 2b - 2) - (nM + 2a - 2b - 2) = 4(b-a) \geq 0.$$

If $a < b$, we have that $|F'| < |F|$, a contradiction to the minimality of F. In the case when $a = b$, namely the value of $|V_l \cap S|$ is equal amongst all the clusters, we have that $|F'| = |F|$. We build a set S'' from S' using the same process as above, and since $|V_l \cap S'|$ is not equal amongst the clusters, it follows that the 2-cluster editing set F'' that corresponds to S'' satisfies $|F''| < |F'| = |F|$, and again we arrive at a contradiction.

Now suppose that the partition $(S, V' \setminus S)$ splits exactly one cluster, and denote this cluster by V_i. Let $a = |V_i \cap S|$. Out of the rest $n-1$ clusters, suppose that r clusters are contained in S, and $n-r-1$ clusters are contained in $V' \setminus S$. W.l.o.g. suppose that $n-r-1 \leq r$, and since n is even we have $n-r-1 \leq r-1$. Define $S' = S \setminus V_i$, and let F' be the corresponding 2-cluster editing set. For each $v \in V_i \cap S$, there are at least $rM - 1$ edges in F between v and $S \setminus V_i$, and $M - a$ edges between v and $V_i \setminus S$, so the number of edges in F that are incident on v is at least $rM - 1 + M - a$. On the other hand, an edge in F' that is incident on v is either between v and n_v, or between v and $(V' \setminus S) \setminus V_i$. The number of edges of the latter type is $(n-1-r)M$, so the number of edges in F that are incident on v is at most $(n-1-r)M + 1 \leq (r-1)M + 1$. It follows that

$$|F| - |F'| \geq a\,(rM - 1 + M - a - ((r-1)M + 1)) = a\,(2M - a - 2) > 0,$$

contradicting the minimality of F. Therefore, F splits no cluster.

We now claim that the number of clusters that are contained in S is exactly $n/2$. Conversely, suppose w.l.o.g. that $r > n/2$. Let V_i be some cluster contained in S. Let $S' = S \setminus V_i$ and let F' be the corresponding 2-cluster editing set. Similarly to the above, we have that

$$|F| - |F'| \geq M((r-1)M - 1 - ((n-r)M + 1)) \geq M(M-2) > 0,$$

a contradiction. Hence, S contain $n/2$ clusters.

Define a coloring $f: V \to \{0,1\}$ by $f(i) = 0$ iff $V_i \subseteq S$. By the argument above, f is balanced. It remains to show that f is a legal 2-coloring. For a hyperedge $(i,j,k) \in E$, if i,j,k have the same color then $|F \cap E_{i,j,l}| = 3$. Otherwise, $|F \cap E_{i,j,l}| = 1$ since two of the edges in $E_{i,j,l}$ must connect vertices in clusters on different sides of the partition $(S, V' \setminus S)$. Hence, each monochromatic hyperedge adds two to the size of F. By the first direction of the proof, for a legal 2-coloring, the corresponding editing set is of size exactly k, and thus no monochromatic hyperedge is possible in f. It follows that f is a balanced 2-coloring of G. □

Corollary 1. *p-Cluster Editing is NP-complete for any $p \geq 2$.*

Proof. Fix $p > 2$. We provide a reduction from 2-Cluster Editing. Given an input instance $\langle G = (V,E), k\rangle$ of 2-Cluster Editing, $|V| = n$, we form an instance $\langle G' = (V', E'), k\rangle$ of p-Cluster Editing as follows: Define $V' = V \cup \cup_{i=1}^{p-2} V_i$ where $V_i = \{w_{i,j} : j = 1, \ldots, n^2\}$. The edges of G' include all the edges in E and a clique on each V_i.

Clearly, every 2-cluster editing set of G is a p-cluster editing set of G' (of the same size). Conversely, suppose that F' is a p-cluster editing set of G' of size at most k, and let $(S_1, \ldots, S_p)$ be the corresponding partition. We show that F' is also a 2-cluster editing set for G.

If there is a set V_i such that $V_i \cap S_j \neq \phi$ and $V_i \not\subseteq S_j$ for some j, then F' contains $E_{V_i \cap S_j, V_i \setminus S_j}$. The number of such edges is at least $n^2 - 1 > k$, a contradiction. Therefore, every set V_i is contained in some set S_j. Furthermore, every set S_j contains at most one set V_i since otherwise we have $|F'| \geq n^4 > k$, a contradiction. It follows that all edges in F' are incident on vertices of V, which implies that F' is a 2-cluster editing set of G. □

3.2 A 0.878-Approximation Algorithm

We give in this section a polynomial approximation algorithm for a weighted variant of 2-Cluster Editing which is defined as follows:

Problem 5 (Weighted 2-Cluster Editing). Given a graph G and a weight function on vertex pairs $w: E(G) \cup E(\overline{G}) \to \mathcal{N}$, find in G a 2-cluster editing set with maximum total weight of unedited vertex pairs.

Note, that the decision version of Weighted 2-Cluster Editing reduces to that of 2-Cluster Editing when $w \equiv 1$ (i.e., $w(e) = 1$ for every $e \in E(G) \cup E(\overline{G})$).

Let $n = |V|$ and let S_n denote the n-dimensional unit sphere. We define the following semi-definite relaxation of Weighted 2-Cluster Editing:

$$\max \frac{1}{2}[\sum_{(i,j)\in E}(w_{ij}(1+v_i\cdot v_j)) + \sum_{(i,j)\notin E}(w_{ij}(1-v_i\cdot v_j))]$$
$$\text{s.t. } v_i \in S_n \ \forall i$$

We claim that this is indeed a relaxation of Weighted 2-Cluster Editing, that is, for every partition $P = (A, B)$ of G there exist vectors $v_1, \ldots, v_n \in S_n$ such that the total weight of unedited vertex pairs as implied by P is $\frac{1}{2}[\sum_{(i,j)\in E}(w_{ij}(1+v_i\cdot v_j)) + \sum_{(i,j)\notin E}(w_{ij}(1-v_i\cdot v_j))]$. Indeed, let (A, B) be a partition of G. Let v_0 be any unit vector in S_n. For every $i \in A$ set $v_i = v_0$, and for every $i \in B$ set $v_i = -v_0$. The claim follows.

Our approximation algorithm solves this semi-definite relaxation and then rounds the solution obtained using the random hyperplane technique [5].

Theorem 3. *The algorithm approximates Weighted 2-Cluster Editing with an expected approximation ratio of at least* 0.878.

Proof. Follows directly from [5, Theorem 6.1]. □

4 Cluster Completion

The Cluster Completion problem is trivially polynomial: The optimum solution is obtained by simply transforming each connected component of the input graph into a complete graph. In this section we give a polynomial algorithm for p-Cluster Completion, for any fixed $p \geq 2$.

Let $G = (V, E)$ be an input graph with n vertices and t connected components. If $t < p$ we output *False*. We assume henceforth that $t \geq p$. To find the optimum completion set we compute partitions of the t components of G into p sets (splitting no connected components) and choose the partition which results in a minimum completion set. Using dynamic programming, we only need to consider a polynomial number of partitions. Note that since we only add edges, we seek to minimize the sum of the number of edges in each of the p sets of the partition, or equivalently, the sum of the squared sizes of the sets.

Let $C_1, \ldots, C_t$ be the cardinalities of the connected components in G. Our algorithm will denote each possible partition by a $(p-1)$-long vector of integers which describes the sizes of the sets in the partition (the size of the last set is the difference from n). We will maintain a set S_i of the vectors that correspond to all possible partitions of the first i connected components. The algorithm is given in Figure 1. The actual partition can be obtained by maintaining for each $v \in S_i$ a pointer to its parent vector in S_{i-1}.

Theorem 4. *The algorithm correctly solves the p-Cluster Completion problem in $O(tn^{p-1})$ time.*

$S_0 = \{(0, \ldots, 0)\}$
For $i = 1$ to t **do:**
$S_i = S_{i-1} \cup \{v + C_i e_j : v \in S_{i-1}, j = 1, \ldots, p-1\}$
Pick in S_t a vector v^* minimizing $\sum_{i=1}^{p-1} v_i^2 + (n - \sum_{i=1}^{p-1} v_i)^2$.

Fig. 1. An algorithm for p-Cluster Completion. e_j denotes a $(p-1)$-dimensional unit vector with 1 in position j.

5 Cluster Deletion

We now focus on the cluster deletion problem. We shall give a gap preserving reduction (cf. [10]) from a restricted version of SET-COVER to Cluster Deletion. This reduction implies that there is some constant $\epsilon > 0$ such that it is NP-hard to approximate Cluster Deletion to within a factor of $1 + \epsilon$. We begin by introducing the SET-COVER restriction.

Problem 6 (Minimum Restricted Exact Cover (REC)). The input is a set of elements $U = \{u_1, \ldots, u_t\}$, and a collection C of subsets of U which satisfies the following conditions:

- There is a constant $k_1 > 0$ such that for each $S \in C$, $|S| \leq k_1$.
- There is a constant $k_2 > 0$ such that for all $u \in U$, $|\{S \in C : u \in S\}| \leq k_2$.
- If $S \in C$ and $S' \subset S$ then $S' \in C$.

The goal is to find a sub-collection $I \subseteq C$ of minimum cardinality, such that $\bigcup_{S \in I} S = U$, and the sets in I are pairwise-disjoint.

Note, that the third condition guarantees that a solution to REC always exists (we assume that $\bigcup_{S \in C} S = U$). REC can be shown to be MAX-SNP complete by a simple L-reduction from a restriction of SET-COVER in which the size of every set is bounded and each element occurs in a bounded number of sets. This latter problem is known to be MAX-SNP complete [14]. Hence, there is a constant $\delta_{REC} > 0$ such that it is NP-hard to approximate REC to within a factor of $1 + \delta_{REC}$.

Theorem 5. *There is some constant $\epsilon > 0$ such that it is NP-hard to approximate Cluster Deletion to within a factor of $1 + \epsilon$.*

Proof. By a gap preserving reduction from REC (similar to the one in Theorem 1). For an instance I_{REC} of REC, the reduction produces in polynomial time an instance I_{CD} of Cluster Deletion such that $opt(I_{REC}) \leq c$ implies $opt(I_{CD}) \leq c'$ and $opt(I_{REC}) > (1 + \delta_{REC})c$ implies $opt(I_{CD}) > (1 + \epsilon)c'$, where $opt(I)$ denotes the optimal value for instance I.

We now describe the reduction. Let $I_{REC} = \langle U, C \rangle$, and let $|U| = t$. Suppose that each set in C has size at most k_1, and each element occurs in at most k_2 sets. Let $m = \frac{k_1^2 k_2}{\delta_{REC}}$ and let $q = \sum_{S \in C} |S|$. We build an instance $I_{CD} = \langle G = (V, E) \rangle$ of Cluster Deletion as follows:

$$
\begin{aligned}
V &= \bigcup_{S \in C} \{v_{S,1}, \ldots, v_{S,m}, w_S\} \cup U \ , \\
E &= E_1 \cup E_2 \cup E_3 \cup E_4 \ , \\
E_1 &= \{(v_{S,i}, u) : S \in C, 1 \leq i \leq m, u \in S\} \ , \\
E_2 &= \{(v_{S,i}, v_{S,j}) : S \in C, 1 \leq i < j \leq m\} \ , \\
E_3 &= \{(u, u') : \exists S \in C \text{ s.t. } u, u' \in S\} \ , \\
E_4 &= \{(v_{S,i}, w_S) : S \in C, 1 \leq i \leq m\} \ .
\end{aligned}
$$

In words, for each $S \in C$ we form a clique on S and a set of m new vertices, and also connect all the new vertices to a single extra vertex w_S. For each subset $S \in C$ we denote $V_S = \{v_{S,1}, \ldots, v_{S,m}\}$ and call the elements of V_S, S-vertices. Note, that $|E_3| \leq (k_1 - 1)k_2 t/2 < k_1 k_2 t/2$ and $q \leq k_2 t$. Clearly, $t/k_1 \leq opt(I_{REC}) \leq t$. Let c be any constant such that $t/k_1 \leq c \leq t$. Define $c' \equiv (q - t + c)m + |E_3|$ and $\epsilon \equiv \frac{\delta_{REC}}{2k_1 k_2 + \delta_{REC}}$. We prove that this reduction is gap preserving:

($\Rightarrow$) Suppose that $opt(I_{REC}) \leq c$. Let $I \subseteq C$ be an exact cover of U, $|I| \leq c$. For $u \in U$ denote by I_u the set in I which contains u. Let $\bar{I} = C \setminus I$.

To obtain a cluster subgraph G' of G we delete the following edges:

1. For all $S \in \bar{I}, u \in S$ delete all the edges in $E_{V_S,\{u\}}$.
2. For all $S \in I$ delete all the edges in $E_{V_S,\{w_S\}}$.
3. For all $u \in U, u' \in U \setminus I_u$ delete the edge (u, u') if it exists.

One can easily verify that G' is a cluster graph, and therefore, $opt(I_{CD}) \leq (q - t + c)m + |E_3| = c'$.

