

Jiří Fiala  
Jan Kratochvíl  
Mirka Miller (Eds.)

LNCS 5874

# Combinatorial Algorithms

20th International Workshop, IWOCA 2009  
Hradec nad Moravicí, Czech Republic, June/July 2009  
Revised Selected Papers

 Springer

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

## Editorial Board

David Hutchison

*Lancaster University, UK*

Takeo Kanade

*Carnegie Mellon University, Pittsburgh, PA, USA*

Josef Kittler

*University of Surrey, Guildford, UK*

Jon M. Kleinberg

*Cornell University, Ithaca, NY, USA*

Alfred Kobsa

*University of California, Irvine, CA, USA*

Friedemann Mattern

*ETH Zurich, Switzerland*

John C. Mitchell

*Stanford University, CA, USA*

Moni Naor

*Weizmann Institute of Science, Rehovot, Israel*

Oscar Nierstrasz

*University of Bern, Switzerland*

C. Pandu Rangan

*Indian Institute of Technology, Madras, India*

Bernhard Steffen

*University of Dortmund, Germany*

Madhu Sudan

*Microsoft Research, Cambridge, MA, USA*

Demetri Terzopoulos

*University of California, Los Angeles, CA, USA*

Doug Tygar

*University of California, Berkeley, CA, USA*

Gerhard Weikum

*Max-Planck Institute of Computer Science, Saarbruecken, Germany*

Jiří Fiala Jan Kratochvíl  
Mirka Miller (Eds.)

# Combinatorial Algorithms

20th International Workshop, IWOCA 2009  
Hradec nad Moravicí, Czech Republic, June 28–July 2, 2009  
Revised Selected Papers

## Volume Editors

Jiří Fiala

Jan Kratochvíl

Charles University

Institute for Theoretical Computer Science and

Department of Applied Mathematics

Malostranské nám. 25, 118 00 Prague 1, Czech Republic

E-mail: {fiala,honza}@kam.mff.cuni.cz

Mirka Miller

The University of Newcastle

School of Electrical Engineering and Computer Science

University Drive, Callaghan NSW 2308, Australia

E-mail: mirka.miller@newcastle.edu.au

Library of Congress Control Number: 2009938817

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

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

ISSN 0302-9743

ISBN-10 3-642-10216-6 Springer Berlin Heidelberg New York

ISBN-13 978-3-642-10216-5 Springer Berlin Heidelberg New York

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

springer.com

© Springer-Verlag Berlin Heidelberg 2009

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper SPIN: 12784363 06/3180 5 4 3 2 1 0



# Preface

The 20th International Workshop on Combinatorial Algorithms was held during June 28 – July 2, 2009 in the picturesque castle of Hradec nad Moravicí, located in the north-east corner of the Czech Republic.

IWOCA — the workshop that originated 19 years ago as AWOCA — made a big step towards globalization this year. After 19 conferences held in Australia, Indonesia, Korea, and Japan, the 20th anniversary was celebrated by taking the conference outside the Australasian region for the first time. Another novelty this year was that the proceedings are being published by Springer in the LNCS series.

Our Call for Papers brought an overwhelming response of the combinatorial community. IWOCA 2009 received over 100 submissions, more than twice the amount it received before. Most of the submissions were of exceptionally high quality and thus the Program Committee was faced with hard work and sometimes hard decisions. Many very good papers had to be rejected because of the limited capacity of the conference schedule. In the end, 41 contributed talks were presented during the conference — the maximum number that we could fit in the program.

We would like to thank all who sent their submissions and to congratulate all the authors of the accepted papers. They contributed to what was a most successful conference. We also thank all the authors who submitted posters for the poster session (not included in the proceedings).

We extend special thanks to the distinguished invited speakers. We believe that we offered the participants a rich selection of plenary speakers who are all well-known experts in the area of algorithmic combinatorics and graph theory. Sue Whitesides introduced the issues of computational geometry. Giovanni Manzini gave a splendid overview of data compression algorithms on strings and trees. Robin Thomas revealed intriguing details of clique minors in highly connected graphs. Mike Fellows, one of the founders of fixed parameter complexity theory, surveyed recent progress in this area of theoretical computer science. A cherry on the cake was undoubtedly the participation by Jack Edmonds, the guru of algorithmic mathematics. We were very happy that Jack accepted our invitation and agreed to give an honorary lecture on branching systems, with glimpses of matching and matroid theory as well as lush memories of the early days of combinatorial optimization. We trust that the special setting of Jack's lecture in the historical premises of the castle was a fitting gift for his recent 75th birthday. Jack's interaction with other participants during scientific discussions and social events was one of the highlights of the conference.

We were very pleased to see a large number of students and young researchers among the participants at IWOCA 2009. For the first time in the history of IWOCA, the PC considered awarding a prize for the best student paper. How-

ever, since the quality of the top three student papers was uniformly very high, it was decided to share the prize (sponsored by DIMATIA Charles University) between the authors of these papers. We extend our congratulations to Thomas Feierl for his paper “The Height and Range of Watermelons Without Wall,” to Paweł Gawrychowski and Travis Gagie for “Minimax Trees in Linear Time with Applications,” and to Petr Škoda for “Computability of Branch-Width of Submodular Partition Functions”.

We thank all the members of the Program Committee, all referees and all the members of the Organizing Committee for all the hard work they have done. While all committee members worked well as a team, some names must be singled out: Special thanks go to Yuqing Lin for editing the proceedings that were made available during the conference, to Alan Gibbons and Joe Ryan for organizing the problem session (see [graphtheorygroup.com/iwocaproblems.html](http://graphtheorygroup.com/iwocaproblems.html)), and to Dalibor Fronček and Petr Kovář for the exceptionally successful running of the local organization. We also gratefully acknowledge sponsorship of the conference and preparation of the proceedings by Czech research grants 1M0545 and MSM0021620838.

August 2009

Jiří Fiala  
Jan Kratochvíl  
Mirka Miller

# Conference Organization

## Program Committee

- |                                  |  |
|----------------------------------|--|
| L. Brankovic (Newcastle)         | B. Melichar (Prague)                               |
| H. Broersma (Durham)             | M. Miller (Co-chair,<br>Newcastle, Pilsen, London) |
| C. Colbourn (Tempe)              | R. Nedela (Banská Bystrica)                        |
| G. Di Battista (Rome)            | M. Noy (Barcelona)                                 |
| M. Drmota (Vienna)               | C. Pandu Rangan (Madras)                           |
| J. Fiala (Prague)                | K. Park (Seoul)                                    |
| D. Fronček (Duluth and Opava)    | D. Paulusma (Durham)                               |
| T. Fujito (Toyohashi)            | A. Proskurowski (Eugene)                           |
| P. Hliněný (Brno)                | Z. Ryjáček (Pilsen)                                |
| J. Holub (Prague)                | J. Seberry (Wollongong)                            |
| X. D. Hu (Beijing)               | J. Širáň (Bratislava and<br>Milton-Keynes)         |
| L. Kang (Shanghai )              | J. A. Telle (Bergen)                               |
| M. Klazar (Prague)               | A. K. Tsakalidis (Patras)                          |
| J. Kratochvíl (Co-chair, Prague) | P. Valtr (Prague)                                  |
| S. Kucukcifici (Istanbul)        | K. Wada (Nagoya)                                   |
| G. Landau (Haifa)                | X. Zhu (Kaohsiung)                                 |
| T. Lecroq (Rouen)                |  |
| Z. Lonc (Warsaw)                 |  |
| D. Marx (Budapest)               |  |

## Problem Session Chairs

- A. Gibbons (London)
- J. Ryan (Newcastle)

## Steering Committee

- C. Iliopoulos (London)
- M. Miller (Newcastle, Pilsen, London)
- B. Smyth (Hamilton and Perth)

## Organizing Committee

- |                                      |                       |
|--------------------------------------|-----------------------|
| J. Fiala (Prague)                    | T. Kovářová (Ostrava) |
| D. Fronček (Chair, Duluth and Opava) | M. Kubesa (Ostrava)   |
| P. Kabelíková (Ostrava)              | Y. Lin (Newcastle)    |
| P. Kovář (Ostrava)                   |                       |

## Organizing Institutions

Combinatorica NPO

Department of Applied Mathematics, VŠB-TU Ostrava — the conference was partially supported by the institutional project MSM6198910027.

## Sponsors

DIMATIA and ITI, Charles University, Prague  
GTA, University of Newcastle

## External Reviewers

S. Abdeddaïm	M. Giraud	A. Piperno
P. Adonellis	P. Golovach	E. Prieur
P. Angelini	D. J. Guan	M. Raffinot
T. Araki	O. Guth	I. Razgon
M. Balík	H. Hajiabolhassan	A. Rosa
P. Bonsma	C. Hespel	P. Rossmanith
N. R. Brisaboa	P. Holub	F. Ruskey
J. Brousek	S.-W. Huang	J. Sawada
B.-M. Bui-Xuan	T. Ishii	J. Sgall
L. Cai	T. Izumi	E. Shan
W. Chen	J. Jaroš	N. N. Schraudolph
Y. Cheng	V. Jelínek	B. Smith
M. Chimani	T. Kaiser	E. Sopena
J. Cibulka	M. Kamiński	S. Souissi
S. Clark	Y. Katayama	R. van Stee
R. Čada	J. Kolář	M. Škoviera
W. Didimo	P. Kolman	O. Šuch
M. Dom	R. Kužel	J. Tang
Z. Dvořák	A. Lefebvre	D. Tang
T. Ebenlendr	Y. Lin	J. Teska
T. Ekim	C. Liu	C. Tsihlas
S. El-Zanati	M. Loeb	N. Tsirakis
E. Elkind	K. Marshall	M. Villanueva
A. Farley	D. Meister	S. Wagner
C. Fernández-Córdoba	Y. Nekrich	W.-H. Wang
H. Fernau	M. Noellenburg	M. Weller
T. Flouri	J. Obdržálek	K. Yamazaki
F. Frati	H. Ono	W. Yang
T. Fukunaga	G. Ottaviano	E. Yazici
E. Fusy	S. Oum	H.-G. Yeh
R. Ganian	S. Ozkan	A. Zelikovsky
T. Gavenčiak	M. Patrignani	L. Zhao
P. Giannopoulos	O. Pikhurko	J. Žd'árek

# Table of Contents

## Invited Talks

Branching Systems (Abstract) . . . . .	1
<i>Jack Edmonds</i>	
Towards Fully Multivariate Algorithmics: Some New Results and Directions in Parameter Ecology . . . . .	2
<i>Michael Fellows</i>	
Succinct Representations of Trees . . . . .	11
<i>Giovanni Manzini</i>	
$K_t$ Minors in Large $t$ -Connected Graphs (Abstract) . . . . .	19
<i>Robin Thomas</i>	
Intractability in Graph Drawing and Geometry: FPT Approaches . . . . .	20
<i>Sue Whitesides</i>	

## Contributed Talks

Evaluation of Recoverable-Robust Timetables on Tree Networks . . . . .	24
<i>Gianlorenzo D'Angelo, Gabriele Di Stefano, and Alfredo Navarra</i>	
Weighted LCS (Extended Abstract) . . . . .	36
<i>Amihoud Amir, Zvi Gotthilf, and B. Riva Shalom</i>	
Integrality Properties of Certain Special Balanceable Families . . . . .	48
<i>Nicola Apollonio and Massimiliano Caramia</i>	
Forbidden Subgraph Colorings and the Oriented Chromatic Number . . . . .	60
<i>N.R. Aravind and C.R. Subramanian</i>	
Polynomial Kernels for 3-Leaf Power Graph Modification Problems . . . . .	72
<i>Stéphane Bessy, Christophe Paul, and Anthony Perez</i>	
Approximating the Max Edge-Coloring Problem . . . . .	83
<i>Nicolas Bourgeois, Giorgio Lucarelli, Ioannis Milis, and Vangelis Th. Paschos</i>	
Three Complexity Results on Coloring $P_k$ -Free Graphs . . . . .	95
<i>Hajo Broersma, Fedor V. Fomin, Petr A. Golovach, and Daniël Paulusma</i>	
Fully Decomposable Split Graphs . . . . .	105
<i>Hajo Broersma, Dieter Kratsch, and Gerhard J. Woeginger</i>	

Feedback Vertex Set on Graphs of Low Cliquewidth . . . . .	113
<i>Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle</i>	
Note on Decomposition of $K_{n,n}$ into $(0, j)$ -prisms . . . . .	125
<i>Sylwia Cichacz, Dalibor Fronček, and Petr Kovář</i>	
Edge-Simple Circuits through 10 Ordered Vertices in Square Grids . . . . .	134
<i>David Coudert, Frédéric Giroire, and Ignasi Sau</i>	
Efficient Neighborhood Encoding for Interval Graphs and Permutation Graphs and $O(n)$ Breadth-First Search . . . . .	146
<i>Christophe Crespelle and Philippe Gambette</i>	
LPF Computation Revisited . . . . .	158
<i>Maxime Crochemore, Lucian Ilie, Costas S. Iliopoulos, Marcin Kubica, Wojciech Rytter, and Tomasz Waleń</i>	
Limiting Distribution for Distances in $k$ -Trees . . . . .	170
<i>Alexis Darrasse and Michèle Soria</i>	
Gray Code Compression . . . . .	183
<i>Darko Dimitrov, Tomáš Dvořák, Petr Gregor, and Riste Škrekovski</i>	
Embedded Trees and the Support of the ISE . . . . .	194
<i>Michael Drmota</i>	
Combinatorial Models for Cooperation Networks . . . . .	206
<i>Michael Drmota, Bernhard Gittenberger, and Reinhard Kutzelnigg</i>	
Polar Permutation Graphs . . . . .	218
<i>Tınaz Ekim, Pinar Heggernes, and Daniel Meister</i>	
A New Algorithm for Efficient Pattern Matching with Swaps . . . . .	230
<i>Matteo Campanelli, Domenico Cantone, and Simone Faro</i>	
The Height and Range of Watermelons without Wall (Extended Abstract) . . . . .	242
<i>Thomas Feierl</i>	
Fast Convolutions and Their Applications in Approximate String Matching . . . . .	254
<i>Kimmo Fredriksson and Szymon Grabowski</i>	
Better Polynomial Algorithms on Graphs of Bounded Rank-Width . . . . .	266
<i>Robert Ganian and Petr Hliněný</i>	
Minimax Trees in Linear Time with Applications . . . . .	278
<i>Paweł Gawrychowski and Travis Gagie</i>	
Planar Biconnectivity Augmentation with Fixed Embedding . . . . .	289
<i>Carsten Gutwenger, Petra Mutzel, and Bernd Zey</i>	

Trivially-Perfect Width . . . . .	301
<i>Ling-Ju Hung, Ton Kloks, and Chuan Min Lee</i>	
Lightweight Parameterized Suffix Array Construction . . . . .	312
<i>Tomohiro I, Satoshi Deguchi, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda</i>	
On the Crossing Numbers of Cartesian Products of Stars and Graphs on Five Vertices . . . . .	324
<i>Marián Klešč</i>	
Factorizations of Complete Graphs into Spanning Trees with All Possible Maximum Degrees . . . . .	334
<i>Petr Kovář and Michael Kubesa</i>	
On the Maximal Number of Cubic Subwords in a String . . . . .	345
<i>Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń</i>	
Solution of Peter Winkler's Pizza Problem . . . . .	356
<i>Josef Cibulka, Jan Kynčl, Viola Mészáros, Rudolf Stolař, and Pavel Valtr</i>	
An $O(n)$ -Time Algorithm for the Paired-Domination Problem on Permutation Graphs . . . . .	368
<i>Evangelos Lappas, Stavros D. Nikolopoulos, and Leonidas Palios</i>	
Simpler Parameterized Algorithm for OCT . . . . .	380
<i>Daniel Lokshтанov, Saket Saurabh, and Somnath Sikdar</i>	
Bipartite Graphs of Large Clique-Width . . . . .	385
<i>Nicholas Korpelainen and Vadim V. Lozin</i>	
Kernel in Oriented Circulant Graphs . . . . .	396
<i>Paul Manuel, Indra Rajasingh, Bharati Rajan, and Joice Punitha</i>	
Randomized Postoptimization of Covering Arrays . . . . .	408
<i>Peyman Nayeri, Charles J. Colbourn, and Goran Konjevod</i>	
New Word-Based Adaptive Dense Compressors . . . . .	420
<i>Petr Procházka and Jan Holub</i>	
Rainbow Connection in Graphs with Minimum Degree Three . . . . .	432
<i>Ingo Schiermeyer</i>	
The Complexity of Almost Perfect Matchings in Uniform Hypergraphs with High Codegree . . . . .	438
<i>Edyta Szymańska</i>	

Computability of Width of Submodular Partition Functions . . . . .	450
<i>Petr Škoda</i>	
The Guarding Problem – Complexity and Approximation . . . . .	460
<i>Thirumala Reddy T.V., Sai Krishna D., and Pandu Rangan C.</i>	
Antibandwidth of d-Dimensional Meshes . . . . .	471
<i>Ľubomír Török and Imrich Vrto</i>	
<b>Author Index</b> . . . . .	479



# Branching Systems

Jack Edmonds

Equipe Combinatoire et Optimisation  
Universite Paris VI

For a digraph  $G = (V, E)$  with a specified subset  $R(j)$  of  $V$ , its nodes, a branching  $B(j)$  rooted at  $R(j)$  is a forest in  $G$  such that for each node  $u$  in  $V - R(j)$  there is exactly one edge of  $B(j)$  entering  $u$ . A branching system  $B = [B(j) : j \in J]$  is a collection of edge-disjoint branchings, with specified root-sets, in  $G$ . Given costs  $c(i)$  on the edges  $i$  of  $G$ , and given root sets  $R(j)$ , we survey the use of matroids to find a least cost branching system,  $B$ .



Jack Edmonds giving his plenary lecture at IWOCA 2009.  
(Photo ©Rudolf Stolař)

# Towards Fully Multivariate Algorithmics: Some New Results and Directions in Parameter Ecology

Michael Fellows

University of Newcastle, Callaghan, Australia  
michael.fellows@newcastle.edu.au

**Abstract.** This paper reports on three recent research directions that begin to explore the subject of fully multivariate algorithmics, meaning by this the largely uncharted theoretical landscape that lies beyond parameterized complexity and algorithmics — itself a natural two-dimensional generalization of the familiar one-dimensional framework of  $\mathcal{P}$  versus  $\mathcal{NP}$ .

## 1 Introduction

Parameterized complexity starts from the premise that there are usually secondary measurements, apart from the primary measurement of overall input size, that can significantly affect the computational complexity of a problem, in qualitatively different ways that merit systematic investigation. Parameterized complexity makes room for one such measurement, the *parameter*, and turns on the contrast between the classes of bivariate functions defining  $\mathcal{FPT}$  and  $\mathcal{XP}$ , respectively.

This is formalized by saying that a *parameterized problem*  $\Pi$  takes as input a pair  $(x, k)$  where  $k$  is the *parameter* (usually a positive integer). The problem  $\Pi$  is *fixed-parameter tractable* if it can be solved in time  $f(k)n^c$  where  $n$  is the overall input size, that is,  $n = |(x, k)|$ ,  $c$  is a constant, and  $f(k)$  is some function of the parameter  $k$ .  $\mathcal{XP}$  is the class of parameterized problems that are solvable in time  $O(n^{g(k)})$ .

Viewing complexities theories as driven by contrasting function classes, the classical framework of  $\mathcal{P}$  versus  $\mathcal{NP}$  is “about” the contrast between the univariate function classes:  $\mathcal{P}$ , solvability in time  $O(n^c)$ , and functions of the form  $O(2^{n^c})$ . Hardness for  $\mathcal{NP}$  indicates that you will probably be stuck with a running time of the latter kind.

$\mathcal{FPT}$  generalizes  $\mathcal{P}$ , and parameterized complexity is a natural 2-dimensional sequel to the familiar one-dimensional classical framework. Of course, there may be more than one relevant measurement beyond the overall input size. So what is the natural 3-dimensional sequel? Could there be a mathematically elegant and useful *fully multivariate* framework? Presently, there are no satisfactory answers to these questions. In view of the theoretical and practical successes of parameterized complexity and algorithmics it is natural to investigate them. We report here on three recent research directions that begin to explore the possibilities for *fully multivariate algorithmics*.

## 2 Background on Parameterized Complexity

Parameterized complexity analysis unfolds in the contrast between the “good class” of bivariate functions FPT, and the “bad class” of runtimes of the form  $O(n^{g(k)})$  — solvability in such time defines the parameterized complexity class XP. To emphasize the contrast, one could also consider defining FPT *additively* as solvability in time  $f(k) + n^c$ . It turns out it makes no difference whether the parameterized complexity class FPT is defined multiplicatively or additively [13,10]. The basic issue in parameterized complexity is whether any exponential costs of the problem can be confined to the parameter.

### 2.1 Some Motivating Examples

#### TYPE CHECKING IN ML.

ML is a logic-based programming language for which relatively efficient compilers exist. One of the problems the compiler must solve is the checking of the compatibility of type declarations. This problem is known to be complete for EXP (deterministic exponential time) [20], so the situation appears discouraging from the standpoint of classical complexity theory. However, the implementations work well in practice because the ML TYPE CHECKING problem is FPT with a running time of  $O(2^k n)$ , where  $n$  is the size of the program and  $k$  is the maximum nesting depth of the type declarations [22]. Since normally  $k \leq 5$ , the algorithm is clearly practical on the natural input distribution.

The central issue in parameterized complexity can also be illustrated by fundamental problems about graphs. Consider the following well-known problems:

#### VERTEX COVER

*Input:* A graph  $G = (V, E)$  and a positive integer  $k$ .

*Question:* Does  $G$  have a vertex cover of size at most  $k$ ? (A *vertex cover* is a set of vertices  $V' \subseteq V$  such that for every edge  $uv \in E$ ,  $u \in V'$  or  $v \in V'$  (or both).)

#### DOMINATING SET

*Input:* A graph  $G = (V, E)$  and a positive integer  $k$ .

*Question:* Does  $G$  have a dominating set of size at most  $k$ ? (A *dominating set* is a set of vertices  $V' \subseteq V$  such that  $\forall u \in V: u \in N[v]$  for some  $v \in V'$ .)

Although both problems are NP-complete, the input *parameter*  $k$  contributes to the complexity of these two problems in two qualitatively different ways.

1. There is a simple *bounded search tree* algorithm for VERTEX COVER that runs in time  $O(2^k n)$
2. The best known algorithm for DOMINATING SET is only a minor improvement on the brute force algorithm of trying all  $k$ -subsets. For a graph on  $n$  vertices this approach has a running time of  $O(n^{k+1})$ .

The table below shows the contrast between these two kinds of complexity.

In these two example problems, the parameter is the size of the solution being sought. But a parameter that affects the complexity of a problem can be many

**Table 1.** The Ratio  $\frac{n^{k+1}}{2^{kn}}$  for Various Values of  $n$  and  $k$ 

	$n = 50$	$n = 100$	$n = 150$
$k = 2$	625	2,500	5,625
$k = 3$	15,625	125,000	421,875
$k = 5$	390,625	6,250,000	31,640,625
$k = 10$	$1.9 \times 10^{12}$	$9.8 \times 10^{14}$	$3.7 \times 10^{16}$
$k = 20$	$1.8 \times 10^{26}$	$9.5 \times 10^{31}$	$2.1 \times 10^{35}$

things. The quest for FPT algorithms leads to a *positive toolkit* of methods for obtaining FPT algorithms. For the most up-to-date overview of such methods, see the surveys in [6].

Classically, evidence that a problem is unlikely to have an algorithm with a runtime in the good class is given by determining that it is NP-hard, PSPACE-hard, EXP-hard, etc. In parameterized complexity analysis there are analogous means to show likely parameterized intractability (the *negative toolkit*). The current tower of the main parameterized complexity classes is:

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

Parameterized by the size  $k$  of a solution, the familiar INDEPENDENT SET problem is complete for  $W[1]$  and DOMINATING SET is complete for  $W[2]$ . The naturally parameterized BANDWIDTH problem is hard for  $W[t]$  for all  $t$  [2]. The best known algorithms for the parameterized INDEPENDENT SET and DOMINATING SET problems are slight improvements on the brute-force approach of trying all  $k$ -subsets. The parameterized class  $W[1]$  is strongly analogous to NP, because the  $k$ -STEP HALTING PROBLEM for Turing machines of unlimited non-determinism is complete for  $W[1]$  [11]. FPT is equal to  $M[1]$  if and only if the so-called *Exponential Time Hypothesis* fails [21,9]. There is an algorithm for the  $k$ -INDEPENDENT SET problem that runs in time  $O(n^{o(k)})$  if and only if FPT is equal to  $M[1]$ , and there is an algorithm for the  $k$ -DOMINATING SET problem that runs in time  $O(n^{o(k)})$  if and only if FPT is equal to  $M[2]$  [5].

For further background on parameterized complexity we refer the reader to the textbooks [10,15,23], and the double-special issue of surveys of aspects of the field and various application areas [6].

Relevant secondary measurements that affect problem complexity can be *many different things*, such as the size of the solution, aspects of the structure of typical instances, aspects of the algorithmic approach, or the goodness of an approximation. For any real-world problem, there may be several such secondary measurements in various ranges of magnitude that should be considered. Parameterized complexity establishes a qualitative framework where there is a place for only one such secondary measurement. It is quite natural to investigate the prospects for *fully multivariate frameworks* for qualitatively assessing computational complexity. We report here on three recent research directions that begin to explore the possibilities:

- *Parameter ecology*, where the two different measurements of input structure and solution size interact.
- *Algorithmic metatheorems* in the context of structural parameterizations.
- *Relationships between structural parameterizations*.

### 3 Parameter Ecology

Why *ecology*?

Let us consider again the example of TYPE CHECKING IN ML. Although the problem is highly intractable from the classical point of view, the implemented ML compilers (that include type-checking subroutines) work efficiently. The explanation is that human-composed programs typically have a maximum type-declaration nesting depth of  $k \leq 5$ . The FPT type-checking subroutine that runs in time  $O(2^{kn})$  is thus entirely adequate in practice. The reason that naturally occurring programs have small nesting depth is because the programs would otherwise risk becoming incomprehensible to the programmer creating them!

What this example points to (we think) is that often the “inputs” to one computational problem of interest to real-world algorithmics are not at all arbitrary, but rather are produced by other natural computational processes (e.g., the thinking processes and abilities of the programmer) that are themselves subject to computational complexity constraints. In this way, the natural input distributions encountered by abstractly defined computational problems often have inherited structural regularities and restrictions (relevant parameters, in the sense of parameterized complexity) due to the natural complexity constraints on the generative processes. This connection is what we refer to as the *ecology* of computation.

It therefore seems to be useful to know how all the various parameterized structural notions interact with all the other computational objectives one might have. The familiar paradigm of efficiently solving various problems for graphs of bounded treewidth just represents one row of a matrix of algorithmic questions. Table 1 illustrates the idea. We use here the shorthand: TW is TREEWIDTH, BW is BANDWIDTH, VC is VERTEX COVER, DS is DOMINATING SET, G is GENUS and ML is MAX LEAF. The entry in the 2nd row and 4th column indicates that there is an FPT algorithm to optimally solve the DOMINATING SET problem for a graph  $G$  of bandwidth at most  $k$ . The entry in the 4th row and second column indicates that it is unknown whether BANDWIDTH can be solved optimally by an FPT algorithm when the parameter is a bound on the domination number of the input. An entry in the table describes the current state of knowledge about the complexity of the problem where the input graph is assumed to have a structural bound described by the *row*, and the problem described by the *column* is to be solved to optimality. The table just gives a few examples of the unbounded conceptual matrix that we are concerned with.

The complexity of graph problems, for graphs of bounded treewidth, is well-developed and supports many systematic approaches, such as described in [7,14,10,23,3]. For example, MINIMUM COLORING can be solved in time  $O(n)$  for

**Table 2.** The Complexity Ecology of Parameters

	TW	BW	VC	DS	G	ML
TW	<i>FPT</i>	$W[1]$ -hard	<i>FPT</i>	<i>FPT</i>	?	<i>FPT</i>
BW	<i>FPT</i>	$W[1]$ -hard	<i>FPT</i>	<i>FPT</i>	?	<i>FPT</i>
VC	<i>FPT</i>	<i>FPT</i>	<i>FPT</i>	<i>FPT</i>	?	<i>FPT</i>
DS	$W[1]$ -hard	?	$W[1]$ -hard	$W[1]$ -hard	?	?
G	$W[1]$ -hard	$W[1]$ -hard	$W[1]$ -hard	$W[1]$ -hard	<i>FPT</i>	$W[1]$ -hard
ML	<i>FPT</i>	<i>FPT</i>	<i>FPT</i>	<i>FPT</i>	<i>FPT</i>	<i>FPT</i>

graphs of treewidth at most  $k$ . In the terminology of parameterized complexity MINIMUM COLORING is *fixed parameter tractable* for the parameter *treewidth*.

Highly structured graph problems lead to the consideration of structural parameters that are more restrictive than treewidth: other rows of the matrix. One might ask whether these rows are really interesting, since a graph of bounded *max leaf number* is severely restricted in its structure. To be fair, however, a graph of bounded treewidth is also severely restricted, in contrast to an arbitrary graph. How to determine whether a graph of bounded *max leaf number* is 3-colorable in the “best possible” FPT runtime is an easily stated problem for which the answer is not obvious. Another observation that points to the interest in these rows is that there are now known to be many examples of problems that are  $W[1]$ -hard parameterized by a bound on treewidth, including BANDWIDTH [2]; LIST COLORING, PRE-COLORING EXTENSION and EQUITABLE COLORING [14]; GENERAL FACTOR [26]; and MINIMUM MAXIMUM OUTDEGREE [27]. One must therefore look “below treewidth” for FPT structural parameterizations for these problems. Note that while both bounded *vertex cover number* and bounded *max leaf number* imply bounded treewidth, neither of these structural bounds implies a bound on the other. LIST COLORING even remains  $W[1]$ -hard for graphs of bounded *vertex cover number* [18]. Lastly, it seems that for severe structural parameterizations such as bounded *vertex cover number* and bounded *max leaf number*, different FPT techniques are brought forward to importance, such as well-quasi-ordering and bounded variable integer linear programming.

Some recent positive results include:

**Theorem 1.** [18,19] *The MINIMUM BANDWIDTH problem can be solved to optimality for graphs of bounded vertex cover number (bounded max leaf number) parameterized by the vertex cover number (max leaf number).*

**Open Problem.** The OPTIMAL LINEAR ARRANGEMENT problem, that asks for an injective layout function  $f : V \rightarrow N$  that minimizes  $\sum_{u \neq v} |f(u) - f(v)|$ , is open, parameterized by the vertex cover and max leaf numbers.

### 3.1 How to Parameterize?

There are two principles of how to find relevant parameterizations that seem worthwhile to articulate, and that naturally lead in multivariate directions.

**Principle 1: Model Enrichment.** We have noted above that the graph MINIMUM COLORING problem is FPT when parameterized by the treewidth of a graph. Coloring models scheduling, but in the real world, it is often the case that not every timeblock (color) can be assigned to every task (because, for example, Bob may not be available for any meetings on Thursday). In the interests of greater realism and a better fit to applications, *we should always explore more richly structured variations of tractable problems.* Following this example, since MINIMUM COLORING is FPT parameterized by graph treewidth, it is worth exploring the complexity of MINIMUM LIST COLORING parameterized by treewidth, as this more structured problem has better traction on real applications. Unfortunately, in this case, we have the following negative result.

**Theorem 2.** [17] (MINIMUM) LIST COLORING is hard for  $W[1]$ , parameterized by treewidth, and even when parameterized by vertex cover number.

**Principle 2: Deconstruction of Intractability.** One way to find relevant parameterizations of hard problems is to examine the proofs of the intractability results, and ask, “Why are the instances produced in this argument *unreasonable*?” An examination of the negative result(s) for (MINIMUM) LIST COLORING in [14] shows that the reduction from MULTICOLOR CLIQUE produces instances of LIST COLORING where “most” vertices has lists of allowable colors of size 2. This is clearly unreasonable. Bob may not be available on Thursdays, but it seems unlikely that everybody is available only two hours per week! This in turn suggests richer parameterizations, such as treewidth, together with a bound on the average number of colors *not* on the list.

The interplay between these two principles of parameterization, inevitably leads to us to consider highly structured problems for parameters more restrictive than treewidth.

## 4 Algorithmic Metatheorems: Well-Quasiordering

One of the most powerful FPT classification tools is *well-quasiordering*. Graphs in general are well-quasiordered by minors (the celebrated Graph Minor Theorem), and also importantly, determining whether a graph  $H$  is a minor of a graph  $G$ , parameterized by  $H$ , is FPT (we say that *the minor order has FPT order tests*) [24,25].

The study of general algorithmic machineries, such as well-quasiordering, for rows of the parameter ecology matrix other than bounded treewidth has scarcely begun. One recent positive result is described as follows.

By a *3-star path* we refer to the graph formed by adding a path linking two copies of  $K(1,3)$ . The resulting graph has two pairs of vertices of degree 1, each pair adjacent to a vertex of degree 3 (there are two of these) and all other vertices have degree 2. Let  $\mathcal{U}$  be the set of graphs that consists of all the cycles, together with all of the *3-star paths*.

It is easy to check that  $\mathcal{U}$  is an antichain in the induced subgraph order; it is in the sense of the following theorem a *universal antichain*.

**Theorem 3.** [8,16,17]

(1) A family of graphs  $\mathcal{F}$  that is a lower ideal in the (ordinary) subgraph order is well-quasiordered by the induced subgraph order, and has FPT order tests, if and only if  $\mathcal{F} \cap \mathcal{U}$  is finite (where the order tests are parameterized by the size of the intersection).

(2) For any induced subgraph order antichain  $\mathcal{U}'$  for which (1) holds, the symmetric difference  $\Delta(\mathcal{U}, \mathcal{U}')$  is finite.

As a corollary, graphs of bounded vertex cover number are well-quasiordered by induced subgraphs and have FPT order tests.

## 5 Relationships between Structural Parameters

Some rows of the parameter ecology matrix are *better* than others in a certain sense.

**Example: Topological Bandwidth.** The *topological bandwidth* of a graph is defined to be the minimum, taken over all subdivisions  $G'$  of a graph  $G$ , of the bandwidth of  $G'$ . Determining whether a graph has topological bandwidth bounded by  $k$ , is hard for  $W[t]$  for all  $t$  [2]. However, the topological bandwidth of a graph is “sandwiched” by its cutwidth:  $tbw(G) \leq cw(G)$  and  $cw(G) \leq tbw(G)^2$ , and furthermore, it is FPT to determine whether a graph has cutwidth at most  $k$  [10]. Consequently, a family of graphs has *bounded cutwidth* if and only if it has *bounded topological bandwidth*, and the former is easier to determine.

These examples raise fundamental questions about general relationships between rows of the parameter ecology matrix. *When a row is hard to determine exactly (that is, NP-hard, or hard for  $W[1]$ ) — when can we find a sandwiching structural parameter that is easier to determine?*

We currently know only one concrete example of when such “easier” structural sandwiching is (likely) impossible: the parameter is the *minimum independent domination number* of a graph  $id(G)$  defined to be the minimum size of an independent dominating set of  $G$ .

**Theorem 4.** [12] *There is no FPT sandwiching parameter for  $id(G)$  (for any computable sandwich bound) unless  $FPT = W[2]$ .*

## 6 Concluding Remark

We have surveyed some of the issues involved in the quest to develop *fully multivariate algorithmics* and a few recent research directions that lead us to consider how several different secondary measures with different roles interact in shaping the complexity of computational problems, and thereby the angles we have available to exploit in order to design useful algorithms for NP-hard general problems, in situations where we are attentive to natural input distributions. Much more remains to be explored.



**Acknowledgments.** Many thanks to Frances Rosamond for conversations and assistance with this paper, and the invited talk at IWOCA 2009 on which it is based.

## References

1. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. *J. Algorithms* 12, 308–340 (1991)
2. Bodlaender, H., Fellows, M., Hallett, M.: Beyond NP-completeness for problems of bounded width: hardness for the  $W$  hierarchy. In: *Proceedings of the ACM Symposium on the Theory of Computing (STOC)*, pp. 449–458 (1994)
3. Bodlaender, H.L., Koster, A.M.: Combinatorial optimisation on graphs of bounded treewidth. *The Computer Journal* 51, 255–269 (2007)
4. Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small width. *SIAM J. Computing* 25, 1305–1317 (1996)
5. Chen, J., Chor, B., Fellows, M., Huang, X., Juedes, D., Kanj, I., Xia, G.: Tight lower bounds for certain parameterized NP-hard problems. *Information and Computation* 201, 216–231 (2005)
6. Fellows, M., Downey, R., Langston, M. (Guest eds.): *The Computer Journal: Two special issues of surveys of various aspects of parameterized complexity and algorithmics*. *The Computer Journal* 51(1&3) (2008)
7. Courcelle, B.: The monadic second order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation* 85, 12–75 (1990)
8. Ding, G.: Subgraphs and well-quasi-ordering. *J. Graph Theory* 16, 489–502 (1992)
9. Downey, R., Estivill-Castro, V., Fellows, M., Prieto-Rodriguez, E., Rosamond, F.: Cutting up is hard to do: the parameterized complexity of  $k$ -cut and related problems. *Electron. Notes Theor. Comp. Sci.* 78, 205–218 (2003)
10. Downey, R.G., Fellows, M.R.: *Parameterized Complexity*. Springer, Heidelberg (1999)
11. Downey, R., Fellows, M., Hallett, M., Kapron, B., Wareham, H.T.: The parameterized complexity of some problems in logic and linguistics. In: Matiyasevich, Y.V., Nerode, A. (eds.) *LFCS 1994*. LNCS, vol. 813, pp. 89–100. Springer, Heidelberg (1994)
12. Parameterized approximation for dominating set problems. *Information Processing Letters* 109, 68–70 (2008)
13. Downey, R., Fellows, M., Stege, U.: Parameterized complexity: a framework for systematically confronting computational intractability. In: Graham, R., Kratochvil, J., Nesetril, J., Roberts, F. (eds.) *Contemporary Trends in Discrete Mathematics*, *Proceedings of the DIMACS-DIMATIA Workshop on the Future of Discrete Mathematics*, Prague, 1997. AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 49, pp. 49–99 (1999)
14. Fellows, M., Fomin, F., Lokshtanov, D., Rosamond, F., Saurabh, S., Szeider, S., Thomassen, C.: On the complexity of some colorful problems parameterized by treewidth. In: Dress, A.W.M., Xu, Y., Zhu, B. (eds.) *COCOA 2007*. LNCS, vol. 4616, pp. 366–377. Springer, Heidelberg (2007)
15. Flum, J., Grohe, M.: *Parameterized Complexity Theory*. Springer, Heidelberg (2006)
16. Fellows, M., Hermelin, D., Rosamond, F.: Well-quasi-ordering bounded treewidth graphs. In: *Proceedings of IWPEC 2009*. LNCS, vol. 5917. Springer, Heidelberg (to appear)

17. Fellows, M., Hermelin, D., Rosamond, F.: Parameter ecology, well-quasi-ordering and universal antichains (manuscript, 2009)
18. Fellows, M., Lokshtanov, D., Misra, N., Rosamond, F., Saurabh, S.: Graph Layout Problems Parameterized by Vertex Cover. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 294–305. Springer, Heidelberg (2008)
19. Fellows, M., Lokshtanov, D., Misra, N., Mnich, M., Rosamond, F., Saurabh, S.: The complexity ecology of parameters: an illustration using bounded max leaf number. *Theory of Computing Systems* (2009)
20. Henglein, F., Mairson, H.G.: The complexity of type inference for higher-order typed lambda calculi. In: Proc. Symp. on Principles of Programming Languages (POPL), pp. 119–130. ACM Press, New York (1991)
21. Impagliazzo, R., Paturi, R.: Which problems have strongly exponential complexity? *J. Computer and Systems Sciences* 63, 512–530 (2001)
22. Lichtenstein, O., Pnueli, A.: Checking That Finite-State Concurrents Programs Satisfy Their Linear Specification. In: Proceedings of the 12th ACM Symposium on Principles of Programming Languages, pp. 97–107 (1985)
23. Niedermeier, R.: Invitation to Fixed Parameter Algorithms. Oxford University Press, Oxford (2006)
24. Robertson, N., Seymour, P.: Graph minors: a survey. In: Anderson, J. (ed.) *Surveys in Combinatorics*, pp. 153–171. Cambridge University Press, Cambridge (1985)
25. Robertson, N., Seymour, P.: Graph minors XX. Wagner’s conjecture. *J. Comb. Th. Series B* 92, 325–357 (2004)
26. Szeider, S.: Monadic second order logic on graphs with local cardinality constraints. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 601–612. Springer, Heidelberg (2008)
27. Szeider, S.: Not so easy problems for tree decomposable graphs. In: Proc. ICDM 2008 (to appear)

# Succinct Representations of Trees

Giovanni Manzini

Department of Computer Science,  
University of Eastern Piedmont, Italy  
giovanni.manzini@mf.n.unipmn.it

## 1 Introduction

Trees are by far the most common data structures to represent/organize/search large amount of data. Part of the trees' success is probably due to the simplicity of the usual pointer-based implementation in which to move from parent to child we simply follow a pointer. Unfortunately, a simple counting argument shows that the pointer-based implementation is highly redundant. The number of distinct trees with  $n$  nodes is given by the  $n$ -th Catalan number:

$$C_n = \frac{1}{2n+1} \binom{2n+1}{n}.$$

Hence, provided  $n$  is known,  $\log C_n = 2n - \Theta(\log n)$  bits<sup>1</sup> suffice to identify a particular  $n$ -node tree. A pointer-based representation takes at least  $n \log n$  bits; one could think that the extra space is needed in order to support constant time navigation between tree nodes, but this is not the case. In this paper we review many different *succinct representations* of trees which take only  $2n + o(n)$  bits and support constant time computation of (a subset of) the following navigation/query operations:

- **Parent**( $x$ ): Return the parent of node  $x$ ;
- **Degree**( $x$ ): Return the degree (number of children) of node  $x$ ;
- **Child**( $x, i$ ): Return the  $i$ -th child of node  $x$ ;
- **SubtreeSize**( $x$ ): Return the size (number of nodes) of the subtree rooted at node  $x$ ;
- **Depth**( $x$ ): Return the depth (distance from the root) of node  $x$ ;
- **LevelAncestor**( $x, d$ ): Return the ancestor of  $x$  with depth  $d$ ;
- **LowestCommonAncestor**( $x, y$ ): Return the lowest common ancestor of nodes  $x$  and  $y$ .

In the following we assume that a succinct representation is allowed to use an arbitrary numbering of the nodes (a bijection between nodes and the set  $\{1, 2, \dots, n\}$ ); but the operations above must be consistent with this numbering. For example, if nodes are numbered as in Fig. 1, we must have **Parent**(6) = 2 (the parent of node **f** is node **b**) and **LowestCommonAncestor**(4, 10) = 1 (the lowest

---

<sup>1</sup> All logarithms in this paper are to the base 2.

common ancestor of nodes  $d$  and  $j$  is node  $a$ ). The node numbering is also used to access any auxiliary information stored in tree nodes.

In this paper we review the main techniques for representing *static unlabeled ordinal trees*. An ordinal tree is a rooted tree of arbitrary degree in which the children of each node are ordered (so we have a first child, a second child, and so on). We do not consider *cardinal trees* in which each node has up to  $k$  children each one labeled by a unique integer from the set  $\{1, 2, \dots, k\}$  (for example, a binary tree in which we distinguish between left-child and right-child is a cardinal tree of degree 2). Succinct representations of cardinal trees are described in [4,6,21]. Another important family of trees are the (multi)labeled trees in which each node has assigned one (or more) symbols. In this case we want to represent the tree succinctly and, in addition to the navigation operations, we want to support path-search queries, that is, we want to find all the occurrences in the tree of a given labeled path. Succinct representations of labeled and multi-labeled trees, and their applications to XML storage and processing, are described in [1,2,3,7,8,10,16].

In the following we assume a RAM model with a word size of  $\Theta(\log n)$  bits.

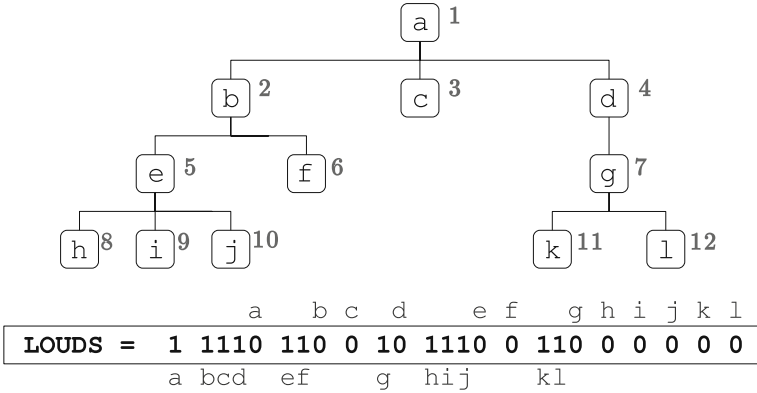
## 2 Representation of Binary Strings

Many tree representations are based on binary strings, and in particular on the possibility of performing the Rank and Select queries on binary strings in constant time. Given a binary string  $B[1, n]$  and  $c \in \{0, 1\}$  we denote by  $\text{Rank}_c(B, i)$  the number of occurrences of the bit  $c$  in  $B[1, i]$ , and by  $\text{Select}_c(B, j)$  the position in  $B$  of the  $j$ -th occurrence of bit  $c$  (hence  $\text{Rank}_c(\text{Select}_c(B, j)) = j$ ). We will make use of the following result:

**Lemma 1** ([5,15,18]). *For any binary string  $B[1, n]$  we can build an auxiliary data structure of  $o(n)$  bits supporting Rank and Select queries in  $O(1)$  time.*

*Proof.* To give an idea of the proof we describe the data structure supporting the  $\text{Rank}_1$  query. We logically split  $B[1, n]$  into *mini-blocks* of length  $\lceil \log^2 n \rceil$ . At the beginning of each mini-block we store the number of **1**'s in the previous mini-blocks. Note that this takes  $\log n$  bits per mini-blocks; since there are  $O(n/\log^2 n)$  mini-blocks these entries take  $o(n)$  bits overall. Then, we logically split each mini-block into *micro-blocks* of length  $\lceil (\log n)/2 \rceil$ . At the beginning of each micro-block we store the number of **1**'s since the beginning of the current mini-block. This number is at most  $\lceil \log^2 n \rceil$  so we can represent it using  $O(\log \log n)$  bits. Since there are  $O(n/\log n)$  micro-blocks, the overall space is again  $o(n)$ .