($\Leftarrow$) Suppose that $opt(I_{REC}) > (1 + \delta_{REC})c$. In any cluster subgraph of G, every $u \in U$ is adjacent to the S-vertices of at most one set $S \in C$. Therefore, $opt(I_{CD}) \geq (q - t)m$. Furthermore, there is an optimum solution F of I_{CD} for which the following is true: If a vertex $u \in U$ is adjacent to an S-vertex in $(V, E \setminus F)$, for some $S \in C$, then F contains all the edges in $E_{V_S,\{w_S\}}$. Indeed, if F' is a cluster deletion set such that $u_1, \ldots, u_r$ $(1 \leq r \leq k_1)$ are adjacent to an S-vertex in $(V, E \setminus F')$, then $F'' = (F' \cup E_{V_S,\{w_S\}}) \setminus (\bigcup_{i=1}^{r} E_{V_S,\{u_i\}} \cup \{v_{S,i}, v_{S,j} : i \neq j\})$ is also such a cluster deletion set, and $|F''| \leq |F'|$. Examine now F. For each $u \in U$, either F contains all edges connecting u to vertices in $V \setminus U$, or there is a single set $S \in C$ such that $E_{V_S,\{u\}} \cap F = \phi$ and $E_{V_S,\{w_S\}} \subseteq F$. Let k be the number of vertices $u \in U$ for which the latter case applies, and let $\mathcal{T}$ be the collection of all sets S such that $(v_{S,i}, u) \in E \setminus F$ for some $u \in U, i$. It follows that $|F| \geq (q - k + |\mathcal{T}|)m$. Since $|\mathcal{T}| \geq opt(I_{REC}) + (k - t)$ for every choice of F, we have $opt(I_{CD}) \geq (q - t + opt(I_{REC}))m > (q - t + (1 + \delta_{REC})c)m$. We conclude that

$$
\begin{aligned}
opt(I_{CD}) &> (q - t + (1 + \delta_{REC})c)m = c' + (\delta_{REC} cm - |E_3|) \\
&> c'(1 + \frac{\delta_{REC} cm - |E_3|}{qm + |E_3|}) > c'(1 + \frac{\delta_{REC}(t/k_1)m - k_1 k_2 t/2}{k_2 tm + k_1 k_2 t/2}) \\
&= c'(1 + \frac{2\delta_{REC} m/k_1 - k_1 k_2}{2k_2 m + k_1 k_2}) = c'(1 + \frac{\delta_{REC}}{2k_1 k_2 + \delta_{REC}}) = c'(1 + \epsilon) \ . \square
\end{aligned}
$$

Let $C_1, \ldots, C_t$ be the connected components of $\overline{G}$.
For $i = 1, \ldots, t$ **do:**
    If C_i is not bipartite **then** output *False* and halt.
    Else find a bipartition (A_i, B_i) of C_i such that $|A_i| \geq |B_i|$.
Output the set that corresponds to $(A_1 \cup \ldots \cup A_t, B_1 \cup \ldots \cup B_t)$.

Fig. 2. An algorithm for 2-Cluster Deletion.

5.1 p-Cluster Deletion

We give in this section a polynomial algorithm for the optimization version of 2-Cluster Deletion. We then show that p-Cluster Deletion is NP-complete for every $p > 2$.

Let $G = (V, E)$ be an input graph with n vertices. W.l.o.g., G is connected, as otherwise, either G is already a 2-cluster graph, or we output *False*. The algorithm is described in Figure 2. Recall that $\overline{G}$ is the complement of G.

Theorem 6. *The algorithm correctly solves 2-Cluster Deletion in* $O(n+|E(\overline{G})|)$ *time.*

Proof. Since the complement of a 2-cluster graph is a complete bipartite graph, a solution exists if and only if $\overline{G}$ is bipartite. Hence, the algorithm outputs *False* iff no solution exists. Moreover, the partition produced by the algorithm has the property that if two vertices are assigned to the same set then they are adjacent. Therefore, the set of edges F returned by the algorithm is a 2-deletion set of G. Hence, it suffices to prove that F is optimal.

Denote $S_1 = A_1 \cup \ldots \cup A_t$ and $S_2 = B_1 \cup \ldots \cup B_t$. Clearly, F consists of edges in G with one endpoint in S_1 and the other in S_2. Therefore,

$$|F| = |E_{S_1,S_2}| = |S_1||S_2| - E(\overline{G}) = |S_1|(n - |S_1|) - E(\overline{G}).$$

Let F^* be an optimal 2-deletion set of G, and let $P(F^*) = (S_1^*, S_2^*)$, where $|S_1^*| \leq |S_2^*|$. We have that $|F^*| = |S_1^*|(n - |S_1^*|) - E(\overline{G})$. For every $i \leq t$, either $A_i \subseteq S_1^*$ or $B_i \subseteq S_1^*$ and, therefore, $|S_1| \leq |S_1^*| \leq n/2$. It follows that $|F| \leq |F^*|$. Hence, F is an optimal 2-deletion set of G.

The bottleneck in the complexity of the algorithm is computing the connected components of $\overline{G}$ and finding a bipartition for each of them. Each of these tasks can be performed in $O(n + |E(\overline{G})|)$ time. □

Theorem 7. *p-Cluster Deletion is NP-complete for any $p \geq 3$.*

Proof. Membership in NP is trivial. We provide a reduction from p-Coloring. Given an input graph $G = (V, E)$, the reduction outputs its complement $\overline{G}$ and a bound $k = |\overline{E}|$. A p-coloring f of G trivially translates into a p-deletion set $\{(u, v) \notin E : f(u) \neq f(v)\}$ of $\overline{G}$ of size at most k. Conversely, suppose that F is a p-deletion set of $\overline{G}$ with $|F| \leq k$, and let $C_1, \ldots, C_p$ be the cliques of $(V, \overline{E} \setminus F)$. The coloring f defined by $f(v) = i$ for all $v \in C_i$ is a p-coloring of G. □

Note that the reduction works with any $k \geq |\overline{E}|$ and in fact shows that even deciding whether a graph has a p-cluster deletion set is NP-hard, for $p \geq 3$.

References

1. A. A. Alizadeh, M. B. Eisen, et al. Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling. *Nature*, 403(6769):503–511, 2000.
2. A. Ben-Dor, R. Shamir, and Z. Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3/4):281–297, 1999.
3. C. Berge. *Graphs and Hypergraphs*. North-Holland, Amsterdam, 1973.
4. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., San Francisco, 1979.
5. M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.
6. T. R. Golub, D. K. Slonim, et al. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286:531–537, October 1999.
7. C. Hagen and A.B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(9):1074–1085, 1992.
8. P. Hansen and B. Jaumard. Cluster analysis and mathematical programming. *Mathematical Programming*, 79:191–215, 1997.
9. J.A. Hartigan. *Clustering Algorithms*. John Wiley and Sons, 1975.
10. D. S. Hochbaum, editor. *Approximation Alogrithms for NP-Hard Problems*. PWS Publishing, Boston, 1997.
11. L. Lovasz. Covering and coloring of hypergraphs. In *Proc. 4th Southeastern Conf. on Combinatorics, Graph Theory, and Computing*. Utilitas Mathematica Publishing, 1973.
12. A. Natanzon. Complexity and approximation of some graph modification problems. Master's thesis, Department of Computer Science, Tel Aviv University, 1999.
13. A. Natanzon, R. Shamir, and R. Sharan. Complexity classification of some edge modification problems. *Discrete Applied Mathematics*, 113:109–128, 2001.
14. C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. of Computer and System Science*, 43:425–440, 1991.
15. R. Sharan and R. Shamir. CLICK: A clustering algorithm with applications to gene expression analysis. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 307–316, 2000.
16. Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113, 1993.

Two Counterexamples in Graph Drawing

O. Sýkora[1,*], L.A. Székely[2,**], and I. Vrťo[3,***]

[1] Department of Computer Science, Loughborough University
Loughborough, Leicestershire LE11 3TU, The United Kingdom
[2] Department of Mathematics, University of South Carolina
Columbia, SC 29208, USA
[3] Department of Informatics, Institute of Mathematics, Slovak Academy of Sciences
Dúbravská 9, 842 35 Bratislava, Slovak Republic

Abstract. We provide counterexamples to two conjectures known in the field of graph drawing. The first conjecture (made by J. Halton ten years ago) asserts that the thickness of any graph of maximum degree Δ is at most $\lceil(\Delta+2)/4\rceil$. We give an existence proof that there are graphs of the thickness $\lceil\Delta/2\rceil$—this is known to be the best possible upper bound. The second conjecture (made by F. Shahrokhi recently) proposes a relation between the crossing number of a graph and the optimal linear arrangement of that graph. We construct a graph which does not satisfy this relation.

1 Halton's Conjecture

1.1 Introduction

The *thickness* of a graph G, denoted by $\Theta(G)$, is the minimum number of planar graphs, whose union is G. Thickness is one of the classical and standard measures of non-planarity of graphs. The problem of finding the thickness of a graph can be thought of as drawing of the graph on the smallest possible number of planes without edge crossings. Multilayer embedding appears naturally in applications, like printed circuit design [11] and multilayer VLSI layouts [1]. The thickness has been determined exactly only for: complete graphs, (almost all) complete bipartite graphs, hypercubes, and graphs of orientable genus 1 and 2. See Figure 1 with an example of a drawing of $K_{6,7}$ in two planes.

Computing the thickness is NP-hard [8]. For further results on thickness, see the survey papers [2,10]. Wessel [14] proved that

$$\left\lceil\frac{\delta+1}{6}\right\rceil \le \Theta(G) \le \left\lceil\frac{\Delta}{2}\right\rceil, \tag{1}$$

* This research was supported in part by the EPSRC grant GR/R37395/01.
** This research was supported in part by the NSF contract 007 2187.
*** This research was partially supported by the DFG grant No. Hr14/5-1 and the VEGA grant No. 02/7007/20.

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 391–398, 2002.

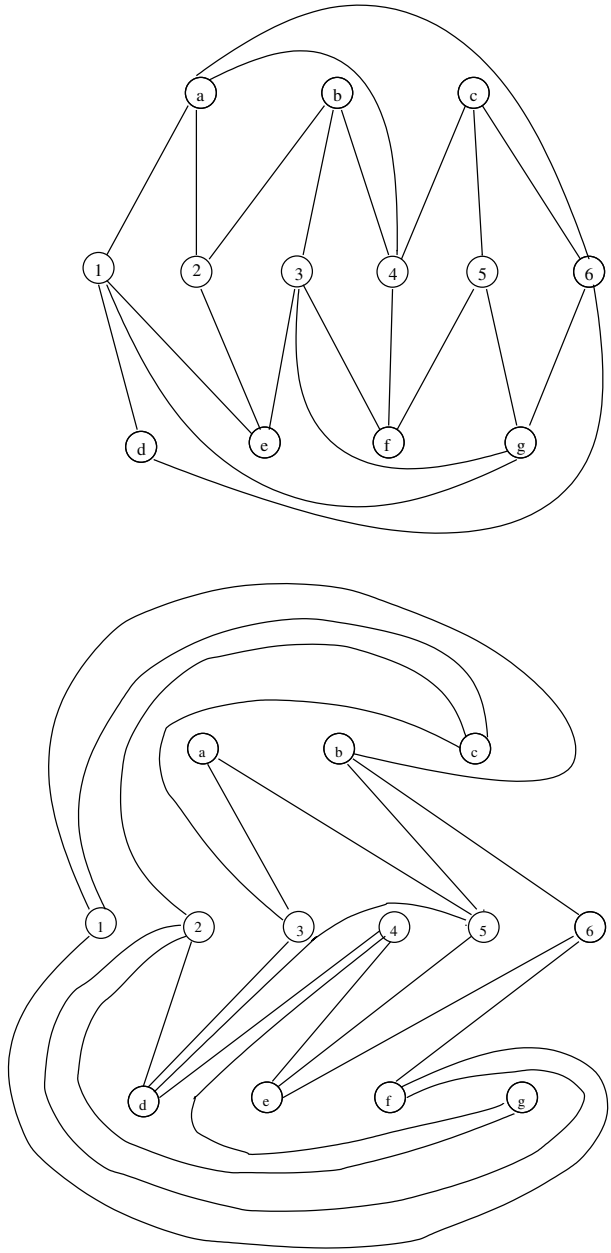


Fig. 1. Drawing of $K_{6,7}$ in two planes

where δ and Δ are minimum and maximum vertex degrees of the graph. Halton [7] independently proved the same upper bound, and in addition, he conjectured a stronger upper bound:

$$\Theta(G) \leq \left\lceil \frac{\Delta + 2}{4} \right\rceil .$$

In particular, Halton conjectured that any graph of maximum degree 6 has thickness at most 2, i.e. it is *biplanar*. A supporting argument for the conjecture comes from the fact that the thickness of $K_{n,n}$ is $\lceil (n+2)/4 \rceil$. Mutzel et al. [10] thought that Halton's conjecture might influence the design of integrated circuits, since current chip designers mainly use only two layers for designing a chip. Halton's conjecture was restated also in the survey of Beineke [2], although he doubted the conjecture.

We disprove Halton's conjecture and show that the upper bound in (1) is the best possible.

1.2 An Existence Proof

Theorem 1. *For any fixed $\Delta \geq 3$ and sufficiently large n there exists an n-vertex regular graph of degree Δ with thickness $\lceil \Delta/2 \rceil$.*

We have to recall some results from graph enumeration. Denise et al. [5] proved:

Lemma 1. *The number of unlabelled planar graphs on n vertices is at most 76^n, for sufficiently large n.*

This immediately implies:

Lemma 2. *The number of labelled n-vertex planar graphs is at most $76^n n!$, for sufficiently large n.*

McKay [9] showed:

Lemma 3. *Let R denote the set of all unlabelled Δ-regular simple n-vertex graphs. If $\varepsilon > 0$, and $3 \leq \Delta = O(n^{1/2-\varepsilon})$, then uniformly*

$$|R| = \frac{(\Delta n)!}{(\frac{\Delta n}{2})! 2^{\frac{\Delta n}{2}} (\Delta!)^n n!} \exp\left(-\frac{\Delta^2 - 1}{4} + O\left(\frac{\Delta^3}{n} \right) \right).$$

This implies that

$$|R| \geq \alpha \frac{(\Delta n)!}{(\frac{\Delta n}{2})! 2^{\frac{\Delta n}{2}} (\Delta!)^n n!} \exp\left(-\frac{\Delta^2 - 1}{4} \right), \tag{2}$$

for some absolute constant $\alpha > 0$ and sufficiently large n.

Let L denote the set of vertex labelled planar graphs on vertices $\{1, 2, ..., n\}$, and let U denote the set of unlabelled planar graphs on n vertices.

Now we return to the proof of Theorem 1. We assume that Δ is even. The odd case is similar. The proof goes by contradiction, so we make the assumption that every Δ-regular graph on n vertices has thickness at most $(\Delta - 2)/2$. Then,

of course, the same holds for unlabelled graphs, and we will reach contradiction by counting unlabelled graphs.

Next, based on the assumption, we will construct a partial function

$$F:\ U \times \overbrace{L \times \cdots \times L}^{(\Delta-4)/2 \text{ times}} \to R, \tag{3}$$

which is a surjection. The existence of such a function yields that

$$|U| \cdot |L|^{(\Delta-4)/2} \geq |R|. \tag{4}$$

Observe that Lemmata 1, 2 provide upper bound for the LHS of (4), and (2) provides lower bound for the RHS of (4). Hence we have

$$76^n (76^n n!)^{\frac{\Delta-4}{2}} \geq \alpha \frac{(\Delta n)!}{(\frac{\Delta n}{2})! 2^{\frac{\Delta n}{2}} (\Delta!)^n n!} \exp\left(-\frac{\Delta^2-1}{4}\right)$$

Regrouping terms we have an equivalent inequality:

$$\frac{1}{\alpha}(\Delta!)^n 76^{\frac{(\Delta-2)n}{2}} 2^{\frac{\Delta n}{2}} \exp\left(\frac{k^2-1}{4}\right) \geq \frac{(\Delta n)!}{(\frac{\Delta n}{2})!(n!)^{\frac{\Delta-2}{2}}}.$$

Now realize that the RHS is at least $n!$, i.e.

$$\frac{1}{\alpha}(\Delta!)^n 76^{\frac{(\Delta-2)n}{2}} 2^{\frac{\Delta n}{2}} \exp\left(\frac{k^2-1}{4}\right) \geq n!$$

Taking the natural logarithm of both sides we can easily see that the leading term on the LHS is $n\Delta \ln \Delta$ and the leading term on the RHS is $n \ln n$. Thus for $n >> \Delta$ we reach a contradiction.

Finally, we construct the partial function. For every $G \in U$ fix an arbitrary numbering of the vertices with $\{1, 2, ..., n\}$. Although the numbering is arbitrary, we use the *same* numbering whenever we evaluate F on a vector whose first coordinate is G and next $(\Delta-4)/2$ coordinates are graphs in the other planes. Keep $F(G, H_1, ..., H_{(\Delta-4)/2})$ undefined, if any edge ij is present in at least two of $G, H_1, ..., H_{(\Delta-4)/2}$, or if the graph

$$K = G \bigcup \left(\cup_{i=1}^{(\Delta-4)/2} H_i \right)$$

is not Δ-regular. If no edge occurs twice and K is Δ-regular, then set $F(G, H_1, ..., H_{(\Delta-4)/2})$ as the isomorphism class of K (i.e. delete labels from the vertices).

Finally, we have to show that F is surjection. Since elements of R are isomorphism classes of graphs, we have to show that any $J \in R$ is an image under F. Take any labelled graph $J^* \in J$. According to our hypothesis, J^* is of thickness at most $(\Delta-2)/2$, therefore any candidate for J^* can be written as $\cup_{i=0}^{(\Delta-4)/2} H_i$ with some labelled planar graphs $H_0, H_1, ..., H_{(\Delta-4)/2}$. There is a permutation π

of the set $\{1, 2, 3, ..., n\}$ such that $\pi(H_0) \in U$ and $\pi(H_0)$ has the labelling what this graph has in U. Then

$$F\left(\pi(H_0), \pi(H_1), ..., \pi(H_{(\Delta-4)/2})\right) = J.$$

□

Remark. It remains an open problem to construct explicitely Δ-regular graphs with thickness $\lceil \Delta/2 \rceil$.

2 Shahrokhi's Conjecture

2.1 Introduction

The *planar crossing number* $cr(G)$ of a graph $G = (V, E)$ is the minimum number of crossings over all drawings of G on a plane. This problem has been extensively studied by mathematicians, computer scientists and VLSI people. It is of high interest to compare the crossing number to other graph invariants. In [13] an unexpected relation between the so called bipartite crossing number and the optimal linear arrangement value was revealed. This motivated Shahrokhi to make a conjecture which would relate similarly the (ordinary) crossing number of a graph to the optimal linear arrangement value. Given an arbitrary graph $G = (V, E)$, and a bijection $f : V \to \{1, 2, 3, ..., |V|\}$, called *linear arrangement*, the *value* of this linear arrangment is

$$\sum_{uv \in E} |f(u) - f(v)|;$$

and the *optimal linear arrangement* problem for G is to find f which minimizes this value. The minimum value $L(G)$ is called the *length* of the graph G.

Let us call a graph family near $\Delta-$regular, if Δ is the maximum degree and the ratio Δ over the minimum degree is bounded by a constant. Shahrokhi [12] conjectured that for a near $\Delta-$regular $n-$vertex graph family $G = (V, E)$, the following relation holds between the planar crossing number and the length:

$$cr(G) > \alpha \frac{(L(G) - \gamma n^2)\Delta}{\log^{\beta} n}, \tag{5}$$

with some absolute positive constants α, β, and γ. If such an estimation exists, with possibly small β, it could provide a good approximation algorithm for the crossing number for a large class of graphs. Note that $\Delta L(G)$ is an obvious upper bound for the crossing number. So far the best approximation algorithm for crossing numbers has an approximation factor of $O(\log^3 n)$, [6].

We construct a graph G which fails the inequality (5).

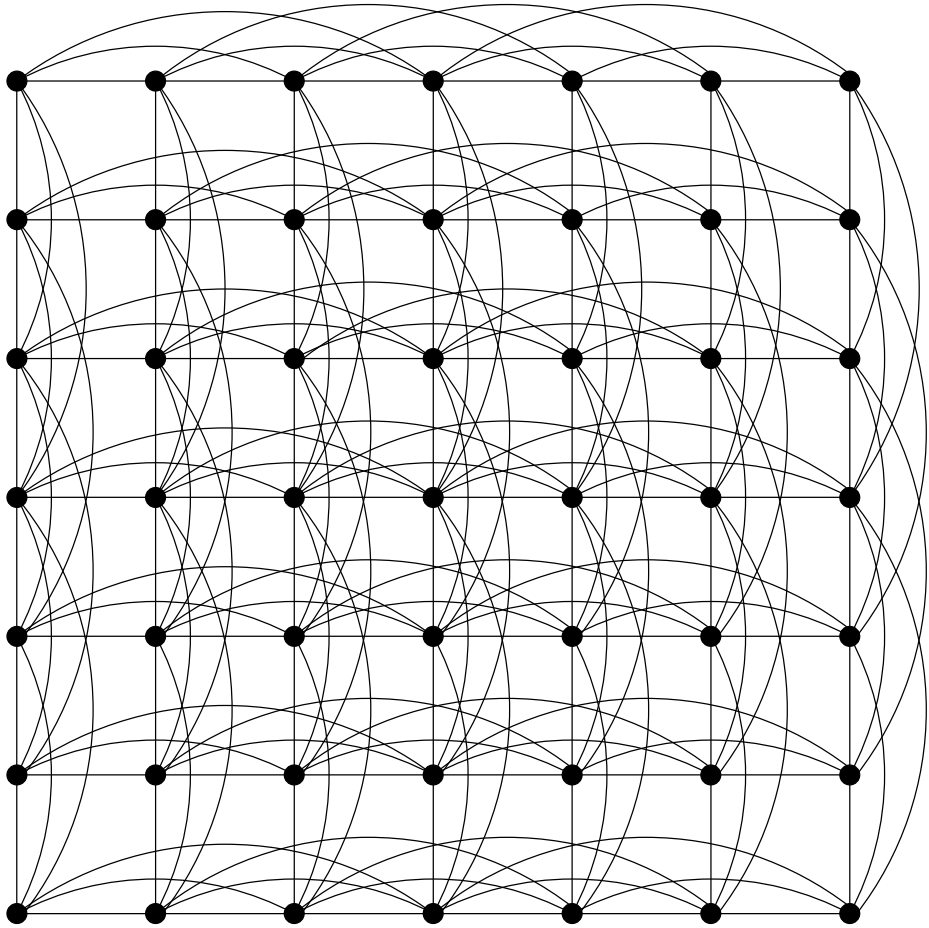


Fig. 2. A drawing of the graph G for $n = 49$.

2.2 Counterexample

Let r be sufficiently large integer and let k be an integer such that

$$\sqrt{10\gamma r} \leq k \leq r^{1-\varepsilon},$$

for some fixed $0 < \varepsilon < 1/2$. Let H denote a graph constructed from the r-vertex path by joining vertices if their distance is at most k. Define $G = H \times H$, where $\times$ denotes the Cartesian product of graphs. The graph G has $n = r^2$ vertices and is near regular with maximum degree $\Delta = 4k$. Figure 2 just illustrates the structure of such graphs for $n = 49$ and $k = 3$, without satisfying the condition for k.

Theorem 2. *The graph G does not satisfy the inequality (5).*

Proof. Place the vertices of G in the plane in an $r \times r$ grid manner and draw the edges in a natural way (see Figure 2). One can easily check that the number of crossings of "horizontal" edges with "vertical" edges is at most $\frac{1}{4}r^2k^4(1+o(1))$. The number of remaining crossings is at most r^2k^3. Hence $cr(G) < \frac{1}{2}r^2k^4$. Let $B(G)$ denote the *bisection width* of G, i.e. the minimum number of edges whose removal divides $G = (V, E)$ into two parts having at most $\frac{2}{3}|V|$ vertices each. It is easy to see that $L(G) \geq \frac{n}{3}B(G)$. We need to estimate $B(G)$. Let $\partial(A)$ denote the edge boundary of a vertex set A in a graph, i.e. the set of edges having one endpoint in A and the other endpoint in $V - A$. For a graph $F = (V_F, E_F)$, define its isoperimetric number as

$$i(F) = \min_A \left\{ \frac{|\partial(A)|}{|A|} : A \subset V_F, |A| \leq \frac{|V_F|}{2} \right\}.$$

It is easy to see by shifting elements in one direction that for our graph H,

$$i(H) \geq \min\left(\frac{2k^2}{r}, \min_{1 \leq i \leq k} k - \frac{i}{2} \right) = \frac{2k^2}{r}.$$

Chung and Tetali [4] proved that for any graph F

$$i(F \times F) \geq \frac{1}{2}i(F).$$

For our graph G we have

$$i(G) = i(H \times H) \geq \frac{1}{2}i(H) \geq \frac{k^2}{r}.$$

Now realize that

$$\min_A \left\{ \frac{|\partial(A)|}{|A|} : A \subset V_G, r^2/3 \leq |A| \leq r^2/2 \right\} \geq i(G) \geq \frac{k^2}{r}.$$

This implies that any subset of vertices of G of cardinality from the interval $[r^2/3, r^2/2]$ has an edge boundary of the cardinality at least $\frac{1}{3}rk^2$. Hence $B(G) \geq \frac{1}{3}rk^2$ and $L(G) \geq \frac{1}{9}r^3k^2$.

To get a contradiction it is sufficient to show that the function

$$f(x) = \frac{4\alpha x(\frac{1}{9}r^3x^2 - \gamma r^4)}{2^\beta \log^\beta r} - \frac{1}{2}r^2x^4$$

is positive for $x \in [\sqrt{10\gamma r}, r^{1-\varepsilon}]$ and sufficiently large r. In fact, one can check that $f(x)$ is nondecreasing in this interval and that

$$f(\sqrt{10\gamma r}) > 0.$$

□

References

1. Aggarwal, A., Klawe, M., Shor, P.: Multi-Layer Grid Embeddings for VLSI. Algorithmica **6** (1991) 129–151
2. Beineke, L. W.: Biplanar Graphs: A Survey. Computers and Mathematics with Applications **34** (1997) 1–8
3. Beineke, L. W., Harary, F., Moon, W.: On the Thickness of the Complete Bipartite Graph. Proc. Cambridge Philosophical Society **60** (1964) 1–5
4. Chung, F. R. K., Tetali, P.: Isoperimetric Inequalities for Cartesian Products of Graphs. Combinatorics, Probability and Computing **7** (1998) 141–148
5. Denise, A., Vasconcellos, M., Welsh, D.: The Random Planar Graphs. Congressus Numerantium 113 (1996) 61-79.
6. Even, G., Guha, S., Schieber, B.: Improved Approximations of Crossings in Graph Drawing. In: 32th Annual Symposium on Theory of Computing. ACM Press (2000)
7. Halton, J. H.: On the Thickness of Graphs of Given Degree. Information Sciences **54** (1991) 219–238
8. Mansfield, A.: Determining the Thickness of Graphs is NP-hard. Mathematical Proceedings of Combridge Philosohical Society **93** (1983) 9–23
9. McKay, B. D.: Asymptotics for Symmetries 0-1 Matrices with Prescribed Row Sums. Ars Combinatoria **19** (1985) 15–25
10. Mutzel, P., Odenthal, T., Scharbrodt, M.: The Thickness of Graphs: A Survey. Graphs and Combinatorics **14** (1998) 59–73
11. Owen, A.: The Biplanar Crossing Number. IEEE Transactions on Circuit Theory **18** (1971) 277–280
12. Shahrokhi, F.: Open Problem Presented at Special Session on Combinatorics and Graph Theory. 963 Sectional Meeting of AMS. Columbia SC. March 16-18 (2001)
13. Shahrokhi, F., Sýkora, O., Székely, L.A., Vrťo, I.: On Bipartite Drawings and the Linear Arrangement Problem. SIAM J. Computing **30** (2001) 1773–1789
14. Wessel, W.: Über die Abhängigkeit der Dicke eines Graphen von seinen Knotenpunktvalenzen. In: Geometrie und Kombinatorik'83. Vol. 2. Karl-Marx-Stadt Uni. (1984) 235–238

Connected and Loosely Connected List Homomorphisms

Narayan Vikas

School of Computing Science, Simon Fraser University, Burnaby
British Columbia, Canada V5A 1S6
vikas@cs.sfu.ca

Abstract. In this paper, we study a special graph colouring problem, called the list homomorphism problem, which is a generalisation of the list colouring problem. Several variants of the list homomorphism problem have been considered before. In particular, a complete complexity classification of the connected list homomorphism problem for reflexive graphs has been given before, according to which the problem is polynomial time solvable for reflexive chordal graphs, and NP-complete for reflexive non-chordal graphs. A natural analogue of this result is known not to hold for this problem for bipartite graphs. We observe that the notion of list connectivity in the problem needs to be modified for bipartite graphs. We introduce a new variant called the bipartite loosely connected list homomorphism problem for bipartite graphs. We give a complete complexity classification of this problem, showing that it is polynomial time solvable for chordal bipartite graphs, and NP-complete for non-chordal bipartite graphs. This result is analogous to the result for the connected list homomorphism problem for reflexive graphs. We present a linear time algorithm for the bipartite loosely connected list homomorphism problem for chordal bipartite graphs, as well as for the connected list homomorphism problem for reflexive chordal graphs, showing that the algorithms can decide just by testing whether or not the corresponding consistency tests succeed.

1 Introduction

In this section, we first give some definitions that we use in this paper, next we describe the list homomorphism problem and some of its variants including our new variant, and discuss results on these problems.

1.1 Definitions

Let G and H be graphs. A *homomorphism* $f : G \to H$, of G to H, is a mapping f of the vertices of G to the vertices of H, such that $f(g)$ and $f(g')$ are adjacent vertices of H whenever g and g' are adjacent vertices of G. Now, for each vertex v of G, let $L(v)$ be a list of vertices of H. We denote by L the entire set of lists $L(v)$ for the vertices v of G. A *list homomorphism* $l : G \to H$, of G to H with

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 399–412, 2002.

respect to L, is a homomorphism of G to H, such that $l(v) \in L(v)$, for every vertex v of G. If $l : G \to H$ is a list homomorphism with respect to L then we say that $l : G \to H$ is a list-L-homomorphism.