To answer the  $\text{Rank}_1(B, i)$  query we first find the mini-block  $m$  and micro-block  $\mu$  containing position  $i$ . Accessing the mini-block and micro-block tables we get the number of **1**'s up to the beginning of  $\mu$ , so we are left with the task of counting the number of **1**'s from the beginning of  $\mu$  up to position  $i$ . The crucial observation is that a micro-block, being a bit-string of length  $\lceil (\log n)/2 \rceil$ , is identified by a positive integer of size at most  $2^{\lceil (\log n)/2 \rceil} = O(\sqrt{n})$ . Hence, we can build a two dimensional table  $T$  such that  $T(p, q)$  stores the number of **1**'s



**Fig. 1.** An unlabeled tree and the corresponding LOUDS binary sequence. The figure also shows the correspondence between 0's 1's and tree nodes.

among the first  $q$  bits of the micro-block represented by the integer  $p$ . Table  $T$  has  $O(\sqrt{n} \times \log n)$  entries each one of size  $O(\log \log n)$  so it also takes  $o(n)$  bits.

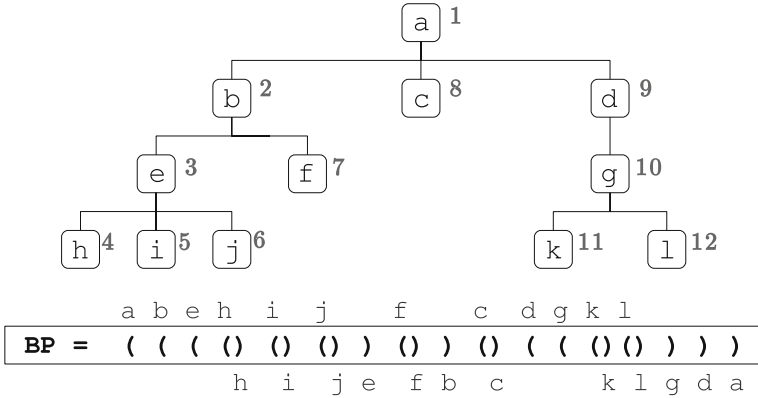
Summing up, using  $o(n)$  additional space, computing  $\text{Rank}_1(B, i)$  reduces to three table lookups which takes constant time as claimed. To compute  $\text{Rank}_0(B, i)$  we use the formula  $\text{Rank}_0(B, i) = i - \text{Rank}_1(B, i)$ . For Select queries we use a similar approach with a different set of auxiliary data structures.  $\square$

Although the above result is all we need for most tree representations, it is worth mentioning that it has been recently improved in different directions. For example, assuming the binary string  $B[1, n]$  contains exactly  $m$  1's, the usual counting argument gives a lower bound for representing  $B$  of  $\mathcal{B}(n, m) = \lceil \log \binom{n}{m} \rceil$  bits. In [21] it is shown how to represent  $B$  in  $\mathcal{B}(n, m) + o(n)$  bits. This result has been later generalized to non-binary sequences [9,12,22], and optimal lower bounds have been given in [11].

### 3 Jacobson LOUDS Representation

The first succinct representation of ordinal trees was proposed in 1989 by Jacobson in [14,15]. Jacobson's representation is called Level Order Unary Degree Sequence (LOUDS) since it is obtained by traversing the tree in level order and writing the degree of each node in unary (see Fig. 1). For example, for a node with 3 children we write the string 1110. Since we write a 0 for each node and an 1 for each child, for an  $n$ -node tree the resulting string has  $n$  0's and  $n-1$  1's. By adding an initial 1 to this string—corresponding to an hypothetical super-root node—we get a string with  $n$  0's and  $n$  1's in which both the  $i$ -th 0 and the  $i$ -th 1 correspond to the  $i$ -th node in the level order. Because of these correspondences Parent and Child operations can be expressed in terms of Rank/Select queries. Indeed, we have:

$$\begin{aligned} \text{Parent}(x) &= 1 + \text{Rank}_0(\text{Select}_1(x)), \\ \text{Child}(x, k) &= \text{Rank}_1(\text{Select}_0(x-1) + k). \end{aligned}$$



**Fig. 2.** An unlabeled tree and the corresponding BP sequence. The figure also shows the correspondence between open/closing parentheses and tree nodes.

For example, in Fig. 1 to compute  $\text{Parent}(7)$ , i.e. the parent of node  $g$ , we first compute  $\text{Select}_1(7) = 10$ , which is the  $\mathbf{1}$  corresponding to node  $g$ . Then,  $\text{Rank}_0(10) = 3$  gives the number of  $\mathbf{0}$ 's preceding the unary degree of  $g$ 's parent. Hence,  $g$ 's parent has position  $1 + \text{Rank}_0(10) = 4$  in level 10 and is indeed node  $d$ .

Jacobson's representation of an  $n$ -node tree consists of the  $2n$  bit LOUDS string together with auxiliary  $o(n)$  bits to support  $\text{Rank}/\text{Select}$  operations in  $O(1)$  time (Lemma 1). Clearly such representation offers constant time<sup>2</sup> tree navigation, but none of the other tree operations mentioned in Section 1.

## 4 Balanced Parentheses Representation

Ten years after Jacobson's results, Munro and Raman [19,20] proposed an alternative tree representation based on strings of open and closing parentheses. Note that parentheses are used for convenience: being strings over a two-symbol alphabet these are equivalent to ordinary binary strings. The Balanced Parentheses (BP) representation is obtained by traversing the tree in depth-first order writing an open parenthesis when a node is first encountered and a closing parenthesis when the same node is encountered again while going up after traversing its subtree (see Fig. 2 for an example). Clearly, for an  $n$ -node tree the BP representation consists of a string containing  $n$  open and  $n$  closing parentheses that respect the usual nesting rule. In addition to the  $o(n)$  bits auxiliary data structures for  $\text{Rank}/\text{Select}$  of Lemma 1, Munro and Raman's representation makes use of  $o(n)$  bits data structures supporting in  $O(1)$  time the operations:

<sup>2</sup> Jacobson's representation precedes the results on constant-time  $\text{Rankselect}$ ; Jacobson's original navigation operations take, in our model,  $O(\log \log n)$  time.

- $\text{FindClose}(i)$ : Return the position of the closing parenthesis matching the open parenthesis in position  $i$ .
- $\text{FindOpen}(i)$ : Return the position of the open parenthesis matching the closing parenthesis in position  $i$ .
- $\text{Enclose}(i)$ : Return the position of the closest open parenthesis enclosing the open parenthesis in position  $i$ .

Assuming that each node is identified by its depth-first rank, it is straightforward to verify that the following identities hold:

$$\begin{aligned} \text{Parent}(x) &= \text{Rank}_{\text{open}}(\text{Enclose}(\text{Select}_{\text{open}}(x))); \\ \text{SubtreeSize}(x) &= (\text{FindClose}(\text{Select}_{\text{open}}(x)) - \text{Select}_{\text{open}}(x) + 1)/2. \end{aligned}$$

For example, in Fig. 2 to compute  $\text{Parent}(6)$ , i.e. the parent of node  $j$ , we first compute  $\text{Select}_{\text{open}}(6) = 8$ , which is the position of the open parenthesis associated to node  $j$ . Then we compute  $\text{Enclose}(8) = 3$  which is the position of the closest open parenthesis enclosing the one in position 8 and is therefore the open parenthesis associated to  $j$ 's parent. Finally, by computing  $\text{Rank}_{\text{open}}(3) = 3$  we get the identifier of  $j$ 's parent, which is of course node  $e$ . As another example, to compute  $\text{SubtreeSize}(9)$ , i.e. the size of the subtree rooted at node  $d$ , we compute  $\text{Select}_{\text{open}}(9) = 16$ , which is the position of the open parenthesis associated to node  $d$ , and  $\text{FindClose}(16) = 23$ , which is the position of the closing parenthesis associated to  $d$ . The size of the subtree is half of the number of parenthesis between these two, hence  $(23 - 16 + 1)/2 = 4$ .

Unfortunately, the BP representation does not support  $\text{Child}(x, i)$  directly. Instead, we can compute in constant time the first child and the right sibling of a node using the identities

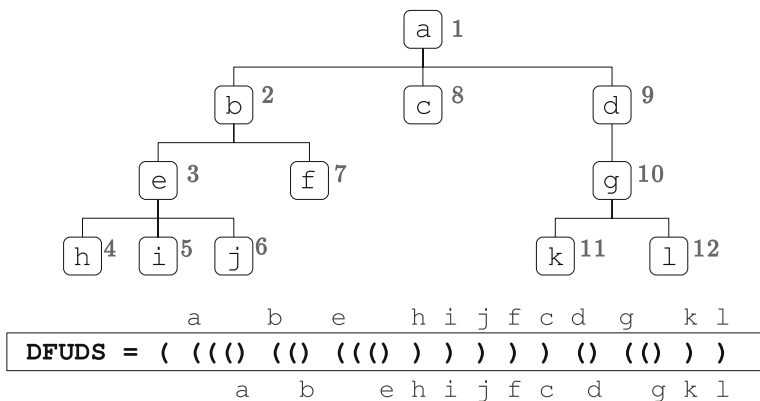
$$\begin{aligned} \text{FirstChild}(x) &= x + 1, \\ \text{RightSibling}(x) &= \text{Rank}_{\text{open}}(\text{FindClose}(\text{Select}_{\text{open}}(x)) + 1). \end{aligned}$$

Used together these formulas can be used to compute  $\text{Child}(x, i)$  in  $O(i)$  time.

Summing up, the BP representation takes  $2n + o(n)$  bits and supports tree navigation and subtree size, provided that the tree's degree is bounded by a constant.

## 5 DFUDS Representation

In [4] Benoit *et al.* introduce the Depth-First Unary Degree Sequence (DFUDS) representation of an  $n$ -node tree. DFUDS combines the LOUDS and BP representations and offers the benefits of both. As in the BP representation, nodes are considered in depth-first order. As in the LOUDS representation, for each visited node we write its degree in unary, but using open and closing parentheses instead of  $\mathbf{1}$ 's and  $\mathbf{0}$ 's respectively. As in LOUDS, we include an initial open parenthesis corresponding to the hypothetical super-root, thus obtaining a string of  $2n$  balanced parentheses (see Fig. 3 for an example). Benoit *et al.* shows that



**Fig. 3.** An unlabeled tree and its DFUDS representation. The figure shows the starting and ending position of the unary encoding of the degree of each node.

the  $Parent(x)$ ,  $Child(x, i)$ , and  $SubtreeSize(x)$  operations can all be implemented via a constant number of  $Rank$ ,  $Select$ ,  $FindOpen$ ,  $FindClose$ ,  $Enclose$  operations on the above string. Thus, the DFUDS representation offers constant time basic navigation operations using  $2n + o(n)$  bits of storage.

Recently, Jansson *et al.* [16] have shown that using the DFUDS representation is possible to implement also the  $Depth$ ,  $LevelAncestor$ ,  $LowestCommonAncestor$  operations within the same space bound of  $2n + o(n)$  bits. This makes DFUDS the first succinct representation able to support the whole set of navigation/query operations mentioned in Section 4.

## 6 Ultra-Succinct Representation

In [16], Jansson *et al.* have not only extended the range of operations supported by the DFUDS representation, but also used DFUDS to derive a new representation which breaks the  $2n - \Theta(\log n)$  bits lower bound established in Section 4. This was possible because  $2n - \Theta(\log n)$  bits is a lower bound for representing a generic  $n$ -node tree; if we have additional information on the tree structure we can devise more economical encodings. For example, it is easy to show that an  $n$ -node full binary tree (in which every internal node has degree 2) can be represented using  $n$  bits. The following result [3] is the key to generalize this observation.

**Lemma 2.** *The number of ordered trees with  $n$  nodes having  $n_i$  nodes with  $i$  children is given by*

$$\frac{1}{n} \binom{n}{n_0 \quad n_1 \quad \dots \quad n_{n-1}} \tag{1}$$

*provided that  $\sum_{i \geq 0} n_i(i - 1) = -1$ . If the equality does not hold no such tree exists. □*

<sup>3</sup> See for example [http://www.emis.de/journals/SLC/wpapers/s38pr\\_rote.pdf](http://www.emis.de/journals/SLC/wpapers/s38pr_rote.pdf).



Using an ingenuous encoding of the DFUDS sequence, and appropriate auxiliary data structures, Jansson *et al.* [16] have been able to represent an  $n$ -node tree with degree distribution  $n_0, n_1, \dots$  with a number of bits equal to the base 2 logarithm of (1) plus  $o(n)$  bits.

## 7 Representations Based on Tree Covering

A completely different approach to the succinct representation of trees is the one based on tree covering [10,13,17]. This approach is similar to the techniques described in Section 2 for supporting constant time Rank/Select over binary strings. An  $n$ -node tree is decomposed into  $O(n/\log^4 n)$  mini-trees of  $O(\log^4 n)$  nodes each. Each mini-tree is then decomposed into  $O(\log^3 n)$  micro-trees of at most  $(\log n)/8$  nodes each. If we collapse each mini-tree into a single node, the original tree has size  $O(n/\log^4 n)$ , so even a pointer-based representation takes  $o(n)$  bits. Similarly, if we collapse micro-trees into single nodes each mini-tree has size  $O(\log^3 n)$  and they can be represented in overall  $o(n)$  bits. Finally, micro-trees are represented using for example DFUDS using a total storage of  $2n + o(n)$  bits. Tree operations are computed working at three levels: the original tree, mini-trees, and micro-trees. The crucial point is that, since micro-trees are very small, for each tree operation we can store in a  $o(n)$  bits table the answer of each possible query on each possible micro-tree (compare with the proof of Lemma 1 where we pre-compute a table of the answers of each possible Rank<sub>1</sub> query on each possible micro-block).

Recently, [6] has refined this approach showing that it can be used to represent in an uniform way ordinal, cardinal, and free trees, and that its space usage matches the tighter lower bound induced by Lemma 2.

## References

1. Barbay, J., Golynski, A., Munro, J.I., Rao, S.S.: Adaptive searching in succinctly encoded binary relations and tree-structured documents. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 24–35. Springer, Heidelberg (2006)
2. Barbay, J., Golynski, A., Munro, J.I., Rao, S.S.: Adaptive searching in succinctly encoded binary relations and tree-structured documents. *Theoretical Computer Science* 387(3), 284–297 (2007)
3. Barbay, J., He, M., Munro, J.I., Rao, S.S.: Succinct indexes for string, binary relations and multi-labeled trees. In: Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 680–689 (2007)
4. Benoit, D., Demaine, E., Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Representing trees of higher degree. *Algorithmica* 43, 275–292 (2005)
5. Clark, D.: Compact Pat Trees. PhD thesis, University of Waterloo (1996)
6. Farzan, A., Ian Munro, J.: A uniform approach towards succinct representation of trees. In: Gudmundsson, J. (ed.) SWAT 2008. LNCS, vol. 5124, pp. 173–184. Springer, Heidelberg (2008)

7. Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Structuring labeled trees for optimal succinctness, and beyond. In: Proc. 46th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 184–193 (2005)
8. Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Compressing and searching XML data via two zips. In: Proc. 15th International World Wide Web Conference (WWW), pp. 751–760 (2006)
9. Ferragina, P., Manzini, G., Mäkinen, V., Navarro, G.: Compressed representations of sequences and full-text indexes. *ACM Transactions on Algorithms* 3(2) (2007)
10. Geary, R.F., Raman, R., Raman, V.: Succinct ordinal trees with level-ancestor queries. *ACM Transactions on Algorithms* 2(4), 510–534 (2006)
11. Golynski, A.: Optimal lower bounds for rank and select indexes. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 370–381. Springer, Heidelberg (2006)
12. Golynski, A., Munro, J.I., Rao, S.S.: Rank/select operations on large alphabets: a tool for text indexing. In: Proc. 17th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 368–373 (2006)
13. He, M., Munro, I., Rao, S.: Succinct ordinal trees based on tree covering. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 509–520. Springer, Heidelberg (2007)
14. Jacobson, G.: Space-efficient static trees and graphs. In: Proc. 30th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 549–554 (1989)
15. Jacobson, G.: Succinct Static Data Structures. PhD thesis, Carnegie Mellon University (1989)
16. Jansson, J., Sadakane, K., Sung, W.K.: Ultra-succinct representation of ordered trees. In: Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 575–584 (2007)
17. Munro, I., Raman, V., Storm, A.: Representing dynamic binary trees succinctly. In: Proc. 12th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 529–536 (2001)
18. Munro, J.I.: Tables. In: Chandru, V., Vinay, V. (eds.) FSTTCS 1996. LNCS, vol. 1180, pp. 37–42. Springer, Heidelberg (1996)
19. Munro, J.I., Raman, V.: Succinct representation of balanced parentheses, static trees and planar graphs. In: Proc. 38th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 118–126 (1997)
20. Munro, J.I., Raman, V.: Succinct representation of balanced parentheses and static trees. *SIAM J. Computing* 31, 762–776 (2001)
21. Raman, R., Raman, V., Rao, S.S.: Succinct indexable dictionaries with applications to encoding  $k$ -ary trees and multisets. In: Proc. 13th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 233–242 (2002)
22. Sadakane, K., Grossi, R.: Squeezing succinct data structures into entropy bounds. In: Proc. 17th ACM-SIAM Symposium on Discrete Algorithm (SODA), pp. 1230–1239 (2006)

# $K_t$ Minors in Large $t$ -Connected Graphs<sup>\*</sup>

## (Abstract)

Robin Thomas

School of Mathematics  
Georgia Institute of Technology  
Atlanta, GA 30332  
thomas@math.gatech.edu

A graph  $G$  has a  $K_t$  minor if a graph isomorphic to  $K_t$ , the complete graph on  $t$  vertices, can be obtained from a subgraph of  $G$  by contracting edges. A long-standing conjecture of Hadwiger states that every graph with no  $K_t$  minor is  $(t - 1)$ -colorable. Hadwiger's conjecture is known for  $t \leq 6$ , and open for all  $t > 7$ .

A deep theorem of Robertson and Seymour describes the structure of graphs with no  $K_t$  minor. The theorem is very powerful, but it is fairly complicated to state, and the condition it gives is necessary, but not sufficient, for the exclusion of a  $K_t$  minor.

We prove a necessary and sufficient condition under additional restrictions on the graph  $G$ . We prove that for every integer  $t$  there exists an integer  $N$  such that every  $t$ -connected graph on at least  $N$  vertices with no  $K_t$  minor has a set of at most  $t - 5$  vertices whose deletion makes the graph planar. This is best possible in the sense that neither  $t$ -connectivity nor the size of the deleted set can be lowered, and for  $t > 7$  some lower bound on the number of vertices is needed.

---

<sup>\*</sup> This is joint work with Sergey Norin.

# Intractability in Graph Drawing and Geometry: FPT Approaches

Sue Whitesides

Department of Computer Science, University of Victoria, Victoria BC, Canada  
sue@uvic.ca

**Abstract.** The fixed parameter tractability (FPT) approach pioneered by Downey and Fellows provides an algorithm design philosophy for solving special cases of intractable problems. Here we review several examples from geometry and graph drawing, in particular layered graph drawing, that illustrate fixed parameter tractability techniques.

What follows is a brief summary of the topics presented in an invited talk at IWOCA 09, and some follow-up, but far from comprehensive, references. The author thanks the conference organizers and the program committee for their kind invitation to speak.

The fixed parameter tractability (FPT) approach of Downey and Fellows [4], has produced a substantial body of literature that provides both lower bound and algorithmic results. On the algorithmic side, the philosophy is to identify a parameter  $k$  associated with a given hard problem and then to design an algorithm whose running time is polynomial in the problem size while depending exponentially or worse on  $k$ . For example, a running time of  $O(2^k n)$  reflects fixed parameter tractability, whereas a running time of  $O(n^k)$  does not. An algorithm with a running time of this form could possibly give useful results for small values of  $k$ . Sometimes this is the case in practice, and sometimes such results are of mainly theoretical interest. Experimentations with implementations appear to be uncommon in the literature.

From the perspective of computational geometry and graph drawing, it thus natural to ask whether FPT techniques can be applied in those domains. See [11] for a recent survey article on this topic and for further elaboration and references for most of the examples mentioned here.

In honour of the transformation and evolution of AWOCA into IWOCA, a problem in visibility representations of graphs is also included; this is a topic from previous AWOCA's and ACCMCC's and GD's.

## 1 Graph Drawing

The field of graph drawing seeks good layouts and embeddings for graphs. The motivation for these layouts may be physically based (e.g., arrangements of wires, roads, pipes, laser beams) or visually based (e.g., diagrams or maps to be drawn on paper, viewed on screens, experienced in a 3D virtual reality environment).

The desired properties of the layout depend on the intended use. One may wish to realize edges as straight lines, paths that follow grid lines, or arcs or polylines. One may aim to minimize the number of bends in edges or the area in a grid layout, or the number of edge crossings. One may seek good angular resolution. Electrical properties or readability properties of the resulting layout may be the concern. The desired properties may conflict with one another, generating trade-off issues between, for example, layout readability and area. Many optimization problems in graph drawing are NP-hard.

## 2 Visibility Representations

While one typically lays out graphs by drawing vertices as points and edges as curves of some sort (straight lines, arcs, polylines, gridlines), more geometric representations of graphs have also been studied. Here, vertices are represented as geometric objects such as line segments or axis-aligned boxes in 3D, and edges are represented as a binary geometric relation such as intersection, tangency, or visibility.

One example is as follows. If one represents vertices as axis-aligned unit squares in 3D floating above the  $xy$ -plane, and edges as lines of sight parallel to the  $z$ -axis, what is the largest complete graph that can be represented? The answer is 7. While specific constructions show that 7 is possible, the proof that  $K_8$  has no such representation so far involves a non-trivial computer search (see [3]). If axis aligned rectangles having possibly different sizes are allowed, the size of the largest representable complete graph  $K_n$  increases to at least  $n = 22$ . The upper bound on  $n$  has decreased in stages from 102 to 50. Hence a substantial gap between the best known lower and upper bounds continues to exist. Other representations for vertices have also been considered, including equal regular  $n$ -gons. For example, in this situation, an upper bound of  $O(n^4)$  has been shown by Štola (see [14], [15]).

## 3 Layered Graph Drawing Problems

Given a graph  $G = (V, E)$ , where  $V$  is partitioned into  $h$  subsets  $V_1, \dots, V_h$  to be drawn as points on  $h$  horizontal layers  $L_1, \dots, L_h$ , and edges are to be drawn as straight line segments joining endpoints in adjacent layers, the problem of minimizing crossings in the drawing becomes one of choosing permutations  $\sigma_i$  for the vertices  $V_i$  assigned to each layer  $L_i$ . A related problem is to choose permutations so as to minimize the number of edges whose removal leaves a crossing-free drawing. Indeed there are a number of variants of this type of problem. One can consider two layers at a time, with the permutation on one of the two layers fixed, or drop the pre-assignment of vertices to the layers. In any case, natural parameters to associate with these types of problems are the number of crossings allowed, the number of edges allowed whose removal results in a planar layout, or the number of layers available.

Hard problems abound. For example, TWO-LAYER PLANARIZATION is NP-complete; that is, to determine, given a bipartite graph  $G = (V_1 \cup V_2, E)$  and non-negative integer  $k$ , whether there exist orderings for  $V_1$  and  $V_2$  so that after at most  $k$  edges are removed, the graph can be drawn without crossings on two layers. See [8], [2].

A number of fixed parameter tractable algorithms are known for layered graph drawing, often tracing their roots to a workshop at the Bellairs Research Institute that investigated the application of FPT methods to layered graph drawing. The workshop was the result of the author's discussions with Mike Fellows at conferences in Australia, including AWOCA Hunter Valley. See [5], [6], [7]. Two by now classic methods of FPT algorithm design, the bounded search tree method and the problem kernelization method (and their combination) are used to obtain the results. Later implementation and experimentation work [13] showed that the FPT approach for two-layer planarization led to results that are comparable but complementary to ILP results for the same problem. The results for  $h$ -layer graph drawing exploit the fact that  $h$ -layer planar graphs (and graphs that can be made  $h$ -layer planar after removal of at most  $k$  edges), have path width  $w$  expressible in terms of  $h$  (or  $h$  and  $k$ ). Then a path decomposition of width at most  $w$  is sought, leading to an FPT result with an algorithm with impractical running time even for small parameter values.

See also the chapter on graph drawing problems in the habilitation thesis of Henning Fernau [9].

## 4 FPT Algorithms for Geometry Problems

We mention two examples that are particularly accessible. See [11] for more details and references, as well as more examples.

The Independent Set Problem on Disc Graphs is as follows. Given  $n$  disks  $D_i$  in the plane, where the radius  $r_i$  of disc  $D_i$  lies in the range  $[1, \sigma]$ , and given that the distance  $d_{ij}$  between the centers of discs  $D_i$  and  $D_j$  is at least  $\lambda > 0$  for each pair of discs  $D_i, D_j$  are there at least  $k$  non-intersecting discs? Taking  $\sigma$  and  $\lambda$  as parameters, Alber and Fiala used the kernelization method to obtain an FPT algorithm ([11]).

The Independent Set Problem for Segment Graphs has been studied by Kara and Kratochvil, who parameterize on  $d$ , the number of different directions of the segments.

The Crossing Free Spanning Tree Problem is as follows. Given a graph  $G = (V, E)$  together with a drawing  $\Gamma(G)$  of  $G$  (the drawing need not be crossing-free), the problem is to determine whether  $\Gamma(G)$  contains a crossing-free spanning tree for  $G$ . FPT algorithmic results based on kernelization have been obtained for this problem, using the number of crossings in  $\Gamma(G)$  as a parameter.

## 5 Conclusion

The exploration of FPT methods in geometry and graph drawing is presently at an early stage, with a small but growing number of results. It is hoped that this presentation will serve as an invitation to explore the area further and to experiment with implementations of FPT algorithmic results.

## References

1. Alber, J., Fiala, J.: Geometric separation and exact solutions for the parameterized independent set problem on disk graphs. *J. Algorithms* 52(2), 134–151 (2004)
2. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.: *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, Englewood Cliffs (1999)
3. Bose, P., Everett, H., Fekete, S., Houle, M.E., Lubiw, A., Meijer, H., Romanik, K., Rote, G., Shermer, T., Whitesides, S., Zelle, C.: A visibility representation for graphs in three dimensions. *J. Graph Algorithms Appl.* 2(2) (1998)
4. Downey, R., Fellows, M.: *Parameterized Complexity*. Springer, Heidelberg (1999)
5. Dujmović, V., Fellows, M., Kitching, M., Liotta, G., McCartin, C., Nishimura, N., Ragde, P., Rosamond, F., Whitesides, S., Wood, D.R.: On the parameterized complexity of layered graph drawing. *Algorithmica* 52(2), 167–176 (2008)
6. Dujmović, V., Fellows, M., Kitching, M., Liotta, G., McCartin, C., Nishimura, N., Ragde, P., Rosamond, F., Whitesides, S., Wood, D.R.: A fixed parameter approach to 2-layer planarization. *Algorithmica* 45(2), 159–182 (2006)
7. Dujmović, V., Whitesides, S.: An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. *Algorithmica* 40(1), 15–31 (2004)
8. Eades, P., Whitesides, S.: Drawing graphs in two layers. *Theoret. Comput. Sci.* 131(2), 361–374 (1994)
9. Fernau, H.: *Parameterized Algorithmics: a Graph Theoretic Approach*. Habilitationsschrift, U. Tübingen (2005)
10. Flum, J., Grohe, M.: *Parameterized Complexity Theory*. Springer, Heidelberg (2006)
11. Giannopoulos, P., Knauer, C., Whitesides, S.: Parameterized complexity of geometric problems. *Comput. J.* 51(3), 372–384 (2008)
12. Niedermeier, R.: *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, Oxford (2006)
13. Suderman, M., Whitesides, S.: Experiments with the fixed-parameter approach for two-layer planarization. *J. Graph Algorithms Appl.* 9(1), 149–163 (2005)
14. Štola, J.: Unimaximal sequences of pairs in rectangle visibility drawing. In: Tollis, I.G., Patrignani, M. (eds.) *GD 2008*. LNCS, vol. 5417, pp. 61–66. Springer, Heidelberg (2009)
15. Štola, J.: 3D visibility representations of complete graphs. In: Liotta, G. (ed.) *GD 2003*. LNCS, vol. 2912, pp. 226–237. Springer, Heidelberg (2004)

# Evaluation of Recoverable-Robust Timetables on Tree Networks<sup>\*</sup>

Gianlorenzo D'Angelo<sup>1</sup>, Gabriele Di Stefano<sup>1</sup>, and Alfredo Navarra<sup>2</sup>

<sup>1</sup> Department of Electrical and Information Engineering, University of L'Aquila  
{gianlorenzo.dangelo,gabriele.distefano}@univaq.it

<sup>2</sup> Department of Mathematics and Computer Science, University of Perugia  
navarra@dmi.unipg.it

**Abstract.** In the context of scheduling and timetabling, we study a challenging combinatorial problem which is interesting from both a practical and a theoretical point of view. The motivation behind it is to cope with scheduled activities which might be subject to unavoidable disturbances, such as delays, occurring during the operational phase. The idea is to preventively plan some extra time for the scheduled activities in order to be “prepared” if a delay occurs, and to absorb it without the necessity of re-scheduling the activities from scratch. This realizes the concept of designing so called *robust timetables*. During the planning phase, one has to consider recovery features that might be applied at runtime if delays occur. Such recovery capabilities are given as input along with the possible delays that must be considered. The objective is the minimization of the overall needed time. The quality of a robust timetable is measured by the *price of robustness*, i.e. the ratio between the cost of the robust timetable and that of a non-robust optimal timetable. The considered problem is known to be *NP-hard*. We propose a pseudo-polynomial time algorithm and apply it on random networks and real case scenarios provided by Italian railways. We evaluate the effect of robustness on the scheduling of the activities and provide the price of robustness with respect to different scenarios. We experimentally show the practical effectiveness and efficiency of the proposed algorithm.

## 1 Introduction

In this work, we investigate an important combinatorial problem in the context of scheduling: the *timetable planning* of public transportation systems. It arises, for instance, in the planning phase of railway systems, requiring to compute a timetable for passenger trains that determines minimal passenger waiting times. However, many disturbing events might occur during the operational phase, that is when the system is running. Such events, whose main effect is the arising of delays, might make unfeasible the scheduled timetables. Hence, it is important to take them into account in advance. A schedule that lets vehicles sit at stations for

---

<sup>\*</sup> This work was partially supported by the Future and Emerging Technologies Unit of EC (IST priority - 6th FP), under contract no. FP6-021235-2 (project ARRIVAL).



some time will not suffer from small delays of arriving vehicles, because delayed passengers can still catch potential connecting vehicles. On the other hand, big delays can cause passengers to lose vehicles and hence imply extra traveling time. The problem of deciding when to guarantee connections from a delayed vehicle to a connecting vehicle is known in the literature as *delay management problem* [1,2,3,4,5,6,7]. Although the problem has a natural formalization, it turns out to be very complicated to be optimally solved. In fact, it has been shown to be *NP*-hard, while it is polynomial in some particular cases (see [4,5,7]).

In order to cope with the management of delays we follow the *recoverable robust optimization* approach provided in [8,9]. The aim is to design timetables in the planning phase in order to be “prepared” to react against possible disturbances. E.g., if a delay occurs, the designed timetable should guarantee the recovery of the scheduled events by means of allowed operations represented by given recovery algorithms. In [10], interesting theoretical results were presented, assuming that only one delay might occur at a generic event of the scheduled event activity network (see [17]) which is a directed graph that represents the sequence and the dependencies of scheduled events. The attention was restricted to event activity networks whose topology is a tree. In this context, it is worth noting that the assumption concerning one single delay does not constitute a restriction to the problem as  $k$  delays of size at most  $\alpha$  can be modelled as one delay of size at most  $k\alpha$ . An event either absorbs the entire delay or propagates it to all the subsequent events. In order to absorb the delay, an event is associated with a so called *slack time* of  $\alpha$  time. Clearly, by associating a slack time of  $\alpha$  to each event, every delay of duration  $\alpha$  can be locally absorbed. However, this approach is not practical as the overall duration time of the scheduled events would increase too much. The planned timetables should be instead able to absorb the possible occurring delay in a fixed amount of steps,  $\Delta$ . This means that if a delay occurs, it is not required that the delay is immediately absorbed (unless  $\Delta = 0$ ), but it can propagate to a limited number of events in the network. Namely, the propagation might involve at most  $\Delta$  events. The objective function to minimize is the total time required by the events in order to serve all the scheduled activities and to be robust with respect to one possible delay of  $\alpha$  time. That is, a timetable is robust if it can be recovered by postponing at most a fixed number of events in case of a delay.

In [11], the authors show that the described problem is *NP*-hard when the event activity network topology is a DAG with weights associated to nodes. In the same paper, it is shown that the problem where the weights are associated to the arcs is a subproblem. Even for this subproblem, in [11], the authors show that the problem remains *NP*-hard, and they provide polynomial time algorithms which cope with the case of  $\Delta = 0$ . In [11], the authors provide algorithms for any  $\Delta$  when the event activity network is a path with positive weights on nodes.

In this paper, we keep on investigating the complexity of the problem. In particular, we study event activity networks which have a tree topology and positive weights. In [10], this problem has been shown to be *NP*-hard even in this restricted scenario. We provide an algorithm that solves the problem in  $O(n^{\Delta+1})$

time where  $n$  is the number of events in the input event activity network. The result implies that the problem can be solved in pseudo-polynomial time for constant  $\Delta$ , i.e. when  $\Delta$  is fixed a priori. The algorithm is not polynomial, since it has been shown in [10] that some instances can be represented in  $O(\log n)$  space and the problem restricted to these instances remains  $NP$ -hard. Hence, when  $\Delta$  is fixed, the proposed algorithm requires a time which is polynomial in  $n$ , that is pseudo-polynomial in the size of the instances. For practical contexts,  $\Delta \ll n$  is a reasonable restriction, since we may require that in at most  $\Delta$  steps (independently from the input instance) a possible delay must be absorbed. We evaluate the proposed algorithm both theoretically and experimentally. The latter evaluation is done by using random networks and real world instances of the problem provided by the Italian railways company, Trenitalia [12]. It turns out that, even if the algorithm is not polynomial, it is very efficient in practice. Moreover, we show the applicability and the low costs in terms of slack times needed for making robust the considered timetables.

## 2 Recoverable Robust Timetabling Problem

In railway systems, events and dependencies among events are modeled by *event activity networks* (see [7]). They are directed graphs where nodes represent arrival or departure events of trains and arcs represent activities occurring between events (waiting in a train, driving between stations or changing to another train). Event activity networks are a particular class of direct acyclic graphs (DAGs).

Given a DAG  $G = (V, A)$ , the timetabling problem consists in assigning a time to each event in such a way that all the constraints provided by the set of activities are respected. A function  $L : A \rightarrow \mathbb{N}$  assigns the minimal duration time to each activity. A timetable  $\Pi \in \mathbb{R}_{\geq 0}^{|V|}$  for  $G$  is an assignment of a time  $\Pi(u)$  to each event  $u \in V$  such that  $\Pi(v) - \Pi(u) \geq L(a)$ , for all  $a = (u, v) \in A$ .

Given a function  $w : V \rightarrow \mathbb{R}_{\geq 0}$  that assigns a weight to each event, an optimal solution to the timetabling problem minimizes the total weighted time for all events. Formally,  $TT$  is as follows.

---

### $TT$

---

GIVEN: A DAG  $G = (V, A)$ , and functions  $L : A \rightarrow \mathbb{N}$  and  $w : V \rightarrow \mathbb{R}_{\geq 0}$ .

PROBLEM: Find a function  $\Pi : V \rightarrow \mathbb{R}_{\geq 0}$  such that  $\Pi(v) - \Pi(u) \geq L(a)$  for all  $a = (u, v) \in A$  and  $f(\Pi) = \sum_{v \in V} w(v)\Pi(v)$  is minimal.

---

An instance  $i$  of  $TT$  is a triple  $(G, L, w)$ , where  $G$  is a DAG,  $L$  associates a minimal duration time to each activity, and  $w$  associates a weight to each event. The set of instances for  $TT$  is denoted by  $I$ . The set of feasible solutions for  $i \in I$  is:  $F(i) = \{\Pi : \Pi(u) \in \mathbb{R}_{\geq 0}, \forall u \in V \text{ and } \Pi(v) - \Pi(u) \geq L(a), \forall a = (u, v) \in A\}$ .

A solution  $\Pi$  for  $TT$  may induce a positive slack time  $s(a)$  for each  $a \in A$ . In particular, since the planned duration of an activity  $a = (u, v)$  is given by  $\Pi(v) - \Pi(u)$ , then  $s(a) = \Pi(v) - \Pi(u) - L(a)$ . Problem  $TT$  can be solved in linear time by assigning the minimal possible time to each event (e.g. by using the Critical Path Method [11,13]). However, in practical context, delays on the

scheduled activities may occur. In this case, an optimal solution for  $TT$  could result unfeasible and recovery (on-line) strategies become necessary.

Given an instance  $i = (G, L, w)$  for  $TT$ , and a constant  $\alpha \in \mathbb{R}_{\geq 0}$ , we consider a single delay of at most  $\alpha$  time. This is modeled as an increase on the minimal duration time of the delayed activity. We denote the set of instances of  $TT$  that can be obtained by applying all possible modifications to  $i$  as a function  $M(i)$  which is formally defined as follows:  $M(i) = \{(G, L', w) : \exists \bar{a} \in A : L(\bar{a}) \leq L'(\bar{a}) \leq L(\bar{a}) + \alpha, L'(a) = L(a) \forall a \neq \bar{a}\}$ .

Recovery capabilities against delays are modeled as a class  $\mathbb{A}$  of algorithms whose definition depends on the concept of *events affected by one delay* as follows.

**Definition 1.** *Given a DAG  $G = (V, A)$ , a function  $s : A \rightarrow \mathbb{R}_{\geq 0}$ , and a number  $\alpha \in \mathbb{R}_{\geq 0}$ , a node  $x$  is  $\alpha$ -affected by  $a = (u, v) \in A$  ( $a$   $\alpha$ -affects  $x$ ) if there exists a path  $p = (v_0 \equiv u, v_1 \equiv v, \dots, v_k \equiv x)$  in  $G$ , such that  $\sum_{i=1}^k s((v_{i-1}, v_i)) < \alpha$ . The set of nodes  $\alpha$ -affected by an arc  $a = (u, v)$  is denoted as  $\text{Aff}(a)$ .*

In the following, given a solution  $\Pi$  for  $i = (G, L, w)$ , the slack time function induced by  $\Pi$  is used as the function  $s$  in the previous definition. It is assumed that the recovery capabilities allow to change the time of at most  $\Delta$  events. Formally, each algorithm in  $\mathbb{A}$  is able to compute a solution  $\Pi' \in F(j)$  if  $|\text{Aff}(a)| \leq \Delta$ , where  $\Delta \in \mathbb{N}$ . This implies that a robust solution must guarantee that, if a delay of at most  $\alpha$  time occurs, then it affects at most  $\Delta$  events.

We define the recoverable robust timetabling problem  $\mathcal{RTT}$  as the problem of finding a timetable that can be recovered by changing the time of at most  $\Delta$  events when a delay of at most  $\alpha$  time occurs. According to the recoverable robustness model in [89], such a problem is defined as  $\mathcal{RTT} = (TT, M, \mathbb{A})$ .

In other words, a solution  $\Pi$  for an instance  $i$  is feasible for  $\mathcal{RTT}$  if it can be *recovered* by applying an algorithm in  $\mathbb{A}$  which changes the time of at most  $\Delta$  events for each possible disturbance  $j \in M(i)$ . The solution  $\Pi$  is called a *robust solution* for  $i$  w.r.t. the original problem  $TT$ .

A *robust algorithm* for  $TT$  is any algorithm  $A_{rob}$  such that, for each  $i \in I$ ,  $A_{rob}(i)$  is a robust solution for  $i$  w.r.t.  $TT$ . The quality of a robust solution is measured by the *price of robustness*. The price of robustness of  $A_{rob}$  is the worst case ratio between the robust solution for an instance of  $\mathcal{RTT}$  and the corresponding optimal solution for the underlying  $TT$  problem. Formally,  $P_{rob}(\mathcal{RTT}, A_{rob}) = \max_{i \in I} \left\{ \frac{f(A_{rob}(i))}{\min\{f(x) : x \in F(i)\}} \right\}$ . The *price of robustness*  $P_{rob}(\mathcal{RTT})$  of problem  $\mathcal{RTT}$  is defined as the minimum price of robustness among all the robust algorithms. An algorithm  $A_{rob}$  is  *$\mathcal{RTT}$ -optimal* if  $P_{rob}(\mathcal{RTT}, A_{rob}) = P_{rob}(\mathcal{RTT})$ . A robust solution  $\Pi$  for an instance  $i$  of  $\mathcal{RTT}$  is  *$\mathcal{RTT}$ -optimal* if:  $f(\Pi) = \min\{f(\Pi') : \Pi' \text{ is a feasible solution for } \mathcal{RTT}\}$ .

### 3 Pseudo-polynomial Algorithms for Fixed $\Delta$

In this section, we give some theoretical results, whose omitted proofs can be found in [14]. We concentrate our attention to instances of  $\mathcal{RTT}$  where the DAG

is a tree with  $n$  nodes. Hence, in the remainder of the paper, we will refer as  $\mathcal{RTT}$  to the problem restricted to trees. We denote as  $T = (V, A)$  a tree rooted in  $r$ . If  $v \in V$ ,  $\text{deg}(v)$  denotes the degree of  $v$ ,  $T_v$  denotes the subtree of  $T$  rooted in  $v \in V$ . Given a subtree  $T_v$ ,  $N_o(T_v)$  denotes the set of nodes  $y$  such that  $(x, y) \in A$ ,  $x \in T_v$  and  $y \notin T_v$ . We denote by  $w(T_v)$  the value  $\sum_{x \in T_v} w(x)$  and by  $|T_v|$  the number of nodes contained in  $T_v$ . Note that, in order to compute all the values  $w(T_v)$  for each  $v \in V$ , one visit of  $T_r$  is sufficient.

In the next lemma, we prove that for any instance, there exists a  $\mathcal{RTT}$ -optimal solution which assigns only slack times of size  $\alpha$ .

**Lemma 1.** *Given an instance  $i$  of  $\mathcal{RTT}$ , for each solution  $\Pi$  for  $i$ , there exists a solution  $\Pi'$  for  $i$  such that  $f(\Pi') \leq f(\Pi)$  and, for each arc  $a = (x, y)$ , either  $\Pi'(y) = \Pi'(x) + L(a)$  or  $\Pi'(y) = \Pi'(x) + L(a) + \alpha$ .*

Denoted by  $\mathcal{RTT}_\Delta$  the problem  $\mathcal{RTT}$  when the maximal number  $\Delta$  of affected nodes allowed is fixed a priori, algorithm  $\text{SA}_\Delta$ , provided in Figure 1, is  $\mathcal{RTT}_\Delta$ -optimal for any fixed  $\Delta \geq 1$ . The computational complexity of  $\text{SA}_\Delta$  is  $O(n^{\Delta+1})$ . In order to characterize a solution  $\Pi$ , we need the following definition and lemma.

**Definition 2.** *Given a solution  $\Pi$  of  $\mathcal{RTT}_\Delta$  and a node  $v \in V$ , a ball  $B_\Pi(v)$  is the maximal subtree rooted in  $v$  s. t. for each arc  $a = (x, y)$  in  $B_\Pi(v)$ ,  $s(a) = 0$ .*

**Lemma 2.** *For each instance of  $\mathcal{RTT}_\Delta$ , there exists a  $\mathcal{RTT}_\Delta$ -optimal solution  $\Pi$  such that for each  $v \in V$ ,  $B_\Pi(v)$  cannot be extended by adding any node from  $N_o(B_\Pi(v))$  while keeping feasibility and, unless  $\Delta = 0$ , at most one of two consecutive arcs has a slack time of  $\alpha$ .*

Then, for any  $\Delta \geq 1$ , there exists a  $\mathcal{RTT}_\Delta$ -optimal solution  $\Pi$  with the following structure. By Lemma 2, for each arc  $a$  outgoing from the root  $r$ ,  $s(a) = 0$ . Then, for each  $v \in N_o(r)$ ,  $\Pi$  induces a ball  $B_\Pi(v)$  such that  $|B_\Pi(v)| \leq \Delta$ . In particular,  $|B_\Pi(v)| < \Delta$  only if  $|T_v| < \Delta$ . As a consequence,  $|B_\Pi(r)| \leq 1 + \Delta \cdot \text{deg}(r)$ . For each arc  $a = (x, y)$  such that  $x \in B_\Pi(r)$  and  $y \notin B_\Pi(r)$ ,  $s(a) = \alpha$ . By Lemma 2, for each arc  $a$  outgoing from  $y$ ,  $s(a) = 0$  and the same arguments used for  $B_\Pi(r)$  can be used to characterize  $B_\Pi(y)$ .

A possible approach can be that of enumerating all the solutions with the above structure and choosing the cheapest one. Note that, such an approach has a computational time which is exponential in  $n$ . In what follows, we show a recursive approach which avoids to consider a large number of solutions and thus reducing the computational time to a polynomial in  $n$ . The algorithm  $\text{SA}_\Delta$  works as follows. It assigns  $\Pi(r) = 0$  and no slack times to arcs outgoing from  $r$ . Then, for each  $v \in N_o(r)$  it has to decide which subtree of  $T_v$  belongs to  $B_\Pi(r)$ . To do this, it evaluates the cost, in terms of the value of the objective function, of any possible subtree  $B$  of  $T_v$  rooted at  $v$  of size at most  $\Delta$  and then chooses the subtree which implies the cheapest solution.

For each already defined ball  $B_\Pi$ , this procedure is then repeated for each  $v \in N_o(B_\Pi)$  which does not belong to a defined ball by using  $v$  as the root.

**Algorithm SA $_{\Delta}$** **Input:**  $v \in V$ **Output:**  $(\Pi, f(\Pi))$ , times assigned at nodes in  $T_v$  with  $\Pi(v) = 0$  and  $f(\Pi)$ 

1.  $\Pi(v) = 0, f_{\Pi} = 0$
2. **for each**  $v_i \in N_o(v)$
3.      $f_{\min} = +\infty$
4.     **for each** maximal subtree  $B$  rooted at  $v_i$  of  $T_{v_i}$ , such that  $|B| \leq \Delta$
5.          $\Pi_B(v_i) = L((v, v_i)), f_B = L((v, v_i)) \cdot w(v_i)$
6.         **for each**  $(x, y) \in B$
7.              $\Pi_B(y) = \Pi_B(x) + L((x, y))$
8.              $f_B = f_B + \Pi_B(y) \cdot w(y)$
9.         **for each**  $(u, z) \in A$ , such that  $u \in B$  and  $z \notin B$
10.              $(\Pi_{T_z}, f_{T_z}) = \mathbf{SA}_{\Delta}(z)$
11.              $f_B = f_B + f_{T_z} + (\Pi_B(u) + L((u, z)) + \alpha) \cdot w(T_z)$
12.         **if**  $f_B < f_{\min}$  **then**
13.              $f_{\min} = f_B$
14.              $B_{\min} = B$
15.          $\Pi(v_i) = L((v, v_i))$
16.         **for each**  $(x, y) \in B_{\min}$
17.              $\Pi(y) = \Pi(x) + L((x, y))$
18.         **for each**  $(u, z) \in A$ , such that  $u \in B_{\min}$  and  $z \notin B_{\min}$
19.             **for each**  $x \in T_z$
20.                  $\Pi(x) = \Pi_{T_z}(x) + \Pi(u) + L((u, z)) + \alpha$
21.          $f_{\Pi} = f_{\Pi} + f_{\min}$
22. **return**  $(\Pi, f_{\Pi})$

**Fig. 1.** Recursive algorithm to compute a robust timetable on a tree

The cost of a subtree  $B$  rooted at  $v$  is computed as the value of the objective function when  $B$  is chosen as a ball rooted in  $v$ . That is, for each arc  $a \in B$ ,  $s(a) = 0$ ; for each  $a = (x, y) \in A$  such that  $x \in B$  and  $y \notin B$ ,  $s(a) = \alpha$ ; and for each node in  $T_y$ , an optimal solution is chosen. Computing this cost requires to know the optimal solution of a subtree, this is done by using recursively  $\mathbf{SA}_{\Delta}$ .

Formally,  $\mathbf{SA}_{\Delta}$  is given in Figure 1. It takes a node  $v$  as input and it returns a pair  $(\Pi, f(\Pi))$  where  $\Pi$  is a  $\mathcal{RTT}$ -optimal timetable for  $T_v$  and  $f(\Pi)$  is its value of the objective function. A solution for  $\mathcal{RTT}_{\Delta}$  is computed by calling  $\mathbf{SA}_{\Delta}(r)$ .

In detail, Lines 1 assigns  $\Pi(v)$  and initializes  $f_{\Pi}$ . For each  $v_i \in N_o(v)$ , Lines 3–21 compute  $\Pi$  for the subtree  $T_{v_i}$ . Lines 3–14 compute a subtree  $B_{\min}$  of  $T_{v_i}$  rooted at  $v_i$  of size at most  $\Delta$  which implies the cheapest solution of cost  $f_{\min}$ . To do this, Line 4 enumerates all the possible subtree  $B$  rooted at  $v_i$  of  $T_{v_i}$  of size at most  $\Delta$ . Then, Lines 5–8 compute the cost  $f_B$  of nodes in  $B$  and Lines 9–11 compute the cost of nodes not in  $B$  by summing for each  $(u, z) \in A$ , such that  $u \in B$  and  $z \notin B$  the cost  $f_{T_z}$  of a solution of  $T_z$  computed by recursively calling  $\mathbf{SA}_{\Delta}(z)$ . If  $f_{T_z}$  is the cost of a solution of a subtree  $T_z$ , then the contribution to  $f_B$  of all nodes in  $T_z$  is  $(\Pi_B(u) + L((u, z)) + \alpha) \cdot w(T_z)$  where  $\Pi_B(u)$  is the time assigned to  $u$  if  $B$  is chosen as a ball rooted in  $v_i$ . In fact, each time assigned

to such nodes have been shifted of  $\Pi_B(u) + L((u, z)) + \alpha$ . Finally, Lines 15–21 assign  $\Pi$  for the subtree  $T_{v_i}$  by choosing  $B_{\min}$  as a ball rooted in  $v_i$ .

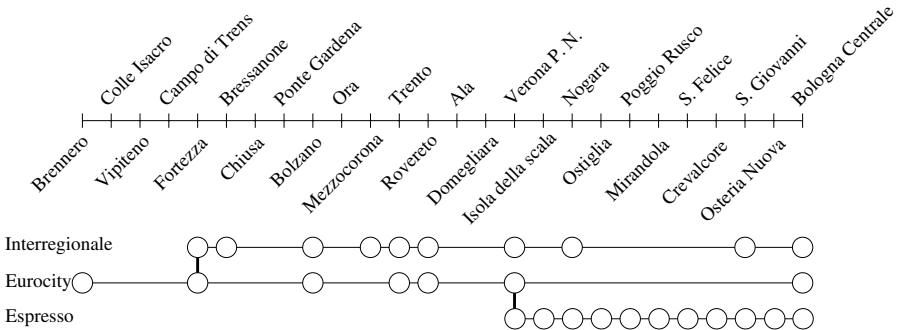
**Theorem 1.**  $SA_\Delta$  is  $\mathcal{RTT}_\Delta$ -optimal and requires  $O(n^{\Delta+1})$  time and  $O(n^2)$  space, where  $n$  is the number of nodes.  $P_{rob}(\mathcal{RTT}_\Delta, SA_\Delta) \leq 1 + \frac{\alpha}{2}$ .  $P_{rob}(\mathcal{RTT}_\Delta) \geq 1 + \frac{\alpha}{\Delta+1}$ .

## 4 Experimental Study

### 4.1 Real World Data

We consider real case scenarios of *Single-Line Corridors*. A corridor is a sequence of stations represented by a line. The stations are linked by multiple tracks and each station is served by many trains of different types. Types of trains concern the locations that each train serves and its maximal speed. For an example, see Figure 2. In these systems, it is a practical evidence that slow trains wait for faster trains in order to serve passengers to small stations. This situation is modelled with the only assumption that the changes of passengers from one train to another at a station must be guaranteed only when the second train is starting its journey from the current station. In practice, we do not require as a constraint the possibility for passengers to change for a train which has already started its journey. This does not mean that passengers cannot change train at some station in the middle of a train journey, but only that this is not considered as a constraint. Further motivations for this model can be found in [15,16].

Let us consider the real world example provided in Figure 2 where three trains serve the same line. The slowest train, the Espresso, goes from Verona to Bologna, the Interregionale goes from Fortezza to Bologna, and the fastest one, the Euro-City, goes from Brennero to Bologna. The Euro-City starts its journey before all the other trains, and it arrives at Fortezza station before the departure event of the Interregionale. At Verona Station, the Espresso is scheduled to start its journey after the arrival event of the Euro-City. Hence, there is an arc between the Euro-City and the starting event corresponding to the Interregionale at Fortezza station, and another arc connecting the Euro-City



**Fig. 2.** Example of three trains serving a line. For each station and for each train, we represent only one circle which indeed corresponds to an arrival and a departure event.

**Table 1.** Data used in the experiments

Corridor	Line	N. of Stations	N. of Trains
BrBo	Brennero–Bologna	48	68
MdMi	Modane–Milano	54	291
BzVr	Bolzano–Verona	27	65
PzBo	Piacenza–Bologna	17	25

**Table 2.** Sizes of the trees

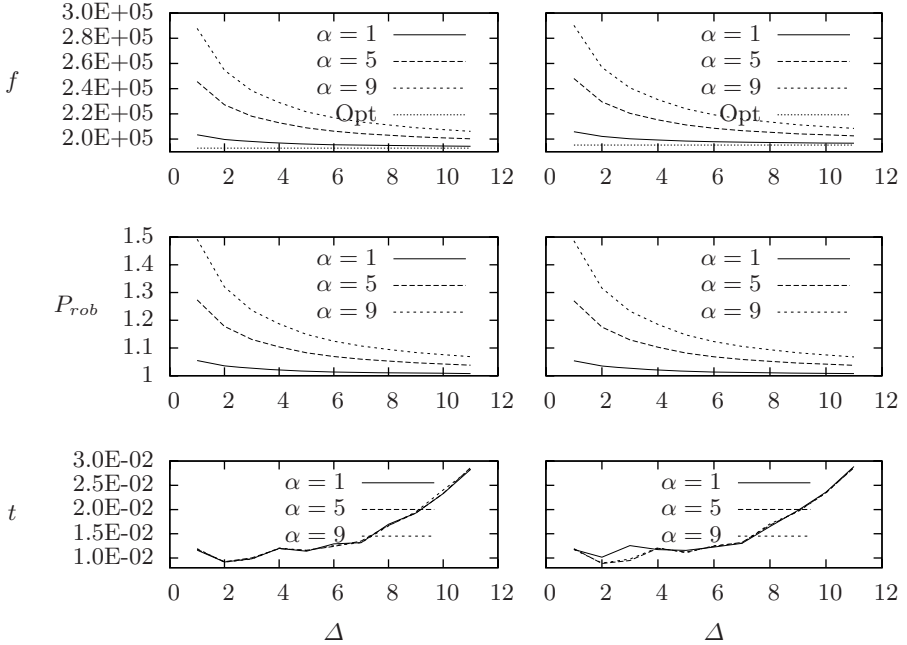
Corridor	N. of Nodes	Max. Traveling Time	Avg. Activity Time
BrBo	1103	516	9
MdMi	4358	318	8
BzVr	648	197	5
PzBo	163	187	10

to the starting event of the Espresso at Verona station. An arc which represents a changing activity can only connect one node to the head of a branch. The DAG obtained by this procedure is a tree, as shown in Figure 2. In general, the result of this procedure is a forest and we link the roots of the trees in this forest to a unique root event (for details, see [15]). The weights on the events represent the relevance of the trains which they belong to, the weight of the root is 0.

Table 1 shows the data used in the experiments referring to four corridors provided by Trenitalia [12]. Starting from the provided data and according to the described requirements, we derived event activity networks having tree topologies whose sizes are reported in Table 2. We then apply the  $SA_{\Delta}$  algorithm on different scenarios, comparing the obtained robust timetables with the optimal non-robust ones.

Our experiments are based on three parameters. Namely, we vary on the maximum number  $\Delta$  of events that can be affected by a delay, the maximum time delay  $\alpha$ , and the case of average or real times  $L$  needed to perform the scheduled activities. In what follows, all the activities times and the delays are expressed in minutes. In order to obtain  $RIT$  instances, for each corridor among BrBo, MdMi, BzVr and PzBo, we vary  $\Delta \in \{1, 2, \dots, 11\}$  and  $\alpha \in \{1, 5, 9, 13, 17\}$ . Moreover, we use two different functions  $L$ : the first one is based on the real values obtained by available data; the second one is the constant average function which assigns to each activity the same duration time obtained as the average among all real values of each instance. This second function is used to test the behavior of the algorithm based only on the network topology, in order to understand the dependability w.r.t. real values. The average activity times are shown in Table 2.

For each corridor we show three diagrams concerning the objective function  $f$ , the price of robustness  $P_{rob}$  of  $SA_{\Delta}$ , and the computational time  $t$  needed by  $SA_{\Delta}$  in the mentioned cases. In each diagram, we show three curves which represent the results obtained by setting  $L$  to real values and  $\alpha \in \{1, 5, 9\}$ . Results obtained by assigning  $\alpha \in \{13, 17\}$  are not shown as they are less significant being  $\alpha$  too large compared with the average activity time. Furthermore, for the instance



**Fig. 3.** Corridor BrBo with real values of minimum activity times (left) and average activity times (right)

BrBo, we give the three diagrams obtained by setting  $L$  to the corresponding average activity time. For any other instance we do not give these diagrams as the inferred properties do not change. The full set of results can be found in [14]. All the experiments have been carried out on a workstation equipped with a 2,66 GHz Intel Core2 processor, 8Gb RAM, Linux (kernel 2.6.27) and gcc 4.3.3.

In the obtained diagrams, the values of the objective function  $f$  of the robust problem are compared to the optimum value of  $f$  for the non-robust problem. As  $\Delta$  increases, the curves tend rapidly to the optimum. For small values of  $\alpha$ , the price of robustness is very low. Concerning the diagrams representing the computational times, we can see that our tests required a very small amount of time. However, the exponential growth of the curves as  $\Delta$  increases is already evident. Surprisingly, for practical purposes, our experiments show that algorithm  $SA_{\Delta}$  can be safely applied without requiring ages of computation.

**Corridor BrBo** (see Figure 3, left). This corridor is quite large in terms of stations and trains as shown in Table 1. We can see that the price of robustness is very close to 1 when  $\alpha = 1$  while it is almost 1.5 when considering big delays of  $\alpha = 9$  and  $\Delta = 1$ . When  $\Delta = 1$ , the algorithm adds one slack time for each pair of consecutive arcs. Then, the value of  $P_{rob}$ , when  $\Delta = 1$ , is about  $\frac{2L_{avg} + \alpha}{2L_{avg}} = 1 + \frac{\alpha}{2L_{avg}}$ , where  $L_{avg}$  is the average activity time, as shown in Figure 3. It is interesting to note how the values of  $f$  and  $P_{rob}$  decrease quickly



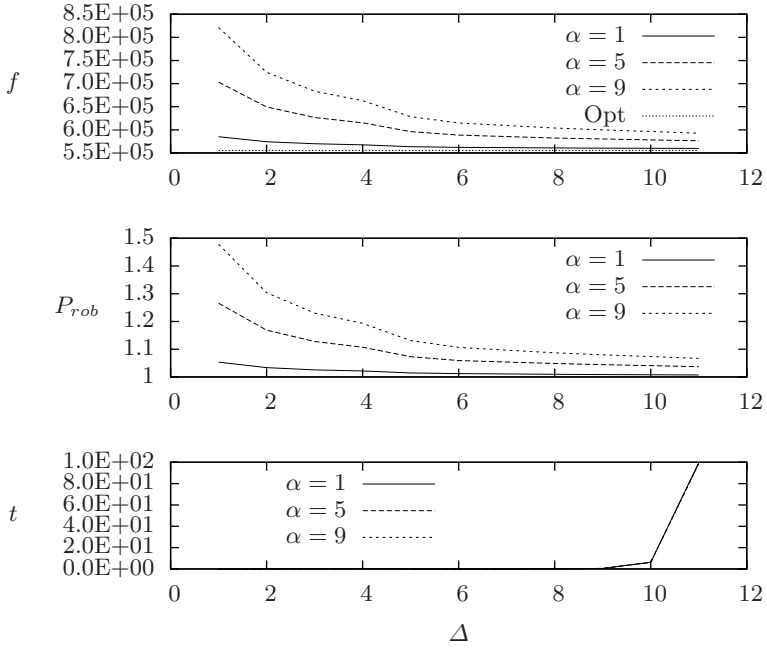


Fig. 4. Corridor MdMi

with  $\Delta$ . In particular, the price of robustness is between 1.00754 and 1.06785, when  $\Delta = 11$ . This implies that adding robustness reflects an increasing in the costs of just 0.8 – 6.8%. The computational time increases with  $\Delta$  but it is less than 30 milliseconds in the worst case. In detail, for  $\Delta = 11$  we need about 28 milliseconds to achieve a price of robustness of at most 1.06785.

**Corridor BrBo (average activity times).** (see Figure 3, right). In this case, the objective function takes almost identical values w.r.t. the previous case. As we expect, the value of the objective function does not depend on the value of  $L$ , but only on the structure of the tree and on the size of the delay.  $P_{rob}$  strictly depends on  $\alpha/L_{avg}$  which can be considered a parameter for evaluating the magnitude of a delay. For  $\Delta = 1$  an optimal robust timetable has to assign one slack time of size  $\alpha$  for each pair of consecutive activities. It follows that, if  $\alpha = 9$  and  $L_{avg} = 9$ , the price of robustness is 1.5, as it can be seen in Figure 3. The same happens for  $\alpha = 5$  where the expected value is about 1.27.

**Corridor MdMi** (see Figure 4). This corridor is the biggest in terms of stations and trains. As shown in Table 1, the number of considered trains is more than four times the one in BrBo, while the number of stations is slightly more. Still, we can see comparable performances for the price of robustness even though the incidence of the required computational time is more evident. However, as the

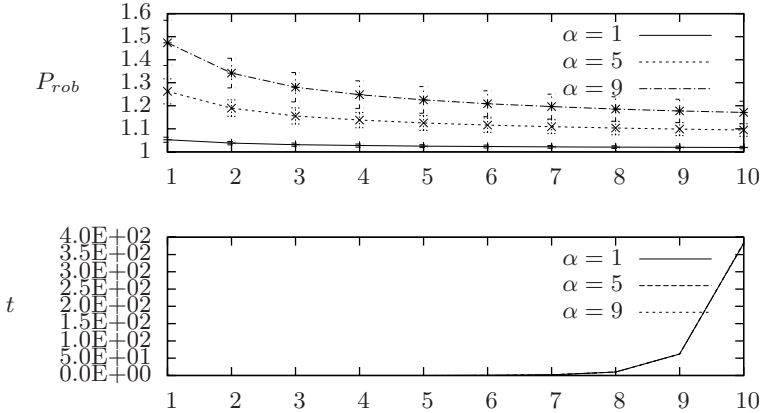


Fig. 5. Randomly generated trees

timetables are calculated at the planning phase and not at runtime, the required time is still of an acceptable order being about 96 seconds in the worst case. Results regarding corridors BzVr and PzBo are reported in [14].

## 4.2 Randomly Generated Data

By analyzing the obtained results, we note that the time required by the algorithm is negligible w.r.t. the theoretical bound. This suggests that those real world instances have some hidden properties. One cause might be the almost linearity of the input trees which are made of long paths and whose nodes have low outdegree. In order to investigate on this matter, we test the algorithm on a set of five randomly generated trees. Each tree contains 1000 nodes and is generated starting from a single node and then by linking a new generated node to an existing one extracted uniformly at random. The node weights randomly range in  $[1, 10]$ , and the minimum duration times of activities randomly range in  $[1, 18]$ . In this way, the average activity duration time is comparable with that of the real world instances. Finally,  $\Delta \in \{1, 2, \dots, 10\}$  and  $\alpha \in \{1, 5, 9\}$ . For each pair  $(\Delta, \alpha)$ , we performed one test for each generated tree.

In Figure 5, we show the average values of price of robustness and computational time, and the standard deviation of the price of robustness. The results confirm our intuition that the almost linear structure of the real world data heavily influences the computational times. In fact, in this case the time elapsed is about 10000 times worse than that of the corridor BrBo which have comparable size. However, the price of robustness is kept low as in the previous instances.

## 5 Conclusion

We have presented algorithm  $SA_{\Delta}$  for solving the problem of planning robust timetables when the input event activity network topology is a tree. The algorithm ensures that, if a delay occurs, no more than  $\Delta$  activities are influenced by

the propagation of such a delay. We have shown the performances of  $SA_{\Delta}$  both theoretically and experimentally. Despite the problem is proved to be *NP*-hard, the obtained results show the applicability of the algorithm to ensure robust timetables with respect to bounded delays.

## References

1. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Navarra, A.: Delay Management Problem: Complexity Results and Robust Algorithms. In: Yang, B., Du, D.-Z., Wang, C.A. (eds.) COCOA 2008. LNCS, vol. 5165, pp. 458–468. Springer, Heidelberg (2008)
2. De Giovanni, L., Heilporn, G., Labbé, M.: Optimization models for the delay management problem in public transportation. *European Journal of Operational Research* 189(3), 762–774 (2007)
3. Gatto, M., Jacob, R., Peeters, L., Schöbel, A.: The Computational Complexity of Delay Management. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 227–238. Springer, Heidelberg (2005)
4. Gatto, M., Jacob, R., Peeters, L., Widmayer, P.: Online Delay Management on a Single Train Line. In: Geraets, F., Kroon, L.G., Schoebel, A., Wagner, D., Zaroliagis, C.D. (eds.) Railway Optimization 2004. LNCS, vol. 4359, pp. 306–320. Springer, Heidelberg (2007)
5. Ginkel, A., Schöbel, A.: The bicriteria delay management problem. *Transportation Science* 41(4), 527–538 (2007)
6. Schöbel, A.: A model for the delay management problem based on mixed integer programming. *ENTCS* 50(1), 1–10 (2004)
7. Schöbel, A.: Integer programming approaches for solving the delay management problem. In: Geraets, F., Kroon, L.G., Schoebel, A., Wagner, D., Zaroliagis, C.D. (eds.) Railway Optimization 2004. LNCS, vol. 4359, pp. 145–170. Springer, Heidelberg (2007)
8. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Navarra, A.: Robust Algorithms and Price of Robustness in Shunting Problems. In: Proc. of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, pp. 175–190 (2007)
9. Liebchen, C., Lübbecke, M., Möhring, R.H., Stiller, S.: Recoverable robustness. Technical Report ARRIVAL-TR-0066, ARRIVAL Project (2007)
10. D'Angelo, G., Di Stefano, G., Navarra, A.: Recoverable robust timetables on trees. Technical Report ARRIVAL-TR-0163, ARRIVAL project (2008)
11. Cicerone, S., Di Stefano, G., Schachtebeck, M., Schöbel, A.: Dynamic Algorithms for Recoverable Robustness Problems. In: Proc. of the 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (2008)
12. Trenitalia, <http://www.trenitalia.com/>
13. Levy, F., Thompson, G., Wies, J.: The ABCs of the Critical Path Method. Graduate School of Business Administration. Harvard University (1963)
14. <http://informatica.ing.univaq.it/misc/TimetablingTree2009/>
15. D'Angelo, G., Di Stefano, G., Navarra, A.: Recoverable-robust timetables for trains on single-line corridors. Technical Report ARRIVAL-TR-0180, ARRIVAL project (2008)
16. D'Angelo, G., Di Stefano, G., Navarra, A.: Recoverable-robust timetables for trains on single-line corridors. In: 3rd International Seminar on Railway Operations Modelling and Analysis, RailZurich (2009)

# Weighted LCS

## (Extended Abstract)

Amihood Amir<sup>1,\*</sup>, Zvi Gotthilf<sup>2</sup>, and B. Riva Shalom<sup>3</sup>

<sup>1</sup> Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel  
and Department of Computer Science, Johns Hopkins University,  
Baltimore, MD 21218  
`amir@cs.biu.ac.il`

<sup>2</sup> Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel  
`gotthiz@cs.biu.ac.il`

<sup>3</sup> Department of Software Engineering, Shenkar College, Ramat-Gan 52526, Israel  
`riva.shalom@gmail.com`

**Abstract.** The Longest Common Subsequence (LCS) of two strings  $A$  and  $B$  is a well studied problem having a wide range of applications. When each symbol of the input strings is assigned a positive weight the problem becomes the *Heaviest Common Subsequence* (HCS) problem. In this paper we consider a different version of weighted LCS on *Position Weight Matrices* (PWM). The Position Weight Matrix was introduced as a tool to handle a set of sequences that are not identical, yet, have many local similarities. Such a weighted sequence is a ‘statistical image’ of this set where we are given the probability of every symbol’s occurrence at every text location. We consider two possible definitions of LCS on PWM. For the first, we solve the weighted LCS problem of  $z$  sequences in time  $O(zn^{z+1})$ . For the second, we prove  $\mathcal{NP}$ -hardness and provide an approximation algorithm.

## 1 Introduction

The *Longest Common Subsequence* problem, whose first famous dynamic programming solution appeared in 1974 [14], is one of the classical problems in Computer Science. The widely known string version appears in Definition 1.

**Definition 1.** The String Longest Common Subsequence (*LCS*) Problem:

*Input:* Two strings  $A, B$  of length  $n$  over alphabet  $\Sigma$ .

*Output:* The length of the longest subsequence common to both strings.

The LCS problem has been very well studied. For a survey, see [5]. The problem is mainly motivated in measuring the similarity over the input strings. An immediate example from computational biology is measuring the commonality of two DNA molecules or proteins, which may yield functional similarity between them. The well known dynamic programming solution [7] requires a running time of

---

\* Partly supported by ISF grant 35/05.

$O(n^2)$ , for two input strings of length  $n$ . The LCS problem had also been investigated on more general structures such as trees and matrices [2], run-length encoded strings [4], and more.

Another structure, useful in molecular biology, is the weighted sequence. This is defined as a sequence  $S = s_1, \dots, s_{|S|}$  where a value is associated to every  $s_i$ ,  $i = 1..|S|$ . While comparing two weighted sequences we define a weight function,  $W$ , assigning a value to every possible match between two characters one from the first sequence and the other from the second sequence. The LCS variant for these weighted sequences aims at maximizing the weight of the common subsequence, instead of its length as hereafter defined:

**Definition 2.** The Heaviest Common Subsequence (HCS) Problem:

*Input:* Two strings  $A = a_1..a_n$ ,  $B = b_1..b_n$  of length  $n$  over alphabet  $\Sigma$  and a weight function  $W : a_i \times b_j \rightarrow N$ .

*Output:* A common subsequence of length  $l$   $a_{i_1}..a_{i_l} = b_{j_1}..b_{j_l}$  maximizing the sum  $\sum_{k=1}^l W(a_{i_k}, b_{j_k})$

Note that in contrast to sequence alignment problem, where we have a single weight for the matching of two characters, in the HCS problem the weight of the match depends on the position of the symbols in the input sequences *as well as* on the characters themselves.

Recently, another model of weighted sequences was introduced in which, at each position of the sequence, any symbol of the alphabet can occur with a certain probability. To prevent ambiguity, we refer to such sequences as *p-weighted sequences*, though in the literature they are both named weighted sequences.

**Definition 3.** ([9]) A *p-weighted sequence*  $A = a_1..a_n$  over alphabet  $\Sigma$ , is a sequence of sets  $a_i$ ,  $1 \leq i \leq n$ . Every  $a_i$  is a set of pairs  $(s_j, \pi_i(s_j))$ , where  $s_j \in \Sigma$  and  $\pi_i(s_j)$  is the probability of having symbol  $s_j$  at location  $i$ .

Formally,  $a_i = \{(s_j, \pi_i(s_j)) \mid s_j \neq s_l \text{ for } j \neq l, \text{ and } \sum_j \pi_i(s_j) = 1\}$ .

The concept of p-weighted sequences was introduced as a tool for motif discovery and local alignment. A weighted sequence is called in the biological literature a “Position Weight Matrix” (PWM) [12]. A p-weighted sequence of length  $m$  is a  $|\Sigma| \times m$  matrix that reports the frequency of each symbol in a finite alphabet  $\Sigma$  for every possible location.

The first usage of PWM sequences was for relative short sequences, for example binding sites, sequences resulting from multiple alignment etc. Iliopoulos et. al. [9] considered building very large Position Weight Matrices that correspond, for example, to complete chromosome sequences that have been obtained using a whole-genome shotgun strategy [13]. By keeping all the information the whole-genome shotgun produces, it is possible to ferret out information that has been previously undetected after being faded during the consensus step. This concept is true for other applications where local similarities are thus encoded. Therefore, the necessity of developing adequate algorithms for p-weighted sequences increases.

It is natural to extend the LCS definition to p-weighted strings as a means of measuring their similarity. However the PWM model deals with probabilities,

thus values smaller than 1 are multiplied as a subsequence is extended. The heaviest common p-weighted subsequence will always be of length 1, since every added symbol reduces the total weight. Therefore, we define a **new** but related problem named *Longest Common Weighted Subsequence*, in which the weight is allowed to decrease till a certain bound, and under this restriction the longest common subsequence is sought.

The bound is set according to the certainty level required in the application. Since we consider two p-weighted sequences, we differentiate between their probabilities by denoting  $\pi_i^A$  the probability of occurring at the  $i$ th location of sequence  $A$ . The formal definition appears below.

**Definition 4.** The Longest Common Weighted Subsequence (*LCWS*) Problem:

*Input:* Two p-weighted strings  $A, B$  of length  $n$  over alphabet  $\Sigma$ , and a constant  $\alpha$ ,  $0 < \alpha \leq 1$ .

*Output:* The maximal  $l$  such that there is a common subsequence of length  $l$ ,

$$a_{i_1}..a_{i_l} = b_{j_1}..b_{j_l}, \text{ where } \prod_{y=1}^l (\pi_{i_y}^A(a_{i_y}) \cdot \pi_{j_y}^B(b_{j_y})) \geq \alpha.$$

Though the *LCWS* problem seems natural for the position weighted matrices input, in case the probabilities of the characters of one input sequence are far from being uniformly distributed, the results may be biased and not reflect a real relation between the weighted sequences. In order to prevent this effect, and obtain informative results we suggest an additional definition to the *LCWS* problem, Longest Common Weighted Subsequence with two thresholds, referred to as *LCWS2*. In the *LCWS2* problem, a separate probability bound is set for each of the p-weighted sequences.

**Definition 5.** The Longest Common Weighted Subsequence 2 (*LCWS2*) Problem:

*Input:* Two p-weighted strings  $A, B$  of length  $n$  over alphabet  $\Sigma$ , and constants  $\alpha_1, \alpha_2$ ,  $0 < \alpha_i \leq 1$ .

*Output:* The maximal  $l$  such that there is a common subsequence of length  $l$ ,

$$a_{i_1}..a_{i_l} = b_{j_1}..b_{j_l}, \text{ where } \prod_{y=1}^l \pi_{i_y}^A(a_{i_y}) \geq \alpha_1 \text{ AND } \prod_{y=1}^l \pi_{j_y}^B(b_{j_y}) \geq \alpha_2.$$

In real-world applications it is rarely the case that one needs to compare only two data instances. Rather, it is important to be able to compare multiple sequences. Consequently, we generalize the *LCWS* problems to multiple sequences and show that our algorithm generalizes in the natural way.

This paper is organized as follows: Section 2 describes related work. The *LCWS* problem solution and its extension appear in Section 3. We consider the *LCWS2* problem and its hardness in Section 4. Section 5 concludes the paper and poses some open questions.

## 2 Related Work

Jacobson and Vo [10] solved the Heaviest Common Subsequence problem by reducing it to the Heaviest Increasing Subsequence problem (HIS). Their algorithm for the Heaviest Common Subsequence runs in  $O((r+n) \log n)$  time, where

$r$  is the number of matches between  $A$  and  $B$  and  $n$  is the length of the input sequences. For small alphabets with a uniform distribution, the time may be  $O(n^2 \log n)$ . Recently Li [11] gave a linear space algorithm for the HCS problem.

Regarding the p-weighted sequences, Iliopoulos et al. [8] defined the problem of longest common *substring* of p-weighted sequences, where the common sequence is *consecutive*. They suggested solving the problem using a p-weighted generalized suffix tree, in which the longest branch common to both strings is the answer. Their problem is a special case of the LCWS problem.

Amir et. al. [1] showed some conditions where p-weighted matching problems can be reduced to ordinary pattern matching problems. In their model, the probability is fixed, and the text is p-weighted while the pattern is an ordinary string. Both these assumptions are not valid for the *LCWS* problem.

Finally, Amir et. al. [3] have defined weighted Hamming and edit distances. Although edit distance and LCS are known to be related, our model and that of [3] are different in that they consider a p-weighted text and a regular pattern. The case of Amir et. al. [3] is the special case in our model where all probabilities of the sequences equal one.

### 3 Longest Common Weighted Subsequence (LCWS)

The resemblance between the HCS and LCWS problems lies in the weight demands on the common subsequence. However, there is a substantial difference between the problems. The HCS maximizes a single parameter – the weight – whereas the LCWS maximizes the length under a weight restriction.

The weight bound does force the algorithm to maximize the weight at every step, yet not as a goal but rather as a byproduct. Consider the example in Fig. 1. Let the associated weight function of the HCS be multiplying the probabilities of the symbols, as given in the third table. The HCS result will be a common subsequence of length one, obtained from matching  $a_1$  to  $b_3$ , with weight  $56/81$ . Nevertheless, the LCWS for  $\alpha = 1/9$  will return length 2 obtained from matching  $a_2$  to  $b_1$  and  $a_3$  to  $b_2$ , which has a lower probability (weight) yet respects the threshold and yields a longer subsequence. As a consequence, a new method for solving the Longest Common Weighted Subsequence problem is required.

We present a dynamic programming algorithm for the *LCWS* problem. We construct a two dimensional table, where the columns represent the characters of the  $A$  sequence, and the rows refer to the characters of sequence  $B$ . A character in a p-weighted sequence is a table containing all symbols of  $\Sigma$  and the probability of appearing at that location.

As above mentioned, the core of the LCWS problem is maximizing the LCS length under a weight restriction. Consequentially, we cannot save at every entry merely the highest probability achieved so far as it may, in the future, degrade below  $\alpha$  and would have to be discarded. We therefore save at entry  $i, j$ , for every possible length, the highest probability of a common subsequences that can be obtained from  $A[1..j]$  and  $B[1..i]$ . We denote the variables containing this information by  $l_{i,j}^k$ , where  $k$  represents the length of the common subsequence. Saving these probabilities, when some  $l_{i,j}^k$  is too small, we still have the

A	$\Pi_1^A(0) = 1/9$	$\Pi_2^A(0) = 2/3$	$\Pi_3^A(0) = 2/3$
	$\Pi_1^A(1) = 8/9$	$\Pi_2^A(1) = 1/3$	$\Pi_3^A(1) = 1/3$
	$a_1$	$a_2$	$a_3$

B	$b_1$	$b_2$	$b_3$
	$\Pi_1^B(0) = 1/2$	$\Pi_2^B(0) = 2/3$	$\Pi_3^B(0) = 2/9$
	$\Pi_1^B(1) = 1/2$	$\Pi_2^B(1) = 1/3$	$\Pi_3^B(1) = 7/9$

$prob(a_i, b_j)$	$a_1$	$a_2$	$a_3$
$b_1$	4/9	1/3	1/3
$b_2$	8/27	4/9	4/9
$b_3$	56/81	7/27	7/27

**Fig. 1.** An example of two p-weighted sequences

information regarding  $l_{i,j}^{k-1}$ , which may increase its length in future steps and still exceed  $\alpha$  in weight.

As each position in a p-weighted sequence consists of  $|\Sigma|$  symbols and their probabilities, when considering the matching of  $a_i$  and  $b_j$  we compute for each symbol  $\sigma \in \Sigma$  the product  $\pi_i^A(\sigma)\pi_j^B(\sigma)$  and select the highest value. We denote the selected value of entry  $i, j$  as  $best_{i,j}$  and save the symbol yielding this probability.

We can fill the dynamic programming table in row-major order. Computing an entry  $i, j$  implies computing the most probable common subsequence of  $A[1..j]$  and  $B[1..i]$  of length  $k$ ,  $1 \leq k \leq \min\{i, j\}$ . Considering  $l_{i,j}^k$ , the correlated subsequence can be constructed by matching the  $a_j$  and  $b_i$ , selecting their  $best$  symbol, and by this extending a smaller subsequence, or by matching one of  $b_i$  and  $a_j$  to a previous character from the counterpart sequence. Lemma 1 formally defines the computation required for filling an entry in the dynamic programming table.

**Lemma 1**

$$LCWS(B[1..i], A[1..j]) = \{l_{i,j}^k\}_{k=1}^{\min\{i,j\}} = \max\{l_{i,j-1}^k, l_{i-1,j}^k, best_{i,j} \cdot l_{i-1,j-1}^{k-1}\}.$$

Proof: An LCWS entry contains probabilities of most probable common subsequences of length  $k$ .  $k$  must start from 1, which means that only a single element was used for the common subsequence, and is bounded by the length of the longest possible common subsequence of  $A[1..j]$  and  $B[1..i]$ , implying it cannot exceed  $\min\{i, j\}$ .

Computing a certain  $l_{i,j}^k = x$  we will prove the optimality of  $x$  inductively on  $i, j$ . The base case is  $l_{1,1}^1$  when the common subsequence consists of a single symbol obtained by matching  $b_1$  to a character from  $a_1$ . Obviously  $best_{1,1}$ , yields the proper value.

Consider now  $l_{i,j}^k$ . Suppose to the contrary, that the values of  $l_{i',j'}^k, i' < i$ , or  $j' < j$  are optimal, but  $x$  is not the optimal probability of a common subsequence



of length  $k$  of  $A[1..j]$  and  $B[1..i]$ , implying there exists another common subsequence of length  $k$  with probability  $x'$  such that  $x < x'$ . The  $x'$  subsequence can be either obtained by a previous computed subsequence not including a match of  $A[j]$  and  $B[i]$ , or by adding the current match to a  $k - 1$  common subsequence of  $B[1..f]$  and  $A[1..h]$ . In the former case, the  $x'$  subsequence can include a match of  $B[i]$ , a matching of  $A[j]$  or neither of them. Since  $l_{i,j}^k$  maximizes the values of  $l_{i-1,j}^k$ ,  $l_{i,j-1}^k$ , the assumption implies that there exists another subsequence of length  $k$  with probability  $x'$  where  $\max\{l_{i-1,j}^k, l_{i,j-1}^k\} < x'$  contradicting the induction hypothesis of optimal value of  $l_{i',j'}^k$ ,  $i' < i$ , or  $j' < j$ . Note, that  $l_{i-1,j-1}^k$  needs no separate discussion, as it is considered when computing both  $l_{i-1,j}^k$ ,  $l_{i,j-1}^k$ .

The second possible case where the common subsequence has probability  $x'$ , includes matching  $A[j]$  and  $B[i]$ . The fact that  $x < x'$  yields  $\text{best}_{i,j} \cdot l_{i-1,j-1}^{k-1} < \text{best}_{i,j} \cdot l_{f,h}^{k-1}$ ,  $f < i$ ,  $h < j$ , contradicting the optimality of  $l_{i-1,j-1}^{k-1}$ , therefore this possibility is impossible as well. ■

We can fill the whole table and then go over  $\{l_{n,n}^k\}$  in decreasing order of  $k$ , and check whether  $l_{n,n}^k \geq \alpha$ . The first value satisfying the inequality, the relevant  $k$  is returned, as the length of the longest common subsequence under the  $\alpha$  demands.

An example of a LCWS table where  $\alpha = 0.002$  appears in Fig. 2.

Filling the table in this fashion implies computing every entry of the table requires  $O(n + |\Sigma|)$  time for finding the *best* symbol and all  $O(n)$  relevant probabilities, using Lemma 1. So the time complexity is  $O(n^3 + |\Sigma|n^2)$ .

	$\pi_1^A(a) = 0.5$ $\pi_1^A(b) = 0.4$ $\pi_1^A(c) = 0.1$	$\pi_2^A(a) = 0.3$ $\pi_2^A(b) = 0.2$ $\pi_2^A(c) = 0.5$	$\pi_3^A(a) = 0.1$ $\pi_3^A(b) = 0.1$ $\pi_3^A(c) = 0.8$	$\pi_4^A(a) = 0.4$ $\pi_4^A(b) = 0.3$ $\pi_4^A(c) = 0.3$	$\pi_5^A(a) = 0.3$ $\pi_5^A(b) = 0.7$ $\pi_5^A(c) = 0$
$\pi_1^B(a) = 0.2$ $\pi_1^B(b) = 0.4$ $\pi_1^B(c) = 0.4$	(best - 0.16) $l^1 \searrow [b].16$	(best - 0.2) $l^1 \searrow [c].2$	(best - 0.32) $l^1 \searrow [c].32$	(best - 0.12) $l^1 \rightarrow [c].32$	(best - 0.28) $l^1 \rightarrow [c].32$
$\pi_2^B(a) = 0.5$ $\pi_2^B(b) = 0.1$ $\pi_2^B(c) = 0.4$	(best - 0.25) $l^1 \searrow [a].25$	(best - 0.2) $l^1 \rightarrow [a].25$	(best - 0.32) $l^1 \searrow [c].32$	(best - 0.2) $l^1 \rightarrow [c].32$	(best - 0.15) $l^1 \rightarrow [c].32$
$\pi_3^B(a) = 0$ $\pi_3^B(b) = 0.9$ $\pi_3^B(c) = 0.1$	(best - 0.36) $l^1 \searrow [b].36$	(best - 0.18) $l^1 \rightarrow [b].36$	(best - 0.09) $l^2 \searrow [cc].064$	(best - 0.27) $l^2 \rightarrow [cc].064$	(best - 0.63) $l^2 \searrow [cb].2016$
$\pi_4^B(a) = 0.6$ $\pi_4^B(b) = 0.1$ $\pi_4^B(c) = 0.3$	(best - 0.3) $l^1 \downarrow [b].36$	(best - 0.18) $l^2 \searrow [ab].045$	(best - 0.24) $l^2 \downarrow [cc].064$	(best - 0.24) $l^3 \searrow [ccb].0173$	(best - 0.18) $l^3 \searrow [cab].0403$
		(best - 0.18) $l^2 \searrow [ba].0648$	(best - 0.24) $l^3 \searrow [abc].0108$	(best - 0.24) $l^4 \searrow \text{---}$	(best - 0.18) $l^4 \searrow [ccba].0031$

Fig. 2. A LCWS Table

The space required is  $O(n^2)$ . Though each of the  $n^2$  entries contains  $O(n)$  probabilities and their origin. Nevertheless, due to Lemma [□](#), during the computation of  $l_{i,j}^k$ , we need only the cells adjacent to the current. Therefore when filling the  $h$ th row we keep only rows  $h, h - 1$  in the memory. As a consequence, at each step we save only  $O(n)$  activated entries implying the space requirement is  $O(n^2)$ .

The time complexity can be improved if we note that the dependency on adjacent entries holds for each of the  $l_{i,j}^k$ s separately. In other words, we do not have to compute  $l_{i,j}^k$  for all possible  $k$ s in the same iteration. We suggest improving the algorithm, by filling the table layer after layer. After the initialization of  $l_{i,j}^1$  with  $best_{i,j}$  values, at every step, we will have  $l_{i,j}^k$ , for a single  $k$ , computed for the entire table, and we will compute  $l_{i,j}^{k+1}$ , as these computation will be possible, according to Lemma [□](#).

At the end of iteration  $k + 1$ , we check whether  $l_{n,n}^{k+1} \geq \alpha$ . In case the inequality is valid we consider  $k + 1$  as a possible answer, as we have just found that there exists a common subsequence of this length with a proper probability. We continue to compute  $l_{i,j}^{k+2}$ s and discard all  $l_{i,j}^k$ s, as their information is useless from now on.

If the contrary holds and  $l_{n,n}^{k+1} < \alpha$  we return  $k$  as the length of the longest common weight subsequence, as  $l_{n,n}^{k+1}$  contains the highest probability of a common subsequence of length  $k + 1$ , due to Lemma [□](#). In case its probability is less than expected, there would be no other common subsequence of length  $k + 1$  or more respecting the weight demand.

**Theorem 1.** *The LCWS problem is solvable in  $O(Ln^2)$  time and  $O(n^2)$  space, where  $L$  is the length of the longest common weighted subsequence of the input.*

*Proof:* The Algorithm stops after an iteration in which the weight bound is not respected. Therefore the number of iterations performed is  $L + 1$ . In each of them  $l_{i,j}^k$  is computed for all  $n^2$  entries of the table. This computation involves a constant number of operation, as detailed in Lemma [□](#). In a addition,  $best_{i,j}$  is determined once in time  $O(|\Sigma|n^2)$ .

All in all we have  $O((L + |\Sigma|)n^2)$  time requirements. Since in most usages of the Position Weight Matrix,  $|\Sigma|$  is rather small, and actually a constant, the time complexity is converted to  $O(Ln^2)$ .

Regarding space, at each iteration we consider two probabilities  $l_{i,j}^k$  and  $l_{i,j}^{k+1}$ . As the table consists of  $n^2$  entries, we get space requirement of  $O(n^2)$ .

## 4 Longest Common Weighted Subsequence with Two Thresholds (LCWS2)

The LCWS2 problem, defined in Section [□](#) in which the probability of the common subsequence in each of the sequences, must exceed its  $\alpha_i$  threshold cannot be solved in the same manner as the LCWS is solved. This is due the difference between the problems that can be intuitively summarized by the following two observations.

**Observation 1.** *The LCWS problem allows its optimal solution to consider at every step increasing prefixes of the input strings.*

Proof: The dynamic programming solution has a single possible direction of enlarging the substrings to which it computes their *LCWS*, since all probabilities are associatively multiplied together. Therefore, computing  $LCWS(A[1, i], B[1, j])$  depends merely on the LCS of prefixes of  $A$  and  $B$  shorter by one or zero symbols.

**Observation 2.** *It does not seem sufficient to consider at every step increasing prefixes of the input strings in order to obtain an optimal solution for the LCWS2 problem.*

Intuition: In this problem we execute two distinct probability multiplications and want to obtain the longest common subsequence satisfying the thresholds demand. Consequentially, we would like to multiply high probabilities in both sides. In case the current characters  $A[i], B[j]$  agree, i.e., a single symbol,  $\sigma \in \Sigma$ , whose probability is highest for both positions, then adding this symbol as the match of the characters does not change the invariant of optimal solution so far.

However, when  $A[i], B[j]$  do not agree, where there is a  $\sigma_1$  whose probability is maximal in  $A[i]$  but  $\sigma_2 \neq \sigma_1$  has maximal probability in  $B[j]$ , it is not clear which symbol one should choose for the common subsequence. It may be more profitable to choose  $\sigma_1$ , even causing the  $B$  probability to decrease a lot, since later on a reversed case will occur and balance the probabilities. It, therefore, seems intuitive that local considerations do not suffice for computing the *LCWS2* problem. This intuition is proven in the next subsection.

#### 4.1 LCWS2 Is $\mathcal{NP}$ -Hard

We prove that the *LCWS2* problem is  $\mathcal{NP}$ -hard for unbounded alphabets. To this aim we define the *CWS2* decision version:

**Definition 6.** The Common Weighted Substructure with 2 thresholds (*CWS2*):

*Input:* Two  $p$ -weighted strings  $A, B$  of length  $n$  over alphabet  $\Sigma$ , and constants  $L, \alpha_1, \alpha_2, 0 < \alpha_i \leq 1$ .

*Output:* Does there exists a common weighted subsequence of length  $L$ , where  $a_{i_1}..a_{i_L}=b_{j_1}..b_{j_L}$ , where  $(\prod_{y=1}^L \pi_{i_y}(a_{i_y})) \geq \alpha_1$  AND  $(\prod_{y=1}^L \pi_{i_y}(b_{j_y})) \geq \alpha_2$ .

**Theorem 2.** *The LCWS2 problem is  $\mathcal{NP}$ -hard.*

Proof: We prove the hardness using a Turing reduction from the Partition problem.

**Definition 7.** The Partition problem: [\[6\]](#)

*Input:* A finite set  $S$  and a "value"  $v(s) \in Z^+$  for each  $s \in S$ .

*Output:* Is there a subset  $S' \subseteq S$  such that  $\sum_{s \in S'} v(s) = \sum_{s \in S-S'} v(s)$  ?

**Lemma 2.** *Partition  $\leq_T^p$  CWS2.*

Given set  $S = s_1, s_2, \dots, s_n$  of integers, we construct two weighted sequences  $A = A_1..A_n$ ,  $B = B_1..B_n$  both over alphabet of size  $n + 2$ . In addition we need to set a pair of thresholds  $\alpha_1, \alpha_2$  and  $L$ .