The *consistency test* for G with respect to H and L produces a set L^* of lists $L^*(v) \subseteq L(v)$, for all $v \in V(G)$, such that for any edge ab of G, if $h \in L^*(a)$ then there exists a vertex $h' \in L^*(b)$ such that hh' is an edge of H, and if $z \in L^*(b)$ then there exists a vertex $z' \in L^*(a)$ such that zz' is an edge of H; the set L^* is obtained in such a way that minimum number of vertices are removed from $L(v)$, for all $v \in V(G)$. Note that there exists a list-L-homomorphism of G to H if and only if there exists a list-L^*-homomorphism of G to H. The consistency test we have mentioned is in essence the *arc consistency test* in artificial intelligence [Mackworth, 1977]. It follows from the result of [Mackworth and Freuder, 1985] that if H is fixed then L^* is obtained in time linear in the size of G. We say that the consistency test for G with respect to H and L *succeeds* if $L^*(v) \neq \phi$, for all $v \in V(G)$, otherwise we say it does not succeed. We shall be using the consistency test in our proofs. The consistency test has been an important tool in artificial intelligence and has been widely used there since a long time including in earlier studies on constraint satisfaction problems [Montanari, 1974]. In graph homomorphism problems, the consistency test has been increasingly used in the last few years including in [Gutjahr, Welzl, and Woeginger, 1992], [Hell, Nesetril, and Zhu, 1996].

If vv is an edge of a graph then vv is called a *loop*, and the vertex v is said to have a loop. A graph is said to be *reflexive* if every vertex of the graph has a loop, and *irreflexive* if none of its vertices has a loop. A graph in general is *partially reflexive*, meaning that its individual vertices may or may not have loops. A *bipartite graph* G is a graph whose vertex set can be partitioned into two subsets G_A and G_B, such that each edge of G has one endpoint in G_A and the other endpoint in G_B; we say that (G_A, G_B) is a bipartition of G. Thus a bipartite graph is irreflexive by definition.

A graph in which every two distinct vertices are adjacent is called a *complete graph*. A bipartite graph, with bipartition (G_A, G_B), in which every vertex of G_A is adjacent to every vertex of G_B is called a *complete bipartite graph*. For a graph G, we use $V(G)$ and $E(G)$ to denote its vertex set and edge set respectively. In the following definitions we assume that G is a graph.

If H is a subgraph of G such that $E(H)$ contains all the edges of G that have both endpoints in $V(H)$, then H is called the subgraph of G *induced* by $V(H)$, and we say that H is an *induced subgraph* of G. A *chordal graph* is a graph which does not contain any induced cycle of length greater than three. A *chordal bipartite graph* is a bipartite graph which does not contain any induced cycle of length greater than four. A *clique* of G is a subset of $V(G)$ that induces a complete subgraph of G or is empty.

A vertex v of G is said to be an *isolated vertex* of G, if v is not adjacent to any other vertex v' of G, $v \neq v'$. If a vertex u is adjacent to a vertex v in G then u is said to be a *neighbour* of v in G, and v is said to be a neighbour of u in G. The *neighbourhood* of a vertex v of G, denoted as $Nbr(v)$, is the set of all neighbours

of v in G (note that if v has a loop then $v \in Nbr(v)$). The *neighbourhood* of a set S of vertices of G, denoted as $Nbr(S)$, is the set of all vertices of G which have a neighbour in S.

Two vertices u and v of G are said to be *connected* in G, if there exists a path from u to v in G; otherwise u and v are said to be *disconnected* in G. We say that G is *connected*, if every pair of vertices in G is connected; otherwise we say that G is *disconnected*. A *component* of G is a maximal connected subgraph of G. Let $S \subseteq V(G)$. We say that S is *connected* in G, if S induces a connected subgraph of G or $S = \phi$; otherwise we say that S is *disconnected* in G. We say that two vertices x and x' of G are *connected* in S, if there exists a path from x to x' in G all of whose vertices belong to S; otherwise we say that x and x' are *disconnected* in S. When we say that there is a path P connecting two vertices x and x' in S, this means that P is a path from x to x' in G, with $V(P) \subseteq S$.

We now introduce our new term which we shall use in our new variant of the list homomorphism problem. Let (H_A, H_B) be a bipartition of a bipartite graph H. Let $X \subseteq H_A$, and $Y \subseteq H_B$. If $(Nbr(X) \cap Y) \cup (Nbr(Y) \cap X)$ is connected in H then we say that $X \cup Y$ is *loosely connected* in H. Thus if $X \cup Y$ is connected in H then it is also loosely connected in H.

We may say that a set is connected, disconnected, or loosely connected without explicitly mentioning the graph in which it is which may be understood from the context. We now give below the definitions for perfect edge elimination scheme and perfect vertex elimination scheme that we use in our algorithms in this paper.

Suppose that Z is a bipartite graph. An edge $e = ab$ of Z is called a bisimplicial edge if $Nbr(a) \cup Nbr(b)$ induces a complete bipartite subgraph of Z. Let $S =< e_1, e_2, \ldots, e_m >$ be an ordering of the edges of Z, where $E(Z) = \{e_1, e_2, \ldots, e_m\}$ (every edge of Z is in S). Let $e_i = a_i b_i$, for all $i = 1, 2, \ldots, m$. We say that S is a *perfect edge elimination scheme* for Z, if e_i is a bisimplicial edge of the subgraph Z_i of Z containing only the edges $e_i, e_{i+1}, \ldots, e_m$, that is, every vertex in $A_i = \{a_j \in Nbr(b_i) | j > i\}$ is adjacent to every vertex in $B_i = \{b_j \in Nbr(a_i) | j > i\}$, for all $i = 1, 2, \ldots, m$. It can be shown that every chordal bipartite graph has a perfect edge elimination scheme. See [Golumbic, 1980] for a study on chordal bipartite graphs. In [Golumbic, 1980], another definition for perfect edge elimination scheme is given. The fact that every chordal bipartite graph has a perfect edge elimination scheme, as defined above, is also mentioned in [Brandstadt, 1993]. According to [Brandstadt, 1993], several researchers, including Tse Heng Ma and Haiko Muller, observed this result. The author would like to thank Pavol Hell and Jing Huang for pointing this definition of perfect edge elimination scheme.

A vertex s of G is called a *simplicial vertex* of G, if its neighbourhood, $Nbr(s)$, is a clique of G. Let $S =< v_1, v_2, \ldots, v_n >$ be an ordering of the vertices of G, where $V(G) = \{v_1, v_2, \ldots, v_n\}$ (every vertex of G is in S). We say that S is a *perfect vertex elimination scheme* for G, if v_i is a simplicial vertex of the subgraph G_i of G induced by the vertices $v_i, v_{i+1}, \ldots, v_n$, that is, the set $X_i = \{v_j \in Nbr(v_i) | j > i\}$ is a clique of G, for all $i = 1, 2, \ldots, n$. It can be

shown that a graph is chordal if and only if it has a perfect vertex elimination scheme. See [Golumbic, 1980] for a study on chordal graphs.

1.2 List Homomorphism Problems and Results

Let H be a fixed graph. The *list homomorphism problem for H*, denoted as *LHOM-H*, is the following :

Instance : A graph G and lists $L(v) \subseteq V(H)$, for all $v \in V(G)$.
Question : Does there exist a list-L-homomorphism of G to H?

Let S be a set of n different colours. Let $M(v) \subseteq S$, for each vertex v of a graph G. The *list colouring problem for n colours* asks whether each vertex v of G can be assigned a colour from $M(v)$ such that no adjacent vertices are assigned the same colour. The classic *n-colourability* problem is a special case when $M(v) = S$, for all $v \in V(G)$. Further, note that the list colouring problem for n colours is a special case of *LHOM-H* when H is an irreflexive complete graph with n vertices.

The list homomorphism problem for reflexive and irreflexive graphs H has been studied in [Feder & Hell, 1998] and [Feder, Hell, and Huang, 1999] respectively. It is shown in [Feder & Hell, 1998] that *LHOM-H* is polynomial time solvable for any reflexive interval graph H, and NP-complete for any reflexive non-interval graph H. For irreflexive graphs H, it is shown in [Feder, Hell, and Huang, 1999] that *LHOM-H* is polynomial time solvable if H is the complement of a circular arc graph of clique covering number two, and is NP-complete otherwise. Thus the complexity of the problem *LHOM-H* is completely classified for reflexive and irreflexive graphs H.

The *connected list homomorphism problem for H*, denoted as *CL-H*, is a restriction of the problem *LHOM-H* where each list $L(v)$ induces a connected subgraph of H. This problem was introduced in [Feder & Hell, 1998]. We generalise the problem *CL-H* as follows. Let $H_1, H_2, \ldots, H_s$ be the components of H. The *extended connected list homomorphism problem for H*, denoted as *ECL-H*, is a restriction of the problem *LHOM-H* where each list $L(v)$ is such that the list $L_i(v) = L(v) \cap V(H_i)$ is connected, i.e., it is either empty or induces a connected subgraph of H_i, for all $i = 1, 2, \ldots, s$. Note that for each vertex v of the input graph, $L_i(v) \subseteq V(H_i)$, for all $i = 1, 2, \ldots, s$, and $L_1(v), L_2(v), \ldots, L_s(v)$ is a partition of $L(v)$.

It follows from [Feder & Hell, 1998] that if H is a reflexive non-chordal graph then *ECL-H* is NP-complete, and if H is a reflexive chordal graph then *ECL-H* is solvable in linear time. Thus we have a complete complexity classification of *ECL-H* for reflexive graphs H. We also present an alternate linear time algorithm for the problem *ECL-H*, when H is a reflexive chordal graph. Our main reason for giving an alternate algorithm is to show the use of the consistency test technique as a tool for some list homomorphism problems, and the other reason is to provide a direct formula for finding a list homomorphism if there exists one. In [Feder & Hell, 1998], an algorithmic way is described for finding a possible list homomorphism.

It follows from the result of [Hell & Nesetril, 1990] that *ECL-H* is NP-complete for any non-bipartite irreflexive graph H. Further, it follows from [Feder, Hell, and Huang, 1999] that *ECL-H* is NP-complete for any non-chordal bipartite graph H also. However unlike the case for reflexive graphs, it is shown in [Feder, Hell, and Huang, 1999] that there are chordal bipartite graphs H for which the problem *ECL-H* is NP-complete. Thus we do not have a complete complexity classification of *ECL-H* for bipartite graphs H.

We note that under any homomorphism of a graph G to another graph H, if H is bipartite then it is necessary for G to be bipartite also, and all non-isolated vertices of each subset in a bipartition of G map to the same subset in a bipartition of H, and adjacent vertices of G map to different subsets in the bipartition of H (the isolated vertices of G can map to any subset in the bipartition of H). This suggests to consider modifying the notion of connectivity of lists for bipartite graphs.

We introduce our new variant called the bipartite loosely connected list homomorphism problem for bipartite graphs to resolve this situation. We first introduce some notation. Let H be a fixed bipartite graph with bipartition (H_A, H_B). Let $H_1, H_2, \ldots, H_s$ be the components of H, and $(H_{i,A}, H_{i,B})$ be a bipartition of H_i with $H_{i,A} \subseteq H_A$ and $H_{i,B} \subseteq H_B$, for all $i = 1, 2, \ldots, s$. Let a graph G with lists $L(v) \subseteq V(H)$, for all $v \in V(G)$, be an instance of *LHOM-H*. Let $L_A(v) = L(v) \cap H_A$ and $L_B(v) = L(v) \cap H_B$, for all $v \in V(G)$. Thus $L_A(v)$, $L_B(v)$ is a partition of $L(v)$, for all $v \in V(G)$. Further, let $L_{i,A}(v) = L_A(v) \cap H_{i,A}$ and $L_{i,B}(v) = L_B(v) \cap H_{i,B}$, for all $i = 1, 2, \ldots, s$, $v \in V(G)$. Thus $L_{i,A}(v) \subseteq H_{i,A}$ and $L_{i,B}(v) \subseteq H_{i,B}$, for all $i = 1, 2, \ldots, s$, $v \in V(G)$. Also, $L_{1,A}(v), L_{2,A}(v), \ldots, L_{s,A}(v)$ is a partition of $L_A(v)$, and $L_{1,B}(v), L_{2,B}(v), \ldots, L_{s,B}(v)$ is a partition of $L_B(v)$, with $v \in V(G)$.

The *bipartite loosely connected list homomorphism problem for H*, denoted as *BLCL-H*, is a restriction of the problem *LHOM-H* where for each edge vv' of G, the sets $L_{i,A}(v) \cup L_{i,B}(v')$ and $L_{i,B}(v) \cup L_{i,A}(v')$ are both loosely connected, for all $i = 1, 2, \ldots, s$. This completes the description of *BLCL-H*.

We present a linear time algorithm for *BLCL-H* when H is a chordal bipartite graph. We utilize the consistency test in our algorithm. We point out that *BLCL-H* is NP-complete for any non-chordal bipartite graph H. Thus we achieve a complete complexity classification of *BLCL-H* for bipartite graphs H which is analogous to the result of the complete complexity classification of the problem *ECL-H* for reflexive graphs H.

In order to show that *BLCL-H* is NP-complete for any non-chordal bipartite graph H, we need to consider the following problem. The *one-or-all list homomorphism problem for H*, denoted as *OAL-H*, is a restriction of *LHOM-H* where each list $L(v)$ contains either a single vertex of H or all the vertices of H. The problem *OAL-H* was introduced in [Feder & Hell, 1998]. Clearly, the problem *OAL-H* is a restriction of both the problems *BLCL-H* and *ECL-H* (with H being bipartite in case of the problem *BLCL-H*).

The problem *OAL-H* has its own significance, as it is polynomially (in fact linearly) equivalent to a well studied problem of retraction. This can be eas-

ily proved, and a proof for this can be found in [Feder & Hell, 1998]. Retraction problems have been of continuing interest in graph theory for a long time and have been studied in various literature including [Hell, 1972], [Hell, 1974], [Nowakowski & Rival, 1979], [Pesch and Poguntke, 1985], [Bandelt, Dahlmann, and Schutte, 1987], [Hell & Rival, 1987], [Pesch, 1988], [Bandelt, Farber, and Hell, 1993], [Feder & Hell, 1998], [Feder & Vardi, 1998], [Feder, Hell, and Huang, 1999]. Another problem, called the compaction problem [Vikas, 1999, 2001], polynomially transforms to *OAL-H* and the retraction problem. Hence polynomial results on *OAL-H* and the retraction problem are also helpful for the compaction problem.