**Observation 3.** *The requirement that the product of the probabilities of the common sequence be higher than  $\alpha_i$  is equivalent to demanding that the sum of the logarithm of the probabilities will be higher than  $\log \alpha_i$ .*

Proof: The observation is a direct result of the fact that the logarithm of a product equals the sum of logarithms. A special case is a zero probability that is converted to infinity. Note that the logarithms of probabilities are all negative numbers. We can simply invert the signs of all numbers, making them all positive, and require adding as many numbers as possible without exceeding (the inverted)  $\log \alpha_i$ . ■

We are now ready to define the reduction. Given a set  $S = \{s_1, \dots, s_n\}$ , we set alphabet of the LCWS2 problem to be  $\Sigma = \{\sigma_1, \dots, \sigma_{n+2}\}$ . We define two  $p$ -weighted sequences,  $A$  and  $B$ , of length  $n$ . By definition [3], location  $i$  in a  $p$ -weighted sequence is the set of all pairs  $(\sigma, \pi_i(\sigma))$ , where  $\sigma \in \Sigma$  and  $\pi_i(\sigma)$  is the probability of having symbol  $\sigma$  at location  $i$ . We define the probabilities of the symbols of  $\Sigma$  in the following manner.

Let  $sum = \sum_{s \in S} s$ , the sum of all elements of  $S$ .

$$\pi_i^A(\sigma_j) = \begin{cases} s_i & j = i \\ x_i & j = n + 1 \\ \infty & \text{otherwise} \end{cases} \quad \pi_i^B(\sigma_j) = \begin{cases} sum - s_i & j = i \\ y_i & j = n + 2 \\ \infty & \text{otherwise} \end{cases}$$

The value of  $x_i$  is such that  $2^{-s_i} + 2^{-x_i} = 1$ , and the value of  $y_i$  is such that  $2^{s_i - sum} + 2^{-y_i} = 1$ . They are necessary because in each location there is a single element with probability non-zero, thus we need to add a probability that, with it, will add up to 1.

Obviously the construction is done in polynomial time in the size of  $n$ , as  $|\Sigma| = n + 2$ . For an example of the construction for set  $S = \{6, 3, 4, 7\}$  see Fig. [3].

From the probabilities definition we get that the only possible symbols that can potentially be chosen for any weighted LCS with finite threshold are choosing  $\sigma_i$  of  $a_i$  with  $\sigma_i$  of  $b_i$  for  $1 \leq i \leq n$ .

We proceed with the Turing reduction. we perform up to  $n/2$  iterations. In the  $i$ th iteration we set  $\alpha_1 = sum/2$ ,  $\alpha_2 = sum \cdot (i - 1/2)$ , and  $L = i$ . We check whether there is a CWS2 with these parameters. If the answer is negative we increment  $i$  by one and start a new iteration. If no CWS2 was found after the  $n/2$  iteration we terminate the search. If there is a CWS2 in iteration  $i$ , we declare a partition of  $S$  into sizes  $i$  and  $n - i$ .

*Claim.* A partition of  $S$  into size  $i$  and  $n - i$  exists iff a CWS2 of length  $i$  was found on the  $i$ th iteration.

A =	$\pi_1^A(a) = 6$	$\pi_2^A(a) = \infty$	$\pi_3^A(a) = \infty$	$\pi_4^A(a) = \infty$
	$\pi_1^A(b) = \infty$	$\pi_2^A(b) = 3$	$\pi_3^A(b) = \infty$	$\pi_4^A(b) = \infty$
	$\pi_1^A(c) = \infty$	$\pi_2^A(c) = \infty$	$\pi_3^A(c) = 4$	$\pi_4^A(c) = \infty$
	$\pi_1^A(d) = \infty$	$\pi_2^A(d) = \infty$	$\pi_3^A(d) = \infty$	$\pi_4^A(d) = 7$
	$\pi_1^A(e) = x_1$	$\pi_2^A(e) = x_2$	$\pi_3^A(e) = x_3$	$\pi_4^A(e) = x_4$
	$\pi_1^A(f) = \infty$	$\pi_2^A(f) = \infty$	$\pi_3^A(f) = \infty$	$\pi_4^A(f) = \infty$
B =	$\pi_1^B(a) = 14$	$\pi_2^B(a) = \infty$	$\pi_3^B(a) = \infty$	$\pi_4^B(a) = \infty$
	$\pi_1^B(b) = \infty$	$\pi_2^B(b) = 17$	$\pi_3^B(b) = \infty$	$\pi_4^B(b) = \infty$
	$\pi_1^B(c) = \infty$	$\pi_2^B(c) = \infty$	$\pi_3^B(c) = 16$	$\pi_4^B(c) = \infty$
	$\pi_1^B(d) = \infty$	$\pi_2^B(d) = \infty$	$\pi_3^B(d) = \infty$	$\pi_4^B(d) = 13$
	$\pi_1^B(e) = \infty$	$\pi_2^B(e) = \infty$	$\pi_3^B(e) = \infty$	$\pi_4^B(e) = \infty$
	$\pi_1^B(f) = y_1$	$\pi_2^B(f) = y_2$	$\pi_3^B(f) = y_3$	$\pi_4^B(f) = y_4$

**Fig. 3.** The constructed sequences, according to the set  $\{6, 3, 4, 7\}$

Proof: ( $\Rightarrow$ ) Suppose there is a partition of  $S$  into two subsets  $S_1 = \{s_{g_1}, s_{g_2}, \dots, s_{g_i}\}$  and  $S_2$ . As  $S_1$  is a subset of the partition we know that the  $\sum_{s_g \in S_1} s_g = sum/2$ . Consequently, considering the  $g_1, g_2, \dots, g_i$  characters of  $A$  and their corresponding symbol as the common subsequence, their log probabilities will sum up to  $\alpha_1 = sum/2$ . Due to the probabilities allocation in our construction, we are bound to select the same indices  $g_1, \dots, g_i$  in the counterpart sequence  $B$ . Note that adding their new probabilities we get,  $sum - s_{g_1} + sum - s_{g_2} + \dots + sum - s_{g_i} = i \cdot sum - \sum_{s_g \in S_1} s_g = i \cdot sum - 1/2 \cdot sum = \alpha_2$ . All in all, the existence of a partition of subsets  $i, n - i$  in the set, implies a common subsequence of length  $i$  respecting the thresholds.

Clearly, there cannot be a longer  $CWS2$ , as addition of a single character to the common subsequence implies adding to the calculations of both sequences altogether  $sum$ , which result in  $(i + 1) \cdot sum$  while the sum of  $\alpha_i$ s is merely  $i \cdot sum$ .

( $\Leftarrow$ ) We perform up to  $n/2$  iterations. In the  $i$ th iteration we check whether there is a  $CWS2$  of length  $i$ . In case the answer is negative we increment  $i$  by one and start a new iteration. If no  $CWS2$  was found after the  $n/2$  iteration we terminate the search, as the largest size of the smaller subset of the partition is  $n/2$ , so we have covered all relevant sizes.

Suppose we find a  $CWS2$  of length  $i$ , where the common weighted subsequence is  $A_{g_1}, A_{g_2}, \dots, A_{g_i}$ . Due to the construction, the common subsequence will appear in  $B$  at the same indices as in  $A$ . Obtaining the  $CWS2$  implies that  $\sum_{k=1}^i \pi_{g_k}^A(\sigma_{g_k}) \leq sum/2$  and due to the construction we get  $s_{g_1} + \dots + s_{g_i} \leq sum/2$ . In addition, the occurrence of the  $CWS2$  in  $B$  implies that  $\sum_{k=1}^i \pi_{g_k}^B(\sigma_{g_k}) \leq sum(i - 1/2)$  which means that  $sum - s_{g_1} + \dots + sum - s_{g_i} = i \cdot sum - (s_{g_1} + \dots + s_{g_i}) \leq sum(i - 1/2)$ . Since we have just claimed that the sum of the chosen numbers from set  $S$  are less than or equal to  $sum/2$ , subtracting it from  $i \cdot sum$  we get a result greater or equal to  $sum(i - 1/2)$ , contradicting the requirement of not exceeding  $\alpha_2$ . Hence, it must be the case that  $s_{g_1} + \dots + s_{g_i} = sum/2$ . Thus, these numbers form a subset of the partition problem. ■

The above lemma concludes the proof of the theorem. ■

## 4.2 Approximation Algorithm

Having proved that the *LCWS2* problem for unbounded  $\Sigma$  is  $\mathcal{NP}$ -hard, we provide an approximation algorithm *LCWS2<sub>A</sub>*.

The approximation algorithm considers each symbol  $\sigma \in \Sigma$  separately. For a fixed  $\sigma \in \Sigma$ , let  $i_1, \dots, i_k$  be the indices of the longest possible sequence of  $\sigma$ 's in  $A$  such that  $\prod_{\ell=1}^k \pi_{i_\ell}^A(\sigma) \geq \alpha_1$ , and let  $j_1, \dots, j_m$  be the indices of the longest possible sequence of  $\sigma$ 's in  $B$  such that  $\prod_{\ell=1}^m \pi_{j_\ell}^B(\sigma) \geq \alpha_2$ . Take the minimum of  $k$  and  $m$  as *counter <sub>$\sigma$</sub>* .

Choose the symbol  $\sigma$  with the largest *counter <sub>$\sigma$</sub>*  and output  $\sigma^{\text{counter}_\sigma}$ .

**Observation 4.** *The approximation algorithm requires time  $O(|\Sigma|n \log n)$ .*

Proof: The input is a p-weighted sequence of length  $n$ , where each character contains  $\Sigma$  probabilities. We therefore construct  $\Sigma$  lists of length at most  $n$  and we sort each list. ■

**Theorem 3.** *The approximation ratio of *LCWS2<sub>A</sub>* is  $\frac{1}{|\Sigma|}$ .*

Proof: Suppose the optimal length of the *LCWS2* is *OPT*, and that the *LCWS2<sub>A</sub>* algorithm returned *counter <sub>$i$</sub>* . This implies that the symbol that can be repeated most frequently, without decreasing beneath the thresholds is  $\sigma_i$ . The optimal solution to the *LCWS2* problem suggested by the *OPT* algorithm may include several symbols. Let  $\sigma_j$  be the most frequent symbol in the optimal solution. Note that  $\text{counter}_{\sigma_j} \leq \text{counter}_{\sigma_i}$ . In addition, the number of  $\sigma_j$ 's in the optimal solution is at least  $\frac{1}{|\Sigma|} \text{OPT}$ . We get  $\frac{1}{|\Sigma|} \text{OPT} \leq \text{counter}_{\sigma_j} \leq \text{counter}_{\sigma_i}$ . ■

## 5 Conclusions and Open Problems

The main contribution of this paper is in applying the Longest Common Subsequence to a new useful structure. We define the problem of Longest Common Weighted Subsequence, considering the LCS problem applied to the important structure of p-weighted sequences. We give two possible definitions to the problem. For the first, we present a simple dynamic programming algorithm that generalizes to higher dimensions. For the second we proved  $\mathcal{NP}$ -hardness for unbounded alphabets, and described a proper approximation algorithm. It remains unclear what is the actual complexity class of the *LCWS2* problem over unbounded alphabet, since we used a Turing reduction for the hardness proof.

## References

1. Amir, A., Chencinski, E., Iliopoulos, C.S., Kopelowitz, T., Zhang, H.: Property Matching and Weighted Matching. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 188–199. Springer, Heidelberg (2006)
2. Amir, A., Hartman, T., Kapah, O., Shalom, B.R., Tsur, D.: Generalized LCS. In: Ziviani, N., Baeza-Yates, R. (eds.) SPIRE 2007. LNCS, vol. 4726, pp. 50–61. Springer, Heidelberg (2007)

3. Amir, A., Iliopoulos, C.S., Kapah, O., Porat, E.: Approximate Matching in Weighted Sequences. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 365–376. Springer, Heidelberg (2006)
4. Apostolico, A., Landau, G.M., Skiena, S.: Matching for run-length encoded strings. *Journal of Complexity* 15(1), 4–16 (1999)
5. Bergroth, L., Hakonen, H., Raita, T.: A survey of longest common subsequence algorithms. In: Proc. 7th Symposium on String Processing and Information Retrieval (SPIRE), pp. 39–48 (2000)
6. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., New York (1979)
7. Hirschberg, D.S.: A Linear space algorithm for Computing Maximal Common Subsequences. *Commun. ACM* 18(6), 341–343 (1975)
8. Iliopoulos, C., Makris, C., Panagis, Y., Perdikuri, K., Theodoridis, E., Tsakalidis, A.K.: Efficient Algorithms for Handling Molecular Weighted Sequences. In: IFIP TCS, pp. 265–278 (2004)
9. Iliopoulos, C.S., Mouchard, L., Pedikuri, K., Tsakalidis, A.K.: Computing the repetitions in a weighted sequence. In: Proceedings of the 2003 Prague Stringology Conference (PSC 2003), vol. 10, pp. 91–98 (2003)
10. Jacobson, G., Vo, K.P.: Heaviest Increasing/Common Subsequence Problems. In: Apostolico, A., Galil, Z., Manber, U., Crochemore, M. (eds.) CPM 1992. LNCS, vol. 644, pp. 52–66. Springer, Heidelberg (1992)
11. Li, R.: A Linear Space Algorithm for the Heaviest Common Subsequence Problem. *Utilitas Mathematica* 75, 13–20 (2008)
12. Thompson, J.D., Higgins, D.G., Gibson, T.J.: CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.* 22, 4673–4680 (1994)
13. Venter, J.C., Celera Genomics Corporation: The Sequence of the Human Genome. *Science* 291, 1304–1351 (2001)
14. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. *J. ACM* 21, 168–173 (1974)

# Integrality Properties of Certain Special Balanceable Families

Nicola Apollonio<sup>1</sup> and Massimiliano Caramia<sup>2</sup>

<sup>1</sup> Istituto per le Applicazioni del Calcolo, M. Picone, Via G. Amendola, 122/D 70126  
Bari, Italy

`nicola.apollonio@uniroma1.it`

<sup>2</sup> Dipartimento di Ingegneria dell'Impresa, Università di Roma "Tor Vergata", Via  
del Politecnico 1, 00133 Roma, Italy

`caramia@disp.uniroma2.it`

**Abstract.** Balanceable clutters are clutters whose bipartite representation contains no odd wheel and no odd 3-path configuration as induced subgraph (this is Truemper's characterization of balanceable matrices). In this paper we study a proper subclass of balanceable clutters called quasi-graphical defined by forbidding one-sided even wheels and one-sided even 3-path configurations. We characterize Mengerian quasi-graphical clutters and, as a consequence, we show that a recent conjecture in [5] is true for quasi-graphical clutters.

**Keywords:** Wheels, 3-path configurations, Mengerian Clutters.

## 1 Introduction

An *unbalanced hole submatrix* of a  $\{-1, 0, 1\}$  matrix  $A$  is a square submatrix of  $A$  having exactly two nonzero entries per row and per columns whose sum of the entries is not divisible by four and minimal with this property. A  $\{-1, 0, 1\}$  matrix  $A$  is *balanced* if it does not contain any unbalanced hole submatrix. A *binary* matrix is a matrix with  $0, 1$  entries. A binary matrix is *balanceable* if it can be *signed* to become balanced, where *signing* a binary matrix  $A$  consists of multiplying some of its entries by  $-1$ . A finite family of subsets of a finite ground set is *balanceable* if so is its *incidence* matrix, i.e., the binary matrix whose columns are the incidence vectors of the members of the family (taken over the ground set). The *bipartite graph* of a finite family  $\mathcal{C} = (L_j \mid j \in P)$  of subsets of  $V$  is the bipartite graph  $B(\mathcal{C})$  with color classes  $V$  and  $P$  in which  $v \in V$  and  $j \in P$  are connected by an edge if  $v \in L_j$ . Truemper characterized balanceable families as those finite families whose bipartite graph contains neither *odd wheels* nor *odd 3-path-configuration* as induced subgraphs (see e.g., [6]). Recall that a (bipartite) *uv-3-path configuration* ( $3PC(u, v)$ ) is a bipartite graph consisting of three internally vertex-disjoint *uv*-paths  $P_1$ ,  $P_2$  and  $P_3$  such that  $V(P_i) \cup V(P_j)$ ,  $i \neq j$ , induces a chordless cycle and  $u$  and  $v$  are not adjacent. A *3-path configuration* ( $3PC$ ) is a  $3PC(u, v)$  for some  $u$  and  $v$ . Since  $3PC(u, v)$  is a bipartite graph, the length of each of the three *uv*-paths



is odd or even accordingly to whether  $u$  and  $v$  belong to different color classes or to the same color class, respectively. In the former case each path has length at least three and the 3PC is said to be *odd*. In the latter case, if each path has length at least four, we say that the 3PC is *even*<sup>1</sup>. A (bipartite) *wheel* is a bipartite graph  $(C, v)$  consisting of a chordless cycle  $C$  and a vertex  $v \notin V(C)$  that has at least three neighbors on  $C$ ;  $C$  and  $v$  are referred to as the *rim* and the *center* of the wheel, respectively. Each edge of the wheel incident to the center is called a *spoke*. The wheel is *odd* if it has an odd number of spokes. It is *even* otherwise. A  $k$ -wheel is a bipartite wheel with  $k$  spokes. In this paper we study a proper subclass of balanceable matrices, namely, the class of quasi-graphical families defined as follows: a finite family  $\mathcal{C} = (L_j \mid j \in P)$  of subsets of  $V$  is quasi-graphical if it is balanceable and its bipartite graph contain neither even  $\mathcal{C}$ -wheel nor even  $\mathcal{C}$ -3PC as induced subgraph where an even  $\mathcal{C}$ -wheel is an even wheel whose center is in the color class  $P$  and an even  $\mathcal{C}$ -3PC is an even 3PC whose vertices of degree three are both in the color class  $P$ . The choice of the term quasi-graphical is explained in Remark 2. With any finite family  $\mathcal{C}$  of subsets of a common ground set  $V$  and a function  $w \in \mathbb{Z}_+^V$  we can associate the following pair of dual linear programs:

$$\begin{aligned} & \text{minimize } wy \text{ subject to} \\ & \sum (y(v) : v \in L) \geq 1 \quad \forall L \in \mathcal{C} \\ & y \in \mathbb{R}_+^V, \end{aligned} \tag{1}$$

$$\begin{aligned} & \text{maximize } 1x \text{ subject to} \\ & \sum (x(L) : v \in L \in \mathcal{C}) \leq w(v) \quad \forall v \in V \\ & x \in \mathbb{R}_+^{\mathcal{C}}, \end{aligned} \tag{2}$$

The main aim of this paper is to characterize Mengerian quasi-graphical families, namely, those quasi-graphical families  $\mathcal{C}$  for which problem (2) has an integral optimal solution for any  $w \in \mathbb{Z}_+^V$ , i.e, the defining system of (1) is Totally Dual Integral. Our result relies on the notion of pie introduced by Golubic and Jamison in [8] in the context of Edge-Path-Tree graphs and closely follow a similar characterization for Edge-Path-Tree families given in [1] to which it specializes. Odd pies can be viewed as natural generalizations of odd circuits in graphs. A pie is a collection of subsets of a common ground set whose members can be cyclically ordered so that each member intersects exactly its two neighbors in the order and each element of the ground set occurs in at most two members of the collection. We base the characterizations of Mengerian quasi-graphical families on Lovász's 2-matching characterization of Mengerianity (see Theorem 1) and the additional observation (see Theorem 2) that if a quasi-graphical family  $\mathcal{C}$  does not contain any odd pie as minor either it contains the  $Q_6$  clutter as minor or the members of certain 2-matchings in  $\mathcal{C}$  can be chosen "as uncrossing as possible". Recall that the  $Q_6$  clutter is the clutter whose members are the

<sup>1</sup> We stress here that if  $H$  is a 3PC( $u, v$ ), with  $u$  and  $v$  belonging to the same color class, but  $u$  and  $v$  are linked by a path of length two, then  $H$  must not be considered an even wheel.

edge sets of the four triangles of the complete graph on four vertices. A Venn representation of the  $Q_6$  is given in Figure 3 (a).

*Terminology.* Throughout the rest of the paper  $\mathcal{C} = (L_j \mid j \in P)$  denotes a finite family of subsets of a finite ground set  $V$ . We also denote  $\cup(L \mid L \in \mathcal{C})$  by  $V(\mathcal{C})$  and we say that  $\mathcal{C}$  is a family on  $V$  if  $V = V(\mathcal{C})$ . We use the term *collection* for families with no repeated members. A *clutter* is a collection whose elements are inclusionwise incomparable. For  $X, Y \subseteq V$  and  $X \cap Y = \emptyset$  the family of the (inclusionwise) minimal members in  $\{L - Y \mid L \cap X = \emptyset, L \in \mathcal{C}\}$  is denoted by  $\mathcal{C} \setminus X/Y$  and is referred to as a *minor* of  $\mathcal{C}$ . If  $\mathcal{C}$  is a clutter so is  $\mathcal{C} \setminus X/Y$ . It is well known that  $\mathcal{C} \setminus X/Y = \mathcal{C}/Y \setminus X$ . When  $X = \emptyset$  or  $Y = \emptyset$  the notation will be abridged to  $\mathcal{C} \setminus X$  (*deletion minor*) and  $\mathcal{C}/X$  (*contraction minor*), respectively. In a graph, a chordless cycle on four or more vertices is called a *hole*. In a bipartite graph, a hole is *odd* if its length is not divisible by four. Throughout the rest of the paper we use the following concrete coloring for the bipartite graph  $B(\mathcal{C})$  of  $\mathcal{C} = (L_j \mid j \in P)$ : the vertices in the color class  $P$  are represented by solid circles; those in the color class  $V$  are represented by empty circles.

In a graph every odd cycle contains an odd circuit, i.e., a subgraph where each vertex occurs in two edges. The natural generalization to families of the notion of circuit in a graph, is the notion of pie introduced in [8]. A *pie* is a collection  $\mathcal{P} = (L_j \mid j \in N)$  on some finite ground set  $V$  such that  $n := |N| \geq 3$  and

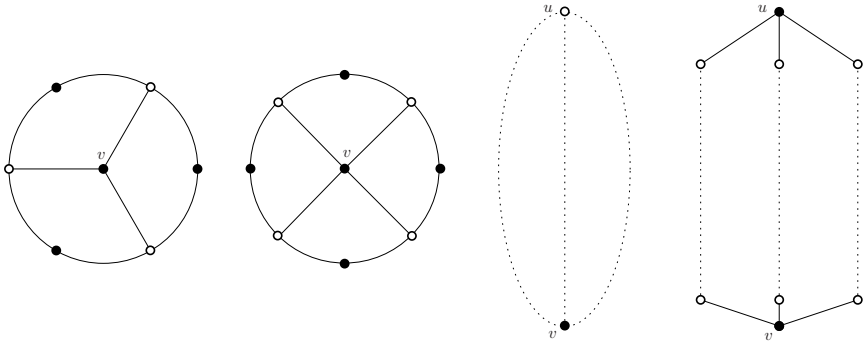
- for some permutation  $(j_1, \dots, j_n)$  of  $N$  one has  $L_{j_i} \cap L_{j_{i+1}} \neq \emptyset$  and  $L_{j_h} \cap L_{j_i} = \emptyset$  if  $|i - h| \notin \{1, n - 1\}$ , (addition over indices is modulo  $n$ );
- if  $n = 3$  then  $\cap_{j \in N} L_j = \emptyset$ .

Two members  $L_h$  and  $L_i$  of a pie are *consecutive* if  $L_h \cap L_i \neq \emptyset$ . The number  $n$  is the *size* of the pie; a pie of size  $n$  is a  $n$ -*pie*. The pie is *odd* if  $n$  is odd and *even* otherwise. We set  $B_{j_i} = L_{j_i} \cap L_{j_{i+1}}$ ,  $i = 1, \dots, n$  (addition over the indices is taken modulo  $n$ ) and we call  $B_{j_i}$  the  $i$ -th *branch* of the pie. Observe that by the definition of pie one has  $B_i \cap B_j = \emptyset$ , for  $i \neq j$ ,  $i, j \in N$ . If  $\mathcal{P}$  is a pie in  $\mathcal{C}$  we say that  $\mathcal{C}$  *contains* a pie. Notice that a family might contain odd pies without containing odd pies as minor: in a  $Q_6$  the collection formed by any three of its members is a 3-pie though no minor of the  $Q_6$  is an odd pie.

*Organization.* The rest of the paper goes as follows. In the next section we give the characterization of Mengerian quasi-graphical families and discuss some consequences—mainly the fact that the Conjecture in [5] asserting that every minimal non-packing clutter has a transversal of size 2 holds true within quasi-graphical families—. The characterization uses Theorem 2 which is technical and hence proved in Section 2.1

## 2 Mengerian Quasi-graphical Families

In this section we characterize Mengerian quasi-graphical clutters. The characterization closely follows the characterization of Mengerian Edge-Path-Tree families given in [1]. We need the following two results. Recall that a  $w$ -*matching*  $x$



**Fig. 1.** An odd wheel, an even  $\mathcal{C}$ -wheel, an odd  $3PC(u, v)$  and an even  $\mathcal{C}$ - $3PC(u, v)$ . Solid lines represent edges and dotted lines represent paths.

of  $\mathcal{C}$  is any integral point in the polyhedron of (2). The number  $\sum(x(L) \mid L \in \mathcal{C})$  is called the *size* of  $x$  and the maximum size of a  $w$ -matching of  $\mathcal{C}$  is denoted by  $\nu_w(\mathcal{C})$ .

**Theorem 1 (Lovász).** *A family  $\mathcal{C}$  of subsets of a given ground set  $V$  is Mengerian if and only if  $\nu_{2w}(\mathcal{C}) = 2\nu_w(\mathcal{C})$  for each  $w \in \mathbb{Z}_+^V$ .*

**Theorem 2.** *Let  $\mathcal{C}$  be a quasi-graphical family. Assume that  $\mathcal{C}$  is a clutter without any odd pie as minor and let  $\mathcal{P}$  be an odd pie in  $\mathcal{C}$ . Then either  $\mathcal{C}$  contains the  $Q_6$  clutter as minor or  $F_0 \cup F_1 \subseteq L\Delta L'$  for some two members  $L$  and  $L'$  of  $\mathcal{C}$  which are consecutive in  $\mathcal{P}$  and some two disjoint members  $F_0$  and  $F_1$  of  $\mathcal{C}$ .*

Theorem 1 is Lovász’s 2-matching characterization of Mengerian families. Theorem 2 is technical and its proof will be postponed after the characterization. Recall that a family  $\mathcal{C}$  is *balanced* if  $B(\mathcal{C})$  does not contain any induced odd hole. Balanced families are Mengerian (see, e.g., [9]).

**Theorem 3.** *Let  $\mathcal{P}$  be a quasi-graphical family. Then  $\mathcal{P}$  is Mengerian if and only if  $\mathcal{P}$  contains neither odd pies nor the  $Q_6$  clutter as minors.*

*Proof.* Neither odd pies nor the  $Q_6$  clutter are Mengerian—in particular odd pies are not even ideal: if  $\mathcal{P}' = \{L'_1, \dots, L'_k\}$  is an odd pie minor of  $\mathcal{P}$ , then  $\mathcal{P}'$  can be contracted to the edge set of an odd polygon—hence necessity follows. To prove sufficiency we need the following fact whose proof can be found in [3].

*Claim.* Let  $\mathcal{E} = (E_j \mid j \in P)$  be a quasi-graphical family on  $V$ . If  $\mathcal{E}$  does not contain any odd pie then  $\mathcal{E}$  is balanced.

Without loss of generality  $\mathcal{C}$  is a clutter on  $V$ . Suppose that  $\mathcal{C}$  contains neither odd pies nor the  $Q_6$  clutter as minors but it is not Mengerian. By Theorem 1 one has  $\nu_{2w}(\mathcal{C}) > 2\nu_w(\mathcal{C})$  for some  $w \in \mathbb{Z}_+^V$ . Let  $w$  be chosen so as to minimize  $\sum_{v \in V} w(v)$  and let  $V^* := \{v \in E \mid w(v) \geq 1\}$  be its support. Therefore, for

$v \in V^*$ ,  $\nu_{2(w-\chi_v)}(\mathcal{C}) = 2\nu_{w-\chi_v}(\mathcal{C})$ ,  $\chi_v \in \mathbb{Z}_+^V$ , being the incidence vector of edge  $v$  over  $V$ . Let  $x \in \mathbb{Z}_+^{\mathcal{C}}$  be a  $2w$ -matching of size  $\nu_{2w}(\mathcal{C})$  and let  $\mathcal{M} = \{L \in \mathcal{C} \mid x(L) \geq 1\}$  be its support. The clutter  $\mathcal{M}$  must contain some odd pie otherwise, by Claim 2, we would have  $\nu_{2w}(\mathcal{C}) = \nu_{2w}(\mathcal{M}) = 2\nu_w(\mathcal{M}) \leq 2\nu_w(\mathcal{C})$ . Let  $\mathcal{P} = \{L_1, \dots, L_n\} \subseteq \mathcal{M} \subseteq \mathcal{C}$  be any odd pie in  $\mathcal{M}$ . Clearly  $\mathcal{P}$  is an odd pie in  $\mathcal{C}$ . Notice that  $V(\mathcal{P}) \subseteq V^*$ . Possibly after renumbering we may suppose that  $L_i$  and  $L_j$  are consecutive in  $\mathcal{P}$  if and only if  $|i - j| \in \{1, n - 1\}$ . Therefore, by Theorem 2, there are disjoint members  $F_0$  and  $F_1$  of  $\mathcal{C}$  such that, for some  $j = 1, \dots, n$ , one has  $F_i \subseteq L_j \Delta L_{j+1}$ ,  $i = 0, 1$ . Define  $\bar{x}$  as follows:

$$\bar{x}(L) = \begin{cases} x(L) - 1 & \text{if } L \in \{L_j, L_{j+1}\} \\ x(L) + 1 & \text{if } L \in \{F_0, F_1\} \\ x(L) & \text{otherwise.} \end{cases}$$

By construction,

$$\sum_{L \ni v} \bar{x}(L) = \begin{cases} \sum_{L \ni v} x(L) - 1 & \text{if } v \in (L_j \Delta L_{j+1}) - (F_0 \cup F_1) \\ \sum_{L \ni v} x(L) - 2 & \text{if } v \in L_j \cap L_{j+1} \\ \sum_{L \ni v} x(L) & \text{otherwise.} \end{cases}$$

Let  $v_j \in P_j \cap L_{j+1}$ . It follows that  $\bar{x}$  is a  $2(w - \chi_{v_j})$ -matching of size

$$\sum_{L \in \mathcal{M} \cup \{F_0, F_1\}} \bar{x}(L) = \sum_{L \in \mathcal{M}} x(L),$$

contradicting the minimality of  $w$ . □

**Corollary 1.** *An ideal quasi-graphical family is Mengerian if and only if it does not contain the  $Q_6$  clutter as minor.*

*Remark 1.* A clutter has the packing property if Problem (1) has an integral optimal dual solution for all  $w \in \{0, 1, +\infty\}^V$  (see [6]). In [5] the authors conjecture that every minimally non packing clutter has a transversal of size 2. In the same paper the authors show that the conjecture implies the replication conjecture of Conforti and Cornuejols for packing clutters which is in turn equivalent to the conjecture that a clutter is Mengerian if and only if it is packing [6]. By corollary 1 all these conjectures hold true for quasi-graphical families.

Corollary 1 can be specialized to *Edge-Path-Tree* families. A family  $\mathcal{E} = (E_i \mid i \in P)$  is an *Edge-Path-Tree* family if there exists a tree  $T = (V, E)$  such that  $E_i$  is the edge set of some path in  $T$ . To see this we need to recall some well known preliminary notion. With every binary matrix  $A$  with  $m$  rows one can associate the binary matroid  $M(A)$  generated by the columns of  $[I_m, A]$ ,  $I_m$  being the identity matrix of order  $m$ . Such a matroid is defined as the matroid whose circuits are the minimal supports of the vectors in the nullspace of  $[I_m, A]$ ,  $[I_m, A]$  being a viewed as a matrix over  $GF(2)$ . Two binary matrices are *GF(2)-equivalent* if one

arises from the other by a sequence of  $GF(2)$ -pivoting<sup>2</sup>. Any binary matrix  $A$  is  $GF(2)$ -equivalent to itself.  $GF(2)$ -equivalent matrices generate the same binary matroid and, conversely, if  $A$  and  $A'$  have the same order and  $M(A) = M(A')$  then  $A$  and  $A'$  are  $GF(2)$ -equivalent. A *minor* in  $M(A)$  is a matroid of the form  $M(C)$  where  $C$  is a submatrix of some matrix  $A'$  which is  $GF(2)$ -equivalent to  $A$ . The operation of  $GF(2)$ -pivoting a matrix  $A \in \{0, 1\}^{I \times J}$  can be described in terms of the bipartite graph  $B(A)$  as follows: if  $A'$  is the result of  $GF(2)$ -pivoting on a nonzero entry  $a_{i,j}$  of  $A$ , then  $B(A')$  results from  $B(A)$  by complementing the edges between  $N(i) - \{j\}$  and  $N(j) - \{i\}$ , where, for a vertex  $h$  of  $B(A)$ ,  $N(h)$  denotes the set of neighbors of  $h$  (see [4,6]). Let  $W_4$  denote the wheel with four spokes and whose rim has eight vertices. The proof of the following lemma, which is similar to an analogous result in [4] about odd wheels and odd 3PC's, can be found in [3].

**Lemma 1.** *Let  $G$  be a bipartite graph such that  $G$  is either an even 3-path configuration  $3PC(u, v)$  or an even wheel  $(C, v)$ . Then  $G$  can be pivoted into a bipartite graph containing a  $W_4$  whose center is in the same color class of  $u$  and  $v$ , if  $G = 3PC(u, v)$  and in the same color class of  $v$  if  $G = (C, v)$ .*

**Corollary 2 ([1]).** *An Edge-Path-Tree family is Mengerian if and only if it does not contain any odd pie as minor. Consequently, every ideal Edge-Path-Tree family is Mengerian.*

*Proof.* Let  $\mathcal{E} = (E_i \mid i \in P)$  be an edge-path-tree family on  $E$  and let  $A(\mathcal{E})$  be its incidence matrix. Let  $|E| = m$ . Recall that a binary matrix is *regular* if it can be signed to become totally unimodular (see e.g. [6,9]). Since  $\mathcal{E}$  is an Edge-Path-Family it follows that  $A(\mathcal{E})$  is the unsigned pattern of a *network matrix* and any such matrix is a totally unimodular matrix [9]. Therefore  $A(\mathcal{E})$  is regular and, consequently,  $\mathcal{E}$  is balanceable, [6]. In [7], Fournier observed that  $\mathcal{E}$  is an Edge-Path-Tree family if and only if the binary matroid  $M(\mathcal{E})$  generated by the columns of  $[I_m, A(\mathcal{E})]$  is a graphic matroid. By Tutte's deep characterizations of regular and graphic matroids (see e.g., [9])  $M(\mathcal{E})$  is graphic if and only if  $A(\mathcal{E})$  is regular and  $M(\mathcal{E})$  contains neither  $M^*(K_{3,3})$  nor  $M^*(K_5)$  as matroid-minors (the co-graphic matroid of the  $K_{3,3}$  and  $K_5$ , respectively). Let  $\mathcal{W}_4$  be a family on  $F$  with five members such that  $B(\mathcal{W}_4) \cong W_4$ . We show that  $M(\mathcal{W}_4) = M^*(K_{3,3})$ ; let  $\mathcal{W}_4^*$  be the dual of  $\mathcal{W}_4$ , i.e., the family  $(\{L \in \mathcal{W}_4 \mid f \in L\} \mid f \in F)$ ;  $\mathcal{W}_4^*$  is an Edge-Path-Tree family obtained as follows: let  $G$  be a copy of the  $K_{3,3}$  and  $T$  let be spanning tree of  $G$  whose degree sequence is  $(1, 1, 1, 1, 3, 3)$ . Let  $F = E(G) - E(T)$  and for each  $f$  in  $F$  let  $P(f)$  be the unique path of  $T$  connecting the endpoints of  $f$ . Thus  $\mathcal{W}_4^* = (P(f) \mid f \in F)$  and  $M(\mathcal{W}_4^*) = M(K_{3,3})$

<sup>2</sup> Recall that pivoting  $A$  over  $GF(2)$  on a nonzero entry (the pivot element) means replacing

$$A = \begin{pmatrix} 1 & a \\ b & D \end{pmatrix} \quad \text{by} \quad \tilde{A} = \begin{pmatrix} 1 & a \\ b & D + ba \end{pmatrix}$$

where the rows and columns of  $A$  have been permuted so that the pivot element is  $a_{1,1}$  ([6], p. 69, [9], p. 280).

hence  $M(\mathcal{W}_4) = M^*(K_{3,3})$ . Therefore  $A(\mathcal{E})$  contains neither even  $\mathcal{E}$ -wheels nor even  $\mathcal{E}$ -3PC's because, by Lemma 1, these graphs could be pivoted into graphs containing a  $W_4$  as induced subgraph. Accordingly,  $M(\mathcal{E})$  would contain an  $M^*(K_{3,3})$  minor contradicting that  $M(\mathcal{E})$  is a graphic matroid. Therefore every Edge-Path-Tree family is a quasi-graphical family. Moreover, no Edge-Path-Tree family can contain the  $Q_6$  clutter as minor. Indeed every minor of an Edge-Path-Tree family is an Edge-Path-Tree family but the binary matroid generated by  $[I_6, A_{Q_6}]$ ,  $A_{Q_6}$  being the incidence matrix of the  $Q_6$  clutter, is the co-graphic matroid of the  $K_5$ .  $\square$

*Remark 2.* In view of the proof of Corollary 2 the term quasi-graphical is due to the fact that quasi graphical families contain families which generate regular matroids with no  $M^*(K_{3,3})$  minor, that is almost graphic matroids.

### 2.1 Proof of Theorem 2

Throughout the rest of the section we set  $N = \{1, \dots, n\}$  and  $P = \{1, \dots, p\}$ ; moreover,  $\mathcal{P} = \{L_1, \dots, L_n\}$  is an odd pie in  $\mathcal{C} = \{L_1, \dots, L_n, L_{n+1}, \dots, L_p\}$ ,  $\mathcal{C}$  being a clutter on  $V(\mathcal{C}) = V$ . Possibly after renumbering,  $L_i$  and  $L_j$  are consecutive in  $\mathcal{P}$  if and only if  $|i - j| \in \{1, n - 1\}$ . For  $i \in N$  we denote by  $S_i$  the set of elements of  $V(\mathcal{P})$  occurring in  $L_i$  and in no other member of  $\mathcal{P}$ . By the definition of branch it follows that

- $S_i \cap S_j = \emptyset$ ,  $i \neq j$ ,  $i, j \in N$  and  $S_i \cap B_j = \emptyset$ ,  $i, j \in N$ ;
- $\cup_{j \in N} L_j = (\cup_{j \in J} S_j) \cup (\cup_{j \in N} B_j)$ .
- $S_j \cup S_{j+1} \subseteq L_j \Delta L_{j+1}$  (addition over indices is modulo  $n$ ) and  $\cup_{j \in N} S_j = \Delta_{j \in N} L_j$ ,

where  $B_j$  is the  $j$ -th branch of  $\mathcal{P}$ ,  $j \in N$ . We also observe explicitly that if  $v \in B_i$  for some  $i \in N$  then  $v \in P_j$  for some  $j \neq i$  if and only if  $|i - j| \in \{1, n - 1\}$ . We denote by  $[\mathcal{P}]$  the clutter  $\mathcal{P} \setminus (V - V(\mathcal{P}))$  and by  $N^*$  the set of indices of  $[\mathcal{P}]$ . Thus  $N \subseteq N^* \subseteq P$ ,  $[\mathcal{P}] = \{L_j \mid j \in N^*\}$  and  $[\mathcal{P}]$  is the set of members of  $\mathcal{C}$  contained in  $V(\mathcal{P})$ . Finally, for  $s \in N^*$  let

$$\kappa_N(s) = |\{j \in N \mid P_i \cap B_j \neq \emptyset\}|.$$

**Lemma 2.** *If  $\mathcal{C}$  is a quasi-graphical clutter then  $\kappa_N(s)$  is either zero or two for each  $s \in N^*$ .*

*Proof.* For  $j = 1, \dots, n$ , if  $L_s \cap B_j \neq \emptyset$  let  $v_j$  be an element in  $L_s \cap B_j$ , otherwise let  $v_j$  be an element arbitrarily chosen in  $B_j$ . Thus  $\{v_1, \dots, v_n\} \cup N$  induces an odd hole  $C$  in  $B(\mathcal{C})$ . Now  $\kappa_N(s) \leq 2$  otherwise  $V(C) \cup \{s\}$  would induce a  $\mathcal{C}$ -wheel in  $B(\mathcal{C})$  with at least three spokes contradicting the assumption that  $\mathcal{C}$  is quasi-graphical. Thus  $\kappa_N(s) \leq 2$ . Suppose  $\kappa_N(s) = 1$ ; hence for some  $h \in N$ ,  $v_h \in L_s \cap B_h \neq \emptyset$  and  $v_j \notin L_s$ , for  $j \neq h$ . As  $[\mathcal{P}]$  is a clutter  $L_s$  intersects  $\cup_{j \in N} S_j$ . Suppose first that there is  $v \in L_s \cap S_l$  for some  $l \notin \{h, h + 1\}$  (addition is taken modulo  $n$ ). Thus  $V(C) \cup \{v\} \cup \{s\}$  induces a 3PC( $v_h, l$ ), contradicting

that  $\mathcal{C}$  is quasi-graphical. Hence  $L_s \cap S_l = \emptyset$  for every  $l \notin \{h, h+1\}$ . Necessarily there are  $u \in L_s \cap S_h$  and  $z \in L_s \cap S_{h+1}$  (for if not  $L_s$  would be included either in  $L_h$  or in  $L_{h+1}$ ). Possibly after renumbering, we may suppose that  $h = 1$ . Let us consider the graph  $G$  induced by  $V(\mathcal{C}) \cup \{s\} \cup \{u, z\}$  (see Figure 2 (a)). the set  $(V(\mathcal{C}) - \{v_1\}) \cup \{s\} \cup \{u, z\}$  induces a hole  $C'$  in  $G$  and the neighbors of  $v_1$  on  $C'$  are 1, 2 and  $s$ . Therefore  $V(C') \cup \{v_1\}$  induces the odd wheel  $(C', v_1)$  in  $B(\mathcal{C})$  contradicting that  $\mathcal{C}$  is quasi-graphical. We conclude that  $\kappa_N(s) \neq 1$  and hence  $\kappa_N(s) \in \{0, 2\}$  as stated.  $\square$

Let  $N_0^* = \{s \in N^* \mid \kappa_N(s) = 0 \text{ and } L_s \cap L_j \neq \emptyset \forall j \in N\}$ .

**Lemma 3.** *Let  $\mathcal{C}$  be a quasi-graphical clutter. If  $\kappa_N(t) = 0$  for some  $t \in N^*$  then either  $L_t \subseteq L_j \Delta L_{j+1}$  for some  $j \in N$ , or  $|N| = 3$  and  $N_0^* = \{t\}$ .*

*Proof.* Since  $\kappa_N(t) = 0$ ,  $L_t$  does not intersect any branch of  $\mathcal{P}$ . Hence  $L_t \subseteq \bigcup_{j \in N} S_j = \bigtriangleup_{j \in N} L_j$ . Suppose that  $L_t$  intersects two nonconsecutive members of the pie. Then  $|N| = n \geq 5$ . Let  $L_j$  and  $L_l$  be such members and let  $C$  be the hole induced in  $B(\mathcal{C})$  by  $\{v_1, \dots, v_n\} \cup N$  where  $v_i \in B_i$  ( $i = 1, \dots, n$ ). Pick  $u_j \in L_t \cap L_j$  and  $u_l \in L_t \cap L_l$ . Notice that  $u_j, u_l \notin V(\mathcal{C})$  because  $u_j \in S_j$  and  $u_l \in S_l$ . Moreover,  $j$  and  $l$  are at distance at least four on  $C$ . Thus  $V(\mathcal{C}) \cup \{t\} \cup \{u_j, u_l\}$  induces an even 3PC( $j, l$ ), that is, an even  $\mathcal{C}$ -3PC. This contradicts that  $\mathcal{C}$  is quasi-graphical. Therefore for  $n \geq 5$ , if  $\kappa_N(t) = 0$  then  $L_t \subseteq S_j \cup S_{j+1} \subseteq L_j \Delta L_{j+1}$  for some  $j \in N$ . It follows that, if  $\kappa_N(t) = 0$  but  $L_t \not\subseteq L_j \Delta L_{j+1}$  for all  $j \in N$ , then necessarily  $|N| = 3$  and  $L_t \cap S_j \neq \emptyset$  for all  $j \in N$ . Hence  $t \in N_0^*$ . To prove the rest of the lemma we need the following fact whose proof can be found in [2]. A chain is a family of inclusionwise nested members.

*Claim.* Let  $\mathcal{C} = (L_j \mid j \in P)$  be a quasi-graphical family (not necessarily a clutter) and let  $\mathcal{P} = \{L_1, L_2, L_3\}$  be a 3-pie in  $\mathcal{C}$ . Moreover, let  $N_0(\mathcal{C}, \mathcal{P}) \subseteq P$  be the set of indices of those members of  $\mathcal{C}$  which intersect each member of  $\mathcal{P}$  but no branch of  $\mathcal{P}$ . Then the family  $(L_j \mid j \in N_0(\mathcal{C}, \mathcal{P}))$  is a chain.

Since in a clutter nonempty members of nontrivial chains are singletons the last part of the lemma is a straightforward consequence of the claim after noticing that  $N_0(\mathcal{C}, \mathcal{P}) = N_0^*$ .  $\square$

Let  $\mathcal{P}$  be a 3-pie in  $\mathcal{C}$ . For  $s \in N^*$  we say that  $L_s$  wraps  $L_1$  if  $L_s \cap B_1$  and  $L_s \cap B_3$  are both nonempty. Similarly,  $L_s$  wraps  $L_2$  if  $L_s \cap B_1$  and  $L_s \cap B_2$  are both nonempty and  $L_s$  wraps  $L_3$  if  $L_s \cap B_2$  and  $L_s \cap B_3$  are both nonempty.

**Lemma 4.** *Let  $\mathcal{C}$  be a quasi-graphical clutter and  $\mathcal{P}$  be a 3-pie in  $\mathcal{C}$  with  $N_0^* = \{t\}$ . If for  $s \in N^* - \{t\}$  and  $i \in N$ ,  $L_s$  wraps  $L_i$  and meets  $L_t$  then  $L_s \cap L_t \cap L_{i+1}$  and  $L_s \cap L_t \cap L_{i+2}$  are both empty, addition over indices being modulo 3.*

*Proof.* Since  $s \neq t$  it follows that  $\kappa_N(s) = 2$  (by Lemma 2). Possibly after renumbering,  $L_s$  wraps  $L_1$ . Hence  $L_s$  intersects  $B_1$  and  $B_3$ . Let  $v_1 \in L_s \cap B_1$ ,  $v_3 \in L_s \cap B_3$ . Moreover, let  $u \in L_t \cap L_2$  if  $L_s \cap L_t \cap L_3 = \emptyset$  and  $u \in L_s \cap L_t \cap L_2$  otherwise (see Figure 3 (b)). Analogously, let  $v \in L_t \cap L_3$  if  $L_s \cap L_t \cap L_3 = \emptyset$  and



$v \in L_s \cap L_t \cap L_3$  otherwise. Observe that  $|L_j \cap \{v_1, v_3, u, v\}| = 2$  for  $j \in N \cup \{t\}$  and that  $|L_s \cap \{v_1, v_3, u, v\}| = 2$  if and only if  $L_s \cap L_t \cap L_2$  and  $L_s \cap L_t \cap L_3$  are both empty. Therefore if at least one among  $L_s \cap L_t \cap L_2$  and  $L_s \cap L_t \cap L_3$  is nonempty then  $|L_s \cap \{v_1, v_3, u, v\}| \geq 3$  and  $N \cup \{s, t\} \cup \{v_1, v_3, u, v\}$  induces a  $\mathcal{C}$ -wheel with at least three spokes in  $B(\mathcal{C})$  contradicting that  $\mathcal{C}$  is quasi-graphical.  $\square$

**Lemma 5.** *Let  $\mathcal{C}$  be a quasi-graphical clutter and let  $\mathcal{P}$  be a pie in  $\mathcal{C}$ . For  $j \in N$  and  $v \in B_j$  let  $\delta(v) = \{t \in N^* \mid v \in L_t\}$ . Then  $(\delta(v) \mid v \in B_j)$  is a chain for each  $j \in N$ .*

*Proof.* Suppose not. Hence, for some  $i \in N$  there are  $u(r), u(s) \in B_i$  and  $r, s \in N^*$  such that  $L_r \in \delta(u(r)) - \delta(u(s))$  and  $L_s \in \delta(u(s)) - \delta(u(r))$ . By Lemma 2,  $\kappa_N(r), \kappa_N(s) \in \{0, 2\}$ . Hence there are (not necessarily distinct) indices  $h, j \in N - \{i\}$  such that  $L_r \cap B_h \neq \emptyset$  and  $L_s \cap B_j \neq \emptyset$ . For  $l \notin \{h, i, j\}$ , let  $v_l \in B_l$ . Since  $\kappa_N(r) = \kappa_N(s) = 2$  and the branches of a pie are pairwise disjoint it follows that  $v_l \notin L_r$  and  $v_l \notin L_s$  for  $l \in N - \{h, i, j\}$ . Also observe that by the definition of branch  $u(r), u(s) \in L_{i+1}$ . Possibly after renumbering, we may suppose that  $h = 1$  and  $i \neq n$ . Hence  $i + 1 \neq 1$  (modulo  $n$ ). Let us distinguish three cases.

Case (a):  $h = j = 1$  and there is some  $z \in L_r \cap L_s \cap B_1$ . The assumptions on the indices guarantee that  $B_1 \neq B_i$  and  $B_1 \neq B_{i+1}$ . Thus  $\{1, z, r, u(r), i + 1, v_{i+1}, \dots, n, v_n\}$  induces a hole  $C$  in  $B(\mathcal{C})$ . The unique neighbor of  $s$  on  $C$  is  $z$ ; the unique neighbor of  $u(s)$  on  $C$  is  $i + 1$ ; and since  $u(s) \in L_s$ ,  $u(s)$  and  $s$  induce an edge in  $B(\mathcal{C})$ . Thus  $V(C) \cup \{u(s), s\}$  induces a 3PC( $z, i + 1$ ) in  $B(\mathcal{C})$ , contradicting that  $\mathcal{C}$  is quasi-graphical.

Case (b):  $h = j = 1$  and  $L_r \cap L_s \cap B_1 = \emptyset$ . Let  $z(r) \in L_r \cap B_1$  and  $z(s) \in L_s \cap B_1$ . Possibly after renumbering, we may suppose that  $\pi := |\{1, i + 1, i + 2, \dots, n\}| \geq |\{2, 3, \dots, i\}|$ . Thus if  $n = 3$  then  $\pi = 2$  and if  $n \geq 5$  then  $\pi \geq 3$ . Let  $C$  be the hole induced in  $B(\mathcal{C})$  by  $\{1, z(r), r, u(r), i + 1, u(s), s, z(s)\}$  and let  $Q$  be the path induced in  $B(\mathcal{C})$  by  $\{1, v_n, n, v_{n-1}, \dots, v_{i+1}, i + 1\}$ . The length of  $Q$  is  $2(\pi - 1)$ . Thus if  $n \geq 5$  then  $2(\pi - 1) \geq 4$  and  $V(C) \cup V(Q)$  induces an even 3PC( $1, i + 1$ ) in  $B(\mathcal{C})$ . If  $n = 3$  then  $i = 2$ . Hence  $B_1 \cup B_2 \subseteq L_2$  and  $\{u(r), u(s), z(r), z(s)\} \subseteq L_2$ . It follows that  $V(C) \cup \{2\}$  induces a  $\mathcal{C}$ -wheel with four spokes in  $B(\mathcal{C})$ . In either case the fact that  $\mathcal{C}$  is quasi-graphical is contradicted.

Case (c):  $h = 1 \neq j$ . Let  $z(r) \in L_r \cap B_1$  and  $z(s) \in L_s \cap B_j$ . As  $i \neq j$  and  $i \neq n$ , possibly after renumbering, we may suppose that  $i < j$ . Let (see Figure 2(b))

$$D_1 = \{z(s), j + 1, v_{j+1}, \dots, v_n, 1, z(r)\}, \quad D_2 = \{2, v_2, \dots, v_{i-1}, i\},$$

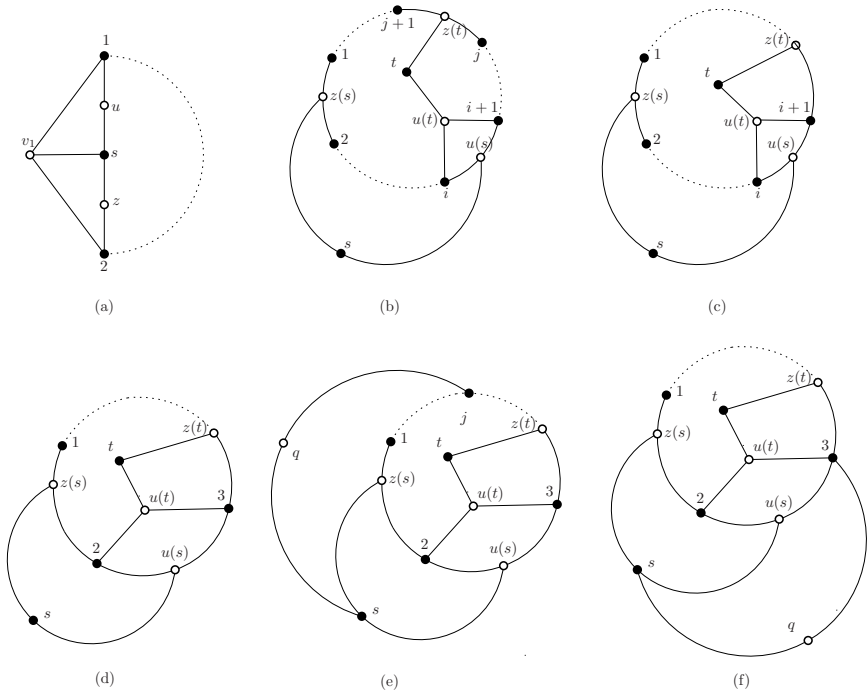
$$D_3 = \{i + 1, v_{i+1}, \dots, v_{j-1}, j\}.$$

We claim that  $j = i + 1$ . For, if not,  $D_1 \cup \{r, u(r)\} \cup D_3$  induces a hole  $C$  in  $B(\mathcal{C})$  and  $V(C) \cup \{s, u(s)\}$  is a 3PC( $i + 1, z(s)$ ), contradicting that  $\mathcal{C}$  is quasi-graphical. Hence  $D_3 = \{i + 1\}$ . Next we claim that  $i = 2$ . For, if not,  $z(r) \notin L_i$  because  $z(r) \in B_1$  and  $L_1$  and  $L_i$  are not consecutive. Hence  $z(r)$  and  $i$  are not adjacent in  $B(\mathcal{C})$ . Thus  $D_1 \cup \{r, u(r), i, u(s), s\}$  induces a hole  $C$  in  $B(\mathcal{C})$  and  $i + 1$  has exactly three neighbors on  $C$ , namely,  $u(r)$ ,  $u(s)$  and  $z(s)$  (Figure 2(c)). Therefore  $(C, i + 1)$  is an odd wheel in  $\mathcal{C}$  contradicting that  $\mathcal{C}$  is quasi-graphical. We conclude that

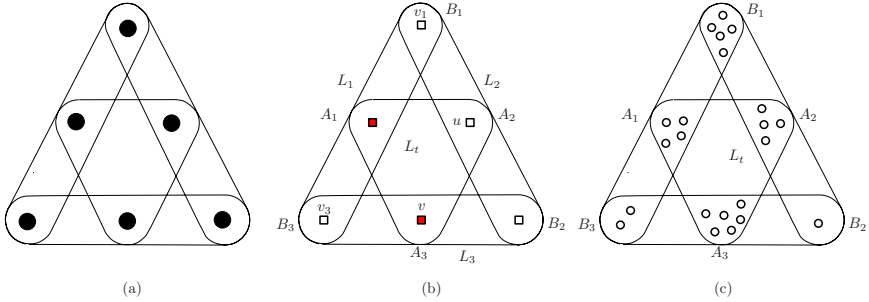


$i = 2$  and, consequently, that  $D_2 = \{2\}$  and  $D_3 = \{3\}$ . Let now  $C$  be the hole induced by  $\{1, z(r), 2, u(r), 3, z(s), \dots, n, v_n\}$  (Figure 2(d)). Since  $L_r$  intersects  $B_1$  and  $B_2$  it follows that  $L_r$  must intersect some  $S_j$  with  $j \neq 2$ . For, if not,  $L_r$  would be included in  $L_2$  and  $[\mathcal{P}]$  would not be a clutter contradicting that  $[\mathcal{P}]$  is a deletion minor of the clutter  $\mathcal{C}$ . Thus there is some  $q \in L_r \cap S_j$  for some  $j \in N - \{2\}$ . Suppose first that  $j \neq 3$  and let  $D = \{z(s), \dots, j, q, r, u(r), 2, u(s), s\}$ . Clearly  $D$  induces a hole  $C'$  in  $B(\mathcal{C})$ . Since 3 has exactly three neighbors on  $C'$ , namely,  $z(s)$ ,  $u(s)$  and  $u(r)$  it follows that  $(C', 3)$  is an induced odd wheel in  $B(\mathcal{C})$  (Figure 2(e)). Thus  $j = 3$ . Let now  $D = D_1 \cup \{r, q, 3\}$ . The graph  $C'$  induced by  $D$  is still a hole. Moreover, 2 and  $u(s)$  have each exactly one neighbor on  $C'$ , namely,  $z(r)$  and 3, respectively. Since  $u(s)$  and 2 are adjacent in  $B(\mathcal{C})$  it follows that  $V(C') \cup \{2, s\}$  induces a 3PC( $z(s), 3$ ) in  $B(\mathcal{C})$  (Figure 2(f)). We conclude that case (c) cannot occur and this completes the proof of the lemma.  $\square$

*Proof of the Theorem.* Let  $\mathcal{P} = (L_j \mid j \in N)$  be a pie in the quasi-graphical clutter  $\mathcal{C} = (L_j \mid j \in P)$  where,  $N = \{1 \dots, n\} \subseteq P = \{1, \dots, p\}$  and, possibly after renumbering,  $L_i$  and  $L_j$  are consecutive in  $\mathcal{P}$  if and only if  $|i - j| \in \{1, n - 1\}$ . Observe first that  $\kappa_N(t)$  must be zero for some  $t \in N^*$  otherwise  $\mathcal{C}$  would contain an odd pie as minor contradicting the hypotheses of the Theorem. To



**Fig. 2.** (a): the odd wheel in Lemma 2 (b)-(f): the various cases occurring in part (c) of Lemma 5. Solid lines represent edges and dotted lines represent paths.



**Fig. 3.** (a): a Venn-representation of the  $Q_6$  clutter; (b): Lemma 4 (red) boxes are (possibly) elements of  $L_s$ ; (c): the factorization on the r.h.s of (3)

see this let us argue as follows. By Lemma 2 if  $\kappa_N(s) > 0$  for each  $s \in N^*$  then  $\kappa_N(s) = 2$  for each  $s \in N^*$ . For  $j = 1, \dots, n$  let  $v_j$  be such that  $\delta(v) \subseteq \delta(v_j)$  for each  $v \in B_j$ . Thus  $|L_s \cap \{v_1, \dots, v_k\}| = 2$  for each  $s \in N^*$ . It follows that  $[\mathcal{P}]/(V(\mathcal{P}) - \{v_1, \dots, v_n\})$  is an  $n$ -pie. A contradiction. Hence  $\kappa_N(t) = 0$ , for some  $t \in N^*$ . By Lemma 3 either

- case (a)  $L_t \subseteq L_j \Delta L_{j+1}$  for some  $t \in N^*$  and some  $j \in N$ ; or
- case (b)  $n = 3$ ,  $N_0^* = \{t\}$  and  $\kappa_N(s) = 2$  for each  $s \in N^* - \{t\}$ .

In case (a), since  $L_t \subseteq L_j \Delta L_{j+1} \subseteq L_j \cup L_{j+1}$  and  $[\mathcal{P}]$  is a clutter, it follows that  $L_t$  meets both  $L_j$  and  $L_{j+1}$ . Thus  $\mathcal{Q} = \{L_j, L_{j+1}, L_t\}$  is a 3-pie with index set  $\{j, j + 1, t\}$ . Possibly after relabelling the members of  $\mathcal{C}$  we may suppose that  $j = 1$ , and  $t = 3$ . Hence  $\mathcal{Q}$  is a 3-pie in  $\mathcal{C}$  whose index set is  $M = \{1, 2, 3\}$  and where  $M^*$  is the index set of  $[\mathcal{Q}]$ . By Lemma 2 one has  $\kappa_M(s) = 0$  for some  $s \in M^*$  otherwise, as above,  $[\mathcal{Q}]$  (and hence  $\mathcal{C}$ ) would contain an odd pie minor. Therefore  $L_s \subseteq L_1 \Delta L_2 \Delta L_3$  for some  $s \in M^*$ . Thus  $L_s \subseteq (L_1 \Delta L_2) - L_3$  (because  $L_t \subseteq L_j \Delta L_{j+1}$ ). Hence  $L_s \cap L_t = \emptyset$  and the theorem is proved in case (a) with  $F_0 = L_s$  and  $F_1 = L_t$ .

Suppose we are in case (b). We show that  $[\mathcal{P}]$  contains the  $Q_6$  clutter as minor. Let  $A_1 = L_t \cap L_1$ ,  $A_2 = L_t \cap L_2$  and  $A_3 = L_t \cap L_3$  (see Figure 3 (c)). We claim that

$$L_s \subseteq A_1 \cup A_2 \cup A_3 \cup B_1 \cup B_2 \cup B_3 \text{ for every } s \in N^* - \{t\}. \tag{3}$$

To see (3) suppose that it does not hold. Thus there exists some  $v$  in  $L_i \cap L_s - (L_{i+1} \cup L_{i+2} \cup L_t)$  for some  $i \in N$  and some  $s \in N^* - \{t\}$ . Observe that  $L_{i+1}$ ,  $L_{i+2}$  and  $L_t$  are members of  $[\mathcal{P}] \setminus v$ . Since  $N_0^* = \{t\}$ , it follows that  $\mathcal{Q} = \{L_{i+1}, L_{i+2}, L_t\}$  is a 3-pie in  $[\mathcal{P}] \setminus v$  and hence in  $[\mathcal{P}]$  and  $\mathcal{C}$ . Let  $M = \{i + 1, i + 2, t\}$ , be the index set of  $\mathcal{Q}$  where  $i + 1$  and  $i + 2$  are modulo 3 and let  $M^*$  be the index set of  $[\mathcal{Q}]$ . Remark that  $[\mathcal{Q}]$  is a minor of  $[\mathcal{P}]$ . For no  $r \in P$  member  $L_r$  can be contained in  $L_{i+1} \Delta L_{i+2} \Delta L_t$  otherwise any such member either would be contained in one among  $S_1 \cup S_2$ ,  $S_1 \cup S_3$  and  $S_2 \cup S_3$  contradicting that we are in case (b) or  $r \in N_0^*$  and  $r \neq t$  contradicting Lemma 3.

Hence  $\kappa_M(s) = 2$  for each  $s \in M^*$  and, as above,  $[Q]$  contains an odd pie as minor contradicting the hypotheses of the theorem. Thus we conclude that (3) holds. Next we claim that

$$N^* = N \cup \{t\}. \quad (4)$$

To prove (4) let us argue as follows. Since  $\kappa_N(s) = 2$  for all  $s \in N^* - \{t\}$  it follows that for each  $s \in N^* - \{t\}$  there is  $i \in N$  such that  $L_s$  wraps  $i$  and meets  $L_t$  (by (3)). By Lemma 4,  $L_s \cap A_{i+1}$  and  $L_s \cap A_{i+2}$  are both empty ( $i+1$  and  $i+2$  are modulo 3). Therefore (still by (3))  $L_s \subseteq A_i \cup B_i \cup B_{i+2} \subseteq L_i$ . Thus  $s = i$  (because  $[P]$  is a clutter) and  $N^* = \{1, 2, 3, t\}$ . Hence (4) holds and  $[P] = \{L_1, L_2, L_3, L_t\}$ .

Let now  $a_i$  and  $b_i$  be arbitrarily chosen elements in  $A_i$  and  $B_i$ , respectively ( $i = 1, 2, 3$ ). Thus, for  $j \in N \cup \{t\}$ ,  $L_j \cap \{a_1, a_2, a_3, b_1, b_2, b_3\}$  is one of the four members of the  $Q_6$  clutter. Hence  $[P]/(V(P) - \{a_1, a_2, a_3, b_1, b_2, b_3\})$  is the  $Q_6$  clutter and the proof of part (b) is completed.  $\square$

## References

1. Apollonio, N.: Integrality properties of edge path tree families. *Discrete Mathematics* 309, 4181–4184 (2009)
2. Apollonio, N., Caramia, M.: A superclass of Edge-Path-Tree graphs with few cliques. *Operations Research Letters* (in press, 2009), doi:10.1016/j.orl.2009.05.002
3. Apollonio, N., Caramia, M.: A local structure result for line graphs of strong quasi-graphical matrices. Technical Report DII RR-08.09, University of Rome Tor Vergata, <http://dspace.uniroma2.it/dspace/handle/2108/655>
4. Conforti, M., Gerards, A.M.H., Kapoor, A.: A Theorem of Truemper. *Combinatorica* 20, 15–26 (2000)
5. Cornuéjols, G., Guenin, B., Margot, F.: The Packing Property. *Mathematical Programming* 89, 113–126 (2000)
6. Cornuéjols, G.: *Combinatorial Optimization. Packing and Covering*. SIAM, CBMS–NSF, Philadelphia (2001)
7. Fournier, J.-C.: Hypergraphes de Chaines d’Aretes d’un Arbre. *Discrete Mathematics* 43, 29–36 (1983)
8. Golumbic, M.C., Jamison, R.E.: The Edge Intersection Graphs of Paths in a Tree. *Journal of Combinatorial Theory, Series B* 38, 8–22 (1985)
9. Schrijver, A.: *Theory of Linear and Integer Programming*. Wiley, Chichester (1986)

# Forbidden Subgraph Colorings and the Oriented Chromatic Number

N.R. Aravind and C.R. Subramanian

The Institute of Mathematical Sciences,  
Taramani, Chennai-600113, India  
{nraravind, crs}@imsc.res.in

**Abstract.** We present an improved upper bound of  $O(d^{1+\frac{1}{m-1}})$  for the  $(2, \mathcal{F})$ -subgraph chromatic number  $\chi_{2, \mathcal{F}}(G)$  of any graph  $G$  of maximum degree  $d$ . Here,  $m$  denotes the minimum number of edges in any member of  $\mathcal{F}$ . This bound is tight up to a  $(\log d)^{1/(m-1)}$  multiplicative factor and improves the previous bound presented in [1].

We also obtain a relationship connecting the oriented chromatic number  $\chi_o(G)$  of graphs and the  $(j, \mathcal{F})$ -subgraph chromatic numbers  $\chi_{j, \mathcal{F}}(G)$  introduced and studied in [1]. In particular, we relate oriented chromatic number and the  $(2, r)$ -treewidth chromatic number and show that  $\chi_o(G) \leq k((r+1)2^r)^{k-1}$  for any graph  $G$  having  $(2, r)$ -treewidth chromatic number at most  $k$ . The latter parameter is the least number of colors in any proper vertex coloring which is such that the subgraph induced by the union of any two color classes has treewidth at most  $r$ .

We also generalize a result of Alon, et. al. [2] on acyclic chromatic number of graphs on surfaces to  $(2, \mathcal{F})$ -subgraph chromatic numbers and prove that  $\chi_{2, \mathcal{F}}(G) = O(\gamma^{m/(2m-1)})$  for some constant  $m$  depending only on  $\mathcal{F}$ . We also show that this bound is nearly tight. We then use this result to show that graphs of genus  $g$  have oriented chromatic number at most  $2^{O(g^{1/2+\epsilon})}$  for every fixed  $\epsilon > 0$ . This improves the previously known bound of  $2^{O(g^{4/7})}$ . We also refine the proof of a bound on  $\chi_o(G)$  obtained by Kostochka, et. al. in [3] to obtain an improved bound on  $\chi_o(G)$ .

## 1 Introduction

We study several variants of proper vertex colorings and present relationships connecting them. The chromatic number  $\chi(G)$  of  $G$  is the least  $k$  such that  $G$  is properly colorable using  $k$  colors. An acyclic vertex coloring (introduced in [4], see also [5]) of  $G = (V, E)$  is a proper coloring of  $V$  in which the subgraph induced by the union of any two color classes is acyclic. The acyclic chromatic number  $a(G)$  is the least  $k$  such that  $G$  admits an acyclic vertex coloring using  $k$  colors.

Sopena, in [6], introduced the notion of oriented chromatic number for oriented graphs (directed graphs having no self-loops and no 2-cycles). The oriented chromatic number of an oriented graph  $\vec{G}$  is the smallest size  $|V(\vec{H})|$  of a  $\vec{H}$  for

which a homomorphism  $\phi : \vec{G} \rightarrow \vec{H}$  exists. Equivalently,  $\chi_o(\vec{G})$  is the smallest  $k \geq 1$  such that there is a proper  $k$ -coloring  $(V_1, \dots, V_k)$  of  $V(\vec{G})$  such that for every  $i \neq j$ , all edges joining  $V_i$  and  $V_j$  are oriented in the same way.

The oriented chromatic number  $\chi_o(G)$  of an undirected graph  $G$  is the maximum value of  $\chi_o(\vec{G})$  where the maximum is over all orientations  $\vec{G}$  of  $G$ . Bounds for the oriented chromatic number have been obtained in terms of the maximum degree and also for special families of graphs such as trees, planar graphs, partial  $k$ -trees ([6]), etc. Of these, the following two results are relevant to the main results of this paper. They are:

- (B1) The result of Sopena in [6] that, for every  $r \geq 1$ , every partial  $r$ -tree has oriented chromatic number at most  $(r + 1)2^r$ .
- (B2) The result of Raspaud and Sopena in [7] that if a graph has acyclic chromatic number at most  $k$ , then  $\chi_o(G) \leq k2^{k-1}$ .

In this paper, we obtain improved bounds on  $(2, \mathcal{F})$ -subgraph chromatic numbers (of which acyclic chromatic number is a special case) and also obtain a relation involving these numbers and oriented chromatic numbers. We then apply these results to obtain new or improved bounds on  $(2, \mathcal{F})$ -subgraph chromatic numbers and oriented chromatic numbers of graphs of bounded genus.

### 1.1 Improved Bounds on $(2, \mathcal{F})$ -Chromatic Numbers

Recently, the present authors [1] studied a generalized notion of proper colorings which impose constraints on the union of any few color classes. Such a notion was first considered by Nesetril and Ossona de Mendez in [8], where it is proved that these numbers are bounded for proper minor-closed families of graphs. For suitably chosen constraints, this general notion specializes to known restricted colorings like acyclic colorings, star colorings, etc. We need the following definitions from [1], which formally define a general restricted coloring.

Given two graphs  $G$  and  $H$ , we say that  $G$  is  $H$ -free if  $G$  has no isomorphic copy of  $H$  as a subgraph (not necessarily induced). Given a family  $\mathcal{F}$  of graphs, we say that  $G$  is  $\mathcal{F}$ -free if  $G$  is  $H$ -free for each  $H \in \mathcal{F}$ .

**Definition 1.** *Let  $j$  be a positive integer and  $\mathcal{F}$  be a family of connected graphs of (usual) chromatic number at most  $j$  such that for each  $H \in \mathcal{F}$ ,  $|V(H)| > j$ . We define a  $(j, \mathcal{F})$ -subgraph coloring to be a proper coloring of the vertices of a graph  $G$  so that the subgraph of  $G$  induced by the union of any  $j$  color classes is  $\mathcal{F}$ -free. We denote by  $\chi_{j, \mathcal{F}}(G)$  the minimum number of colors sufficient to obtain a  $(j, \mathcal{F})$ -subgraph coloring of  $G$ .*

In [1], we obtained the bound of  $\chi_{j, \mathcal{F}}(G) = O\left(d^{\frac{k-1}{k-j}}\right)$  for any graph  $G$  of maximum degree  $d$ , where  $k$  is  $\min_{H \in \mathcal{F}} |V(H)|$ . But this bound is not optimal; for acyclic coloring, we have  $j = 2$ ,  $\mathcal{F} = \{C_4, C_6, \dots\}$  and  $k = 4$  and hence we get a bound of  $O(d^{3/2})$  but it is known [9] that  $a(G) = O(d^{4/3})$ . Our first main result is the following improved bound on the  $(2, \mathcal{F})$ -chromatic numbers. This improves the above bound of [1] for the case  $j = 2$ .

**Theorem 1.** *Let  $\mathcal{F}$  be a family of connected bipartite graphs on 3 or more vertices such that the minimum number of edges in any member of  $\mathcal{F}$  is  $m$ . Then, for any graph  $G$  of maximum degree  $d$ ,  $\chi_{2,\mathcal{F}}(G) < \lceil Cd^{1+\frac{1}{m-1}} \rceil$  where  $C = C(\mathcal{F}) = 64(m+1)^3s$  and  $s$  is the number of bipartite graphs in  $\mathcal{F}$  on at most  $m$  vertices.*

For acyclic vertex coloring, we note that this leads to the optimal bound of  $O(d^{4/3})$  since  $\mathcal{F} = \{C_4, C_6, \dots\}$  so that  $m = 4$  in this case. In fact, for every constant family  $\mathcal{F}$ , the upper bound of Theorem 1 is tight within a multiplicative factor of  $O((\log d)^{1/(m-1)})$ ; this follows from the results of [1].

**1.2 Relating  $\chi_{j,\mathcal{F}}(G)$  and  $\chi_o(G)$**

Our second main result is the following connection between  $(j, \mathcal{F})$ -subgraph colorings and oriented colorings. This generalizes and was inspired by the connection (B2) between  $a(G)$  and  $\chi_o(G)$  established in [7]. For a family  $\mathcal{F}$  of connected graphs, let  $Forb(\mathcal{F}) = \{G : G \text{ is } \mathcal{F}\text{-free}\}$ .

**Theorem 2.** *Let  $\mathcal{F}$  be a family of connected graphs. Suppose there exists a natural number  $t$  such that  $\chi_o(F) \leq t$ , for each  $F \in Forb(\mathcal{F})$ . Suppose  $j \geq 2$ . Then, for any graph  $G \notin Forb(\mathcal{F})$  with  $\chi_{j,\mathcal{F}}(G) \leq k$ , its oriented chromatic number  $\chi_o(G)$  is at most  $kt^{\lceil \frac{2k-j}{j} \rceil}$  if  $j$  is even and is at most  $kt^{\lceil \frac{2k-j+1}{j-1} \rceil}$  if  $j$  is odd.*

In Section 2, we prove this theorem. The special case of this theorem obtained by setting  $j = 2$ , is going to be used later and we state it separately as the following theorem.

**Theorem 3.** *Let  $\mathcal{F}$  be a family of connected graphs. Suppose there exists a  $t$  such that  $\chi_o(F) \leq t$ , for each  $F \in Forb(\mathcal{F})$ . Then, for any graph  $G \notin Forb(\mathcal{F})$  with  $\chi_{2,\mathcal{F}}(G) \leq k$ , its oriented chromatic number  $\chi_o(G)$  is at most  $kt^{k-1}$ .*

Fixing  $\mathcal{F} = \{G : \chi(G) \leq j, tw(G) \geq r + 1\}$ , Definition 1 specializes to the following:

**Definition 2.** *Let  $j, r$  be positive integers such that  $j \leq r + 1$ . We define a  $(j, r)$ -treewidth (vertex) coloring of a graph  $G = (V, E)$  to be a proper coloring of  $V(G)$  so that the subgraph induced by the union of any  $j$  color classes has treewidth at most  $r$ . We denote by  $\chi_{j,r}^{tw}(G)$  the minimum number of colors sufficient to obtain a  $(j, r)$ -treewidth coloring of  $G$ .*

We now specialize Theorem 3 by choosing  $\mathcal{F}$  to be the set of all connected bipartite graphs of treewidth  $r + 1$  and apply the bound (B1) on the oriented chromatic number of partial  $r$ -trees to obtain the following result which we shall later use to bound the oriented chromatic number of graphs on surfaces.

**Corollary 1.** *For  $r \geq 2$ , let  $G$  be any graph with a  $(2, r)$ -treewidth chromatic number at most  $k$ . Then,  $\chi_o(G)$  is at most  $k((r + 1)2^r)^{k-1}$ .*

### 1.3 $(2, \mathcal{F})$ -Subgraph Colorings of Graphs on Surfaces

It is known from the Map Color Theorem of Ringel and Youngs [10] that the chromatic number of an arbitrary surface of Euler characteristic  $-\gamma$  is  $\Theta(\gamma^{1/2})$ . Using the  $O(d^{4/3})$  bound (for general graphs), Alon, Mohar and Sanders proved in [2] that the acyclic chromatic number of a (simple) graph embeddable on a surface of characteristic  $-\gamma (\leq 0)$  is at most  $100\gamma^{\frac{4}{7}} + 10^4$ . It was also shown that this bound is nearly tight.

Generalizing these arguments and by using the bound of Theorem [1], we prove that this result can be extended for  $(2, \mathcal{F})$ -colorings as well provided that  $\mathcal{F}$  does not contain connected graphs with pendant vertices. Our third main result is this extension. which we state below.

**Theorem 4.** *Let  $\mathcal{F}$  be a family of connected bipartite graphs on at least 4 vertices each having minimum degree at least 2. Let  $m$  be the smallest number of edges of any member of  $\mathcal{F}$ . If  $G$  is a (simple) graph embeddable on a surface of Euler characteristic  $-\gamma \leq 0$ , then  $\chi_{2, \mathcal{F}}(G) \leq A\gamma^{\frac{m}{2m-1}} + B$  where  $A$  and  $B$  are constants depending only on  $\mathcal{F}$ .*

For the acyclic chromatic number, we have  $m = 4$  and  $m/(2m - 1) = 4/7$ ; this is the bound obtained in [2]. By choosing  $\mathcal{F} = \mathcal{F}_r$  where  $\mathcal{F}_r$  is the set of all connected bipartite graphs of treewidth  $r + 1$ , we get the following consequence of Theorem [4].

**Corollary 2.** *If  $G$  is a simple graph embeddable on a surface of Euler characteristic  $-\gamma \leq 0$ , then,  $\chi_{2,r}^{tw}(G) \leq A\gamma^{\frac{m_r}{2m_r-1}} + B$  for every  $r \geq 1$ . Here,  $A$  and  $B$  are suitable absolute positive constants and  $m_r$  denotes the minimum number of edges in any member of  $\mathcal{F}_r$ .*

We also establish that the upper bound of Theorem [4] is tight upto a  $\text{polylog}(\gamma)$  multiplicative factor. This generalizes a similar tightness result presented in [2] for acyclic chromatic numbers.

**Theorem 5.** *Let  $\mathcal{F}$  and  $m$  be as described in Theorem [4]. For every sufficiently large  $\gamma \geq 0$ , there is a graph  $G$  embeddable on a surface (orientable or non-orientable) with Euler characteristic  $-\gamma : \chi_{2, \mathcal{F}}(G) \geq c\gamma^{\frac{m}{2m-1}} / (\log \gamma)^{1/(2m-1)}$ , for some positive constant  $c$  which depends only on  $\mathcal{F}$ .*

### 1.4 Oriented Chromatic Numbers on Surfaces

For graphs of Euler characteristic  $-\gamma \leq 0$ , by combining the bound  $a(G) = O(\gamma^{4/7})$  obtained in [2] with the bound (B2) of [7] (mentioned before), we get an upper bound of  $O(\gamma^{4/7} 2^{O(\gamma^{4/7})}) = 2^{O(\gamma^{4/7})}$  for the oriented chromatic number  $\chi_o(G)$ . The fourth main result of this paper is an improvement of this bound and is obtained by combining Corollary [1] and Corollary [2]. Recall that Corollary [1] is a generalization of bound (B2) and Corollary [2] is a generalization of the bound obtained in [2].

**Theorem 6.** *Let  $G$  be a simple graph embeddable on a surface of Euler characteristic  $-\gamma \leq 0$ . Then,*

$$\chi_o(G) \leq O(\gamma^{\frac{m_r}{2m_r-1}})((r+1)2^r)^{O(\gamma^{\frac{m_r}{2m_r-1}})} \leq 2^{O(\gamma^{m_r/(2m_r-1)})}$$

for every fixed  $r \geq 1$ . Thus for every  $\epsilon > 0$ , there exists  $c_\epsilon$  such that  $\chi_o(G) \leq 2^{c_\epsilon \gamma^{(1/2)+\epsilon}}$ .

**Proof:** Follows as a consequence of combining Corollary 1 and Theorem 4 with the bound (B1) (mentioned earlier).

Note that this significantly improves the bound  $2^{O(\gamma^{4/7})}$  mentioned before.

### 1.5 Outline of the Paper

We prove Theorem 1 in Section 2. Theorem 2 is proved in Section 3. Theorem 4 and Theorem 5 are proved in Section 4. In Section 5, we refine the proof of a bound on  $\chi_o(G)$  (in terms of its maximum degree) obtained by Kostochka, Sopena and Zhu in 3 to obtain an improved bound on  $\chi_o(G)$ . Finally, in Section 6, we conclude with remarks on some related issues.

## 2 Proof of Theorem 1

The proof is based on probabilistic arguments. The probabilistic tool we use is an easy-to-use version of the Lovasz Local Lemma, which we state below without proof (will appear in the journal version).

**Lemma 1.** *(Special case of Lovász Local Lemma) Consider a finite set of events which can be partitioned into types  $1, 2, \dots$  such that the probability of any event of type  $i$  is at most  $p_i$  and let the events of type  $i$  be  $A_{i,1}, A_{i,2}, \dots$ . Further, let there be reals  $a_1, a_2, \dots$  and  $b_1, b_2, \dots$  (each  $b_i \geq 1$ ) such that any event of type  $i$  is independent of all but at most  $a_i * b_j$  events of type  $j$ . Suppose, also that (A) :  $\sum_i 2^{(a_i+1)} b_i p_i \leq 1$  holds. Then,*

$$Pr(\cap(\overline{A_{i,j}})) > 0$$

*i.e. with positive probability none of the events  $A_{i,j}$  holds. In particular, if the number of types of events is  $k$  and  $k2^{a_i+1} b_i p_i \leq 1$  for each  $i \in [k]$ , then with positive probability, none of the events  $A_{i,j}$  hold.*

### Proof of Theorem 1:

Choose  $x = \lceil Cd^{1+\beta} \rceil$  where  $\beta = \frac{1}{m-1}$  and  $C = C(\mathcal{F}) = 64(m+1)^3 s$ .

Let  $f : V \rightarrow \{1, 2, \dots, x\}$  be a random vertex coloring of  $G$ , where for each vertex  $v \in V$  independently, the color  $f(v) \in \{1, 2, \dots, x\}$  is chosen uniformly at random. It suffices to prove that with positive probability,  $f$  is a  $(2, \mathcal{F})$ -coloring of  $G$ . To this end, we define a family of bad events whose total failure implies a  $(2, \mathcal{F})$ -coloring and use the Lovasz local Lemma (as stated in Lemma 1) to show



that with positive probability none of them occurs. The events we consider are of the following types.

a) **Type 1:** For each pair of adjacent vertices  $u$  and  $v$ , let  $A_{u,v}$  be the event that  $f(u) = f(v)$ .

To reduce the number of copies of forbidden subgraphs we need to consider, we use a notion similar to the one employed in [9]. A subset of  $k$  vertices is called a special  $k$ -set if there are more than  $d^{1-(k-1)\beta}$  vertices adjacent to each of the  $k$  vertices.

We say that a subset  $S$  of the vertices is *good* if for every vertex  $v \in S$  and for any  $k \in [2, m]$ , the set of neighbors of  $v$  in  $S$  does not contain any special  $k$ -set as a subset.

For each  $k \in [2, m]$ , we define the following events:

b) **Types 2,k:** For each special set  $S$  of  $k$  vertices, let  $B_k(S)$  be the event that the vertices of  $S$  are colored with one common color by  $f$ .

c) **Type 3:** For each connected subset  $L$  of  $V(G)$  such that  $|L| = m + 1$ , let  $C_L$  be the event that the vertices in  $L$  are colored using at most 2 colors in the coloring by  $f$ .

Let the bipartite members of  $\mathcal{F}$  of size at most  $m$  be  $H_1, H_2, \dots, H_s$  where  $s = s(\mathcal{F})$  is the number of such members. For each  $i \in [1, s]$ , we define the following Type 4,  $i$  events:

d) **Type 4,i:** For each good subset  $S$  of vertices of  $G$  such that  $G[S]$  contains  $H_i$  as a spanning subgraph, let  $D_i(S)$  be the event that the random coloring  $f$  uses at most 2 colors on the vertices of  $S$ .

If we forbid all events of Types 1 and  $(2, k)$ , then for any  $S \subseteq V$  such that (i)  $G[S]$  contains some  $H_i$  as a spanning subgraph and (ii)  $S$  is not a good set, there should be some  $v \in S$  and some  $k \in [2, m]$  such that  $N_S(v)$  contains a special  $k$ -set which is not monochromatically colored (since events of Type 2,k are forbidden) and hence  $f$  uses at least 3 colors on  $S$ .

Thus, it follows that if none of the events of the above types occur, then  $f$  is a  $(2, \mathcal{F})$ -coloring. We first estimate upper bounds on the probabilities of each type of events.

- (i) For each Type 1 event  $A$ ,  $p_1 = Pr(A) = \frac{1}{x}$ .
- (ii) For each Type  $(2, k)$  event  $B_k$ ,  $p_{2,k} = Pr(B_k) = \frac{1}{x^{k-1}}$ .
- (iii) For each Type 3 event  $C$ ,  $p_3 = Pr(C) \leq \frac{1}{x^{m-1}}$ .
- (iv) For each Type  $(4, i)$  event  $D_i$ ,  $p_{4,i} = Pr(D_i) \leq \frac{2^{n_i}}{x^{n_i-2}}$ .

Note that any of the events defined above is mutually independent of all events that do not share a vertex in common with the given event. Thus, it suffices to estimate the number of events of each type containing a given vertex. This estimate is given in the following simple lemma.

**Claim 2**

Let  $v$  be an arbitrary vertex of the graph  $G = (V, E)$ . Then the following two statements hold.

- (i)  $v$  belongs to at most  $d$  edges of  $G$ .
- (ii) For each  $k \in [2, m]$ , the number of special  $k$ -sets containing  $v$  is at most  $d^{(k-1)(1+\beta)}$ .
- (iii)  $v$  belongs to at most  $(m + 1)4^{m+1}d^m$  connected subsets of size  $m + 1$  in  $V(G)$ .
- (iv) For each  $i \in [1, m + 1]$ ,  $v$  belongs to at most  $n_i d^{(n_i-2)(1+\beta)}$  subgraphs isomorphic to  $H_i$  where  $n_i = |V(H_i)|$  and such that the vertex set of the subgraph is *good*.

**Proof of Claim 2**

Part (i) follows from the fact that  $\Delta(G) = d$ .

Part (ii) follows from the fact that there are at most  $d^k$  induced stars of size  $k + 1$  in  $G$ , with  $v$  as a leaf, and for each special  $k$ -set there are more than  $d^{1-(k-1)\beta}$  centers of the  $k + 1$ -star. Thus the number of special  $k$ -sets containing  $v$  is at most  $\frac{d^k}{d^{1-(k-1)\beta}} = d^{(k-1)(1+\beta)}$ .

Part (iii) has already been established as part of the proof of Proposition 2.2 in [1].

Part (iv) can be seen as follows: The position of  $v$  in  $H_i$  has atmost  $n_i$  choices. Once  $v$  is identified with a vertex of  $H_i$ , the number of ways of embedding the remaining vertices can be bounded as follows: consider a sequence  $v_2, \dots, v_{n_i}$  of the remaining vertices of  $H_i$  such that each vertex has atleast one neighbour to its left in the sequence. Clearly this is possible since  $H_i$  is connected. Let  $t_i$  denote the number of vertices to the left of  $v_i$  and adjacent to it. Once the vertices to the left of  $v_i$  are embedded in  $G$ , the number of ways of identifying  $v_i$  in  $G$  is at most  $d^{1-(t_i-1)\beta}$  because there is no special  $t_i$  set among these vertices. Thus the number of ways of embedding the remaining vertices of  $H_i$  in  $G$  is at most  $d \sum_{i=2}^{n_i} [1-(t_i-1)\beta]$ . Using the fact that  $\sum_{i=2}^{t_i} |E(H_i)| \geq m$  and  $\beta = \frac{1}{m-1}$ , we see that  $\sum_{i=2}^{n_i} [1-(t_i-1)\beta] \leq (n_i-1)(1+\beta) - m\beta = (n_i-2)(1+\beta)$ . This proves Part (iv) and completes the proof of Claim 2.

Since an event is independent of all other events with which it does not share a vertex, we see that the assumptions of Lemma 1 hold with the following values of  $a_i$ s and  $b_i$ s.

- Type 1 :  $a_1 = 2, b_1 = d$ .
- Type 2,  $k$  :  $a_{2,k} = k, b_{2,k} = d^{(k-1)(1+\beta)}$  for each  $k \in [2, m]$ .
- Type 3 :  $a_3 = m + 1, b_3 = (m + 1)4^{m+1}d^m$ .
- Type 4,  $i$  :  $a_{4,i} = n_i, b_{4,i} = n_i d^{(n_i-2)(1+\beta)}$  for each  $i \in [1, s]$ .

By Lemma 1, to prove that with positive probability none of the "bad" events hold, it suffices to verify the following inequality:

$$8 \frac{d}{x} + \sum_{k=2}^m 2^{(k+1)} \frac{d^{(k-1)(1+\beta)}}{x^{k-1}} + 2(m + 1)8^{m+1} \frac{d^m}{x^{m-1}} + \sum_{i=1}^s 2n_i 4^{n_i} \frac{d^{(n_i-2)(1+\beta)}}{x^{n_i-2}} \leq 1$$

We now substitute  $x = Cd^{1+\frac{1}{m-1}}$  where  $C = 64(m + 1)^3 s$ . Using the facts that  $\beta = \frac{1}{m-1}$  and  $n_i \leq m$  for  $i \in [1, s]$ , we see that it suffices to verify:

$$\frac{1}{8m^3s} + \frac{1}{32ms} + \frac{2(m+1)8^{m+1}}{(4m+4)^{3m-3}s} + \frac{1}{4m^2} \leq 1$$

The above inequality can easily be seen to be true for any  $m \geq 2$ ,  $s \geq 1$ .

Thus by Lemma [1](#), with positive probability, none of the bad events occurs and hence there exists a  $(2, \mathcal{F})$ -coloring using  $O(d^{1+\frac{1}{m-1}})$  colors. This completes the proof of Theorem [1](#).

### 3 Relating $\chi_{j,\mathcal{F}}(G)$ and $\chi_o(G)$

We now prove Theorem [2](#) which relates oriented chromatic number and the forbidden subgraph colorings.

**Proof of Theorem [2](#).** Let  $G = (V, E)$  be an undirected graph such that  $G \notin \text{Forb}(\mathcal{F})$  and let  $\vec{G} = (V, A)$  be an arbitrary orientation of  $E(G)$ . Since  $G \notin \text{Forb}(\mathcal{F})$ , we have  $k \geq \chi_{j,\mathcal{F}} \geq j + 1$ . Let  $V_1, \dots, V_k$  be the color classes of  $V$  with respect to a  $(j, \mathcal{F})$ -subgraph coloring  $c$  of  $V(G)$  using  $k$  colors. Let  $\mathcal{T}$  be the collection of subsets obtained by partitioning  $[1, k]$  into at most  $\lceil \frac{k}{\lfloor j/2 \rfloor} \rceil$  subsets of size at most  $\lfloor j/2 \rfloor$  each. Note that  $|\mathcal{T}|$  is at most  $\lceil \frac{2k}{j} \rceil$  if  $j$  is even and is at most  $\lceil \frac{2k}{j-1} \rceil$  if  $j$  is odd. Let  $\mathcal{S}$  be the collection defined by

$$\mathcal{S} = \{T \cup T' : T, T' \in \mathcal{T}, T \neq T'\}.$$

It follows that

- (i) Each  $S \in \mathcal{S}$  is a set of size at most  $j$ .
- (ii) for every  $l, m \in [1, k]$ , there exists a  $S \in \mathcal{S}$  with  $l, m \in S$ ,
- (iii) for each  $i \in [k]$ ,  $i$  is a member of at most  $\lceil \frac{k}{\lfloor j/2 \rfloor} \rceil - 1$  sets in  $\mathcal{S}$ . Let  $\mathcal{S}_i$  be defined by  $\mathcal{S}_i = \{S \in \mathcal{S} : i \in S\}$ .

For each  $S \in \mathcal{S}$ , let  $\vec{G}_S$  denote the induced subgraph  $\vec{G}[\cup_{i \in S} V_i]$ . Clearly  $G_S \in \text{Forb}(\mathcal{F})$ , since  $(V_1, \dots, V_k)$  is a  $(j, \mathcal{F})$ -subgraph coloring.

Let  $c_S$  be an oriented coloring of  $\vec{G}_S$  using at most  $t$  colors.