Since the consistency test turns out to be a useful technique for list homomorphism problems (including *OAL-H*), as can be noted from the results in this paper, it is hence also useful for retraction problems. Earlier the isometry test has been a commonly used technique for retraction problems. Without giving a formal proof here, the consistency test includes the isometry test, and hence is more powerful. For example, the isometry test does not turn out to be useful for the retraction problem for reflexive chordal graphs H, whereas using the consistency test, we get a polynomial (in fact linear) time algorithm for the problem, as it provides a linear time algorithm for *ECL-H* which includes *OAL-H*.

In Section 2, we present a complete complexity classification of the problem *BLCL-H*. In particular, in Section 2.1, we present a linear time algorithm for *BLCL-H* when H is a chordal bipartite graph. In Section 3, we present a linear time algorithm for *ECL-H* when H is a reflexive chordal graph.

2 A Complete Complexity Classification of the Bipartite Loosely Connected List Homomorphism Problem

Theorem 2.1 *The problem BLCL-H is NP-complete if H is a non-chordal bipartite graph, and is polynomial time solvable if H is a chordal bipartite graph.*

Proof. Let H be a non-chordal bipartite graph. Let H' be an induced cycle of length at least six in H (since H is a non-chordal bipartite graph, we know there exists such an H' in H). As mentioned in Section 1.2, we know that *OAL-H'* is a restriction of *BLCL-H'*. Clearly, for any induced subgraph Z' of a given bipartite graph Z, the problem *BLCL-Z'* is a restriction of the problem *BLCL-Z*. Thus we have that *OAL-H'* is a restriction of *BLCL-H*. It is shown in [Feder, Hell, and Huang, 1999] that *OAL-H'* is NP-complete. It follows that *BLCL-H* is also NP-complete.

When H is a chordal bipartite graph, we prove results in Section 2.1 below, showing that *BLCL-H* is solvable in linear time. □

2.1 A Linear Time Algorithm for Bipartite Loosely Connected List Homomorphism to Chordal Bipartite Graphs

In this section, we give a linear time algorithm for *BLCL-H* when H is a chordal bipartite graph, completing the proof of Theorem 2.1. We first prove the following

theorem Theorem 2.1.1 for chordal bipartite graphs which we use in proving the main theorem Theorem 2.1.2.

Theorem 2.1.1 *Let H be a chordal bipartite graph with bipartition (H_A, H_B). Let $X \subseteq H_A$, and $Y, Z \subseteq H_B$, such that the sets $X \cup Y$ and $X \cup Z$ are both loosely connected. Then $(Nbr(Y) \cap X) \cup Z$ is also loosely connected.*

Proof. Let $X' = Nbr(Y) \cap X$, $X'' = Nbr(Z) \cap X$, $Y' = Nbr(X) \cap Y$, and $Z' = Nbr(X) \cap Z$. Since $X \cup Y$ and $X \cup Z$ both are loosely connected, we have that $X' \cup Y'$ and $X'' \cup Z'$ both are connected.

We have to prove that $X' \cup Z$ is loosely connected. Let $X^* = Nbr(Z) \cap X'$ and $Z^* = Nbr(X') \cap Z$. Thus we have to prove that $X^* \cup Z^*$ is connected.

Suppose that $X^* \cup Z^*$ is disconnected. Then there exist two distinct vertices x and x' in X^* such that x and x' are disconnected in $X^* \cup Z^*$. Clearly, the vertices x and x' are also disconnected in $X^* \cup Z'$, as $X^* \subseteq X'$, $Z^* \subseteq Z'$, and no vertex in X' has a neighbour in $Z - Z^*$. The vertices x and x' also belong to X'', as $X^* \subseteq X''$. Since $X'' \cup Z'$ is connected, the vertices x and x' are connected in $X'' \cup Z'$. Since x and x' are connected in $X'' \cup Z'$ but disconnected in $X^* \cup Z'$, and the fact that $X'' - X^* \subseteq X - X'$, this implies that $(X - X') \cap X'' \neq \phi$, and for any path P connecting x and x' in $X'' \cup Z'$ (thus by definition $V(P) \subseteq X'' \cup Z'$), there is at least one internal vertex of P which belongs to $(X - X') \cap X''$. Moreover, since $X^* = X'' - (X - X')$, we can choose the vertices x and x' such that no internal vertex of P belongs to $X'' - (X - X')$, i.e., every internal vertex of P belongs to either Z' or $(X - X') \cap X''$. Let x and x' be such a pair of vertices, and let $P_{xx'}$ denote a shortest path connecting x and x' in $X'' \cup Z'$ (again $V(P_{xx'}) \subseteq X'' \cup Z'$), where every internal vertex of $P_{xx'}$ belongs to Z' or $(X - X') \cap X''$, and at least one internal vertex of $P_{xx'}$ belongs to $(X - X') \cap X''$. Since $X^* \subseteq X'$, the vertices x and x' also belong to X'. Since $X' \cup Y'$ is connected, there exists a shortest path $P'_{xx'}$ connecting x and x' in $X' \cup Y'$ $(V(P'_{xx'}) \subseteq X' \cup Y')$.

Let $P_{xx'} = xz_1w_1z_2w_2 \ldots z_{p-1}w_{p-1}z_px'$, where $w_i \in (X - X') \cap X''$, for all $i = 1, 2, \ldots, p-1$, $z_j \in Z'$, for all $j = 1, 2, \ldots, p$, where $p > 1$, as at least one internal vertex of $P_{xx'}$ belongs to $(X - X') \cap X''$. Let $P'_{xx'} = xy_1x_1y_2x_2 \ldots y_{q-1}x_{q-1}y_qx'$, where $x_i \in X'$, for all $i = 1, 2, \ldots, q-1$, $y_j \in Y'$, for all $j = 1, 2, \ldots, q$, where $q \geq 1$. We have that $x, x' \in X^* = X'' - (X - X')$ and $x, x' \in X'$. Clearly, the vertices x and x' of these paths are not adjacent to any vertex of these paths other than shown, as these paths are shortest paths connecting x and x' in $X'' \cup Z'$ and $X' \cup Y'$ respectively. The only possibility of adjacency among internal vertices of these paths is due to a possible adjacency between the vertices x_i and z_j, $1 \leq i \leq q-1$, $1 \leq j \leq p$. Clearly, no other pair of internal vertices of these paths can be adjacent.

Suppose that for every subpath $z_jw_jz_{j+1}$ of $P_{xx'}$ $1 \leq j \leq p-1$, there exists a vertex x_i of $P'_{xx'}$, $1 \leq i \leq q-1$, such that x_i is adjacent to both z_j and z_{j+1}. Then there exists a path from x to x' all of whose vertices belong to the set $S = \{x, x', x_1, x_2, \ldots, x_{q-1}, z_1, z_2, \ldots, z_p\}$. Since $S \subseteq X' \cup Z'$, it follows that x and x' are connected in $X' \cup Z'$. Clearly then x and x' are also connected in

$X^* \cup Z'$. Further, this implies that x and x' are also connected in $X^* \cup Z^*$, which contradicts our assumption.

Thus there exists a subpath $z_s w_s z_{s+1}$ of $P_{xx'}$, $1 \leq s \leq p-1$, such that there is no vertex x_i of $P'_{xx'}$ adjacent to both z_s and z_{s+1}, $1 \leq i \leq q-1$. Suppose there exist two distinct vertices t and t' which belong to $T = \{x, x', x_1, x_2, \ldots, x_{q-1}\}$, such that t is adjacent to z_s, and t' is adjacent to z_{s+1}. Let $P'_{tt'}$ be the subpath of $P'_{xx'}$ from t to t'. We choose t and t' to be such that no internal vertex on the subpath $P'_{tt'}$ is adjacent to any vertex of $z_s w_s z_{s+1}$ (we can always choose t and t' to be such). Clearly, $P'_{tt'}$ consists of at least three distinct vertices. Also, the subpath $z_s w_s z_{s+1}$ of $P_{xx'}$, $1 \leq s \leq p-1$, has three distinct vertices, as $p > 1$. Let $P'_{tt'} = t y'_1 x'_1 y'_2 x'_2 \ldots y'_{r-1} x'_{r-1} y'_r t'$, where $x'_i \in \{x_1, x_2, \ldots, x_{q-1}\}$, for all $i = 1, 2, \ldots, r-1$, $y'_j \in \{y_1, y_2, \ldots, y_q\}$, for all $j = 1, 2, \ldots, r$, $r \geq 1$. Thus $t y'_1 x'_1 y'_2 x'_2 \ldots y'_{r-1} x'_{r-1} y'_r t' z_{s+1} w_s z_s t$ is an induced cycle of length greater than four, and we have a contradiction. The case when z_s or z_{s+1} is not adjacent to any vertex in T would mean even fewer edges between $P_{xx'}$ and $P'_{xx'}$, and based on the above case, would obviously imply existence of an induced cycle of length greater than four in H (including z_s, w_s, z_{s+1}), which would again be a contradiction. Hence $X^* \cup Z^*$ must be connected, and therefore $X' \cup Z$ must be loosely connected. □

Theorem 2.1.2 *Let H be a chordal bipartite graph, and let a bipartite graph G with lists $LST(v) \subseteq V(H)$, for all $v \in V(G)$, be an instance of BLCL-H. Then there exists a list-LST-homomorphism of G to H if and only if the consistency test for G with respect to H and LST succeeds.*

Proof. Suppose that H has s components, namely, $H_1, H_2, \ldots, H_s$. Let (H_A, H_B) be a bipartition of H, and $(H_{i,A}, H_{i,B})$ be a bipartition of H_i, with $H_{i,A} \subseteq H_A$, and $H_{i,B} \subseteq H_B$, for all $i = 1, 2, \ldots, s$. Let $LST_A(v) = LST(v) \cap H_A$, and $LST_B(v) = LST(v) \cap H_B$, for all $v \in V(G)$. Further, let $LST_{i,A}(v) = LST_A(v) \cap H_{i,A}$, and $LST_{i,B}(v) = LST_B(v) \cap H_{i,B}$, for all $i = 1, 2, \ldots, s$, $v \in V(G)$. Since the graph G with lists $LST(v)$, for all $v \in V(G)$, is an instance of *BLCL-H*, we have that for each edge vv' of G, the sets $LST_{i,A}(v) \cup LST_{i,B}(v')$ and $LST_{i,B}(v) \cup LST_{i,A}(v')$ are both loosely connected, for all $i = 1, 2, \ldots, s$.

If there exists a list-LST-homomorphism of G to H then clearly the consistency test for G with respect to H and LST succeeds.

Now suppose that the consistency test for G with respect to H and LST succeeds. Let $L'(v)$, for all $v \in V(G)$ be the lists obtained as a result of performing the consistency test for G with respect to H and LST. We have $L'(v) \subseteq LST(v)$, for all $v \in V(G)$, and since the consistency test has succeeded, we have $L'(v) \neq \phi$, for all $v \in V(G)$. Let $L'_A(v) = L'(v) \cap H_A$, and $L'_B(v) = L'(v) \cap H_B$, for all $v \in V(G)$. Let $L'_{i,A}(v) = L'_A(v) \cap H_{i,A}$, and $L'_{i,B}(v) = L'_B(v) \cap H_{i,B}$, for all $i = 1, 2, \ldots, s$, $v \in V(G)$. We have that $L'_{i,A}(v) \subseteq LST_{i,A}(v)$, and $L'_{i,B}(v) \subseteq LST_{i,B}(v)$, for all $i = 1, 2, \ldots, s$, $v \in V(G)$. For each edge vv' of G, since the sets $LST_{i,A}(v) \cup LST_{i,B}(v')$ and $LST_{i,B}(v) \cup LST_{i,A}(v')$ are both loosely connected, it follows from Theorem 2.1.1 and the mechanism of the con-

sistency test that the sets $L'_{i,A}(v) \cup L'_{i,B}(v')$ and $L'_{i,B}(v) \cup L'_{i,A}(v')$ are both connected, for all $i = 1, 2, \ldots, s$.

Assume that G has t components, namely, $G_1, G_2, \ldots, G_t$. Let (G_A, G_B) be a bipartition of G, and $(G_{i,A}, G_{i,B})$ be a bipartition of G_i, with $G_{i,A} \subseteq G_A$, and $G_{i,B} \subseteq G_B$, for all $i = 1, 2, \ldots, t$. Let u be some vertex of G_j, $1 \leq j \leq t$. Let i_j be some value for which either $L'_{i_j,A}(u) \neq \phi$ or $L'_{i_j,B}(u) \neq \phi$ (both may be nonempty), $1 \leq i_j \leq s$. There exists such an i_j, as $L'(u) \neq \phi$. Without loss of generality, suppose that $L'_{i_j,A}(u) \neq \phi$. Thus $L'_{i_j,A}(u)$ contains a vertex of $H_{i_j,A}$, and hence $L'(u)$ contains a vertex of $H_{i_j,A}$. We can always redefine the bipartition of G and G_j such that $u \in G_{j,A}$, and $G_{j,A} \subseteq G_A$, without affecting the bipartition of other components of G. Hence, without loss of generality, we assume that the vertex u of G_j belongs to $G_{j,A}$. Since $L'(u)$ contains a vertex of $H_{i_j,A}$, it follows due to the success of the consistency test that for each $a \in G_{j,A}$, the list $L'(a)$ contains a vertex of $H_{i_j,A}$, and for each $b \in G_{j,B}$, the list $L'(b)$ contains a vertex of $H_{i_j,B}$. This implies that $L'_{i_j,A}(a) \neq \phi$, for all $a \in G_{j,A}$, and $L'_{i_j,B}(b) \neq \phi$, for all $b \in G_{j,B}$. Since j was an arbitrary value, we argue this way for each component G_j of G, $1 \leq j \leq t$. We let $L(a) = L'_{i_j,A}(a)$, for all $a \in G_{j,A}$, and $L(b) = L'_{i_j,B}(b)$, for all $b \in G_{j,B}$, $j = 1, 2, \ldots, t$. Thus $L(a) \subseteq V(H_{i_j,A})$, for all $a \in V(G_{j,A})$, and $L(b) \subseteq V(H_{i_j,B})$, for all $b \in V(G_{j,B})$, $j = 1, 2, \ldots, t$. As mentioned above, $L'_{i_j,A}(a) \cup L'_{i_j,B}(b)$ is connected, for all $ab \in E(G_j)$, $j = 1, 2, \ldots, t$. Hence $L(a) \cup L(b)$ is connected, for all $ab \in E(G)$.