Assume an ordering  $\{S_1, S_2, \dots\}$  on the members of  $\mathcal{S}$ . We now define a new coloring  $\phi$  of  $V(G)$ : Fix any  $i$  and let  $\mathcal{S}_i = \{S_{i_1}, \dots, S_{i_l}\}$  be the members of  $\mathcal{S}_i$  where we have  $l \leq \lceil \frac{k}{\lfloor j/2 \rfloor} \rceil - 1$ . For each  $v \in V_i$ ,

$$\phi(v) = (c(v), c_{S_{i_1}}(v), \dots, c_{S_{i_l}}(v)).$$

Clearly,  $\phi$  is a proper coloring of  $V(\vec{G})$  because of the component  $c$ . We now prove that it is an oriented coloring. If it is not an oriented coloring, then there are four vertices  $x, y, z, t$  of  $\vec{G}$  such that  $(x, y) \in A$  and  $(z, t) \in A$  with  $\phi(x) = \phi(t)$  and  $\phi(y) = \phi(z)$ . By the definition of  $\phi$ ,  $x$  and  $t$  (respectively  $y$  and  $z$ ) belong to the same  $V_i$  (respectively  $V_j$ ) where  $i = c(x) = c(t)$  and  $j = c(y) = c(z)$ . Let  $S$  be

any set in  $\mathcal{S}$  containing  $i$  and  $j$  where  $S \in \mathcal{S}_i \cap \mathcal{S}_j$  and  $x, y, z, t \in V(\vec{G}_s)$ . By the definition of  $\phi$ , we have  $c_S(x) = c_S(t)$  and  $c_S(y) = c_S(z)$ . But this contradicts the fact that  $c_S$  is an oriented coloring of  $\vec{G}_S$ .

The number of possible values of  $\phi(v)$  is at most  $kt^{\lceil \frac{k}{\lfloor j/2 \rfloor} \rceil - 1}$ . This number is  $kt^{\lceil \frac{2k-j}{j} \rceil}$  if  $j$  is even and is  $kt^{\lceil \frac{2k-j+1}{j-1} \rceil}$  if  $j$  is odd. This proves Theorem 2.

## 4 (2, $\mathcal{F}$ )-Subgraph Colorings of Graphs on Surfaces

By applying the bound of Theorem 1 which holds for general graphs, we obtain a bound on  $\chi_{2,\mathcal{F}}(G)$  for graphs embeddable on surfaces, provided the members of  $\mathcal{F}$  have minimum degree at least 2. This bound was stated in Theorem 4 and is proved in this section.

The proof is essentially the proof of 2 extended to a more general setting. Hence, we only provide the sketch of the proof.

### 4.1 Proof of Theorem 4

We follow the proof of 2. Assume the theorem is false for a surface  $S$  with Euler characteristic  $-\gamma \leq 0$ , and let  $G$  be a graph embeddable on it, with a minimum number of vertices, which is a minimal counterexample to the theorem. Let  $H$  be  $G$  with (possibly multiple) edges added to triangulate  $S$ . Clearly  $deg_G(v) \leq deg_H(v)$  for all vertices  $v$  of  $G$ . Suppose  $V(G) = V(H) = \{v_1, \dots, v_n\}$ , where  $deg_H(v_1) \leq deg_H(v_2) \leq \dots \leq deg_H(v_n)$ . If  $\gamma = 0$ , define  $h_1 = 0$  and  $h_2 = 0$ . Otherwise, define  $h_1 := \lceil c\gamma^{\frac{m}{2m-1}} \rceil$  and  $h_2 := \lfloor 6\gamma/h_1 \rfloor (\leq 6\gamma^{\frac{m-1}{2m-1}}/c)$ , where  $c$  is an absolute constant, to be chosen later. Let  $d := deg(v_{n-h_1})$ . The proof will split on the size of  $d$ .

**Case I:**  $d \leq (4/3)h_2 + 9$ . In this case, the induced subgraph of  $G$  on  $\{v_1, \dots, v_n\}$  has maximum degree at most  $d$ , and thus has a  $(2, \mathcal{F})$ -subgraph coloring using at most  $\lceil Cd^{m/(m-1)} \rceil$  colors, by Theorem 1. Coloring the remaining vertices of  $G$  with  $h_1$  new colors that have not been used before gives a  $(2, \mathcal{F})$ -subgraph coloring of  $G$  with at most

$$\lceil C((4/3)h_2 + 9)^{m/(m-1)} \rceil + h_1 \leq C(8\gamma^{\frac{(m-1)}{(2m-1)}}/c + 9)^{m/(m-1)} + c\gamma^{m/(2m-1)} + 2$$

colors. An appropriate choice of constant values (independent of  $\gamma$ ) for  $A, B$  and  $c$  shows that this is smaller than  $A\gamma^{m/(2m-1)} + B$ , implying that in this case  $G$  cannot be a counterexample.

**Case II:**  $d \geq (4/3)h_2 + (28/3)$ . We charge each vertex as follows. Define  $charge'(v_i) = 6 - deg_H(v_i)$  for  $1 \leq i \leq n - h_1$ , and  $charge'(v_i) = -deg_H(v_i)/4$  for  $n - h_1 + 1 \leq i \leq n$ .

In this case, using the discharging method and an inductive argument explained in 2, it can be shown that there exists a vertex  $v$  such that (i)  $G - v$  is  $(2, \mathcal{F})$ -subgraph colorable using  $A\gamma^{m/(2m-1)} + B$  colors, (ii)  $v$  can be properly colored with a color  $i$  so that for any other color  $j$ , any connected component

containing  $v$  in the union of color classes  $i$  and  $j$  must have a vertex of degree one. Thus a  $(2, \mathcal{F})$ -subgraph coloring of  $G - v$  can be extended to a  $(2, \mathcal{F})$ -subgraph coloring of  $G$  using no additional color. This is a contradiction to our assumption that  $G$  is a minimum counter example. This completes the proof.

### 4.2 Proof of Theorem 5

The proof is based on an approach similar to the one used in [2]. It uses the following lemma whose proof follows from the proof of a lower bound presented in [1].

**Lemma 2.** *Let  $\mathcal{F}$  and  $m$  be as described in Theorem 5. Let  $G = G(n, p)$  be the random graph on  $\{1, \dots, n\}$  where each potential edge is chosen independently with probability  $p = c \left(\frac{\log n}{n}\right)^{1/m}$  for suitable positive constant  $c$  which depends only on  $\mathcal{F}$ . Then, almost surely,  $G$  is connected and has at most  $cn^{\frac{(2m-1)}{m}}(\log n)^{\frac{1}{m}}$  edges and satisfies  $\chi_{2,\mathcal{F}}(G) = \Omega(n)$ .*

Let  $G$  be a connected graph on at most  $O(n^{(2m-1)/m}(\log n)^{1/m})$  edges and satisfying  $\chi_{2,\mathcal{F}}(G) = \Omega(n)$  guaranteed by Lemma 2. Let  $G$  be embedded on a surface of characteristic  $-\gamma$  for the smallest  $\gamma \geq 0$  possible. Let  $e = |E(G)|$ . By an application of Euler’s formula, one can show (as shown in [2]) that  $\gamma > n^{(2m-1)/m}$ , and hence  $\log \gamma > (2m - 1)(\log n)/m$  and also that  $\gamma = O\left(n^{\frac{(2m-1)}{m}}(\log \gamma)^{\frac{1}{m}}\right)$ . Hence,  $\chi_{2,\mathcal{F}}(G) = \Omega(n) = \Omega\left(\gamma^{\frac{m}{(2m-1)}}/(\log \gamma)^{\frac{1}{(2m-1)}}\right)$ .

## 5 An Improved Bound on the Oriented Chromatic Number

In [3], Kostochka, Sopena and Zhu showed that the oriented chromatic number of any graph  $G$  of maximum degree  $k$  is at most  $2k^2 2^k$ . They prove this result using probabilistic arguments which can in fact be refined so that we obtain the following improvement of this result.

**Theorem 7.** *If  $G$  is any graph of maximum degree  $k$  and degeneracy  $d$ , then its oriented chromatic number  $\chi_o(G)$  is at most  $16kd2^d$ .*

This replaces a factor  $k2^k$  by  $d2^d$  and will result in a better bound for those  $G$  having  $d \ll k$ .

As in [3], we prove (using probabilistic arguments) the following lemma. Before that, we recall the following notation from [3]. For an oriented graph  $G = (V, A)$  and a subset  $I = \{x_1, \dots, x_i\}$  of  $V$  and a vertex  $v \in V \setminus I$  such that  $v$  is adjacent to each  $x_j$ , we use  $F(I, v, G)$  to denote the vector  $a = (a_1, \dots, a_i)$  where, for each  $j \leq i$ ,  $a_j = 1$  if  $(x_j, v) \in A$  and  $a_j = -1$  if  $(v, x_j) \in A$ .

**Lemma 3.** *Let  $d, k$  be positive integers with  $d \leq k$  and  $k \geq 5$ . There exists a tournament  $T = (V, A)$  on  $t = 16kd2^d$  vertices with the following property:*

For each  $i, 0 \leq i \leq d$ , for each  $I \subseteq V, |I| = i$ , and for each  $a \in \{1, -1\}^i$ , there exist at least  $kd + 1$  vertices  $v \in V \setminus I$  with  $F(I, v, T) = a$ .

We now give the proof of Theorem 7 assuming Lemma 3 whose proof is omitted due to lack of space.

**Proof of Theorem 7.** Let  $G = (V, E)$  be any graph of maximum degree  $k$  and degeneracy  $d$ . If  $d \leq 1$ , then  $G$  is a forest and hence its  $\chi_o(G) \leq 3$  as shown in 6. For  $d \geq 2$  and  $k \leq 4$ , the result follows from a bound of  $(2k - 1)2^{2k-2}$  derived in 6. Hence, we assume that  $k \geq 5$  and  $d \geq 2$ . Consider a linear ordering  $(v_n, \dots, v_1)$  of  $V$  such that for each  $i \leq n, v_i$  has at most  $d$  neighbors in the subgraph  $G_i$  induced by  $V_i = \{v_1, \dots, v_i\}$ . Let  $T$  be the tournament on  $t = 16kd2^d$  vertices specified in Lemma 3. Let  $G'$  be any orientation of  $G$ . We inductively color vertices of  $G'$  in the order  $(1, \dots, n)$  in such a way that after the coloration of the first  $m$  vertices:

- (1) The partial coloring  $f(v_1), \dots, f(v_m)$  is a valid oriented coloring of  $G'_m$  using vertices of  $T$ ;
- (2) For each  $v_j$  with  $j > m$ , all neighbors of  $v_j$  in  $V_m$  are colored with distinct colors.

Now, we need to color  $v_{m+1}$  so that (1) and (2) hold for  $f(v_{m+1})$  as well. For this, let  $\{y_1, \dots, y_i\} \subseteq V_m$  be the neighbors of  $v_{m+1}$  in  $V_m$  each colored with distinct colors (because of (2)) from  $I = \{f(y_1), \dots, f(y_i)\}$ . Note that  $i \leq d$ . Let  $a = F(\{y_1, \dots, y_i\}, v_{m+1}, G'_{m+1})$ . Let  $K = \{w \in V(T) \setminus I : F(I, w, T) = a\}$ . By Lemma 3, we know that  $|K| \geq kd + 1$ . Now, there can be at most  $kd$  paths of the form  $(v_{m+1}, u, v_j)$  such that  $u \in V \setminus V_{m+1}$  is a neighbor of  $v_{m+1}$  in  $G$  and  $v_j, j \leq m$  is a neighbor of  $u$  in  $V_m$ . Let  $B \subseteq V_m$  be the set of all such  $v_j$ 's and let  $f(B)$  be the set of their colors with  $|f(B)| \leq kd$ . Now, color  $v_{m+1}$  with any color from  $K \setminus f(B)$  and one can easily check that  $f(v_{m+1})$  satisfies both (1) and (2), thus extending the coloring inductively. This proves Theorem 7.

## 6 Conclusions and Open Problems

We showed a relation between forbidden subgraph colorings and oriented colorings. In particular, we obtained an upper bound for oriented chromatic number in terms of low treewidth colorings and found an upper bound of  $O(2^{g^{1/2+o(1)}})$  for the oriented chromatic number of graphs of genus  $g$ . However, we believe that this bound is not tight. In fact, we believe in the following

**Conjecture:** There exist absolute positive constants  $c_1, c_2$  such that : if  $G$  is a graph of genus at most  $g$ , then  $\chi_o(G) \leq c_1 2^{c_2 \sqrt{g}}$ .

Further, it would be interesting to obtain bounds for the  $(j, k)$ -treewidth chromatic number (for graphs of bounded genus), when  $j > 2$ . We also pose the following interesting and challenging open problem.

**Open Problem:** Determine if there is a  $k$  such that  $\chi_{2,k}^{tw}(G) \leq 4$  for all planar graphs  $G$  and find the smallest such  $k$  if it exists.

Note that if we replace 4 by 5 in the above inequality, then the answer is yes for  $k = 1$  since it has been shown by Borodin [11] that  $a(G) \leq 5$  for any planar graph  $G$ . Also, this bound is tight as Grünbaum [4] has obtained an infinite family of planar graphs having no acyclic 4-coloring.

## References

1. Aravind, N.R., Subramanian, C.R.: Bounds on proper colorings with restrictions on the union of color classes. Submitted to a Journal
2. Alon, N., Mohar, B., Sanders, D.P.: On acyclic colorings of graphs on surfaces. *Israel Journal of Mathematics* 94, 273–283 (1996)
3. Kostochka, A.V., Sopena, E., Zhu, X.: Acyclic and oriented chromatic numbers of graphs. *Journal of Graph Theory* 24(4), 331–340 (1997)
4. Grünbaum, B.: Acyclic colorings of planar graphs. *Israel Journal of Mathematics* 14, 390–408 (1973)
5. Albertson, M.O., Berman, D.M.: The acyclic chromatic number. *Congr. Numer.* 17, 51–60 (1976)
6. Sopena, E.: The chromatic number of oriented graphs. *Journal of Graph Theory* 25(2), 191–205 (1997)
7. Raspaud, A., Sopena, E.: Good and semi-strong colorings of oriented planar graphs. *Inf. Process. Lett.* 51(4), 171–174 (1994)
8. Nešetřil, J., de Mendez, P.O.: Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.* 27(6), 1022–1041 (2006)
9. Alon, N., McDiarmid, C., Reed, B.: Acyclic coloring of graphs. *Random Struct. Algorithms* 2(3), 277–288 (1991)
10. Ringel, G., Youngs, J.W.T.: Solution of the heawood map coloring problem. *Proc. Nat. Acad. Sci. U.S.A.* 60, 438–445 (1968)
11. Borodin, O.V.: Acyclic colorings of planar graphs. *Discrete Mathematics* 25(3), 211–236 (1979)

# Polynomial Kernels for 3-Leaf Power Graph Modification Problems<sup>\*</sup>

Stéphane Bessy<sup>1</sup>, Christophe Paul<sup>2</sup>, and Anthony Perez<sup>1</sup>

<sup>1</sup> LIRMM - Université Montpellier 2, France

<sup>2</sup> CNRS, LIRMM - Université Montpellier 2, France

**Abstract.** A graph  $G = (V, E)$  is a 3-leaf power iff there exists a tree  $T$  the leaf set of which is  $V$  and such that  $(u, v) \in E$  iff  $u$  and  $v$  are at distance at most 3 in  $T$ . The 3-leaf power edge modification problems, *i.e.* edition (also known as the CLOSEST 3-LEAF POWER), completion and edge-deletion are FPT when parameterized by the size of the edge set modification. However, a polynomial kernel was known for none of these three problems. For each of them, we provide a kernel with  $O(k^3)$  vertices that can be computed in linear time. We thereby answer an open question first mentioned by Dom, Guo, Hüffner and Niedermeier [9].

## 1 Introduction

The combinatorial analysis of experimental data-sets naturally leads to graph modification problems. For example, extracting a threshold graph from a dissimilarity on a set is a classical technique used in clustering and data analysis to move from a numerical to a combinatorial data-set [11,17]. The edge set of the threshold graph aims at representing the pairs of elements which are close to each another. As the dissimilarity reflects some experimental measures, the edge set of the threshold graph may reflect some false positive or negative errors. So for the sake of cluster identification, the edge set of the threshold graph has to be edited in order to obtain a disjoint union of cliques. This problem, known as CLUSTER EDITING, is fixed-parameter tractable (see e.g. [12,14,25]) and efficient parameterized algorithms have been proposed to solve biological instances with about 1000 vertices and several thousand edge modifications [2,6].

The (PROPER) INTERVAL GRAPH COMPLETION problem is another example of graph modification problem which arises in the context of molecular biology for the DNA physical mapping problem [13,18]. So, motivated by the identification of some hidden combinatorial structures on experimental data-sets, edge-modification problems cover a broad range of classical graph optimization problems, among which *completion* problems, *edition* problems and *edge-deletion* problems (see [21] for a recent survey). Though most of the edge-modification problems turn out to be NP-hard problems, efficient algorithms can be obtained to solve the natural parameterized version of some of them. Indeed, as long as the

---

<sup>\*</sup> Work supported by the French research grant ANR-06-BLAN-0148-01 "Graph Decomposition and Algorithms - GRAAL".



number  $k$  of errors generated by the experimental process is not too large, one can afford a time complexity exponential in  $k$ . A problem is *fixed parameterized tractable* (FPT for short) with respect to parameter  $k$  [11,22] whenever it can be solved in time  $f(k) \cdot n^{O(1)}$ , where  $f(k)$  is an arbitrary computable function. The reduction to a polynomial kernel is one of the most efficient fixed parameterized algorithmic techniques. The idea is to preprocess the input in order to reduce its size while preserving the existence of a solution so that the size of the reduced instance is bounded by a polynomial in the parameter  $k$ . More formally, a problem is *kernelizable* if any instance  $(G, k)$  can be reduced in polynomial time (using *reduction rules*) into an instance  $(G', k')$  such that  $k' \leq k$  and the size of  $G'$  is bounded by a function of  $k$ . Clearly having a kernel of small size is highly desirable and is an important issue in the context of applications [6,16].

This paper follows this line of research and studies the kernelization of edge-modification problems related to the family of *leaf powers*, graphs arising from a phylogenetic reconstruction context [19,20,23]. The goal is to extract, from a threshold graph  $G$  on a set  $S$  of species, a tree  $T$ , whose leaf set is  $S$  and such that the distance between two species is at most  $p$  in  $T$  iff they are adjacent in  $G$  ( $p$  being the value used to extract  $G$  from dissimilarity information). If such a tree  $T$  exists, then  $G$  is a *p-leaf power* and  $T$  is its *p-leaf root*. For  $p \leq 5$ , the *p-leaf power* recognition is polynomial time solvable [3,5], whereas the question is still open for  $p$  strictly larger than 5. Parameterized *p-leaf power* edge modification problems have been studied so far for  $p \leq 4$ . The edition problem for  $p = 2$  is known as the CLUSTER EDITING problem for which the kernel size bound has been successively improved in a series of recent papers [12,14,24], culminating in [15] with a kernel with  $4k$  vertices. For larger values of  $p$ , the edition problem is known as the CLOSEST *p*-LEAF POWER problem. For  $p = 3$  and 4, the CLOSEST *p*-LEAF POWER problem is known to be FPT [7,9], while its fixed parameterized tractability is still open for larger values of  $p$ . However, the existence of a polynomial kernel for  $p \neq 2$  remained an open question [8,10]. Though the completion and edge-deletion problems are FPT for  $p \leq 4$  [9,10], no polynomial kernel is known for  $p \neq 2$  [15].

**Our Results.** We prove that the CLOSEST 3-LEAF POWER, the 3-LEAF POWER COMPLETION and the 3-LEAF POWER EDGE-DELETION problems admit a kernel with  $O(k^3)$  vertices. We thereby answer positively to the open question of Dom, Guo, Hüffner and Niedermeier [9,10].

## 2 Preliminaries

The graphs we consider in this paper are undirected and loopless. The vertex set of a graph  $G$  is denoted by  $V(G)$ , with  $|V(G)| = n$ , and its edge set by  $E(G)$ , with  $|E(G)| = m$ . The *open* neighborhood of a vertex  $x$  is denoted by  $N_G(x)$  and its *closed* neighborhood  $N_G(x) \cup \{x\}$  by  $N_G[x]$ . We write  $d_G(u, v)$  for the distance between two vertices  $u$  and  $v$  in  $G$  (in the notations, the reference to the graph  $G$  will be omitted when the context is clear). For a subset  $S \subseteq V$ ,  $d_S(u, v)$  denotes the distance between  $u$  and  $v$  within  $S$ . Two vertices  $x$  and  $y$

of  $G$  are *true twins* if  $N[x] = N[y]$ . A *module* is a set of vertices  $S$  such that for any distinct vertices  $x$  and  $y$  of  $S$ ,  $N(x) \setminus S = N(y) \setminus S$ . The subgraph induced by a subset  $S$  of vertices is denoted  $G[S]$ . If  $H$  is a subgraph of  $G$ ,  $G \setminus H$  stands for  $G[V(G) \setminus V(H)]$ . A graph family  $\mathcal{F}$  is *hereditary* if for any graph  $G \in \mathcal{F}$ , any induced subgraph  $H$  of  $G$  also belongs to  $\mathcal{F}$ .

As the paper deals with undirected graphs, we abusively denote by  $X \times Y$  the set of unordered pairs containing one element of  $X$  and one of  $Y$ . Let  $G = (V, E)$  be a graph and  $F$  be a subset of  $V \times V$ ,  $G + F$  is the graph on vertex set  $V$ , the edge set of which is  $E \Delta F$  (the symmetric difference between  $E$  and  $F$ ). Such a set  $F$  is called an *edition* of  $G$  (we may also abusively say that  $G + F$  is an edition). A vertex  $v \in V$  is *affected* by an edition  $F$  whenever  $F$  contains an edge incident to  $v$ . Given a graph family  $\mathcal{F}$  and given a graph  $G = (V, E)$ , a subset  $F \subseteq V \times V$  is an *optimal  $\mathcal{F}$ -edition* of  $G$  if  $F$  is a set of minimum cardinality such that  $G + F \in \mathcal{F}$ . If we constrain  $F$  to be disjoint from  $E$ , we say that  $F$  is a *completion*, whereas if  $F$  is asked to be a subset of  $E$ , then  $F$  is an *edge deletion*. The problem we mainly consider is thus the following:

PARAMETERIZED CLOSEST 3-LEAF POWER:

**Input** : A graph  $G = (V, E)$ .

**Parameter** :  $k \in \mathbb{N}$ .

**Question** : Is there a 3-leaf power edition  $F$  of  $G$  such that  $|F| \leq k$  ?

If we replace edition by deletion (resp.completion), we get the PARAMETERIZED 3-LEAF POWER EDGE-DELETION (resp. PARAMETERIZED 3-LEAF POWER COMPLETION) problem.

## 2.1 Critical Cliques

The notions of critical clique and critical clique graph, introduced in [20], have been recently successfully used in problems such as CLUSTER EDITING [15] and BICLUSTER EDITING [24].

**Definition 1.** A critical clique of a graph  $G$  is a clique  $K$  which is a module and is maximal under this property.

It follows from definition that the set  $\mathcal{K}(G)$  of critical cliques of a graph  $G$  defines a partition of its vertex set  $V$ .

**Definition 2.** Given a graph  $G = (V, E)$ , its critical clique graph  $\mathcal{C}(G)$  has vertex set  $\mathcal{K}(G)$  and edge set  $E(\mathcal{C}(G))$  with

$$(K, K') \in E(\mathcal{C}(G)) \Leftrightarrow \forall v \in K, v' \in K', (v, v') \in E(G)$$

The following lemma was used in the construction of polynomial kernels for CLUSTER EDITING and BICLUSTER EDITING problems in [24].

**Lemma 1.** Let  $G = (V, E)$  be a graph. If  $H$  is the graph  $G + \{(u, v)\}$  with  $(u, v) \in V \times V$ , then  $|\mathcal{K}(H)| \leq |\mathcal{K}(G)| + 4$ .

We now generalize a result used to obtain FPT algorithms for the CLOSEST 3-LEAF POWER problem [7]. A graph family  $\mathcal{F}$  is said to be closed under *twin addition* if for any graph  $G \in \mathcal{F}$ , adding a twin of any of its vertices yields a graph of  $\mathcal{F}$ .

**Lemma 2.** *Let  $\mathcal{F}$  be an hereditary graph family closed under true twin addition. For any graph  $G = (V, E)$ , there exists an optimal  $\mathcal{F}$ -edition (resp.  $\mathcal{F}$ -deletion,  $\mathcal{F}$ -completion)  $F$  such that any critical clique of  $G + F$  is the disjoint union of a subset of critical cliques of  $G$ .*

In particular, this means that one can find an optimal solution that does not delete any edges within a critical clique. Furthermore, in this optimal solution, either all or no edges are added or deleted between two critical cliques. From now on, every considered optimal edition (resp. deletion, completion) is supposed to verify these two properties.

## 2.2 Leaf Powers

**Definition 3.** *Let  $T$  be an unrooted tree whose leaves are one-to-one mapped to the elements of a set  $V$ . The  $k$ -leaf power of  $T$  is the graph  $T^k$ , with  $T^k = (V, E)$  where  $E = \{(u, v) \mid u, v \in V \text{ and } d_T(u, v) \leq k\}$ . We call  $T$  a  $k$ -leaf root of  $T^k$ .*

It is easy to see that for any  $k$ , the  $k$ -leaf power family of graphs satisfies the conditions of Lemma 2. In this paper we focus on the class of 3-leaf powers for which several characterizations are known, one of which propose a list of forbidden induced subgraphs [8]. The proofs of our kernel for the CLOSEST 3-LEAF POWER problem (or 3-LEAF POWER EDITING) rely on the well-known critical clique graph characterization and on a new one which is based on the join composition of graphs.

**Theorem 1.** [7] *A graph  $G$  is a 3-leaf power iff its critical clique graph  $\mathcal{C}(G)$  is a forest.*

Let  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$  be two disjoint graphs and let  $S_1 \subseteq V_1$  and  $S_2 \subseteq V_2$  be two non-empty subsets of vertices. The *join composition* of  $G_1$  and  $G_2$  on  $S_1$  and  $S_2$ , denoted  $(G_1, S_1) \otimes (G_2, S_2)$ , results in the graph  $H = (V_1 \cup V_2, E_1 \cup E_2 \cup (V(S_1) \times V(S_2)))$ .

**Theorem 2.** *Let  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$  be two connected 3-leaf powers. The graph  $H = (G_1, S_1) \otimes (G_2, S_2)$ , with  $S_1 \subseteq V_1$  and  $S_2 \subseteq V_2$ , is a 3-leaf power if and only if one of the following conditions holds:*

1.  $S_1$  and  $S_2$  are two cliques of  $G_1$  and  $G_2$  respectively, and if  $S_1$  (resp.  $S_2$ ) is not critical, then  $G_1$  (resp.  $G_2$ ) is a clique or,
2. there exists  $v \in V_1$  such that  $S_1 = N[v]$  and  $S_2 = V_2$  is a clique.

The following observation will help proving reduction rules.

**Observation 3.** *Let  $C$  be a critical clique of a 3-leaf power  $G = (V, E)$ . For any  $S \subseteq V$ , if the clique  $C \setminus S$  is not critical in  $G[V \setminus S]$ , then the connected component of  $G[V \setminus S]$  containing  $C \setminus S$  is a clique.*

### 3 A Cubic Kernel for CLOSEST 3-LEAF POWER Problem

In this section, we present five preprocessing reduction rules the application of which leads to a kernel with  $O(k^3)$  vertices for the CLOSEST 3-LEAF POWER problem. We first give two simple reduction rules; note that the second one was already used to obtain a kernel with  $O(k^2)$  vertices for the parameterized CLUSTER EDITING problem [24].

**Rule 1.** *If  $G$  has a connected component  $C$  such that  $G[C]$  is 3-leaf power, then remove  $C$  from  $G$ .*

**Rule 2.** *If  $G$  has a critical clique  $K$  such that  $|K| > k+1$ , then remove  $|K|-k-1$  vertices of  $K$  from  $V(G)$ .*

#### 3.1 Branch Reduction Rules

We now assume that the input graph  $G$  is reduced under Rule 1 and Rule 2. The next three reduction rules use the fact that the critical clique graph of a 3-leaf power is a forest. The idea is to identify induced subgraphs of  $G$ , called *branches*, which correspond to subtrees of  $\mathcal{C}(G)$ . Such a subgraph is already a 3-leaf power.

**Definition 4.** *Let  $G = (V, E)$  be a graph. An induced subgraph  $G[S]$ , with  $S \subseteq V$ , is a branch if  $S$  is the union of critical cliques  $K_1, \dots, K_r \in \mathcal{C}(G)$  such that the subgraph of  $\mathcal{C}(G)$  induced by  $\{K_1, \dots, K_r\}$  is a tree.*

Let  $B = G[S]$  be a branch of a graph  $G$  and let  $K_1, \dots, K_r$  be the critical cliques of  $G$  contained in  $S$ . We say that  $K_i$  ( $1 \leq i \leq r$ ) is an *attachment point* of the branch  $B$  if it contains a vertex  $x$  such that  $N_G(x)$  intersects  $V(G) \setminus S$ . A branch  $B$  is a  *$l$ -branch* if it has  $l$  attachment points. Our next three rules deal with 1-branches and 2-branches. In the following, we denote by  $B^R$  the subgraph of  $B$  in which the vertices of the attachment points have been removed. For an attachment point  $P$  of  $B$ ,  $N_B(P)$  denotes the set of neighbors of vertices of  $P$  in  $B$ .

**Lemma 3.** *Let  $G = (V, E)$  be a graph and  $B$  be a 1-branch of  $G$  with attachment point  $P$ . There exists an optimal 3-leaf power edition  $F$  of  $G$  such that:*

1. *the set of affected vertices of  $B$  is a subset of  $P \cup N_B(P)$  and*
2. *in  $G + F$ , the vertices of  $N_B(P)$  are all adjacent to the same vertices of  $V(G) \setminus V(B^R)$ .*

*Proof.* Let  $F$  be an arbitrary optimal 3-leaf power edition of  $G$ . We construct from  $F$  another optimal 3-leaf power edition which satisfies the two conditions above. Let  $C$  be the critical clique of  $H = G + F$  that contains  $P$  and set  $C' = C \setminus B^R$ . By Lemma 2, the set of critical cliques of  $G$  whose vertices belong to  $N_B(P)$  contains two kind of cliques:  $K_1, \dots, K_c$ , whose vertices are in  $C$  or adjacent to the vertices of  $C$  in  $H$ , and  $K_{c+1}, \dots, K_h$  whose vertices are not

adjacent to the vertices of  $C$  in  $H$ . For  $i \in \{1, \dots, h\}$ , let  $C_i$  be the connected component of  $B^R$  containing  $K_i$ .

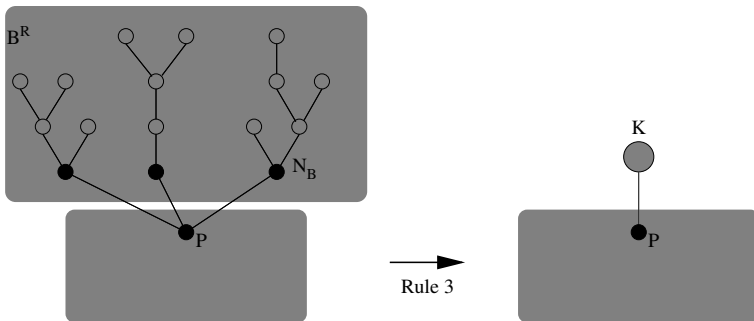
Let us consider the three following induced subgraphs :  $G_1$  the subgraph of  $G$  induced by the disjoint union of  $C_1, \dots, C_c$ ;  $G_2$  the subgraph of  $G$  induced by the disjoint union of  $C_{c+1}, \dots, C_h$ ; and finally  $G'$ , the subgraph of  $H$  induced by  $V(G) \setminus V(B^R)$ . Let us notice that these three graphs are 3-leaf powers. By Observation 3, if  $C'$  is not a critical clique of  $G'$ , then the connected component of  $G'$  containing  $C'$  is a clique. Similarly, if  $K_i$ , for any  $1 \leq i \leq c$ , is not a critical clique of  $G_1$ , then the connected component of  $G_1$  containing  $K_i$  is a clique. Thus, by Theorem 2, the disjoint union  $H'$  of  $G_2$  and  $(G', C') \otimes (G_1, \{K_1, \dots, K_c\})$  is a 3-leaf power. By construction, the edge edition set  $F'$  such that  $H' = G + F'$  is a subset of  $F$  and thus  $|F'| \leq |F|$ . Moreover, the vertices of  $B$  affected by  $F'$  all belong to  $P \cup N_B(P)$ , which proves the first point.

To state the second point, we focus on the relationship between the critical cliques  $K_i$  and  $C'$  in  $H' = G + F'$ . If some  $K_i$  is linked to  $C'$  in  $H'$  (i.e.  $c > 1$ ), it means that the cost of adding the missing edges between  $K_i$  and  $C'$  (which, by Theorem 2, would also result in a 3-leaf power) is lower than the cost of removing the existing edges between  $K_i$  and  $C'$ :  $|K_i| \cdot |C' \setminus P| \leq |K_i| \cdot |P|$ . On the other hand, if some  $K_j$  is not linked to  $C'$  in  $H'$  (i.e.  $c < h$ ), we conclude that  $|P| \leq |C' \setminus P|$ . Finally, if both cases occur, we have  $|P| = |C' \setminus P|$ , and we can choose to add all or none of the edges between  $K_i$  and  $C'$ . In all cases, we provide an optimal edition of  $G$  into a 3-leaf power in which the vertices of  $N_B(P)$  are all adjacent to the same vertices of  $V(G) \setminus V(B^R)$ .  $\square$

The safeness of the first 1-branch reduction rule follows from Lemma 3.

**Rule 3.** *If  $G$  contains a 1-branch  $B$  with attachment point  $P$ , then remove from  $G$  the vertices of  $B^R$  and add a new critical clique of size  $\min\{|N_B(P)|, k + 1\}$  adjacent to  $P$ .*

Our second 1-branch reduction rule considers the case where several 1-branches are attached to the rest of the graph by a join. The following lemma shows that



**Fig. 1.** On the left, a 1-branch  $B$  with attachment point  $P$ . On the right, the effect of Rule 3 which replaces  $B^R$  by a clique  $K$  of size  $\min\{|N_B(P)|, k + 1\}$ .

under some cardinality conditions, the vertices of such 1-branches are not affected by an optimal 3-leaf power edition.

**Lemma 4.** *Let  $G = (V, E)$  be a graph for which a 3-leaf power edition of size at most  $k$  exists. Let  $B_1, \dots, B_l$  be 1-branches, the attachment points  $P_1, \dots, P_l$  of which all have the same neighborhood  $N$  in  $V \setminus \cup_{i=1}^l V(B_i)$ . If  $\sum_{i=1}^l |P_i| > 2k + 1$ , then there is no optimal 3-leaf power edition  $F$  of  $G$  that affects vertices of  $\cup_{i=1}^l V(B_i)$ .*

By Lemma 4, if there exists a 3-leaf power edition  $F$  of  $G$  such that  $|F| \leq k$ , then the 1-branches  $B_1, \dots, B_l$  can be safely replaced by two critical cliques of size  $k + 1$ . This gives us the second 1-branch reduction rule.

**Rule 4.** *If  $G$  has several 1-branches  $B_1, \dots, B_l$ , the attachment points  $P_1, \dots, P_l$  of which all have the same neighborhood  $N$  in  $V \setminus \cup_{i=1}^l V(B_i)$  and if  $\sum_{i=1}^l |P_i| > 2k + 1$ , then remove from  $G$  the vertices of  $\cup_{i=1}^l V(B_i)$  and add two new critical cliques of size  $k + 1$  neighboring exactly  $N$ .*

### 3.2 The 2-Branch Reduction Rule

To complete the set of reduction rules, we need to consider 2-branches. So let  $B$  be a 2-branch with attachment points  $P_1$  and  $P_2$ . The subgraph of  $G$  induced by the critical cliques of the unique path from  $P_1$  to  $P_2$  in  $\mathcal{C}(B)$  is called the *main path* of  $B$  and denoted  $path(B)$ . We say that  $B$  is *clean* if  $P_1$  and  $P_2$  are leaves of  $\mathcal{C}(B)$ , in which case we denote by  $Q_1$  (resp.  $Q_2$ ) the critical clique that neighbors  $P_1$  (resp.  $P_2$ ) in  $B$ .

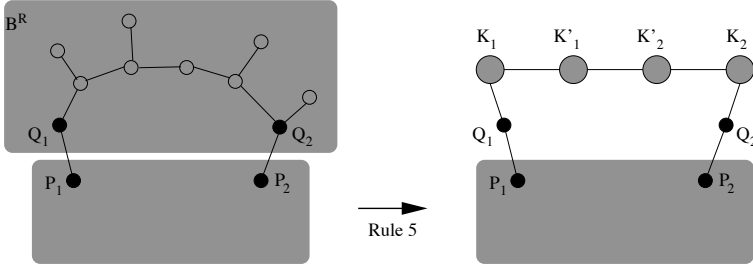
**Lemma 5.** *Let  $B$  be a clean 2-branch of a graph  $G = (V, E)$  with attachment points  $P_1$  and  $P_2$  such that  $path(B)$  contains at least 5 critical cliques. There exists an optimal 3-leaf power edition  $F$  of  $G$  which, if it affects vertices of  $B$  not in  $V(P_1 \cup Q_1 \cup P_2 \cup Q_2)$ , then it contains a min-cut of  $path(B)$ .*

**Rule 5.** *If  $G$  has a clean 2-branch  $B$  such that  $path(B)$  is composed by at least 5 critical cliques, then remove from  $G$  all the vertices of  $V(B)$  except those of  $V(P_1 \cup Q_1 \cup P_2 \cup Q_2)$  and add four new critical cliques:*

- $K_1$  (resp.  $K_2$ ) of size  $k + 1$  adjacent to  $Q_1$  (resp.  $Q_2$ );
- $K'_1$  (resp.  $K'_2$ ) adjacent to  $K_1$  (resp.  $K_2$ ) and such that  $K'_1$  and  $K'_2$  are adjacent and  $|K'_1| \cdot |K'_2|$  equals the min-cut of  $path(B)$ .

### 3.3 Kernel Size and Time Complexity

Let us discuss the time complexity of the reduction rules. The 3-leaf power recognition problem can be solved in  $O(n + m)$  time [4]. It follows that Rule 1 requires linear time. To implement the other reduction rules, we first need to compute the critical clique graph  $\mathcal{C}(G)$ , which, as noticed in [24], can be computed in linear time if we use modular decomposition algorithm (see [26] for a recent paper). Given  $\mathcal{C}(G)$ , which is linear in the size of  $G$ , it is easy to detect the critical



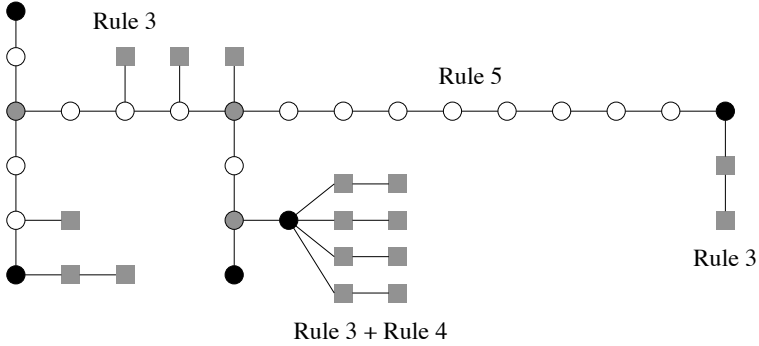
**Fig. 2.** A 2-branch  $B$  on the left (only pendant critical cliques are hanging on  $path(B)$  since we can assume that the graph is reduced by the previous rules). On the right, the way Rule 5 reduces  $B$ .

cliques of size at least  $k + 1$ . So, Rule 2 requires linear time. A search on  $\mathcal{C}(G)$  can identify the 1-branches. It follows that the two 1-branches reduction rules (Rule 3 and Rule 4) can also be applied in  $O(n+m)$  time. Let us now notice that in a graph reduced by the first four reduction rules, a 2-branch is a path to which pendant vertices are possibly attached. It follows that to detect a 2-branch  $B$ , such that  $path(B)$  contains at least 5 critical cliques, we first prune the pendant vertices, and then identify in  $\mathcal{C}(G)$  the paths containing only vertices of degree 2, and at least 5 of them. To do this, we compute the connected components of the graph induced on vertices of degree 2 in  $\mathcal{C}(G)$ . This shows that Rule 5 can be carried out in linear time.

**Theorem 4.** *The PARAMETERIZED CLOSEST 3-LEAF POWER problem admits a kernel with  $O(k^3)$  vertices. Given a graph  $G$ , a reduced instance can be computed in linear time.*

*Proof.* The discussion above established the time complexity to compute a kernel. Let us determine the kernel size. Let  $G = (V, E)$  be a reduced graph (i.e. none of the reduction rules applies to  $G$ ) which can be edited into a 3-leaf power with a set  $F \subseteq V \times V$  such that  $|F| \leq k$ . Let us denote  $H = G + F$  the edited graph. For the sake of simplicity, we assume that  $H$  is connected, and thus  $\mathcal{C}(H)$  is a tree. If  $\mathcal{C}(H)$  is a forest, one has to apply the following arguments to each of its connected component, and then to sum up. We first show that  $\mathcal{C}(H)$  has  $O(k^2)$  vertices (i.e.  $|\mathcal{C}(H)| \in O(k^2)$ ), and then Lemma 1 enables us to conclude.

We say that a critical clique is affected if it contains an affected vertex and denote by  $A$  the set of the affected critical cliques. As each edge of  $F$  affects two vertices, we have that  $|A| \leq 2k$ . Since  $H$  is a 3-leaf power, its critical clique graph  $\mathcal{C}(H)$  is a tree. Let  $T$  be the minimal subtree of  $\mathcal{C}(H)$  that spans the affected critical cliques. Let us observe that if  $B$  is a maximal subtree of  $\mathcal{C}(H) - T$ , then none of the critical cliques in  $B$  contains an affected vertex and thus  $B$  was the critical clique graph of a 1-branch of  $G$ , which has been reduced by Rule 3 or Rule 4. Let  $A' \subset \mathcal{K}(H)$  be the critical cliques of degree at least 3 in  $T$ . As



**Fig. 3.** The black circles are the critical cliques of  $A$ , the grey ones belong to  $A'$ , and the squares are the critical cliques not in  $T$ . On the figure, we can observe a 2-branch of size 8 reduced by Rule 5. Application of Rule 3 may let a path of two critical cliques pendant to the elements of  $A$  and a single critical clique pendant to the elements of the small 2-branches. Finally, Rule 4 can only affect critical cliques of  $A$ .

$|A| \leq 2k$ , we also have  $|A'| \leq 2k$ . The connected components resulting from the removal of  $A$  and  $A'$  in  $T$  are paths. There are at most  $4k$  such paths. Each of these paths is composed by non-affected critical cliques. It follows that each of them corresponds to  $path(B)$  for some 2-branch  $B$  of  $G$ , which has been reduced by Rule 5.

From these observations, we can now estimate the size of the reduced graph. Attached to each of the critical cliques of  $T \setminus A$ , we can have 1 pendant critical clique resulting from the application of Rule 3. Remark that any 2-branch reduced by Rule 5 has no such pendant clique and that  $path(B)$  contains 5 critical cliques. So, a considered 2-branch in  $\mathcal{C}(H)$  is made of at most 8 critical cliques. Finally, attached to each critical clique of  $A$ , we can have at most  $(4k + 2)$  extra critical cliques resulting from the application of Rule 4. See Figure 3 for an illustration of the shape of  $\mathcal{C}(H)$ . Summing up everything, we obtain that  $\mathcal{C}(H)$  contains at most  $4k \cdot 8 + 2k \cdot 2 + 2k \cdot (4k + 3) = 8k^2 + 42k$  critical cliques.

By Lemma 1 we know that for each edited edge in a graph the number of critical cliques increases by at most 4. It follows that  $\mathcal{K}(G)$  contains at most  $8k^2 + 46k$  critical cliques, each of size at most  $k + 1$  (Rule 2). Thus, the reduced graph contains at most  $8k^3 + 54k^2 + 46k$  vertices, proving the  $O(k^3)$  kernel size.  $\square$

### 4 Kernels for Edge Completion and Edge Deletion

We now explain and adapt the previous rules to the cases where only insertions or only deletions of edges are allowed. First, observe that Rules 1 and 2 are also safe for 3-LEAF POWER COMPLETION and 3-LEAF POWER EDGE-DELETION (Rule 2 directly follows from Lemma 2). The same holds for Rules 3 and 4. However, this is not the case for the 2-branch reduction rule (Rule 5), which is safe for 3-LEAF POWER EDGE-DELETION, but not for 3-LEAF POWER COMPLETION.



Nevertheless, in the latter case, the following lemma yields a rule specific to the 3-LEAF POWER COMPLETION.

**Lemma 6.** *Let  $G$  be a graph admitting a clean 2-branch  $B$  such that  $\text{path}(B)$  is composed by at least  $k + 4$  critical cliques. If  $P_1$  and  $P_2$  belong to the same connected component in  $G$ , then there is no 3-leaf power completion of size at most  $k$ .*

**Rule 6.** *Let  $G$  be a graph having a clean 2-branch  $B$  with attachment points  $P_1$  and  $P_2$  such that  $\text{path}(B)$  is composed by at least  $k + 4$  critical cliques.*

- *If  $P_1$  and  $P_2$  belong to the same connected component in  $G \setminus B^R$ , then there is no completion of size at most  $k$ .*
- *Otherwise, remove from  $G$  all the vertices of  $V(B)$  except those of  $P_1 \cup Q_1 \cup P_2 \cup Q_2$  and add all possible edges between  $Q_1$  and  $Q_2$ .*

Using Rules [3](#), [4](#) and [5](#) for deletion and Rules [3](#), [4](#) and [6](#) for completion, we obtain a kernel with  $O(k^3)$  vertices.