We shall prove that there exists a list-L-homomorphism of G to H. Since $L(v) \subseteq LST(v)$, for all $v \in V(G)$, this would imply that there exists list-LST-homomorphism of G to H. We find a list-L-homomorphism of G to H as follows.

Suppose that $E(H) = \{e_1, e_2, \ldots, e_q\}$. Let $S =< e_1, e_2, \ldots, e_q >$ be a perfect edge elimination scheme for H. We shall perform q steps, namely $1, 2, \ldots, q$, in this order. In each step, we consider deleting certain vertices from the older lists and obtain new lists as described below. The original lists are $L(v)$, for all $v \in V(G)$. Let $L^i(v)$, for all $v \in V(G)$, denote the set of lists obtained as a result of executing step i, for all $i = 1, 2, \ldots, q$. We let $L^0(v) = L(v)$, for all $v \in V(G)$. The lists $L^{i-1}(v)$, for all $v \in V(G)$, may be viewed as an input to the i-th step, and the lists $L^i(v)$, for all $v \in V(G)$, may be viewed as an output of the i-th step, for all $i = 1, 2, \ldots, q$.

The i-th step is described as follows, $1 \leq i \leq q$. We consider the edge $e_i = xy$, with $x \in H_A$, $y \in H_B$. Initially, we let $L^i(v) = L^{i-1}(v)$, for all $v \in V(G)$. Next, for every edge ab of G, with $a \in G_A$, $b \in G_B$, $x \in L^{i-1}(a)$, and $y \in L^{i-1}(b)$, we do the following. If y has another neighbour x' in $L^{i-1}(a)$, $x' \neq x$, then we update $L^i(a) = L^{i-1}(a) - \{x\}$. If x has another neighbour y' in $L^{i-1}(b)$, $y' \neq y$, then we update $L^i(b) = L^{i-1}(b) - \{y\}$. This completes the description of step i, $1 \leq i \leq q$. Note that $L^i(v) \subseteq L^{i-1}(v)$, and the two sets differ by at most one element, for all $v \in V(G)$, $i = 1, 2, \ldots, q$.

Let ab be any edge of G. We shall prove by induction that $L^i(a) \cup L^i(b)$ is connected, for all $i = 0, 1, 2, \ldots q$. For $i = 0$, we do know that this is true. Suppose that $L^j(a) \cup L^j(b)$ is connected for all $j < i$, for some $i \geq 1$. We shall show that $L^i(a) \cup L^i(b)$ is also connected. We first show that $L^i(a) \cup L^{i-1}(b)$ is

connected. If $L^i(a) = L^{i-1}(a)$ then by assumption $L^i(a) \cup L^{i-1}(b)$ is connected. Suppose that $L^i(a) \neq L^{i-1}(a)$. It follows from step i that $L^i(a) = L^{i-1}(a) - \{x\}$, where $e_i = xy$, with $x \in H_A$, $y \in H_B$, $x \in L^{i-1}(a)$, and there exists a neighbour b' of a in G with $y \in L^{i-1}(b')$ so that y has another neighbour x' in $L^{i-1}(a)$, $x' \neq x$. We show that if x is an endpoint of any edge $e_j = xz$, with $j < i$, $z \in H_B$, then there exists no neighbour b'' of a in G with $z \in L^{i-1}(b'')$.

Suppose that x is an endpoint of some edge e_j, with $j < i$. Let $e_j = xz$, where $z \in H_B$. Consider when the j-th step is executed. Since $x, x' \in L^{i-1}(a)$, and $L^{i-1}(a) \subseteq L^{j-1}(a)$, we have $x, x' \in L^{j-1}(a)$. Thus x is not the only vertex of $L^{j-1}(a)$, as $x' \in L^{j-1}(a)$, and $x' \neq x$. If there exists no neighbour b'' of a in G with $z \in L^{j-1}(b'')$ then it is also the case that there exists no neighbour b'' of a in G with $z \in L^{i-1}(b'')$, as $L^{i-1}(v) \subseteq L^{j-1}(v)$, for all $v \in V(G)$. Now suppose that there exists a neighbour b'' of a in G, with $z \in L^{j-1}(b'')$, $b'' \in G_B$. Since by assumption $L^{j-1}(a) \cup L^{j-1}(b'')$ is connected, and x is not the only vertex of $L^{j-1}(a)$, either x has another neighbour z' in $L^{j-1}(b'')$, $z' \neq z$, or z has another neighbour x'' (could be x') in $L^{j-1}(a)$, $x'' \neq x$. It follows from step j that either $L^j(a) = L^{j-1}(a) - \{x\}$ or $L^j(b'') = L^{j-1}(b'') - \{z\}$. Since $x \in L^{i-1}(a)$, and $L^{i-1}(a) \subseteq L^j(a)$ (as $j < i$), it must be that $L^j(a) = L^{j-1}(a)$ and $L^j(b'') = L^{j-1}(b'') - \{z\}$. Since b'' was an arbitrary neighbour of a in G, we have that $L^j(b'') = L^{j-1}(b'') - \{z\}$, for all neighbours b'' of a in G. Thus there exists no neighbour b'' of a in G with $z \in L^j(b'')$. Since $L^{i-1}(v) \subseteq L^j(v)$, for all $v \in V(G)$, this implies that there exists no neighbour b'' of a in G with $z \in L^{i-1}(b'')$. Thus for every edge $e_j = xz$, with $j < i$, there exists no neighbour b'' of a in G with $z \in L^{i-1}(b'')$.

It follows from the above result and the definition of perfect edge elimination scheme that for every neighbour b'' of a in G, the vertex x' of $L^{i-1}(a)$ is adjacent to every neighbour of x which occurs in $L^{i-1}(b'')$. Since $L^{i-1}(a) \cup L^{i-1}(b)$ is connected by assumption, this implies that $L^{i-1}(a) - \{x\} \cup L^{i-1}(b)$ is also connected. Thus in all cases $L^i(a) \cup L^{i-1}(b)$ is connected. Using this result, and with arguments analogous to the above when considering the list $L^i(b)$, we can show that $L^i(a) \cup L^i(b)$ is connected (indeed $L^{i-1}(a) \cup L^i(b)$ is also shown to be connected similarly). Thus $L^i(a) \cup L^i(b)$ is connected, for all $i = 0, 1, 2, \ldots, q$. Since ab was an arbitrary edge of G, we have that $L^i(a) \cup L^i(b)$ is connected, for all $ab \in E(G)$, $i = 0, 1, 2, \ldots, q$.

For each isolated vertex v of G, we update $L^q(v)$ by removing all vertices from $L^q(v)$ except one vertex chosen arbitrarily. We now show that $L^q(v)$ is a singleton, for all $v \in V(G)$. We know this is true when v is an isolated vertex of G. Consider an edge $e_i = xy$, with $x \in H_A$, $y \in H_B$, $1 \leq i \leq q$. The edge e_i is considered only in the i-th step. Let ab be an edge of G, with $a \in G_A$, $b \in G_B$, $x \in L^{i-1}(a)$, and $y \in L^{i-1}(b)$. If $L^{i-1}(a) = \{x\}$ or $L^{i-1}(b) = \{y\}$ is a singleton then clearly it follows from step i that accordingly $L^i(a) = \{x\}$ or $L^i(b) = \{y\}$. Now suppose that one or both of the lists $L^{i-1}(a)$ and $L^{i-1}(b)$ is not a singleton. Since we showed that $L^{i-1}(a) \cup L^{i-1}(b)$ is connected, either x has another neighbour y' in $L^{j-1}(b)$, $y' \neq y$, or y has another neighbour x' in $L^{j-1}(a)$, $x' \neq x$ (both cases may hold). It follows from step i that either

$L^i(a) = L^{i-1}(a) - \{x\}$ or $L^i(b) = L^{i-1}(b) - \{y\}$ (both may hold). Thus the size of the lists tend to get smaller, and when a list becomes a singleton, then as explained above, no further vertices are removed from the list. For every non-isolated vertex v of G, each vertex in $L(v)$ is an endpoint of some edge e in the scheme S, and e gets considered in some step. After all the edges of H have been considered (in the order of the scheme S) through the steps $1, 2, \ldots, q$, this mechanism shows that the list $L^q(v)$ must be a singleton, for all $v \in V(G)$.

Let $L^q(v) = \{h_v\}$, where $h_v \in V(H)$, for all $v \in V(G)$. Since we showed that $L^q(a) \cup L^q(b)$ is connected for all $ab \in E(G)$, it follows that the mapping $l : G \to H$, with $l(v) = h_v$, for all $v \in V(G)$, is indeed a list-L^q-homomorphism of G to H. Since $L^q(v) \subseteq L(v)$ and $L(v) \subseteq LST(v)$, for all $v \in V(G)$, it follows that $l : G \to H$ is a list-LST-homomorphism of G to H, and the theorem is proved. □

We outline below our algorithm for *BLCL-H* in a stepwise form which also defines a list homomorphism if there exists one. The correctness of the algorithm readily follows from the proof of Theorem 2.1.2.

Algorithm for $BLCL - H$

Let H be a chordal bipartite graph. Let a bipartite graph G with lists $LST(v) \subseteq V(H)$, for all $v \in V(G)$, be an instance of *BLCL-H*. Our algorithm to decide whether or not there exists a list-LST-homomorphism of G to H, and define a list-LST-homomorphism of G to H if there exists one, is as follows.

1. Perform the consistency test for G with respect to H and LST, and obtain lists $L'(v)$ ($\subseteq LST(v)$), for all $v \in V(G)$.
2. If there exists a vertex v of G for which $L'(v) = \phi$, i.e., if the consistency test in step 1 does not succeed, then report that there does not exist a list-LST-homomorphism of G to H, and stop.
3. Report that there exists a list-LST-homomorphism of G to H, and define one as follows (steps 4 through 10).
4. Let $H_1, H_2, \ldots, H_s$ be the components of H. Let $(H_{i,A}, H_{i,B})$ be a bipartition of H_i, for all $i = 1, 2, \ldots, s$. Let $G_1, G_2, \ldots, G_t$ be the components of G.
5. Let $(G_{j,A}, G_{j,B})$ be a bipartition of G_j (if G_j has only one vertex then either $G_{j,A}$ or $G_{j,B}$ is empty) such that $L(a) = L'(a) \cap V(H_{i_j,A}) \neq \phi$, and $L(b) = L'(b) \cap V(H_{i_j,B}) \neq \phi$, for some i_j (there always exists such an i_j and a bipartition of G_j), $1 \leq i_j \leq s$, for all $a \in V(G_{j,A})$, $b \in V(G_{j,B})$, $j = 1, 2, \ldots, t$. (As shown in the proof of Theorem 2.1.2, using Theorem 2.1.1 and the mechanism of the consistency test, we have that $L(a) \cup L(b)$ is connected, for all $ab \in E(G)$.)
6. Obtain a perfect edge elimination scheme $S =< e_1, e_2, \ldots, e_q >$ for H, where $E(H) = \{e_1, e_2, \ldots, e_q\}$.
7. Let (H_A, H_B) be a bipartition of H, with $H_{i,A} \subseteq H_A$, and $H_{i,B} \subseteq H_B$, for all $i = 1, 2, \ldots, s$. Let (G_A, G_B) be a bipartition of G, with $G_{i,A} \subseteq G_A$, and $G_{i,B} \subseteq G_B$ (note that $G_{i,A}$ and $G_{i,B}$ are computed in Step 5), for all $i = 1, 2, \ldots, t$. Let $L^0(v) = L(v)$, for all $v \in V(G)$.

8. for $i = 1$ to q do
begin
let $e_i = xy$, with $x \in H_A$, $y \in H_B$;
let $L^i(v) = L^{i-1}(v)$, for all $v \in V(G)$;
for each edge ab of G, with $a \in G_A$, $b \in G_B$, do
begin
if $x \in L^{i-1}(a)$ and $y \in L^{i-1}(b)$ then do
begin
if y has a neighbour $x' \in L^{i-1}(a)$, $x' \neq x$, then $L^i(a) = L^{i-1}(a) - \{x\}$;
if x has a neighbour $y' \in L^{i-1}(b)$, $y' \neq y$, then $L^i(b) = L^{i-1}(b) - \{y\}$;
end
end /* for each edge ab */
end /* for $i = 1$ to q */
9. For each isolated vertex v of G, remove all vertices from $L^q(v)$ except one vertex (this vertex is chosen arbitrarily).
10. The mapping $l : G \to H$ with $l(v) =$ the vertex in $L^q(v)$, for all $v \in V(G)$, is a list-L^q-homomorphism, and hence a list-L-homomorphism, and a list-LST-homomorphism of G to H (as shown in the proof of Theorem 2.1.2, we have that $L^q(v)$ is a singleton, for all $v \in V(G)$).

This completes the outline of our algorithm. Since H is fixed, Step 6 takes only a fixed time. As mentioned earlier, Step 1 runs in time linear in the size of G. Note that q is fixed, and Step 8 also runs in time linear in the size of G. It is readily seen that the entire algorithm also runs in time linear in the size of G.

3 A Linear Time Algorithm for Connected List Homomorphism to Reflexive Chordal Graphs

In this section, we give an alternate linear time algorithm for *ECL-H* when H is a reflexive chordal graph. We first prove the following two basic theorems Theorem 3.1 and Theorem 3.2 that we use in proving the main theorem Theorem 3.3. We do not include the proofs of any theorem in this section.

Theorem 3.1 *If H is a reflexive chordal graph then for any connected subsets X and Y of $V(H)$, the subset $Nbr(X) \cap Y$ of $V(H)$ is also connected.*

Theorem 3.2 *Let H be a chordal graph, and S be its perfect vertex elimination scheme. For any two distinct vertices h and h' of H, define $h > h'$ and $h' < h$, if h occurs after h' in S. Let $x, y \in V(H)$. If there exists a path from x to y in H then there exists a path $P = h_1 h_2 \ldots h_k h_{k+1} \ldots h_q$ from $h_1 = x$ to $h_q = y$ in H such that $h_i < h_{i+1}$, for all $i = 1, 2, \ldots, k-1$, and $h_j > h_{j+1}$, for all $j = k, k+1, \ldots, q-1$; $h_t \in V(H)$, for all $t = 1, 2, \ldots, q$.*

Theorem 3.3 *Let H be a reflexive chordal graph, and let a graph G with lists $LST(v) \subseteq V(H)$, for all $v \in V(G)$, be an instance of ECL-H. Then there exists*

a list-LST-homomorphism of G to H if and only if the consistency test for G with respect to H and LST succeeds.