**Theorem 5.** *The PARAMETERIZED 3-LEAF POWER COMPLETION and PARAMETERIZED 3-LEAF POWER EDGE-DELETION problems admit kernels with  $O(k^3)$  vertices. Given a graph  $G$ , a reduced instance can be computed in linear time.*

## References

1. Barthélémy, J.-P., Guénoche, A.: Trees and proximity representations. John Wiley & Sons, Chichester (1991)
2. Böcker, S., Briesemeister, S., Klau, G.W.: Exact algorithms for cluster editing: evaluation and experiments. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 289–302. Springer, Heidelberg (2008)
3. Brandstädt, A., Le, V.B.: Structure and linear time recognition of 4-leaf powers. ACM Transactions on Algorithms (to appear)
4. Brandstädt, A., Le, V.B.: Structure and linear time recognition of 3-leaf powers. Information Processing Letters 98(4), 133–138 (2006)
5. Chang, M.-S., Ko, M.-T.: The 3-steiner root problem. In: Brandstädt, A., Kratsch, D., Müller, H. (eds.) WG 2007. LNCS, vol. 4769, pp. 109–120. Springer, Heidelberg (2007)
6. Dehne, F., Langston, M., Luo, X., Pitre, S., Shaw, P., Zhang, Y.: The cluster editing problem: implementations and experiments. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 13–24. Springer, Heidelberg (2006)
7. Dom, M., Guo, J., Hüffner, F., Niedermeier, R.: Error compensation in leaf root problems. Algorithmica 44, 363–381 (2006)
8. Dom, M., Guo, J., Hüffner, F., Niedermeier, R.: Error compensation in leaf root problems. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 389–401. Springer, Heidelberg (2004)
9. Dom, M., Guo, J., Hüffner, F., Niedermeier, R.: Extending the tractability border for closest leaf powers. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 397–408. Springer, Heidelberg (2005)

10. Dom, M., Guo, J., Hüffner, F., Niedermeier, R.: Closest 4-leaf power is fixed-parameter tractable. *Discrete Applied Mathematics* 156(18), 3345–3361 (2008)
11. Downey, R.G., Fellows, M.R.: *Parameterized complexity*. Springer, Heidelberg (1999)
12. Fellows, M.R., Langston, M., Rosamond, F., Shaw, P.: Efficient parameterized preprocessing for cluster editing. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) *FCT 2007*. LNCS, vol. 4639, pp. 312–321. Springer, Heidelberg (2007)
13. Goldberg, P.W., Golumbic, M.C., Kaplan, H., Shamir, R.: Four strikes against physical mapping of DNA. *Journal of Computational Biology* 2, 139–152 (1995)
14. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems* 38(4), 373–392 (2005)
15. Guo, J.: A more effective linear kernelization for cluster editing. In: Chen, B., Paterson, M., Zhang, G. (eds.) *ESCAPE 2007*. LNCS, vol. 4614, pp. 36–47. Springer, Heidelberg (2007)
16. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. *SIGACT News* 38(1), 31–45 (2007)
17. Hansen, P., Jaumard, B.: Cluster analysis and mathematical programming. *Math. Program.* 79(1-3), 191–215 (1997)
18. Kaplan, H., Shamir, R., Tarjan, R.E.: Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM J. Comput* 28, 780–791 (1999)
19. Kearney, P., Corneil, D.: Tree powers. *Journal of Algorithms* 29(1), 111–131 (1998)
20. Lin, G.H., Kearney, P.E., Jiang, T.: Phylogenetic  $k$ -root and steiner  $k$ -root. In: Lee, D.T., Teng, S.-H. (eds.) *ISAAC 2000*. LNCS, vol. 1969, pp. 539–551. Springer, Heidelberg (2000)
21. Natazon, A., Shamir, R., Sharan, R.: Complexity classification of some edge modification problems. *Discrete Applied Mathematics* 113, 109–128 (2001)
22. Niedermeier, R.: *Invitation to fixed parameter algorithms*. Oxford Lectures Series in Mathematics and its Applications, vol. 31. Oxford University Press, Oxford (2006)
23. Nishimura, N., Ragde, P., Thilikos, D.: On graph powers for leaf-labeled trees. *Journal of Algorithms* 42(1), 69–108 (2002)
24. Protti, F., Dantas da Silva, M., Szwarcfiter, J.L.: Applying modular decomposition to parameterized cluster editing problems. *Theory of Computing Systems* (2007)
25. Shamir, R., Sharan, R., Tsur, D.: Cluster graph modification problems. *Discrete Applied Mathematics* 144(1-2), 173–182 (2004)
26. Tedder, M., Corneil, D., Habib, M., Paul, C.: Simpler linear-time modular decomposition via recursive factorizing permutations. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) *ICALP 2008, Part I*. LNCS, vol. 5125, pp. 634–645. Springer, Heidelberg (2008)

# Approximating the Max Edge-Coloring Problem

Nicolas Bourgeois<sup>1</sup>, Giorgio Lucarelli<sup>2,\*</sup>, Ioannis Milis<sup>2</sup>,  
and Vangelis Th. Paschos<sup>1</sup>

<sup>1</sup> LAMSADE, Université Paris-Dauphine, France

`nbourgeo@phare.normalesup.org`, `paschos@lamsade.dauphine.fr`

<sup>2</sup> Dept. of Informatics, Athens University of Economics and Business, Greece  
`{gluc,milis}@aueb.gr`

**Abstract.** We study the weighted generalization of the edge coloring problem where the goal is to minimize the sum of the weights of the heaviest edges in the color classes. In particular, we deal with the approximability of this problem on bipartite graphs and trees. We first improve the best known approximation ratios for bipartite graphs of maximum degree  $\Delta \geq 7$ . For trees we present a polynomial  $3/2$ -approximation algorithm, which is the first one for any special graph class with an approximation ratio less than the known ratio of two for general graphs. Also for trees, we propose a moderately exponential approximation algorithm that improves the  $3/2$  ratio with running time much better than that needed for the computation of an optimal solution.

## 1 Introduction

In the *max edge-coloring* (MEC) problem we are given a graph  $G = (V, E)$  and a positive integer weight  $w(e)$ , for each edge  $e \in E$ , and we ask for a proper edge-coloring of  $G$ ,  $\mathcal{M} = \{M_1, M_2, \dots, M_k\}$ , such that the sum of the weights of the heaviest edges in the color classes (matchings),  $\sum_{i=1}^k \max_{e \in M_i} \{w(e)\}$ , is minimized.

Clearly, for unit edge weights the MEC problem reduces to the classical edge-coloring problem.

The MEC problem arises in optical communication systems (see for example [9]), where messages are to be transmitted directly from senders to receivers through connections established by an underlying switching network. Any node of such a system cannot participate in more than one transmissions at a time, while the transmission of messages between several pairs of nodes can take place simultaneously. The scheduler of such a system establishes successive configurations of the switching network, each one routing a non-conflicting subset of the

---

\* This work has been funded by the project PENED 2003. The project is cofinanced 75% of public expenditure through EC-European Social Fund, 25% of public expenditure through Ministry of Development-General Secretariat of Research and Technology of Greece and through private sector, under measure 8.3 of Operational Programme "Competitiveness" in the 3rd Community Support Programme.

messages from senders to receivers. Given the transmission time of each message, the transmission time of each configuration equals to the longest message transmitted. The aim is to find a sequence of configurations such that all the messages are transmitted and the total transmission time is minimized. It is easy to see that the above situation corresponds directly to the MEC problem.

**Related Work.** It is well known that the classical edge coloring problem is not approximable within a factor less than  $4/3$  for general graphs [8], unless  $P=NP$ , while it becomes polynomial for bipartite graphs [10]. On the other hand, the MEC problem is known to be not approximable within a factor less than  $7/6$  even for cubic planar bipartite graphs with edge weights  $w(e) \in \{1, 2, 3\}$ , unless  $P=NP$  [3]. The MEC problem is also known to be polynomial for a few special cases including bipartite graphs with edge weights  $w(e) \in \{1, 2\}$  [5], chains [7] (in fact, this algorithm can be also applied for graphs of  $\Delta = 2$ ), stars of chains and bounded degree trees [13].

Concerning the approximability of the MEC problem, a natural greedy 2-approximation algorithm for general graphs has been proposed in [9]. The ratio of this algorithm has been slightly improved to  $2 - \frac{1}{\Delta}$  and  $2 - \frac{2}{\Delta+2}$  in [12]. Especially for bipartite graphs of maximum degree  $\Delta = 3$  an algorithm that attains the  $7/6$  inapproximability bound has been presented in [3]. For bipartite graphs, algorithms have been also presented improving the  $2 - \frac{2}{\Delta+2}$  approximation ratio for general graphs. In fact, algorithms presented in [7] and [13] achieve better ratios for bipartite graphs of  $\Delta \leq 7$ , and  $\Delta \leq 12$ , respectively. However, the ratio of these algorithms exceeds two for larger values of  $\Delta$ . Moreover, two algorithms of approximation ratios  $2 - \frac{2}{\Delta+1}$  and  $\frac{2\Delta^3}{\Delta^3 + \Delta^2 + \Delta - 1}$ , which tend asymptotically to two as  $\Delta$  increases, have been presented in [12].

The analogous to the MEC problem weighted generalization of the classical vertex-coloring problem, known as *max vertex-coloring* (MVC) problem, has been also addressed in the literature [3,5,6,7,15,16]. The MVC problem is not approximable within a factor less than  $8/7$  even for planar bipartite graphs, unless  $P=NP$ , [5,15]. In addition, this bound has been attained for general bipartite graphs [3,15] and a PTAS for trees has been presented in [7,15].

Despite the recent research activity on both the MEC and the MVC problems, their complexity for trees is still unknown. Moreover, no algorithm of approximation ratio  $2 - \delta$ , for any small constant  $\delta > 0$ , for the MEC problem on bipartite graphs or even on trees is known.

**Our Results and Organization of the Paper.** In this paper we contribute on decreasing the above mentioned gap in the approximability of the MEC problem on bipartite graphs and trees. In Section 2, we present an approximation algorithm for bipartite graphs of ratio  $\frac{2(\Delta+1)^3}{\Delta^3 + 5\Delta^2 + 5\Delta + 3 - 2(-1/\Delta)^\Delta}$ , which beats the known ratios for graphs of maximum degree  $\Delta \geq 7$ . In Section 3, we present a polynomial  $3/2$ -approximation algorithm for trees, which improves the known

two approximation ratio for general graphs. Finally, in Section 4, we propose a moderately exponential approximation algorithm for trees that improves the  $3/2$  ratio with running time much better than that needed for the computation of an optimal solution.

**Notation.** In the following we consider the MEC problem on an edge weighted graph  $G = (V, E)$ , where  $|V| = n$  and  $|E| = m$ . A positive integer weight  $w(e)$  is associated with each edge  $e \in E$ . By  $d(v)$  we denote the degree of vertex  $v \in V$  and by  $\Delta$  the maximum degree of the graph  $G$ . For a subset of edges of  $G$ ,  $E' \subseteq E$ , we denote by  $G[E']$  the subgraph of  $G$  induced by the edges in  $E'$ .

For a proper coloring  $\mathcal{M} = \{M_1, M_2, \dots, M_k\}$  of the edges of  $G$ , we denote by  $w_i = \max\{w(e) | e \in M_i\}$  the weight of the color class  $M_i$ ,  $1 \leq i \leq k$ , and by  $W = \sum_{i=1}^k w_i$  the total weight of this coloring. By  $\mathcal{M}^* = \{M_1^*, M_2^*, \dots, M_{k^*}^*\}$  we denote an optimal solution to the MEC problem of weight  $OPT = w_1^* + w_2^* + \dots + w_{k^*}^*$ , where  $w_1^* \geq w_2^* \geq \dots \geq w_{k^*}^*$ .

Finally, given a set  $S$  and a positive integer weight  $w(s)$  for every element  $s \in S$ , we denote by  $\langle S \rangle = \langle s_1, s_2, \dots, s_{|S|} \rangle$  an ordering of  $S$  such that  $w(s_1) \geq w(s_2) \geq \dots \geq w(s_{|S|})$ .

## 2 Improved Approximation for Bipartite Graphs

All known approximation algorithms yielding ratios less than 2 for the MEC problem on a bipartite graph  $G = (V, E)$  [3,7,13,12] are based on the following general idea: Consider an ordering  $\langle E \rangle = \langle e_1, e_2, \dots, e_m \rangle$  of the edges of  $G$ , and let  $E_{p,q} = \{e_p, e_{p+1}, \dots, e_q\}$ ,  $p \leq q$ . Repeatedly, partition the graph  $G$  into three edge induced subgraphs  $G[E_{1,p}]$ ,  $G[E_{p+1,q}]$  and  $G[E_{q+1,m}]$  and produce a solution for the whole graph  $G$  by considering the MEC problem on these three edge induced subgraphs. Depending on how the problem is handled for each subgraph and the analysis followed, this general idea leads to different algorithms of different approximation ratios. Note that the same general approach is also employed by the  $8/7$ -approximation algorithm for the MVC problem [3,15].

Let us denote by  $(p, q)$ ,  $0 \leq p < q \leq m$ , a partition of  $G$  into subgraphs  $G[E_{1,p}]$ ,  $G[E_{p+1,q}]$  and  $G[E_{q+1,m}]$ ; by convention, we define  $E_{1,0} = \emptyset$  and  $E_{0,q} = E_{1,q}$ . By  $\Delta_{1,q}$  we denote the maximum degree of the subgraph  $G[E_{1,q}]$ . For a partition  $(p, q)$  of  $G$ , we call *critical matching* a matching  $M \subseteq E_{p+1,q}$  which saturates all the vertices of  $G[E_{1,q}]$  of degree  $\Delta_{1,q}$ . The proposed algorithm relies on the existence of such a critical matching  $M$ : a solution for the subgraph  $G[E_{1,q}]$  is found by concatenating a  $(\Delta_{1,q} - 1)$ -coloring solution for the subgraph  $G[E_{1,q} \setminus M]$  and the matching  $M$ , if exists, and by a  $\Delta_{1,q}$ -coloring of the subgraph  $G[E_{1,q}]$ , otherwise. For each partition  $(p, q)$ , the algorithm computes a solution for the input graph  $G$  by concatenating a solution for  $G[E_{1,q}]$  and a  $\Delta$ -coloring solution for  $G[E_{q+1,m}]$ . The algorithm computes also a  $\Delta$ -coloring solution for the input graph and returns the best among them.

## Algorithm BIPARTITE

1. Find a  $\Delta$ -coloring solution for  $G$ ;
2. Let  $\langle E \rangle = \langle e_1, e_2, \dots, e_m \rangle$
3. For  $p = 0, 1, 2, \dots, m - 1$  do
4.   For  $q = p + 1$  to  $m$  do
5.     Find, if any, a critical matching  $M$  in  $G[E_{p+1,q}]$ ;
6.     If  $M$  exists
  - then find a  $(\Delta_{1,q} - 1)$ -coloring solution for  $G[E_{1,q} \setminus M]$
  - else find a  $\Delta_{1,q}$ -coloring solution for  $G[E_{1,q}]$ ;
7.     Find a  $\Delta$ -coloring solution for  $G[E_{q+1,m}]$ ;
8.     Find a solution for  $G$  by concatenating the solutions found in Lines 6 and 7 and matching  $M$ , if exists;
9. Return the best among the solutions found in Lines 1 and 8;

The next proposition deals with finding, if any, a critical matching  $M$  in Line 5 of the algorithm.

**Proposition 1.** *For a partition  $(p, q)$  of a graph  $G = (V, E)$ , a critical matching  $M$ , if any, can be found in  $O(n^{2.5})$  time.*

*Proof.* Let  $U$  be the set of vertices of  $G[E_{1,q}]$  of degree  $\Delta_{1,q}$  to be saturated by a critical matching  $M \subseteq E_{p+1,q}$ . Consider the graph  $G' = (V', E')$  where  $V'$  consists of  $V$  and an additional vertex, if  $|V|$  is odd, and  $E'$  consists of  $E_{p+1,q}$  and all the edges between the vertices  $V' \setminus U$  (i.e., the vertices  $V' \setminus U$  induce a clique in  $G'$ ). If there exists a perfect matching in  $G'$ , then there exists a critical matching  $M$ , since no edges adjacent to  $U$  have been added in  $G'$ . Conversely, if there exists a critical matching  $M$ , then there exists a perfect matching in  $G'$ , consisting of the edges of  $M$  plus the edges of a perfect matching in the complete subgraph of  $G'$  induced by its vertices not saturated by  $M$ . Therefore, a critical matching  $M$ , if any, can be found by looking for a perfect matching, if any, in  $G'$ . It is well known that this can be done in  $O(n^{2.5})$  time [14].  $\square$

**Theorem 1.** *Algorithm BIPARTITE achieves an approximation ratio of  $\frac{2(\Delta + 1)^3}{\Delta^3 + 5\Delta^2 + 5\Delta + 3 - 2(-1/\Delta)^\Delta}$  for the MEC problem on bipartite graphs.*

*Proof.* The solution obtained by a  $\Delta$ -coloring of the input graph computed in Line 1 of the algorithm is of weight  $W_1 \leq \Delta \cdot w_1^*$ .

Consider the partition  $(p, q)$  of  $G$  where  $w(e_{p+1}) = w_{i-1}^*$  and  $w(e_{q+1}) = w_i^*$ , for  $2 \leq i \leq \Delta$  (recall that  $w_1^* \geq w_2^* \geq \dots \geq w_{k^*}^*$  and  $k^* \geq \Delta$ ). In such an iteration, all the edges in  $E_{1,q}$  belong to  $i - 1 \geq \Delta_{1,q}$  matchings of an optimal solution  $\mathcal{M}^*$ .

If  $\Delta_{1,q} < i - 1$ , then an  $(i - 2)$ -coloring of  $G[E_{1,q}]$  yields a solution of weight at most  $(i - 2) \cdot w_1^*$  for this subgraph.

If  $\Delta_{1,q} = i - 1$  then a critical matching  $M$  exists. Indeed, in this case the  $(i - 1)$ -th matching of  $\mathcal{M}^*$  always contains some edges from  $E_{p+1,q}$ , for otherwise all the edges in  $E_{1,q}$  belong to  $i - 2$  matchings of  $\mathcal{M}^*$ , a contradiction; these edges

of  $E_{p+1,q}$  could be a critical matching  $M$  for the partition  $(p, q)$ . Thus, a  $(i - 2)$ -coloring solution of  $G[E_{1,q} \setminus M]$  and critical matching  $M$  yield a solution for the subgraph  $G[E_{1,q}]$  of weight at most  $(i - 2) \cdot w_1^* + w_{i-1}^*$ . Finally, a  $\Delta$ -coloring solution for  $G[E_{q+1,m}]$  is of cost at most  $\Delta \cdot w_i^*$ .

Hence, for such a partition  $(p, q)$  the algorithm finds a solution for the whole input graph of weight

$$W_i \leq (i - 2) \cdot w_1^* + w_{i-1}^* + \Delta \cdot w_i^*, \quad 2 \leq i \leq \Delta.$$

As the algorithm returns the best among the solutions found, we have  $\Delta$  different bounds on the weight  $W$  of this solution, that is  $W \leq W_i, 1 \leq i \leq \Delta$  (recall that  $W_1 \leq \Delta \cdot w_1^*$ ). By multiplying both sides of the  $i$ -th bound inequality by

$$x_i = \begin{cases} \frac{1}{\Delta}, & \text{if } i = \Delta \\ \frac{1}{\Delta + 1} \left( 1 - \left( \frac{-1}{\Delta} \right)^{\Delta - i + 1} \right), & \text{if } \Delta - 1 \geq i \geq 2 \\ \frac{1}{\Delta} - \sum_{j=0}^{\Delta-3} \left( \frac{\Delta - (j + 2)}{\Delta} x_{\Delta-j} \right) - \frac{1}{\Delta} x_2, & \text{if } i = 1 \end{cases}$$

and adding them we get

$$\frac{W}{OPT} \leq \frac{2(\Delta + 1)^3}{\Delta^3 + 5\Delta^2 + 5\Delta + 3 - 2(-1/\Delta)^\Delta}. \quad \square$$

Lines 5-8 of the algorithm are repeated  $O(m^2)$  times. Finding a critical matching in Line 5, takes, by Proposition [1](#),  $O(n^{2.5})$  time, while finding the colorings of the bipartite subgraphs of  $G$  in Lines 6 and 7, takes  $O(m \log \Delta)$  time [2](#).

The approximation ratios achieved by Algorithm BIPARTITE are compared, as  $\Delta$  increases, with the best known ones in Table [1](#). Note that our algorithm is of the same complexity with those in [3](#) and [12](#), while the complexity of the algorithm in [13](#) is greater by a factor of  $O(m^2)$ .

**Table 1.** Approximation ratios for bipartite graphs

$\Delta$	Best known	This paper
3	1.17 <a href="#">3</a>	1.42
4	1.32 <a href="#">13</a>	1.50
5	1.45 <a href="#">13</a>	1.55
6	1.56 <a href="#">13</a>	1.60
7	1.65 <a href="#">13</a>	1.64
8	1.74 <a href="#">13</a>	1.67
9	1.78 <a href="#">12</a>	1.69
10	1.80 <a href="#">12</a>	1.71
11	1.82 <a href="#">12</a>	1.73
12	1.84 <a href="#">12</a>	1.75
13	1.85 <a href="#">12</a>	1.76
20	1.90 <a href="#">12</a>	1.83
50	1.96 <a href="#">12</a>	1.93

### 3 A 3/2 Approximation Algorithm for Trees

In this section, we first present an  $(1 + \frac{w_1^* - w_\Delta^*}{OPT})$ -approximation algorithm for the MEC problem on trees. Our algorithm roots the tree in an arbitrary vertex  $r$  and constructs a solution as follows: Traverse the vertices of the rooted tree in pre-order, for each vertex,  $u \in V$ , consider the edges to its children in non increasing order and insert them into the first matching they fit.

Algorithm TREES

1. Let  $T_r$  be the tree rooted in an arbitrary vertex  $r$ ;
2. For each vertex  $u$  in pre-order traversal of  $T_r$  do
3. Let  $\langle E_u \rangle = \langle e_1^u, e_2^u, \dots, e_{d(u)}^u \rangle$  be the edges adjacent to  $u$ ,  
and  $(u, p)$  be the edge from  $u$ ,  $u \neq r$ , to its parent;
4. Using ordering  $\langle E_u \rangle$ , insert each edge in  $E_u$ , but  $(u, p)$ , into  
the first matching which does not contain an edge in  $E_u$ ;

To analyze our algorithm we define  $y_i$ ,  $1 \leq i \leq \Delta$ , to be the weight of the heaviest edge between those ranked  $i$  in each ordering  $\langle E_u \rangle$ ,  $u \in V$ , i.e.,  $y_i = \max_{u \in V} \{w(e_i^u)\}$ .

It is clear that  $y_1 \geq y_2 \geq \dots \geq y_\Delta$ . Next two propositions use these values for bounding the weights of the matchings of both an optimal solution and the solution constructed by our algorithm.

**Proposition 2.** *For all  $1 \leq i \leq \Delta$ , it holds that  $w_i^* \geq y_i$ .*

*Proof.* Let  $e = (u, v)$  be the heaviest edge with rank equal to  $i$ , i.e.,  $y_i = w(e)$ . For at least one of the endpoints of  $e$ , assume, w.l.o.g., for  $u$ , it holds that  $e$  is ranked  $i$  in  $E_u$ , that is  $y_i = w(e_i^u)$ . Therefore, there exist  $i$  edges adjacent to vertex  $u$  of weight at least  $y_i$ . These  $i$  edges belong in  $i$  different matchings in an optimal solution, since they share vertex  $u$  as a common endpoint. Thus, the  $i$ -th matching in an optimal solution is of weight at least  $y_i$ .  $\square$

**Proposition 3.** *Algorithm TREES constructs a solution of exactly  $\Delta$  matchings. For the weight,  $w_i$ , of the  $i$ -th,  $2 \leq i \leq \Delta$ , matching it holds that  $w_i \leq y_{i-1}$ .*

*Proof.* For the first part of the proposition consider first the root vertex of the tree. It has at most  $\Delta$  adjacent edges which the algorithm inserts into at most  $\Delta$  different matchings. Consider, next, any other vertex  $u$  and let  $e$  be the edge between  $u$  and its parent. This edge  $e$  has been already inserted by the algorithm into a matching, say  $M_k$ . The rest, but  $e$ , adjacent to vertex  $u$  edges are at most  $\Delta - 1$  which the algorithm inserts into at most  $\Delta - 1$  matchings different than  $M_k$ . Therefore, the algorithm will use exactly  $\Delta$  matchings  $M_1, M_2, \dots, M_\Delta$ .

We shall prove the second part of the proposition by induction on the vertices in the order they are processed by the algorithm.

For the root  $r$ , the algorithm sorts all adjacent edges to  $r$  and inserts  $e_1^r$  into matching  $M_1$ ,  $e_2^r$  into matching  $M_2$ , and so on. Thus, after the first iteration it holds that  $w_i = w(e_i^r) \leq y_i \leq y_{i-1}$ ,  $2 \leq i \leq \Delta$ .

Assume that the statement of the lemma holds before the iteration processing the vertex  $u \in V$ , that is  $w_i \leq y_{i-1}$ ,  $2 \leq i \leq \Delta$ .



Consider, now, the iteration in which the algorithm processes the vertex  $u$ . Let  $e$  be the edge between  $u$  and its parent,  $j$  be the rank of the edge  $e$  in  $E_u$  and  $M_k$  be the matching where the algorithm has already inserted edge  $e$ . Let us also denote by  $w'_i$  the weight of the matching  $M_i$ ,  $2 \leq i \leq \Delta$ , after processing the vertex  $u$ . We distinguish among three cases, and for each one we prove that  $w'_i \leq y_{i-1}$ ,  $2 \leq i \leq \Delta$ .

- (i) If  $k = j$ , then after this iteration each edge  $e_i^u$  belongs to matching  $M_i$ ,  $1 \leq i \leq d(u)$ . By the inductive hypothesis it follows that  $w'_i = \max\{w_i, w(e_i^u)\}$ , where  $w_i \leq y_{i-1}$ . Since  $w(e_i^u) \leq y_i$ , it holds that  $w'_i \leq y_{i-1}$ ,  $2 \leq i \leq \Delta$ .
- (ii) If  $k > j$ , then after this iteration: for  $1 \leq i \leq j - 1$  and  $k + 1 \leq i \leq d(u)$  each edge  $e_i^u$  belongs to matching  $M_i$ ; for  $j + 1 \leq i \leq k$  each edge  $e_i^u$  belongs to matching  $M_{i-1}$ . For the former case we conclude as in Case (i). For the latter case by the inductive hypothesis it follows that  $w'_i = \max\{w_i, w(e_{i+1}^u)\}$  where  $w_i \leq y_{i-1}$ . Since  $w(e_{i+1}^u) \leq y_{i+1} \leq y_{i-1}$ , it holds that  $w'_i \leq y_{i-1}$ .
- (iii) If  $k < j$ , then after this iteration: for  $1 \leq i \leq k - 1$  and  $j + 1 \leq i \leq d(u)$  each edge  $e_i^u$  belongs to matching  $M_i$ ; for  $k \leq i \leq j - 1$  each edge  $e_i^u$  belongs to matching  $M_{i+1}$ . For the former case we conclude as in Case (i). For the latter case by the inductive hypothesis it follows that  $w'_i = \max\{w_i, w(e_{i-1}^u)\}$  where  $w_i \leq y_{i-1}$ . Since  $w(e_{i-1}^u) \leq y_{i-1}$ , it holds that  $w'_i \leq y_{i-1}$ . □

Using the bounds established in Propositions 2 and 3 we obtain the approximation ratio of our algorithm.

**Lemma 1.** *Algorithm TREES achieves an approximation ratio equal to  $1 + \frac{w_1^* - w_\Delta^*}{OPT}$  for the MEC problem on trees. This is an asymptotically tight 2 approximation ratio.*

*Proof.* For the weight of the first matching obtained by Algorithm TREES it holds that  $w_1 \leq y_1 = w_1^*$ , since both  $y_1$  and  $w_1^*$  are equal to the weight of heaviest edge of the tree. By Proposition 3 it holds that  $w_i \leq y_{i-1}$ ,  $2 \leq i \leq \Delta$  and by Proposition 2 it holds that  $y_i \leq w_i^*$ ,  $1 \leq i \leq \Delta$ . Therefore, the weight

$$\text{of the solution obtained by Algorithm TREES is } W = \sum_{i=1}^{\Delta} w_i \leq y_1 + \sum_{i=2}^{\Delta} y_{i-1} = y_1 + \sum_{i=1}^{\Delta-1} y_i \leq w_1^* + \sum_{i=1}^{\Delta-1} w_i^* \leq w_1^* + OPT - w_\Delta^*, \text{ that is } \frac{W}{OPT} \leq 1 + \frac{w_1^* - w_\Delta^*}{OPT} < 2.$$

Figure 1(a) shows an example for which the algorithm performs a ratio of two. The weight of an optimal solution to this instance is  $C + 2\epsilon$  (Figure 1(b)) and the weight of the solution obtained by Algorithm TREES is  $2C + \epsilon$  (Figure 1(c)). Thus, the approximation ratio for this instance becomes  $\frac{2C + \epsilon}{C + 2\epsilon}$ . □

The most general result we have for the MEC problem is due to Kesselman and Kogan 9. This is a greedy 2-approximation algorithm for general graphs, to which we refer as Algorithm KK. A slightly better analysis of this algorithm presented in 12 leads to the following lemma.

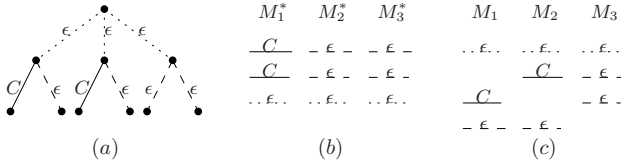


Fig. 1. A tight example for the 2 approximation ratio of Algorithm TREES

**Lemma 2.** [12] *Algorithm KK achieves a tight approximation ratio of  $2 - \frac{w_1^*}{OPT} < 2 - \frac{1}{\Delta}$ .*

To obtain the 3/2 approximation ratio we combine Algorithm KK [9] with Algorithm TREES, i.e., we run both algorithms and we select the best of the two solutions found.

**Theorem 2.** *There is a tight  $\frac{3}{2}$ -approximation algorithm for the MEC problem on trees.*

*Proof.* Let  $W$  be weight of the best of the two solutions found by Algorithm KK and Algorithm TREES. By Lemma 2 it holds that  $\frac{W}{OPT} \leq 2 - \frac{w_1^*}{OPT}$  and by Lemma 1 that  $\frac{W}{OPT} \leq 1 + \frac{w_1^* - w_\Delta^*}{OPT}$ . As the first bound is increasing and the second one is decreasing with respect to  $OPT$ , it follows that the ratio  $\frac{W}{OPT}$  is maximized when  $2 - \frac{w_1^*}{OPT} = 1 + \frac{w_1^* - w_\Delta^*}{OPT}$ , that is  $OPT = 2 \cdot w_1^* - w_\Delta^*$ . Therefore,  $\frac{W}{OPT} \leq 2 - \frac{w_1^*}{OPT} = 2 - \frac{w_1^*}{2 \cdot w_1^* - w_\Delta^*} \leq 2 - \frac{w_1^*}{2 \cdot w_1^*} = \frac{3}{2}$ .

For the tightness of the ratio consider the example shown in Figure 2(a). The weight of an optimal solution to this instance is  $2C + 2\epsilon$  (Figure 2(b)), the weight of the solution created by Algorithm TREES is  $3C$  (Figure 2(c)) and the weight of the solution created by Algorithm KK (Figure 2(d)) is  $3C - \epsilon$ . Our algorithm selects the solution obtained by Algorithm KK of weight  $3C - \epsilon$  and thus approximation ratio becomes  $\frac{3C - \epsilon}{2C + 2\epsilon}$ .  $\square$

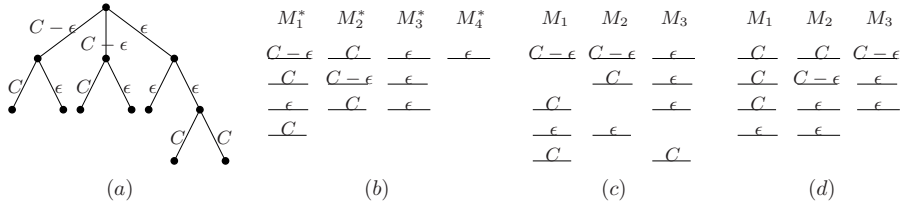


Fig. 2. A tight example for the 3/2-approximation algorithm for trees

## 4 Moderately Exponential Approximation for Trees

In this section, we present a moderately exponential approximation algorithm for the MEC problem on trees. Each step of this algorithm exhaustively considers  $k$  edge weights as the weights of  $k$  matchings of the optimal solution. The range of values of  $k$  (i.e., the number of steps) depends on an integer parameter  $z$ ,  $0 < z \leq \frac{m}{2}$ . More specific, for each value of  $k \leq z$  and  $k \geq m - z$  the algorithm considers all  $\binom{m}{k}$  combinations of edge weights. Thus, both the ratio and the (non polynomial) complexity of the algorithm depend also on  $z$ . In fact, for any combination  $w_1, w_2, \dots, w_k$ , of  $k$  edge weights the algorithm has to answer to the following decision problem: is there a feasible solution  $M_1, M_2, \dots, M_k$ , of the MEC problem where  $\max_{e \in M_i} w(e) = w_i$ ? Moreover, for each  $k'$ ,  $z < k' < m - z$ , a similar decision problem is considered, with weights  $w_1, w_2, \dots, w_z, w_{z+1}, w_{z+2}, \dots, w_{k'}$ , where  $w_{z+1} = w_{z+2} = \dots = w_{k'} = w_z$ . The algorithm returns the minimum feasible solution found among all the iterations. These decision problems encountered in each combination of weights can be answered by transforming them to instances of the next problem:

LIST EDGE-COLORING (LEC)

INSTANCE: A graph  $G = (V, E)$ , a set of colors  $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$  and a list of colors  $\phi(e) \subseteq \mathcal{C}$  for each  $e \in E$ .

QUESTION: Is there a  $k$ -coloring of  $G$  such that each edge  $e$  is assigned a color in its list  $\phi(e)$ ?

It is known that the LEC problem is polynomial for trees [4], but it becomes NP-complete for bipartite graphs [11]. Therefore, our decision problem for trees can be solved in polynomial time, while this method cannot be extended to bipartite graphs.

Algorithm TREES( $z$ )

1. For  $k = 1, 2, \dots, z, m - z, \dots, m$  do
2. Exhaustively search for the weights of the  $k$  heaviest matchings of the optimal solution,  $w_1^* \geq w_2^* \geq \dots \geq w_k^*$ ;
3. Build the input for the List Edge-Coloring algorithm:
  - Set of colors  $\{C_1, C_2, \dots, C_k\}$ ;
  - For each  $e \in E$  set  $\phi(e) = \{C_i : w(e) \leq w_i^*, 1 \leq i \leq k\}$ ;
4. Run the algorithm for the List Edge-Coloring problem;
5. If  $k = z$  then
6. For  $k' = z + 1, z + 2, \dots, m - z - 1$  do
7. Build the input for the List Edge-Coloring algorithm:
  - Set of colors  $\{C_1, C_2, \dots, C_z, \dots, C_{k'}\}$ ;
  - If  $w(e) > w_z^*$  then  $\phi(e) = \{C_i : w(e) \leq w_i^*, 1 \leq i \leq z - 1\}$ ;
  - If  $w(e) \leq w_z^*$  then  $\phi(e) = \{C_1, C_2, \dots, C_{k'}\}$ ;
8. Run the algorithm for the List Edge-Coloring problem;
9. Return the best solution found;

**Theorem 3.** For any  $\rho \geq 1$ , Algorithm TREES( $z$ ) achieves a  $\rho$  approximation ratio for the MEC problem on trees, in polynomial space and running time  $O^*(f(\rho)^m)$ , where  $f(\rho) = \frac{(2\rho-1)^2+1}{(2\rho-1)^2(2\rho-1)^2/(2\rho-1)^2+1}$ .

*Proof.* If  $k^* \leq z$  or  $k^* \geq m - z$  then in an iteration of Lines 2-4 the optimal solution is found.

If  $z < k^* < m - z$  then we consider the following two solutions.

In the  $k = m - z$  step of the algorithm, for a combination  $w_1, w_2, \dots, w_k$  of weights, it holds that  $w_i = w_i^*$ ,  $1 \leq i \leq k^*$ . Hence, for this combination, there is a feasible solution of weight at most  $w_1^* + w_2^* + \dots + w_{k^*}^* + (m - z - k^*)w_{k^*}^* = OPT + (m - z - k^*)w_{k^*}^*$ .

For a combination  $w_1, w_2, \dots, w_k$  of weights, in the iteration of Lines 7-8 where  $k' = k^*$  it holds that  $w_i = w_i^*$ ,  $1 \leq i \leq z$ . Hence, for this iteration there is a feasible solution of weight at most  $w_1^* + w_2^* + \dots + w_z^* + (k^* - z)w_z^* = OPT - \sum_{i=z+1}^{k^*} w_i^* + (k^* - z)w_z^*$ .

Thus, it holds that

$$\begin{aligned} \frac{W}{OPT} &\leq \min \left\{ \frac{OPT + (m - z - k^*)w_{k^*}^*}{OPT}, \frac{OPT - \sum_{i=z+1}^{k^*} w_i^* + (k^* - z)w_z^*}{OPT} \right\} \\ &\leq \min \left\{ 1 + \frac{(m - z - k^*)w_{k^*}^*}{zw_z^* + (k^* - z)w_{k^*}^*}, 1 + \frac{(k^* - z)(w_z^* - w_{k^*}^*)}{zw_z^* + (k^* - z)w_{k^*}^*} \right\}. \end{aligned}$$

As the first value is increasing with  $w_{k^*}^*$  and the second one is decreasing, this quantity is maximized when  $(k^* - z)w_z^* = (m - 2z)w_{k^*}^*$ . Therefore, we have

$$\frac{W}{OPT} \leq 1 + \frac{(m - z - k^*)w_{k^*}^*}{\frac{z(m-2z)}{k^*-z}w_{k^*}^* + (k^* - z)w_{k^*}^*} = \frac{k^*(m - 2z)}{z(m - 2z) + (k^* - z)^2},$$

which is maximized for  $k^* = \sqrt{z(m - z)}$ . Hence,

$$\frac{W}{OPT} \leq \frac{\sqrt{z(m - z)}(m - 2z)}{z(m - 2z) + (\sqrt{z(m - z)} - z)^2} = \frac{m - 2z}{2\sqrt{z(m - z)} - 2z}.$$

By setting  $z = \lambda m$ , where  $0 < \lambda \leq \frac{1}{2}$ , we get

$$\frac{W}{OPT} \leq \frac{m - 2\lambda m}{2\sqrt{\lambda m(m - \lambda m)} - 2\lambda m} = \frac{1 - 2\lambda}{2\sqrt{\lambda(1 - \lambda)} - 2\lambda} = \rho.$$

Therefore, in order to achieve a  $\rho$  approximation ratio we choose  $\lambda = \frac{1}{(2\rho-1)^2+1}$ , that is  $z = \frac{m}{(2\rho-1)^2+1}$ .

Since Lines 3-8 of the algorithm are executed independently for each combination of weights, it follows that the algorithm needs polynomial space. Since the List Edge-Coloring algorithm is polynomial, the complexity of the algorithm

is equal, within a polynomial factor, to the number,  $T(m)$ , of combinations generated. Thus, we have

$$\begin{aligned}
 T(m) &\leq \sum_{i=1}^z \binom{m}{i} + \sum_{i=m-z}^m \binom{m}{i} = 2 \sum_{i=1}^z \binom{m}{i} \leq 2z \binom{m}{z} \leq m \binom{m}{\lambda m} \\
 &\leq m \left( \left( \frac{1}{\lambda} \right)^\lambda \left( \frac{1}{1-\lambda} \right)^{1-\lambda} \right)^m = m \left( \frac{(2\rho-1)^2 + 1}{(2\rho-1)^{2(2\rho-1)^2 / ((2\rho-1)^2 + 1)}} \right)^m \quad \square
 \end{aligned}$$

Some values of  $\rho \leq 3/2$  and  $f(\rho)$  are summarized in the following table:

**Table 2.** Approximation ratios vs. complexity’s base for trees

$\rho$	$OPT$	1.1	1.2	1.3	1.4	1.5
$f(\rho)$	2	1.968	1.896	1.811	1.727	1.649

As it is shown in Table 2, Algorithm TREES( $\frac{m}{2}$ ) computes an exact solution for the MEC problem on trees in  $O^*(2^m)$  time and polynomial space. Note that, in [1] has been presented an algorithm with running time and space  $O^*(2^n)$ , which, for any  $k$ , computes the number of all proper  $k$ -colorings of a graph, and moreover enumerates these colorings. This algorithm can be used to find an optimal solution for the MVC problem on a general graph, by running it for  $1 \leq k \leq n$ . Considering the line graph  $L(G)$  of the input graph  $G$  of the MEC problem, we derive that the MEC problem on general graphs can be optimally solved with running time and space  $O^*(2^m)$ .

**Proposition 4.** *If  $\Delta = o(m)$ , then Algorithm TREES( $\frac{m}{2}$ ) requires subexponential running time  $2^{o(m)}$  in order to compute an exact solution for trees.*

*Proof.* Note that for the number  $k$  of matchings in any optimal solution to the MEC problem it holds that  $k \leq 2\Delta - 1$ . Thus, the number,  $T(m)$ , of combinations of weights generated becomes

$$\begin{aligned}
 T(m) &\leq \binom{m}{2\Delta} \leq \frac{m^m}{(2\Delta)^{2\Delta} (m-2\Delta)^{m-2\Delta}} \\
 &\leq 2^{m \log m - 2\Delta \log(2\Delta) - (m-2\Delta) \log(m-2\Delta)} \\
 &\leq 2^{m \log(1+2\Delta/(m-2\Delta)) + 2\Delta \log(m/2\Delta-1)}
 \end{aligned}$$

Notice first that  $2\Delta/(m-2\Delta)$  tends to 0 for  $m \rightarrow \infty$ , since  $\Delta = o(m)$ , and thus  $m \log \left( 1 + \frac{2\Delta}{(m-2\Delta)} \right) \rightarrow 0$ . Moreover, note that  $2\Delta \log \left( \frac{m}{2\Delta} - 1 \right) = o(m)$ , since  $\frac{2\Delta \log \left( \frac{m}{2\Delta} - 1 \right)}{m}$  tends to 0 as  $m$  increases. Combining the two observations above, we get that  $T(m) = 2^{o(m)}$  and, hence, the running time of Algorithm TREES( $\frac{m}{2}$ ) is  $O^*(2^{o(m)})$ .  $\square$

## References

1. Björklund, A., Husfeldt, T.: Inclusion–exclusion algorithms for counting set partitions. In: 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), pp. 575–582. IEEE Computer Society, Los Alamitos (2006)
2. Cole, R., Ost, K., Schirra, S.: Edge-coloring bipartite multigraphs in  $O(E \log D)$  time. *Combinatorica* 21, 5–12 (2001)
3. de Werra, D., Demange, M., Escoffier, B., Monnot, J., Paschos, V.T.: Weighted coloring on planar, bipartite and split graphs: Complexity and approximation. *Discrete Applied Mathematics* 157, 819–832 (2009)
4. de Werra, D., Hoffman, A.J., Mahadev, N.V.R., Peled, U.N.: Restrictions and pre-assignments in preemptive open shop scheduling. *Discrete Applied Mathematics* 68, 169–188 (1996)
5. Demange, M., de Werra, D., Monnot, J., Paschos, V.T.: Time slot scheduling of compatible jobs. *Journal of Scheduling* 10, 111–127 (2007)
6. Epstein, L., Levin, A.: On the max coloring problem. In: Kaklamanis, C., Skutella, M. (eds.) WAOA 2007. LNCS, vol. 4927, pp. 142–155. Springer, Heidelberg (2008)
7. Escoffier, B., Monnot, J., Paschos, V.T.: Weighted coloring: further complexity and approximability results. *Information Processing Letters* 97, 98–103 (2006)
8. Holyer, I.: The NP-completeness of edge-coloring. *SIAM Journal on Computing* 10, 718–720 (1981)
9. Kesselman, A., Kogan, K.: Nonpreemptive scheduling of optical switches. *IEEE Transactions on Communications* 55, 1212–1219 (2007)
10. König, D.: Über graphen und ihre anwendung auf determinantentheorie und mengenlehre. *Mathematische Annalen* 77, 453–465 (1916)
11. Kubale, M.: Some results concerning the complexity of restricted colorings of graphs. *Discrete Applied Mathematics* 36, 35–46 (1992)
12. Lucarelli, G., Milis, I., Paschos, V.T.: On the maximum edge coloring problem. In: Bampis, E., Skutella, M. (eds.) WAOA 2008. LNCS, vol. 5426, pp. 279–292. Springer, Heidelberg (2009)
13. Lucarelli, G., Milis, I., Paschos, V.T.: On the max-weight edge coloring problem. *Journal of Combinatorial Optimization* (in press)
14. Micali, S., Vazirani, V.V.: An  $O(\sqrt{|V||E|})$  algorithm for finding maximum matching in general graphs. In: 21st Annual IEEE Symposium on Foundations of Computer Science (FOCS 1980), pp. 17–27. IEEE Computer Society, Los Alamitos (1980)
15. Pemmaraju, S.V., Raman, R.: Approximation algorithms for the max-coloring problem. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1064–1075. Springer, Heidelberg (2005)
16. Pemmaraju, S.V., Raman, R., Varadarajan, K.R.: Buffer minimization using max-coloring. In: 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004), pp. 562–571 (2004)

# Three Complexity Results on Coloring $P_k$ -Free Graphs

Hajo Broersma<sup>1,\*</sup>, Fedor V. Fomin<sup>2,\*\*</sup>, Petr A. Golovach<sup>2,\*\*</sup>, and Daniël Paulusma<sup>1,\*</sup>

<sup>1</sup> Department of Computer Science, Durham University, DH1 3LE Durham, United Kingdom

{hajo.broersma, daniel.paulusma}@durham.ac.uk

<sup>2</sup> Department of Informatics, University of Bergen, PB 7803, 5020 Bergen, Norway

{fedor.fomin, petr.golovach}@ii.uib.no

**Abstract.** We prove three complexity results on vertex coloring problems restricted to  $P_k$ -free graphs, i.e., graphs that do not contain a path on  $k$  vertices as an induced subgraph. First of all, we show that the pre-coloring extension version of 5-coloring remains NP-complete when restricted to  $P_6$ -free graphs. Recent results of Hoàng et al. imply that this problem is polynomially solvable on  $P_5$ -free graphs. Secondly, we show that the pre-coloring extension version of 3-coloring is polynomially solvable for  $P_6$ -free graphs. This implies a simpler algorithm for checking the 3-colorability of  $P_6$ -free graphs than the algorithm given by Randerath and Schiermeyer. Finally, we prove that 6-coloring is NP-complete for  $P_7$ -free graphs. This problem was known to be polynomially solvable for  $P_5$ -free graphs and NP-complete for  $P_8$ -free graphs, so there remains one open case.

**Keywords:** graph coloring,  $P_k$ -free graph, computational complexity.

## 1 Introduction

In this paper we consider computational complexity issues related to vertex coloring problems restricted to  $P_k$ -free graphs. Due to the fact that the usual vertex  $\ell$ -coloring problem is NP-complete for any fixed  $\ell \geq 3$ , there has been considerable interest in studying its complexity when restricted to certain graph classes. Without doubt one of the most well-known results in this respect is that  $\ell$ -coloring is polynomially solvable for perfect graphs. More information on this classic result and related work on coloring problems restricted to graph classes can be found in, e.g., [11] and [13]. Instead of repeating what has been written in so many papers over the years, we also refer to these surveys for motivation and background. Here we continue the study of  $\ell$ -coloring and its variants for  $P_k$ -free graphs, a problem that has been studied in several earlier papers by different groups of researchers (see, e.g., [4], [8], [9], [10], [14]).