We outline below our algorithm for *ECL-H* in a stepwise form which also defines a list homomorphism if there exists one using a direct formula. The correctness of the algorithm follows from the proof of Theorem 3.3.

Algorithm for $ECL - H$

Let H be a reflexive chordal graph with components $H_1, H_2, \ldots, H_s$. Let a graph G with lists $LST(v) \subseteq V(H)$, for all $v \in V(G)$, be an instance of *ECL-H*. Let $G_1, G_2, \ldots, G_t$ be the components of G. Our algorithm to decide whether or not there exists a list-LST-homomorphism of G to H, and define a list-LST-homomorphism of G to H if there exists one, is as follows.

1. Perform the consistency test for G with respect to H and LST, and obtain lists $L'(v)$ $(\subseteq LST(v))$, for all $v \in V(G)$.
2. If there exists a vertex v of G for which $L'(v) = \phi$, i.e., if the consistency test in step 1 does not succeed, then report that there does not exist a list-LST-homomorphism of G to H, and stop.
3. Report that there exists a list-LST-homomorphism of G to H, and define one as follows (steps 4, 5, and 6).
4. Obtain a perfect vertex elimination scheme S for H.
5. Let $L(v) = L'(v) \cap V(H_{i_j}) \neq \phi$, for some i_j (there always exists such an i_j), $1 \leq i_j \leq s$, for all $v \in V(G_j)$, $j = 1, 2, \ldots, t$. (It is shown in the proof of Theorem 3.3, using Theorem 3.1 and the mechanism of the consistency test, that $L(v)$ is connected, for all $v \in V(G)$.)
6. The mapping $l : G \to H$ defined below is a list-L-homomorphism, and hence a list-LST-homomorphism, of G to H (we utilize Theorem 3.2 in proving this) : $l(v)$ = the vertex in $L(v)$ which occurs latest in S among the vertices present in $L(v)$.

This completes the outline of our algorithm. A perfect vertex elimination scheme for a chordal graph can be obtained in time linear in the size of the graph (see [Golumbic, 1980]). However since H is fixed, step 4 takes only a fixed time. As mentioned earlier, step 1 runs in time linear in the size of G. Thus it can be seen that the entire algorithm also runs in time linear in the size of G.

References

1. H. J. Bandelt, A. Dahlmann, and H. Schutte, Absolute Retracts of Bipartite Graphs, Discrete Applied Mathematics, 16, 191-215, 1987.
2. H. J. Bandelt, M. Farber, and P. Hell, Absolute Reflexive Retracts and Absolute Bipartite Retracts, Discrete Applied Mathematics, 44, 9-20, 1993.
3. A. Brandstadt, Special graph classes - a survey, Universitat Duisburg Gesamthochschule, 1993.

4. T. Feder and P. Hell, List Homomorphisms to Reflexive Graphs, Journal of Combinatorial Theory, Series B, 72, 236-250, 1998.
5. T. Feder, P. Hell, and J. Huang, List Homomorphisms and Circular Arc Graphs, Combinatorica, 19, 487-505, 1999.
6. T. Feder and M. Y. Vardi, The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory, SIAM Journal on Computing, 28, 57-104, 1998.
7. M. C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, Inc., San Diego, California, 1980.
8. W. Gutjahr, E. Welzl, and G. Woeginger, Polynomial graph-colorings, Discrete Applied Mathematics, 35, 29-45, 1992.
9. P. Hell, Retractions de Graphes, Ph.D. Thesis, Universite de Montreal, 1972.
10. P. Hell, Retracts in Graphs, in Graphs and Combinatorics, Lecture Notes in Mathematics, Springer-Verlag, 406, 291-301, 1974.
11. P. Hell and J. Nesetril, On the Complexity of H-colouring, Journal of Combinatorial Theory, Series B, 48, 92-110, 1990.
12. P. Hell, J. Nesetril, and X. Zhu, Duality and Polynomial Testing of Tree Homomorphisms, Transactions of the American Mathematical Society, 348, 1281-1297, 1996.
13. P. Hell and I. Rival, Absolute Retracts and Varieties of Reflexive Graphs, Canadian Journal of Mathematics, 39, 544-567, 1987.
14. A. K. Mackworth, Consistency in Networks of Relations, Artificial Intelligence, 8, 99-118, 1977.
15. A. K. Mackworth and E. C. Freuder, The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems, Artificial Intelligence, 25, 65-74, 1985.
16. U. Montanari, Networks of Constraints : Fundamental Properties and Applications to Picture Processing, Information Sciences, 7, 95-132, 1974.
17. R. Nowakowski and I. Rival, Fixed-Edge Theorem for Graphs with Loops, Journal of Graph Theory, 3, 339-350, 1979.
18. E. Pesch, Retracts of Graphs, Mathematical Systems in Economics, 110, Frankfurt am Main : Athenaum, 1988.
19. E. Pesch and W. Poguntke, A Characterization of Absolute Retracts of n-Chromatic Graphs, Discrete Mathematics, 57, 99-104, 1985.
20. N. Vikas, Computational Complexity of Compaction to Cycles, Proceedings of Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1999.
21. N. Vikas, Computational Complexity of Compaction to Reflexive Cycles, Accepted in SIAM Journal on Computing, 2001.

Any Load-Balancing Regimen for Evolving Tree Computations on Circulant Graphs Is Asymptotically Optimal

Rolf Wanka*

Dept. of Mathematics & Computer Science and Heinz Nixdorf Institute
Paderborn University, 33095 Paderborn, Germany
wanka@upb.de

Abstract. We analyze evolving tree computations on circulant (rings with "regular" chords) and related graphs. In an evolving α-ary tree computation, a complete tree grows level by level, i. e., every leaf generates α new nodes that become the new leaves. The load balancing task is to spread the new nodes on a network of processors in the moment they were created in such a way that the accumulated number of nodes per processor, i. e., its load, is as close as possible to the average number of nodes per processor. Gao/Rosenberg [2] introduced evolving computations and investigated the growth of complete binary trees on rings of processors. They showed that the so-called KS-regimen behaves optimally in the course of long computations. In this paper, we generalize evolving computations to trees of arbitrary degree and we generalize the regimen notion. We show that *any* regimen behaves optimally. For this purpose, we model the actual load distribution, the generation process, and the distribution regimen by formal infinite polynomials. Then we show that evaluating these polynomials for certain inputs leads to the analysis of these regimens on circulant and related graphs. It is shown that *any* regimen leads to a close to optimal load distribution.

1 Introduction

Background. In the standard abstract formulation of load balancing in a distributed network, processors are modeled as the vertices of a graph and links between them as edges. Each processor initially has a collection of unit-size jobs which we call *tokens*. In a dynamic setting, some of these tokens generate new tokens, so we distinguish between generating and non-generating tokens. The object is to balance the number of tokens by transmitting the new tokens along edges according to some local scheme. This problem has obvious applications in job scheduling and other coordination tasks in parallel and distributed systems. It also arises in the context of finite element computations, and in simulations of physical phenomena.

* Partially supported by DFG SFB 376 "Massively Parallel Computation" and by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

L. Kučera (Ed.): WG 2002, LNCS 2573, pp. 413–420, 2002.

In order to avoid expensive reordering of all tokens or special computations of destinations for every new token individually, it is desirable to have a simple general strategy that governs the load transmission procedure. For example, if every generating token which resides in processor i creates four new tokens, a very simple strategy would be to send one of them to processor $i-1$, to hold two, and to send one to processor $i+2$, where the processor numbers are considered as congruent modulo the size n of the network and having edges to realize these transmissions.

The topic of this paper is the following. We assume the token generation process to grow a tree where, in one round, only the leaf nodes generate new tokens. Every leaf generates the same number of tokens. After the new tokens have been created in the processors of the network it has to be decided which tokens are held and which tokens are sent to which neighboring processors.

Previous Work. Gao/Rosenberg introduced in [2] the notion of *(regularly) evolving (binary) tree computations* on a ring of n processors. In their setting, a complete binary tree is grown. Initially, only one processor holds a generating token. Every generating token creates in a single round two new generating tokens, but will not create further tokens for the rest of the computation. The new tokens have to be spread among the processors according to a *regimen*. Gao/Rosenberg define the KS-regimen ("Keep-left-Send-right") that works on a ring of processors as follows: One new token stays at the processor where it has been generated, one token is sent to the, say, right neighbor of the processor where it has been generated. So, during the computation, the total load of generating and non-generating tokens accumulates in every processor. They show that this policy is asymptotically optimal, i. e., that the tokens are evenly distributed among the processors. More specifically, let the token generation process be run t rounds on a ring of n processors (so the total number of tokens is $M_t = 2^{t+1}-1$), and let $l_i^{(t)}$ the number of tokens in processor i. Then $|l_i^{(t)} - \frac{M_t}{n}| \leq (2\cos(\pi/n))^{t+1} = o(2^{t+1})$, and thus, $l_i^{(t)} = (1 \pm o(1)) \cdot \frac{M_t}{n}$.

In another seminal paper [1], Bhatt/Cai investigate evolving binary tree computations on d-dimensional hypercubes. The tree is allowed to grow arbitrarily, so not only leaves can generate a token in one round. New tokens are sent along a random path of length $O(\log d)$ to their destination. With M denoting the size of the tree, their algorithm ensures that the maximum load per processor is $O(1 + M/2^d)$, with high probability. To obtain this result, random walks on the hypercube are analyzed accurately. This approach has been further investigated by Li [4].

New Result. In this paper, we generalize the regularly evolving tree computations approach to arbitrary α-ary trees that grow level by level. Here, in one round every generating (leaf) token creates $\alpha = \alpha_1 + \cdots + \alpha_k$ new tokens, $k \geq 1$ and $\alpha_i \in \mathbb{N}$ fixed. Let the mentioned tokens be created in processor i. A global regimen states that α_j of these tokens are transmitted to processor $i + \delta_j$, $\delta_j \in \mathbb{Z}$ fixed, where the processor numbers are considered modulo the size n of the network

and having edges to realize these transmissions. For the Gao/Rosenberg KS-regimen, $(\alpha_1, \alpha_2) = (1, 1)$ and $(\delta_1, \delta_2) = (0, 1)$. We show that *any* global regimen on circulant graphs, i. e., rings of length n that have all chords that correspond to the δ_j sequence given above, is asymptotically optimal. That means that when we have after t rounds a tree of size $M_t = (\alpha^{t+1}-1)/(\alpha-1)$, the load in processor i is $l_i^{(t)} = (1 \pm o(1)) \cdot \frac{M_t}{n}$. Furthermore, we determine the parameter that governs the speed of the load balancing.

In order to prove this property, we describe global load-balancing regimens by generating polynomials which simplifies the analysis of the Gao/Rosenberg KS-regimen, or any regimen, considerably by applying algebraic tools.

Organization of Paper. In Section 2, we give a more formal description of evolving computations and regimens and state the result of this paper in this framework. In Section 3, we describe the token generation process by generating polynomials on an infinite linear array with chords. Then, in Section 4, we show that winding-up the infinite array on an n-cycle with chords can be modeled and analyzed by evaluating the generating polynomials for powers of nth primitive roots of unity. Finally, in Section 5 we demonstrate how this approach can be used for analyzing evolving tree computations on multi-dimensional tori with regular chords.

2 Definitions and Result

Let $\alpha \in \mathbb{N}$. An *evolving α-ary tree computation* is a complete α-ary tree that *grows* in rounds. Initially, there is only the root. In one *round*, every leaf of the tree generates α new leaves. So after t rounds, there are α^t leaves, and the size of the tree is $\frac{\alpha^{t+1}-1}{\alpha-1}$. As synonym for the term "node" we use the term "token." The leaves are called *generating* tokens, the interior nodes are *non-generating* tokens.

Let $\alpha = \alpha_1 + \cdots + \alpha_k$, $k \geq 1$, $\alpha_j \in \mathbb{N}$ fixed. The α-ary tree computation grows on a host network of processors that are numbered from 0 through $n-1$. For simplicity, think of the network as an n-cycle, i. e., processor j is connected to processor $(j+1) \bmod n$. The root of the tree computation is in some processor j. Let $(\delta_1, \ldots, \delta_k)$, $-n < \delta_j < n$ be fixed. The global $(\alpha_1, \ldots, \alpha_k; \delta_1, \ldots, \delta_k)$ *regimen* governs one round of the tree computation as follows: For every generating token stored in processor i, α_j new tokens are sent to processor $(i+\delta_j) \bmod n$, for every $j \in \{1, \ldots, k\}$.

If for every processor i the edges $\{i, (i+\delta_j) \bmod n\}$ are added to the n-cycle, we call the resulting graph a *circulant graph*. The graph is a *full* circulant graph, if there is only one connected component induced by the edges defined by the δ-sequence. E. g., when $n = 6$ and $(\delta_1, \delta_2) = (0, 2)$, there are two connected components, whereas, when $n = 5$ there is only one. An example of an evolving 4ary tree computation with $n = 6$ and the $(1, 2, 1; 0, 1, 2)$-regimen is given in Fig. 1.

Note that the dilation of the tree edges in the host network is 1.

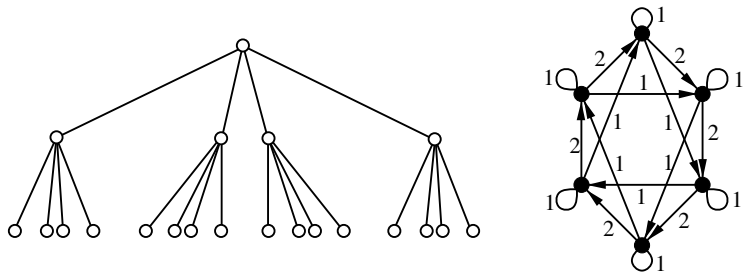


Fig. 1. An evolving 4-ary tree computation after $t = 2$ rounds and the circulant host network with $n = 6$ corresponding to the $(1, 2, 1; 0, 1, 2)$-regimen and to $p(x) = 1 + 2x + x^2$.

In the rest of this paper, let $a_i^{(t)}$ denote the number of generating tokens stored in processor i after round t. Similarly, let $l_i^{(t)}$ denote the total number of tokens stored in processor i.

We prove the following theorem.

Theorem 1. *For any global $(\alpha_1, \ldots, \alpha_k; \delta_1, \ldots, \delta_k)$ regimen on a full circulant graph of size n described by the δ-sequence holds, as t advances:*

$$l_i^{(t)} = \big(1 \pm o(1)\big) \cdot \frac{1}{n} \cdot \frac{\alpha^{t+1} - 1}{\alpha - 1}$$

That means that with increasing number of rounds the load in every processor comes closer and closer to the average load, independent of the used regimen. In a certain way this resembles the properties of random walk on a cycle where after some time the probability distribution comes close to the uniform distribution.

3 The Infinite Setting and the Generating Polynomials

Before we turn to the cycle, we first show how an evolving tree computation and a global $(\alpha_1, \ldots, \alpha_k; \delta_1, \ldots, \delta_k)$-regimen can be modeled on an *infinite* linear array as the tree's host network. In this array, the processors are numbered from $-\infty$ through ∞. As in circulant graphs, we assume that each processor i is connected to all processors $i + \delta_j$. For processor i, let $a_i^{(t)}$ denote the number the generating tokens stored in i after round t, and let $l_i^{(t)}$ denote the total number of tokens in processor i.

As this infinite graph is invariant under shifts right and left, we assume, w. l. o. g., that the root is stored in processor 0. Hence, $a_0^{(0)} = l_0^{(0)} = 1$.

We describe the distributions of the generating tokens and of all tokens among the processors after round t by two generating functions,

$$a^{(t)}(x) = \sum_{i=-\infty}^{\infty} a_i^{(t)} \cdot x^i \qquad \text{and} \qquad l^{(t)}(x) = \sum_{i=-\infty}^{\infty} l_i^{(t)} \cdot x^i \ ,$$

resp., and, for a global $(\alpha_1, \ldots, \alpha_k; \delta_1, \ldots, \delta_k)$-regimen, let

$$p(x) = \sum_{j=1}^{k} \alpha_j \cdot x^{\delta_j} \quad .$$

E. g., for the $(1, 1; 0, 1)$-regimen (i. e., the KS-regimen), $p(x) = 1 + x$, and for the $(1, 2, 1; -1, 0, 1)$-regimen, $p(x) = x^{-1} + 2 + x$.

Note that the total number of tokens is simply $l^{(t)}(1)$.

$p(x)$ describes the regimen completely, and calculations with it can simulate the evolving tree computation because there is the following connection between the regimen, the tree computation and the polynomials.

Lemma 1. *Let the global $(\alpha_1, \ldots, \alpha_k; \delta_1, \ldots, \delta_k)$-regimen in the infinite network setting be given. For all $t \geq 1$,*

(a) $a^{(t)}(x) = p(x)^t$

(b) $l^{(t)}(x) = \sum_{\tau=0}^{t} p(x)^t = \dfrac{p(x)^{t+1} - 1}{p(x) - 1}$

Proof. (a) $a_i^{(t)} \cdot x^i$ is the monomial in $a^{(t)}(x)$ that describes the number $a_i^{(t)}$ of generating tokens in processor i. These tokens create, for $j \in \{1, \ldots, k\}$, $a_i^{(t)} \cdot \alpha_j$ new tokens that have to be transmitted to processor $i + \delta_j$, i. e.,

$$\sum_{j=1}^{k} a_i^{(t)} \cdot \alpha_j \cdot x^{i+\delta_j} = a_i^{(t)} \cdot x^i \cdot \sum_{j=1} \alpha_j \cdot x^{\delta_j} = a_i^{(t)} \cdot x^i \cdot p(x)$$

must, as the contribution of processor i, be added to $a^{(t+1)}(x)$. Thus $a^{(t+1)}(x) = p^{(t)}(x) \cdot p(x)$, which completes the proof of (a).

(b) follows from (a) and the fact that $l^{(t)}(x) = \sum_{\tau=0}^{t} a^{(\tau)}(x)$. □

Hence, the load distribution of the KS-regimen with polynomial $p(x) = 1 + x$ is given by $l^{(t)}(x) = x^{-1} \cdot ((1 + x)^{t+1} - 1)$. Due to symmetry, the "reversed" KS-regimen $p(x) = x^{-1} + 1$ results in the same sequence of loads except that the number of tokens that has been in processor i, is in this case in processor $-i$. So, also $l^{(t)} = x \cdot (1 + 1/x)^{t+1} - x$ returns the same load per processor as the KS-regimen. In general, we can multiply $p(x)$ by x^j, $j \in \mathbb{Z}$, without changing the load sequence. A further consequence is that with $p(x) = 1 + x$ and $\hat{p}(x) = p(x)^2/x = x^{-1} + 2 + x$, the two regimens result, after $2t$ and t rounds, resp., in the same load sequence.

Of course, in the infinite setting the regimen cannot result in a good distribution of the total load that is obviously $l^{(t)}(1)$. E. g., with $p(x) = 1 + x$, in $l^{(t)}(x)$ the largest coefficient is $\binom{t+1}{\lfloor t/2 \rfloor}$, and the smallest non-zero coefficient is 1.

In the next section, we will show that $l^{(t)}(x)$ also can be used to analyze the finite setting.

4 The Finite Setting: Winding up the Infinite Array

In the following, we show that winding up the infinite array and its load on the n-cycle leads to an exact description of the evolving tree computation on the cycle (and its chords). As we have seen above, the load difference can be very large in the infinite setting. However, the winding-up flattens this difference dramatically.

Let $p(x)$ be a regimen, and let $\hat{l}^{(t)}(x) = \sum_{i=-\infty}^{\infty} \hat{l}_i^{(t)} x^k$ the resulting load polynomial in the infinite setting. Consider the same regimen on the n-cycle (with appropriate chords). Then the load $l_i^{(t)}$ of processor i of the cycle is related to the infinite setting as follows.

Lemma 2. *For the load $l_i^{(t)}$ of processor i on the n-cycle, $l_i^{(t)} = \sum_{k=-\infty}^{\infty} \hat{l}_{i+kn}^{(t)}$*

This lemma follows from the fact that generating tokens are not influenced by other generating tokens. So they can be bijectively identified in the cycle and the infinite array.

In the following, we shall see how we can calculate $l_i^{(t)}$ by evaluating $\hat{l}^{(t)}(x)$ for appropriate complex numbers x. Let $\omega_n = e^{2\mathrm{i}\pi/n} = \cos(2\pi/n) + \mathrm{i}\sin(2\pi/n)$ be an nth primitive root of unity. ω_n has the the following important properties:

- $\sum_{j=0}^{n-1} \omega_n^j = 0$ and
- $\omega_n^i = \omega_n^{i+k\cdot n}$ for all k.

Lemma 3. *In the finite setting, the load $l_i^{(t)}$ of processor i after t rounds is*

$$l_i^{(t)} = \frac{1}{n} \cdot \sum_{j=0}^{n-1} \omega_n^{-ji} \cdot \frac{p(\omega_n^j)^{t+1} - 1}{p(\omega_n^j) - 1} \quad .$$

Proof.

$$\frac{1}{n} \cdot \sum_{j=0}^{n-1} \omega_n^{-ji} \cdot \hat{l}^{(t)}(\omega_n^j) = \frac{1}{n} \cdot \sum_{k=-\infty}^{\infty} \hat{l}_k^{(t)} \cdot \underbrace{\sum_{j=0}^{n-1} \omega_n^{j(k-i)}}_{=z} = \sum_{k=-\infty}^{\infty} \hat{l}_{i+kn}^{(t)}$$

By the properties of ω_n, $z = n$, if $k - i = 0 \bmod n$, and $z = 0$ otherwise. By Lemmata 1(b) and 2, the statement follows. □

The technique used above is called *multisection* (e g., see [3, p. 89]).

In the case of the n-cycle and the "two rounds" KS-regimen, i. e., $p(x) = (1+x)(1+x^{-1}) = x^{-1} + 2 + x$, it is particularly easy to get rid of the complex numbers, so we have explicitly

$$\begin{aligned} l_i^{(t)} &= \frac{1}{n} \cdot \sum_{j=0}^{n-1} \omega_n^{-ji} \cdot \frac{(\omega_n^{-j} + 2 + \omega_n^j)^{t+1} - 1}{\omega_n^{-j} + 1 + \omega_n^j} \\ &= \frac{1}{n} \cdot \sum_{j=0}^{n-1} \cos\Big(\frac{2\pi ji}{n}\Big) \cdot \frac{(2 + 2\cos(\frac{2j\pi}{n}))^{t+1} - 1}{1 + 2\cos(\frac{2j\pi}{n})} \quad . \end{aligned}$$

Now we show that $l_i^{(t)}$ comes very close to the average load when t becomes large. More specifically, we show that the deviation from the average load becomes small.

Let $\lambda = \max_{0<j<n} |p(\omega_n^j)| = |p(\omega_n)|$. Note that $\lambda^t = o(\alpha^t)$ for growing t as $\lambda < p(1) = \alpha$, and recall that $\hat{l}^{(t)}(1)$ is the total number of tokens after t rounds.

Let $\Delta_i^{(t)} = l_i^{(t)} - \frac{\hat{l}^{(t)}(1)}{n}$ be the deviation of the load in processor i after round t from the average load.

Lemma 4.

$$|\Delta_i^{(t)}| \leq \frac{n-1}{n} \cdot \frac{\lambda^{t+1}-1}{\lambda-1}$$

Proof. By applying Lemma 3, we obtain

$$l_i^{(t)} = \frac{1}{n} \cdot \sum_{j=0}^{n-1} \omega_n^{-ji} \cdot \hat{l}^{(t)}(\omega_n^j) = \frac{\hat{l}^{(t)}(1)}{n} + \underbrace{\frac{1}{n} \cdot \sum_{j=1}^{n-1} \omega_n^{-ji} \cdot \frac{p(\omega_n^j)^{t+1}-1}{p(\omega_n^j)-1}}_{=\Delta_i^{(t)}} .$$

Now,

$$|\Delta_i^{(t)}| \leq \frac{1}{n} \cdot \sum_{j=1}^{n-1} \left| \frac{p(\omega_n^j)^{t+1}-1}{p(\omega_n^j)-1} \right| \leq \frac{1}{n} \cdot \sum_{j=1}^{n-1} \sum_{\tau=0}^{t} \left| p(\omega_n^j) \right|^{\tau} \leq \frac{n-1}{n} \cdot \frac{\lambda^{t+1}-1}{\lambda-1}$$

The first two estimations are true because of the Triangle Inequality. The ω_n^{-ji} factor can be dropped from the sum because of the Cauchy-Schwarz Inequality. □

As a consequence of Lemma 4, we have that after at most $t^* \approx \log_{\alpha/\lambda}(\frac{1}{4\varepsilon})$ rounds

$$\frac{|\Delta_i^{(t^*)}|}{\hat{l}^{(t^*)}} \leq \varepsilon$$

for arbitrary $\varepsilon > 0$ so that the larger α/λ, the faster the loads approach the average load. Immediately, as $\lambda^t = o(\alpha^t)$, we can conclude:

Theorem 1. *For any global regimen described by $p(x)$ on a full circulant graph holds with total load $\hat{l}^{(t)}(1)$ and as t advances:*

$$l_i^{(t)} = \big(1 \pm o(1)\big) \cdot \frac{\hat{l}^{(t)}(1)}{n}$$

For the "two rounds" KS-regimen mentioned above, $\lambda = 2 + 2\cos(2\pi/n) = 4\cos^2(\pi/n) \approx 4 - \frac{4\pi^2}{n^2}$.

5 Related Networks and Remarks

Now we consider evolving tree computations on a d-dimensional torus as host network. To model the development of the loads, we can first use a d-dimensional

array that is infinite in all directions. For every dimension j, a formal variable x_j can be used, so this time the generating polynomial for the loads is $l^{(t)}(x_1, \ldots, x_d)$. Here, the regimen can be described by a polynomial $p(x_1, \ldots, x_d)$. When we have an $n_1 \times \cdots \times n_d$ torus (with appropriate internal chords reflecting the regimen), we have to do multisection with $\omega_{n_1}, \ldots, \omega_{n_d}$ as described above for the one-dimensional case. Then we can repeat the calculations of the proof of Lemma 4 which also shows that any regimen eventually leads to a good distribution of the tokens.

Our analysis also extends to the case where more than one tree grows. In this case, $a^{(0)}(x)$ is not only just 1, but a polynomial. Also more complex regimens can be analyzed with our approach. Here, a regimen can consist of several polynomials $p_1(x), \ldots, p_m(x)$. Obviously, the development of loads is then described by the polynomial that is the product of the $p_j(x)$.

Evolving tree computations are very similar to Random Walk on the host network. Besides using integers, the difference is the accumulation of load in a single processor. This leads to the following interesting observation: Consider the regimen $p(x) = x^{-1} + x$ on the ring of length 4. After every round, only two processors contain generating tokens, that means, the generating tokens are not balanced among the processors, we have something like a periodic Random Walk. On the other hand, the accumulated number of tokens per processor approaches the average number.

In [5], a Random Walk approach is used to analyze diffusive load-balancing algorithms. Here the number of tokens and their distribution is fixed, and the goal is to distribute the tokens evenly among the processors. In order to upper bound the deviation between the Random Walk where rational number occur and the distribution, in [5] the measure *local divergence* has been introduced which is similar in some respects to our $l_i^{(t)}$. The local divergence accumulates the maximum possible deviation between the integer and the rational process per round. Future work could relate the local divergence to evolving tree computations.

References

1. S. Bhatt and J.-Y. Cai. Taking random walks to grow trees in hypercubes. *J. ACM*, 40:741–764, 1993.
2. L.-X. Gao and A. L. Rosenberg. Toward efficient scheduling of evolving computations on rings of processors. *J. Parallel and Distributed Comp.*, 38:92–100, 1996.
3. D. E. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley, Reading, 3rd edition, 1997.
4. K. Li. Asymptotically optimal randomized tree embedding in static networks. In *Proc. 1st Merged Int. Parallel Processing Symp. and Symp. on Parallel and Distributed Processing (IPPS/SPDP)*, pp. 423–430, 1998.
5. Y. Rabani, A. Sinclair, and R. Wanka. Local divergence of Markov chains and the analysis of iterative load-balancing schemes. In *Proc. 39th IEEE Symp. on Foundations of Computer Science (FOCS)*, pp. 694–703, 1999.

Author Index