### 1.1 Background and Terminology

We refer to [1] for standard graph theory terminology and to [3] for terminology on computational complexity. Let  $G = (V, E)$  be a graph. A (vertex) coloring of  $G$  is a

---

\* The work of DP is supported by EPSRC Grant EP/D053633/1.

\*\* Supported by the Norwegian Research Council.

mapping  $\phi : V \rightarrow \{1, 2, \dots\}$  such that  $\phi(u) \neq \phi(v)$  whenever  $uv \in E$ . Here  $\phi(u)$  is usually referred to as the color of  $u$  in the coloring  $\phi$  of  $G$ . An  $\ell$ -coloring of  $G$  is a mapping  $\phi : V \rightarrow \{1, 2, \dots, \ell\}$  such that  $\phi(u) \neq \phi(v)$  whenever  $uv \in E$ . In list-coloring we assume that  $V = \{v_1, v_2, \dots, v_n\}$  and that for every vertex  $v_i$  of  $G$  there is a list  $L_i$  of admissible colors (a subset of the natural numbers). Given these lists, a list-coloring of  $G$  is a mapping  $\phi : V \rightarrow \{1, 2, \dots\}$  such that  $\phi(v_i) \in L_i$  for all  $i \in \{1, 2, \dots, n\}$  and  $\phi(v_i) \neq \phi(v_j)$  whenever  $v_i v_j \in E$ . In pre-coloring extension we assume that a (possibly empty) subset  $W \subseteq V$  of  $G$  is pre-colored with  $\phi_W : W \rightarrow \{1, 2, \dots\}$  and the question is whether we can extend  $\phi_W$  to a coloring of  $G$ . If  $\phi_W$  is restricted to  $\{1, 2, \dots, \ell\}$  and we want to extend it to an  $\ell$ -coloring of  $G$ , we say we deal with the pre-coloring extension version of  $\ell$ -coloring. In fact, we consider a slight variation on the latter problem which can be considered as list coloring, but which has the flavor of pre-coloring: lists have varying sizes including some of size 1. We will slightly abuse terminology and call these problems pre-coloring extension problems too.

## 1.2 Results of This Paper

We prove the following three complexity results on vertex coloring problems restricted to  $P_k$ -free graphs.

- First of all, in Section 2 we show that the pre-coloring extension version of 5-coloring remains NP-complete when restricted to  $P_6$ -free graphs. Recent results of Hoàng et al. [4] imply that this problem is polynomially solvable on  $P_5$ -free graphs. Their algorithm for  $\ell$ -coloring for any fixed  $\ell$  is in fact a list-coloring algorithm where the lists are from the set  $\{1, 2, \dots, \ell\}$ .
- Secondly, in Section 3 we show that the pre-coloring extension version of 3-coloring is polynomially solvable for  $P_6$ -free graphs. The 3-coloring problem was known to be polynomially solvable for  $P_6$ -free graphs from [10], where the authors use the Strong Perfect Graph Theorem and a result of Tucker [12] to obtain their algorithm. Our algorithm is independent of the Strong Perfect Graph Theorem, and uses a recent structural result of [5]; it reduces the 8 page journal description of the algorithm in [10] to 3 pages.
- Finally, in Section 4 we prove that 6-coloring is NP-complete for  $P_7$ -free graphs. This problem was known to be polynomially solvable for  $P_5$ -free graphs [4] and NP-complete for  $P_8$ -free graphs [14], so there remains one open case.

## 2 Pre-coloring Extension of 5-Coloring for $P_6$ -Free Graphs

In this section we show that the pre-coloring extension version of 5-coloring remains NP-complete when restricted to  $P_6$ -free graphs. We use a reduction from not-all-equal 3-Satisfiability with positive literals only which we denote as NAE 3SATPL (also known as HYPERGRAPH 2-COLORABILITY), which is defined as follows. Given a set  $X = \{x_1, x_2, \dots, x_n\}$  of logical variables, and a set  $C = \{C_1, C_2, \dots, C_m\}$  of three-literal clauses over  $X$  in which all literals are positive, does there exist a truth



assignment for  $X$  such that each clause contains at least one true literal and at least one false literal?

We consider an arbitrary instance  $I$  of NAE 3SATPL and define a graph  $G_I$  and a pre-coloring on some vertices of  $G_I$ , and next we show that  $G_I$  is  $P_6$ -free and that the pre-coloring on  $G_I$  can be extended to a 5-coloring of  $G_I$  if and only if  $I$  has a satisfying truth assignment in which each clause contains at least one true literal and at least one false literal.

## 2.1 The Graph $G_I$ Corresponding to the Instance $I$

Let  $I$  be an arbitrary instance of NAE 3SATPL with variables  $\{x_1, x_2, \dots, x_n\}$  and clauses  $\{C_1, C_2, \dots, C_m\}$ . We define a graph  $G_I$  corresponding to  $I$  and lists of admissible colors for its vertices based on the following construction. We note here that the lists we introduce below are only there for convenience to the reader; it will be clear later that all lists other than  $\{1, 2, \dots, 5\}$  are in fact forced by the pre-colored vertices.

- We introduce one new vertex for each of the clauses, and use the same labels  $C_1, C_2, \dots, C_m$  for these  $m$  vertices; we assume that for each of these vertices there is a list  $\{1, 2, 3\}$  of admissible colors. We say that these vertices are of  $C$ -type.
- We introduce one new vertex for each of the variables, and use the same labels  $x_1, x_2, \dots, x_n$  for these  $n$  vertices; we assume that for each of these vertices there is a list  $\{4, 5\}$  of admissible colors. We say that these vertices are of  $x$ -type.
- We join all  $C$ -type vertices to all  $x$ -type vertices to form a large complete bipartite graph.
- For each clause  $C_j$  containing the variables  $x_i, x_k$  and  $x_r$  we introduce three pairs of new vertices  $\{a_{i,j}, b_{i,j}\}, \{a_{k,j}, b_{k,j}\}, \{a_{r,j}, b_{r,j}\}$ ; we assume the following lists of admissible colors for these three pairs, respectively:  $\{\{1, 4\}, \{2, 5\}\}, \{\{2, 4\}, \{3, 5\}\}, \{\{3, 4\}, \{1, 5\}\}$ . We say that these vertices are of  $a$ -type and  $b$ -type. We add edges between  $x$ -type and  $a$ -type vertices whenever the first index of the  $a$ -type vertex is the same as of the  $x$ -type vertex, and similarly for the  $b$ -type vertices. We add edges between  $C$ -type and  $a$ -type vertices whenever the second index of the  $a$ -type vertex is the same as the index of the  $C$ -type vertex, and similarly for the  $b$ -type vertices. Hence each clause with three variables is represented by three 4-cycles that have one  $C$ -type vertex in common.
- For each  $a$ -type vertex we introduce a copy of a  $K_{2,3}$ , as follows: for  $a_{i,j}$  we add five vertices  $\{p_{i,j,1}, \dots, p_{i,j,5}\}$ , and we add all edges between  $\{p_{i,j,1}, p_{i,j,2}, p_{i,j,3}\}$  and  $\{p_{i,j,4}, p_{i,j,5}\}$ . We say that these vertices are of  $p$ -type. We add edges between each  $a$ -vertex and the  $p$ -vertices of its corresponding  $K_{2,3}$  depending on its list of admissible colors. In particular, we join the  $a$ -vertex to the three  $p$ -vertices of its  $K_{2,3}$  that have a third index which is not in its list of admissible colors. So, if  $a_{i,j}$  has list  $\{1, 4\}$ , we join it to  $p_{i,j,2}, p_{i,j,3}, p_{i,j,5}$ .
- For each  $b$ -type vertex we introduce a new copy of a  $K_{2,3}$  on five vertices of  $q$ -type, in the same way as we introduced the  $p$ -type vertices for the  $a$ -type vertices. Edges are added in a similar way, depending on the indices and the lists.
- We join all the  $p$ -type and  $q$ -type vertices with third indices 1, 2, 3 to all the  $p$ -type and  $q$ -type vertices with third indices 4, 5 to form a huge complete bipartite graph.

- We join all  $x$ -type vertices to all  $p$ -type and  $q$ -type vertices with third indices 1, 2, 3.
- We join all  $C$ -type vertices to all  $p$ -type and  $q$ -type vertices with third indices 4, 5.
- We pre-color all the  $p$ -type and  $q$ -type vertices according to their third index, so  $p_{i,j,\ell}$  will be pre-colored with color  $\ell \in \{1, 2, \dots, 5\}$ . Note that we can now in fact replace all lists introduced earlier by  $\{1, 2, \dots, 5\}$ , since the shorter lists will be forced by the given pre-coloring.

## 2.2 The Proofs for the Result on 5-Coloring

**Lemma 1.** *The graph  $G_I$  is  $P_6$ -free.*

*Proof.* Due to the page restrictions we can only sketch the proofs of this lemma and the claims that follow. We give a proof by contradiction. Suppose the graph  $G_I$  contains an induced subgraph  $H$  which is isomorphic to  $P_6$ . Then  $H$  contains at most three vertices from the set  $S$  of all  $p$ -type and  $q$ -type vertices; otherwise  $H$  would either contain a cycle, or an independent set of four vertices, or a vertex with degree at least three. Analogously,  $H$  contains at most three vertices from the set  $T$  of all  $C$ -type and  $x$ -type vertices. By similar arguments, one can show that  $H$  contains at most three vertices from  $S \cup T$ . We complete the proof by a series of claims followed by proof sketches.

**Claim 1.**  $H$  contains at most two vertices of  $S$ .

*Proof of Claim 1.* Suppose  $|V(H) \cap S| = 3$ . This implies  $H$  does not contain a vertex of  $T$ , so  $H$  contains three vertices from the set  $U$  of all  $a$ -type and  $b$ -type vertices. This is impossible and completes the proof of Claim 1.

**Claim 2.**  $H$  contains at most one vertex of  $S$ .

*Proof of Claim 2.* Suppose  $|V(H) \cap S| = 2$ . Then  $H$  contains at least one vertex of  $T$ ; otherwise  $|V(H)| \leq 4$ . So  $|V(H) \cap T| = 1$ , and  $H$  contains three vertices of  $U$ . If  $V(H) \cap S$  is an adjacent pair, the vertex of  $V(H) \cap T$  is adjacent to precisely one of them, and we easily obtain a contradiction. In the other case,  $(V(H) \cap S) \cup (V(H) \cap T)$  induces either a  $P_3$  or an independent set in  $H$ . Both cases lead to contradictions. This completes the proof of Claim 2.

**Claim 3.**  $H$  contains no vertex of  $S$ .

*Proof of Claim 3.* Suppose  $|V(H) \cap S| = 1$ . Then  $|V(H) \cap T| = 1$  or  $|V(H) \cap T| = 2$ . The first case is impossible since  $U$  is an independent set. For the second case first observe that common neighbors of two vertices from  $U$  can only be in  $T$ , and that two  $x$ -type vertices or two  $C$ -type vertices do not have a common neighbor in  $U$ . Noting that the three vertices of  $V(H) \cap U$  form an independent set, there are two possibilities for the remaining three vertices of  $H$ : they either induce an independent set in  $H$  or a  $P_2$  and a  $P_1$ . In the first case the two vertices of  $V(H) \cap T$  are either both  $x$ -type or both  $C$ -type vertices. This yields a contradiction. In the second case, the induced  $P_1$  can only result from a vertex in  $T$ , so the  $P_2$  is induced by a vertex from  $S$  and a vertex from  $T$ . Now the two vertices from  $V(H) \cap T$  must be both of  $x$ -type or both of  $C$ -type. But then both these vertices are adjacent to the vertex of  $V(H) \cap S$ , a contradiction. This completes the proof of Claim 3.

We conclude that  $H$  contains no vertex of  $S$  and at most three vertices of  $T$ . So  $H$  contains at least three vertices of  $U$  which form an independent set in  $H$ . This yields only one case:  $H$  contains precisely three vertices of  $T$  and precisely three vertices of  $U$ . By previous observations all vertices of  $V(H) \cap T$  must be of the same type, so they form an independent set as well. Recalling that two  $x$ -type vertices or two  $C$ -type vertices have no common neighbors in  $U$ , we obtain a contradiction. This completes the proof of Lemma 1.  $\square$

**Lemma 2.** *If  $I$  has a truth assignment in which each clause contains at least one true and at least one false literal, then the pre-coloring of  $G_I$  can be extended to a 5-coloring of  $G_I$ .*

*Proof.* Suppose  $I$  has a satisfying truth assignment in which each clause contains at least one true and at least one false literal. We use color 4 to color the  $x$ -type vertices representing the true literals and color 5 for the false literals. Now consider the lists assigned to the  $a$ -type and  $b$ -type vertices that come in pairs chosen from  $\{\{1, 4\}, \{2, 5\}\}, \{\{2, 4\}, \{3, 5\}\}, \{\{3, 4\}, \{1, 5\}\}$ . If the adjacent  $x$ -type vertex has color 4, color 1, 2 or 3 is forced on one of the adjacent  $a$ -type or  $b$ -type vertices, respectively, while on the other one we can use color 5; similarly, if the adjacent  $x$ -type vertex has color 5, color 2, 3 or 1 is forced on one of the adjacent  $a$ -type or  $b$ -type vertices, respectively, while on the other one we can use color 4. Since precisely two of the three  $x$ -type vertices of one clause gadget have the same color, this leaves at least one of the colors 1, 2 and 3 admissible for the  $C$ -type vertex representing the clause. By coloring the vertices associated with each clause and variable as described above, a 5-coloring of the pre-colored graph  $G_I$  is obtained. This completes the proof of Lemma 2.  $\square$

**Lemma 3.** *If the pre-coloring of  $G_I$  can be extended to a 5-coloring of  $G_I$ , then  $I$  has a satisfying truth assignment in which each clause contains at least one true and at least one false literal.*

*Proof.* Suppose we have a 5-coloring of the graph  $G_I$  that respects the pre-coloring and all lists assigned as indicated in the previous section. Then each of the  $x$ -type vertices has color 4 or 5, and each of the  $C$ -type vertices has color 1, 2 or 3. We define a truth assignment that sets a variable to TRUE if the corresponding  $x$ -type vertex has color 4, and to FALSE otherwise. Suppose one of the clauses contains only true literals. Then the three  $x$ -type vertices in the corresponding clause gadget of  $G_I$  all have color 4. Now consider the lists assigned to the  $a$ -type and  $b$ -type vertices of this gadget that come in pairs chosen from  $\{\{1, 4\}, \{2, 5\}\}, \{\{2, 4\}, \{3, 5\}\}, \{\{3, 4\}, \{1, 5\}\}$ . Since the adjacent  $x$ -type vertices all have color 4, colors 1, 2 and 3 are forced on three of the  $a$ -type and  $b$ -type vertices adjacent to the  $C$ -type vertex of this gadget, a contradiction, since the  $C$ -type vertex has color 1, 2 or 3. This proves that every clause contains at least one false literal. Analogously, it is easy to show that every clause contains at least one true literal. This completes the proof of Lemma 3.  $\square$

### 3 Pre-coloring Extension of 3-Coloring for $P_6$ -Free Graphs

In this section we show that the pre-coloring extension version of 3-coloring is polynomially solvable for  $P_6$ -free graphs. The key ingredient in our approach is the following

recently obtained characterization of  $P_6$ -free graphs [5]. Here a subgraph  $H$  of a graph  $G$  is said to be a dominating subgraph of  $G$  if every vertex of  $V(G) \setminus V(H)$  has a neighbor in  $H$ .

**Theorem 1** ([5]). *A graph  $G$  is  $P_6$ -free if and only if each connected induced subgraph of  $G$  on more than one vertex contains a dominating induced cycle on six vertices or a dominating (not necessarily induced) complete bipartite subgraph. Moreover, these dominating subgraphs can be obtained in polynomial time.*

A key ingredient in our approach is the following observation: it is checkable in polynomial time whether a pre-coloring of a graph  $G$  can be extended to a proper  $\ell$ -coloring of  $G$  as soon as the uncolored vertices of  $G$  have admissible lists of size at most 2. In this case the remaining decision problem can be modeled and solved as a 2SAT-problem. This approach has been introduced by Edwards [2] and is folklore now. It has been used especially for checking 3-colorability of graphs with small dominating sets (if such dominating sets can be found in polynomial time), e.g., for  $P_5$ -free graphs ([4]) and for  $P_6$ -free graphs ([10]). If there are a polynomial number of possible 3-colorings on the dominating set, then by exhaustively checking all of these colorings combined with solving the 2SAT-problem(s) on the remaining (dominated) vertices, this yields a polynomial time algorithm.

This obviously solves our problem in case the (component of the) instance graph contains a dominating  $C_6$ : all lists of admissible colors on the vertices in the beginning are subsets of  $\{1, 2, 3\}$  and after assuming a coloring on the  $C_6$  (respecting the pre-coloring, i.e., lists of size 1) all lists of admissible colors for the uncolored vertices have size at most 2, and we can model and solve the remaining problem as a 2SAT-problem. Although in the other case we cannot assume that the dominating complete bipartite graph has a bounded size, we can use a similar approach due to the special structure of  $P_6$ -free graphs. We will describe the procedure in more detail. Full details will appear in a journal version.

Suppose our instance graph  $G$  is connected (otherwise we treat the components of  $G$  separately), that we have lists of admissible colors from the set  $\{1, 2, 3\}$  on each vertex of  $G$ , and that we have constructed a dominating complete (not necessarily induced) bipartite graph  $H$  of  $G$  with bipartition classes  $A$  and  $B$ .

If there exists no 3-coloring of  $G$  (respecting a possible pre-coloring, i.e., respecting the given lists) in which one of  $A$  and  $B$  is monochromatic (i.e., every vertex of  $A$  or  $B$  receives the same color), then clearly  $G$  has no 3-coloring extending the pre-coloring, since we have to use at least 4 colors on  $H$ . Then we eventually obtain a NO answer after first trying all cases with  $A$  monochromatic and successively with  $B$  monochromatic, in the way we describe below.

Hence we can assume that  $A$  or  $B$  is monochromatic, and we can guess that  $A$  is monochromatic (if this does not result in a 3-coloring of  $G$  we can repeat the procedure assuming that  $B$  is monochromatic).

From now on we assume that all vertices of  $A$  are colored with color 1 (possibly after renaming the colors). We remove color 1 from all the lists of admissible colors at vertices of  $N(A) = \bigcup_{v \in A} N(v) \setminus A$ , we choose one vertex  $a \in A$  and delete all vertices of  $A \setminus \{a\}$ . We let  $R$  denote the subset of all remaining vertices with admissible lists of size 3. Clearly we are done with the graph (or component)  $G$  if  $R = \emptyset$ , as argued

above, simply by solving a 2SAT-problem defined on the uncolored vertices and all edges incident with these vertices.

So let us assume  $R \neq \emptyset$ . Clearly  $B \cap R = \emptyset$  because all vertices in  $B$  have a neighbor colored with color 1, so their admissible lists have size at most 2. It is now also clear that  $B$  dominates  $R$  (since  $A$  does not dominate any vertex of  $R$ ; otherwise the list of such a vertex would have been updated to size at most 2). Now let us consider the subgraph  $Q$  of  $G' = G - (A \setminus \{a\})$  induced by the vertices of  $V(G') \setminus (\{a\} \cup N(a))$ . In the remainder we redefine  $B := N(a)$  for convenience. If  $Q$  contains an isolated vertex  $v$  (i.e., a vertex with no neighbors in  $Q$ ) with a list containing color 1, then we can use color 1 on  $v$  and remove  $v$ . So, in particular we can assume that all isolated vertices of  $Q$  have admissible lists of size at most 2. We next analyze pairs of adjacent vertices of  $Q$ , and distinguish a number of cases.

**Case 1.**  $Q$  contains an edge  $pq$  such that  $p$  is adjacent to a vertex  $b \in B \setminus N(q)$  and  $q$  is adjacent to a vertex  $c \in B \setminus N(p)$ .

First note that the set  $S = \{a, b, c, p, q\}$  induces a  $C_5$  with possibly an additional edge  $bc$  in  $G'$ . If  $S$  dominates all vertices of  $R$ , we can just guess the eligible 3-colorings on  $S$  and solve our problem for the graph  $G'$  by solving a polynomial number of 2SAT-problems.

Supposing the contrary, let  $x \in R$  be a vertex that is not dominated by  $S$ . Since  $B$  dominates  $R$  there exists a vertex  $y \in B \setminus S$  with  $xy \in E(G')$ . Consider the paths  $xyabpq$  and  $xyacqp$  on six vertices. If  $yb \in E(G')$  or  $yc \in E(G')$ , then guessing a 3-coloring on  $S$  would also fix the eligible color on  $y$ , and reduce the list size on  $x$ . So if this would occur for all possible choices of  $x$  and  $y$ , we could solve our problem in polynomial time. It remains to consider the cases where  $yb \notin E(G')$  and  $yc \notin E(G')$ . Now since  $G'$  is  $P_6$ -free at least one of  $\{yp, yq\}$  is an edge of  $G'$ . If both are edges of  $G'$ , then, since in any 3-coloring of  $G'$  at least one of  $p$  and  $q$  receives color 2 or 3, any eligible 3-coloring on  $S$  will fix the eligible color on  $y$ , and reduce the list size on  $x$ . An analogous situation occurs when  $x, a$  and  $p$  share a common neighbor, and  $x, a$  and  $q$  share another common neighbor.

We next analyze the subcase in which there are pairs of vertices  $p', q' \in R$  not dominated by  $S$ , but where  $p', a, p$  have a common neighbor  $b'$  and  $q', a, q$  have a common neighbor  $c' \neq b'$  such that  $b'$  is not adjacent to either of  $\{b, c, q, q'\}$  and  $c'$  is not adjacent to either of  $\{b, c, p, p'\}$ . Now consider the path  $p'b'pqc'q'$ . If  $b'c' \in E(G')$ , then in any 3-coloring on  $S$  at least one of  $p$  and  $q$  receives color 2 or 3, and the eligible colors on  $b'$  and  $c'$  will be fixed, and the lists on  $p'$  and  $q'$  reduced. So we can deal comfortably with this subcase. Since  $G'$  is  $P_6$ -free, assuming  $b'c' \notin E(G')$  the only other possible subcase is that  $p'q' \in E(G')$ . But this yields a contradiction, since then  $\{q, c, a, b', p', q'\}$  induces a  $P_6$  in  $G'$ .

For the remainder of Case 1, we can now assume that the only subcase that has to be resolved is when all vertices of  $R$  that are not dominated by  $S$  (like  $x$  above) have no neighbor in common with both  $a$  and  $p$ , but only with  $a$  and  $q$ , or symmetrically. Then we can use the same approach as before if  $q$  receives color 2 or 3 in the guessed 3-coloring on  $S$ . If this does not result in a 3-coloring of  $G'$  in the end, we start the whole procedure (with color 1 on each vertex of  $A$ ) again after assigning color 1 to  $q$ ,

adjusting the lists on all vertices in  $N(q)$ , and removing the vertex  $q$ . This clearly yields a polynomial number of cases to check.

Concluding, for all subcases we analyzed in Case 1 except for one, we can propagate any 3-coloring on the set  $S$  to obtain a reduction of the list sizes of all vertices in  $R$ , and solve our problem using 2SAT-formulations (or obtain an obstruction to a 3-coloring at an earlier stage). In the other subcase, we either also get such a reduction or we can pre-color a specific new vertex and start the procedure on a smaller instance.

In the next case we assume that Case 1 does not apply, and we apply similar arguments. We skip the details due to page restrictions.

**Case 2.**  $Q$  contains an edge  $pq$  such that  $p$  is adjacent to a vertex  $b \in B \cap N(q)$  and  $q$  is adjacent to a vertex  $c \in B \setminus N(p)$ .

In the remainder we assume that neither Case 1 nor Case 2 applies. This implies that for each edge  $pq$  in  $Q$ , the vertices  $p$  and  $q$  have exactly the same neighbors in  $B$ , so by repeating the arguments this holds for all vertices in the same component of  $Q$ .

**Case 3.** All vertices in each component of  $Q$  have the same neighbors in  $B$ .

We start with the graph  $G'$  as above. As long as there exist or appear new vertices with lists of size 1 that are not in  $B \cup \{a\}$ , we do the following: for such a vertex  $v$  we adjust the lists of all vertices of  $N(v)$ , and then remove  $v$  (unless we can conclude that we cannot obtain a 3-coloring of  $G'$  extending the pre-coloring; then we stop and return to an earlier stage with a different guess on  $S$  or finally with  $B$  monochromatic instead of  $A$ ). Denote the resulting graph by  $G^*$ , and assume that in the remainder all neighborhoods, lists of admissible colors, subsets of vertices, etc. are with respect to  $G^*$ . In particular, let  $Q$  be the subgraph of  $G^*$  induced by the vertices of  $V(G^*) \setminus (\{a\} \cup B)$ . Recall that if  $Q$  contains an isolated vertex  $v$  with a list containing color 1, then we can use color 1 on  $v$  and remove  $v$ . So we can assume that all isolated vertices of  $Q$  have admissible lists not containing color 1.

Consider the set  $B' \subseteq B$  with vertices that have lists  $\{2, 3\}$ ; the other vertices of  $B$  have a fixed color, so every vertex dominated by such a vertex has a list of size 2.

Suppose  $C$  is a component of the subgraph  $G^*[B']$  induced by  $B'$  in  $G^*$ . Then clearly  $C$  is a bipartite graph (otherwise we arrive at the conclusion that  $G^*$  is not 3-colorable) with all lists equal to  $\{2, 3\}$ . So if we fix one color on a vertex of  $C$ , the other colors on  $C$  will also be fixed. If  $C'$  is another component of  $G^*[B']$  such that  $C$  and  $C'$  are connected by a path with internal vertices in  $Q$ , then fixing one color on a vertex of  $C$  will also fix the colors on  $C'$ : this is clear if  $C$  and  $C'$  have an isolated vertex  $v$  of  $Q$  as a common neighbor, since the list of  $v$  does not contain color 1; in the other case, it follows from the assumption that all vertices in each component of  $Q$  have the same neighbors in  $B$ , so the colors propagate from  $C$  to  $C'$  through subgraphs isomorphic to  $K_4$  minus an edge. We can split the checking whether the pre-coloring can be extended to a 3-coloring of  $G^*$  in separate disjoint problems now. Let  $\mathcal{C}$  denote a maximal set of components of  $G^*[B']$  that are connected by paths with internal vertices in  $Q$  that force the propagation of one fixed color in  $\mathcal{C}$  to fixed colors for all vertices in  $\mathcal{C}$ . Let  $\mathcal{D}$  denote all vertices of  $Q$  dominated by vertices of  $\mathcal{C}$ . Then fixing one color (so all colors) on  $\mathcal{C}$ , we can model the problem on  $\mathcal{D}$  as a 2SAT-problem. If this results in a YES answer, we can check the next maximal set of components, etc.; if for one of the sets we get a



NO answer, we try the swap of colors on this set; if we still get a NO answer, we repeat the whole procedure with color 1 on all vertices of  $B$  instead of  $A$ .

One readily checks that the above arguments can be turned into a polynomial algorithm for checking whether a pre-coloring on a  $P_6$ -free graph  $G$  can be extended to a 3-coloring of  $G$ .

## 4 6-Coloring for $P_7$ -Free Graphs

In this section we sketch how to prove that 6-coloring is NP-complete for  $P_7$ -free graphs. We use a reduction from 3-Satisfiability (3SAT).

We consider an arbitrary instance  $I$  of 3SAT and define a graph  $G_I$ , and next we show that  $G_I$  is  $P_7$ -free and that  $G_I$  is 6-colorable if and only if  $I$  has a satisfying truth assignment. Due to page restrictions we omitted the proofs but only give the construction of  $G_I$ . The complete proofs will appear in a full journal version.

Let  $I$  be an arbitrary instance of 3SAT with variables  $\{x_1, x_2, \dots, x_n\}$  and clauses  $\{C_1, C_2, \dots, C_m\}$ . We define a graph  $G_I$  corresponding to  $I$  based on the following construction.

- We introduce a gadget on 8 new vertices for each of the clauses, as follows: for clause  $C_j$  we introduce a gadget with vertex set:  $\{a_{j,1}, a_{j,2}, a_{j,3}, b_{j,1}, b_{j,2}, b_{j,3}, c_{j,1}, c_{j,2}\}$  and edge set:  $\{a_{j,1}a_{j,2}, a_{j,1}a_{j,3}, a_{j,2}a_{j,3}, a_{j,1}b_{j,1}, a_{j,2}b_{j,2}, a_{j,3}b_{j,3}, b_{j,1}c_{j,1}, b_{j,1}c_{j,2}, b_{j,2}c_{j,1}, b_{j,2}c_{j,2}, b_{j,3}c_{j,1}, b_{j,3}c_{j,2}, c_{j,1}c_{j,2}\}$ . We say that these vertices are of  $a$ -type,  $b$ -type and  $c$ -type.
- We introduce a gadget on 3 new vertices for each of the variables, as follows: for variable  $x_i$  we introduce a complete graph with vertex set  $\{x_i, \bar{x}_i, y_i\}$ . We say that these vertices are of  $x$ -type (both the  $x_i$  and the  $\bar{x}_i$  vertices) and of  $y$ -type.
- If clause  $C_j$  contains the variables  $x_i, x_k$  and  $x_r$ , we add three matching edges between the corresponding literal vertices (so  $x_i$  or  $\bar{x}_i$ , etc., depending on which of them appear in  $C_j$ ) and the three  $b$ -type vertices of the gadget corresponding to  $C_j$ . If  $b_{j,s}x_i$  or  $b_{j,s}\bar{x}_i$  has been added as an edge, we also add the edge  $b_{j,s}y_i$ , and analogously for  $x_k$  and  $x_r$ .
- We introduce three additional vertices  $d_1, d_2$  and  $z$ , and join  $d_1$  and  $d_2$  by an edge. We join all  $x_i$  to  $d_1$  by edges, and all  $\bar{x}_i$  to  $d_2$ .
- We join  $z$  to all vertices of  $y$ -type,  $a$ -type and  $c$ -type, and to  $d_1$  and  $d_2$ .
- We join all the  $x$ -type vertices and  $y$ -type vertices to all the  $a$ -type and  $c$ -type vertices.
- Finally, we join  $d_1$  and  $d_2$  to all the  $a$ -type,  $b$ -type and  $c$ -type vertices.

## 5 Conclusions and Open Problems

We proved that the pre-coloring extension version of 5-coloring remains NP-complete for  $P_6$ -free graphs. Results of Hoàng et al. [4] imply that this problem is polynomially solvable on  $P_5$ -free graphs. They show that  $\ell$ -coloring for any fixed  $\ell$  is polynomially solvable on  $P_5$ -free graphs. In contrast, determining the chromatic number (i.e., the

smallest  $\ell$  such that the graph is  $\ell$ -colorable) is NP-hard on  $P_5$ -free graphs [7]. We also showed that the pre-coloring extension version of 3-coloring is polynomially solvable for  $P_6$ -free graphs. Finally, we proved that 6-coloring is NP-complete for  $P_7$ -free graphs. This problem was known to be polynomially solvable for  $P_5$ -free graphs and NP-complete for  $P_8$ -free graphs. This leaves the natural open problem for 6-coloring on  $P_6$ -free graphs. Also the complexity of 4-coloring and 5-coloring on  $P_6$ -free graphs are open problems. We refer to [9] for the most recent table of the complexity status of  $\ell$ -coloring for  $P_k$ -free graphs: the problem is trivially in P for arbitrary fixed  $\ell$  if  $k \leq 2$ ; it is also in P for fixed  $k \leq 5$  and arbitrary fixed  $\ell$ , and for  $k = 6$  and  $\ell = 3$ ; it is NP-complete for  $\ell = 4$  and any  $k \geq 9$ , for  $\ell = 5$  and  $k \geq 8$ , for  $\ell \geq 6$  and  $k \geq 8$  (and by our result also for  $k = 7$ ). Interesting questions are: what is the complexity of 4-coloring for  $P_6$ -free graphs, of 3-coloring for  $P_7$ -free graphs; does there exist an integer  $k$  such that 3-coloring is NP-complete for  $P_k$ -free graphs? What is the complexity of 5-coloring for  $P_7$ -free graphs, and of 4-coloring for  $P_8$ -free graphs? We finish this paper with two other open problems on 3-coloring that have intrigued many researchers: the complexity of 3-coloring is open for graphs with diameter 2, and for graphs with diameter 3.

## References

1. Bondy, J.A., Murty, U.S.R.: Graph Theory. Springer Graduate Texts in Mathematics, vol. 244 (2008)
2. Edwards, K.: The complexity of coloring problems on dense graphs. *Theor. Comp. Science* 43, 337–343 (1986)
3. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco (1979)
4. Hoàng, C.T., Kamiński, M., Lozin, V., Sawada, J., Shu, X.: Deciding  $k$ -colorability of  $P_5$ -free graphs in polynomial time. *Algorithmica* (in press), doi:10.1007/s00453-008-9197-8
5. van 't Hof, P., Paulusma, D.: A new characterization of  $P_6$ -free graphs. *Discrete Appl. Math.* (in press), doi:10.1016/j.dam.2008.08.025
6. Hujter, M., Tuza, Z.: Pre-coloring extension II: graph classes related to bipartite graphs. *Acta Math. Univ. Comen.* 62, 1–11 (1993)
7. Král', D., Kratochvíl, J., Tuza, Z., Woeginger, G.J.: Complexity of coloring graphs without forbidden induced subgraphs. In: Brandstädt, A., Van Bang Le (eds.) WG 2001. LNCS, vol. 2204, pp. 254–262. Springer, Heidelberg (2001)
8. Kratochvíl, J.: Pre-coloring extension with fixed color bound. *Acta Math. Univ. Comen.* 62, 139–153 (1993)
9. Le, V.B., Randerath, B., Schiermeyer, I.: On the complexity of 4-coloring graphs without long induced paths. *Theor. Comp. Science* 389, 330–335 (2007)
10. Randerath, B., Schiermeyer, I.: 3-Colorability  $\in P$  for  $P_6$ -free graphs. *Discrete Appl. Math.* 136, 299–313 (2004)
11. Randerath, B., Schiermeyer, I.: Vertex colouring and forbidden subgraphs - a survey. *Graphs and Combin.* 20, 1–40 (2004)
12. Tucker, A.: A reduction procedure for coloring perfect  $K_4$ -free graphs. *J. Combin. Theory B* 43, 151–172 (1987)
13. Tuza, Z.: Graph colorings with local restrictions - a survey. *Discuss. Math. Graph Theory* 17, 161–228 (1997)
14. Woeginger, G.J., Sgall, J.: The complexity of coloring graphs without long induced paths. *Acta Cybern.* 15, 107–117 (2001)



# Fully Decomposable Split Graphs

Hajo Broersma<sup>1</sup>, Dieter Kratsch<sup>2</sup>, and Gerhard J. Woeginger<sup>3,\*</sup>

<sup>1</sup> Department of Computer Science, Durham University, Science Laboratories,  
South Road, Durham, DH1 3LE England

`hajo.broersma@durham.ac.uk`

<sup>2</sup> LITA, Université de Metz, 57045 Metz Cedex 01, France

`kratsch@lita.univ-metz.fr`

<sup>3</sup> Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513,  
5600 MB Eindhoven, The Netherlands

`gwoegi@win.tue.nl`

**Abstract.** We discuss various questions around partitioning a split graph into connected parts. Our main result is a polynomial time algorithm that decides whether a given split graph is fully decomposable, i.e., whether it can be partitioned into connected parts of order  $\alpha_1, \alpha_2, \dots, \alpha_k$  for every  $\alpha_1, \alpha_2, \dots, \alpha_k$  summing up to the order of the graph. In contrast, we show that the decision problem whether a given split graph can be partitioned into connected parts of order  $\alpha_1, \alpha_2, \dots, \alpha_k$  for a given partition  $\alpha_1, \alpha_2, \dots, \alpha_k$  of the order of the graph, is NP-hard.

**Keywords:** graph decomposition, integer partition, computational complexity.

## 1 Introduction

Throughout we only consider finite undirected graphs without loops or multiple edges. Let  $G = (V, E)$  be a graph on  $n$  vertices, and let  $\alpha = (\alpha_1, \dots, \alpha_k)$  denote a partition of  $n$ , that is, a sequence of positive integers  $\alpha_1, \dots, \alpha_k$  with  $\sum_{i=1}^k \alpha_i = n$ . The graph  $G$  is called  $\alpha$ -decomposable, if there exists a partition of  $V$  into disjoint subsets  $A_1, \dots, A_k$  of cardinality  $|A_i| = \alpha_i$  for  $1 \leq i \leq k$  such that every set  $A_i$  induces a connected subgraph of  $G$ . Such a partition is called an  $\alpha$ -decomposition of  $G$ , and a (connected) subgraph induced by  $|A_i| = \alpha_i$  vertices is also referred to as an  $\alpha_i$ -component of the  $\alpha$ -decomposition. A graph is called *fully decomposable* (or *arbitrarily vertex decomposable*) if it is  $\alpha$ -decomposable for every partition  $\alpha$  of  $n$ .

Fully decomposable graphs were introduced by Horňák & Woźniak [6]. There are two natural algorithmic questions centered around  $\alpha$ -decompositions of graphs.

---

\* This research has been supported by the Netherlands Organisation for Scientific Research (NWO), grant 639.033.403, and by BSIK grant 03018 (BRICKS: Basic Research in Informatics for Creating the Knowledge Society).

- Q1: Decide whether a given graph  $G$  is  $\alpha$ -decomposable for a given partition  $\alpha$ .
- Q2: Decide whether a given graph  $G$  is fully decomposable.

Question Q1 is notoriously hard. For instance, Dyer & Frieze [3] proved that it is NP-hard to decide whether a planar graph is  $(3, 3, \dots, 3)$ -decomposable. Barth & Fournier [2] showed that Q1 is NP-hard for trees. Generally speaking, Q1 seems to be NP-hard for every natural non-trivial class of specially structured graphs.

The computational complexity of question Q2 is not understood. We are aware of only a single result on question Q2 from the literature: Barth, Baudon & Puech [1] designed a polynomial time algorithm for deciding whether a given tripod (a tree with a single vertex of degree three, and all other vertices of degree one or two) is fully decomposable. Barth & Fournier [2] also proved that every fully decomposable tree has maximum vertex degree at most four. Determining the precise computational complexity of Q2 is an outstanding open problem: The problem is neither known to be NP-hard, nor is it known to be contained in the class NP.

## 2 Results of This Paper

A graph  $G = (V, E)$  is a *split graph* (see for instance Golumbic [5]) if its vertex set can be partitioned into an induced independent set  $I$  and a clique  $C$ . Often split graphs are specified in the form  $G = (C, I, E)$ .

In this paper, we will resolve the computational complexity of questions Q1 and Q2 for split graphs: The following two theorems show that for this graph class Q1 is hard, whereas Q2 is easy.

**Theorem 1.** *It is NP-hard to decide whether a given split graph with  $n$  vertices is  $\alpha$ -decomposable for a given partition  $\alpha$  of  $n$ .*

**Theorem 2.** *It can be decided in polynomial time whether a given split graph with  $n$  vertices is fully decomposable.*

Theorem 1 will be proved in the following Section 3, and the key ingredients for the proof of Theorem 2 will be presented in the remaining part of this paper. The complete proof of the latter result will appear in a full journal version of this paper.

## 3 The Hardness Proof

In this section we will prove Theorem 1. The reduction is done from the following version of the NP-hard  $d$ -DIMENSIONAL MATCHING problem; see Garey & Johnson [4].

PROBLEM:  $d$ -DIMENSIONAL MATCHING ( $d$ -DM)

INPUT: A ground set  $X = \{x_1, \dots, x_{qd}\}$  of  $qd$  elements; a family  $\mathcal{S}$  of  $d$ -element subsets  $S_1, \dots, S_\ell$  of  $X$ .

QUESTION: Can set  $X$  be partitioned into  $q$  disjoint subsets from  $S_1, \dots, S_\ell$ ?

For an instance of this problem  $d$ -DM, we now construct the following corresponding split graph.

- For every element  $x \in X$ , the independent set  $I$  contains a corresponding vertex  $i(x)$ . Furthermore, in the independent set  $I$  there are  $\ell - q$  groups  $D_1, \dots, D_{\ell-q}$  of dummy vertices; every such group consists of exactly  $d - 1$  vertices.
- For every set  $S$  in the family  $\mathcal{S}$ , the clique  $C$  contains a corresponding vertex  $c(S)$ . Furthermore, the clique  $C$  contains  $\ell - q$  dummy vertices  $c_1, \dots, c_{\ell-q}$ .
- There is an edge between any two vertices in the clique  $C$ .
- Whenever  $x \in S$  for some  $x \in X$  and some  $S \in \mathcal{S}$ , there is an edge between  $i(x)$  and  $c(S)$ .
- Furthermore, for  $k = 1, \dots, \ell - q$  the dummy vertex  $c_k$  is joined to the  $d - 1$  dummy vertices in the group  $D_k$ .

The resulting split graph  $G$  has  $2\ell - q$  vertices in  $C$ , has  $(d - 1)\ell + q$  vertices in  $I$ , and thus consists altogether of  $(d + 1)\ell$  vertices. Finally, we define the vector  $\alpha = (d + 1, d + 1, \dots, d + 1)$  that consists of  $\ell$  components of value  $d + 1$ . We claim that the split graph  $G$  is  $\alpha$ -decomposable, if and only if the instance of  $d$ -DM has answer YES.

First assume that the instance of  $d$ -DM has answer YES. Consider the partition of  $X$  into  $q$  subsets from  $\mathcal{S}$ . For every set  $S$  occurring in this partition, we put vertex  $c(S)$  together with all vertices  $i(x)$  with  $x \in S$  into one connected component. For every set  $S$  not occurring in this partition, we put vertex  $c(S)$  together with one of the dummy vertices  $c_k$  and the vertices in the group  $D_k$  into one connected component. This yields that  $G$  is  $\alpha$ -decomposable.

Next assume that the graph  $G$  is  $\alpha$ -decomposable. Every dummy vertex  $c_k$  must be in the same connected component with the vertices in group  $D_k$ , and with exactly one of the vertices  $c(S)$ . This leaves  $q$  of the vertices  $c(S)$  unmatched, and each of them must be in one connected components with  $d$  vertices  $i(x)$  of the independent set. This yields the desired partition of the set  $X$ .

This completes the NP-hardness argument, and the proof of Theorem [□](#). Since  $d$ -DM is NP-hard for every fixed  $d \geq 3$ , we have actually established the following stronger statement.

**Corollary 1.** *For every fixed integer  $f \geq 4$ , it is NP-hard to decide whether a given split graph on  $qf$  vertices is  $\alpha$ -decomposable with respect to the vector  $\alpha = (f, f, \dots, f)$  consisting of  $q$  components of value  $f$ . □*

The following sections will show that the statement in Corollary [□](#) is essentially strongest possible: For  $f \leq 3$ , the corresponding decomposition problem allows a polynomial time solution.

## 4 Primitive Partitions

For  $n \geq 2$ , a partition  $\alpha$  of  $n$  is called *2-3-primitive*, if it is of one of the following forms.

- $\alpha = (1, 3, 3, \dots, 3)$  consists of threes and a single one;
- $\alpha = (2, \dots, 2, 3, 3, \dots, 3)$  only consists of twos and threes.

The following lemma shows that for analyzing the full decomposability of a split graph, we can restrict our attention to 2-3-primitive partitions.

**Lemma 1.** *A split graph on  $n$  vertices is fully decomposable, if and only if it is  $\alpha$ -decomposable for every 2-3-primitive partition  $\alpha$  of  $n$ .*

*Proof.* The only-if-statement is implicit in the definition of a fully decomposable graph. For the if-statement, we recall that every integer  $\ell \geq 2$  can be written in the form  $\ell = 2a + 3b$  with non-negative integers  $a$  and  $b$ . Consider an arbitrary partition  $\alpha = (\alpha_1, \dots, \alpha_k)$  of  $n$ . Replace every  $\alpha_i \geq 2$  in  $\alpha$  by a partition of  $\alpha_i$  into  $a_i$  twos and  $b_i$  threes. Let  $\alpha_0$  denote the number of 1s in the vector  $\alpha$ . If  $\alpha_0 \geq 2$ , then replace the 1s in vector  $\alpha$  by a partition of  $\alpha_0$  into  $a_0$  twos and  $b_0$  threes. If  $\alpha_0 \leq 1$ , then leave the 1s untouched. The resulting new partition  $\alpha' = (\alpha'_1, \dots, \alpha'_m)$  of  $n$  is of the form  $(1, 3, 3, \dots, 3)$  or  $(2, \dots, 2, 3, 3, \dots, 3)$ , and hence 2-3-primitive. By assumption the split graph  $G$  is  $\alpha'$ -decomposable. We let  $A'_1, \dots, A'_m$  denote the corresponding connected vertex sets. Every set  $A'_j$  with  $\alpha'_j = |A'_j| \geq 2$  contains at least one clique-vertex; therefore, the union of the  $a_i$  two-element sets and the  $b_i$  three-element sets corresponding to component  $\alpha_i$  is a connected vertex set  $A_i$  with  $\alpha_i$  elements. This yields that  $G$  is  $\alpha$ -decomposable.  $\square$

We note that Lemma 1 already implies an NP-certificate for deciding whether an  $n$ -vertex split graph is fully decomposable: The certificate lists all 2-3-primitive partitions of  $n$  together with the corresponding decompositions into connected parts. The following sections prove even stronger results.

## 5 Canonical Primitive Partitions

Next let us introduce *canonical primitive partitions* as a crucial subfamily of the 2-3-primitive partitions. Let  $n \geq 2$  be an integer.

- If  $n = 2k$  is even, then the canonical 2-primitive partition of  $n$  consists of  $k$  twos.
  - If  $n = 2k + 1$  is odd, then the canonical 2-primitive partition of  $n$  consists of  $k - 1$  twos and a single three.
- If  $n = 3k$ , then the canonical 3-primitive partition of  $n$  consists of  $k$  threes.
  - If  $n = 3k + 1$ , then the canonical 3-primitive partition of  $n$  consists of  $k$  threes and a single one.
  - If  $n = 3k + 2$ , then the canonical 3-primitive partition of  $n$  consists of  $k$  threes and a single two.

The following lemma strengthens the statement of Lemma 1.

**Lemma 2.** *A split graph with  $n$  vertices is fully decomposable, if and only if it is  $\alpha$ -decomposable for the canonical 2-primitive partition  $\alpha$  of  $n$  and for the canonical 3-primitive partition  $\alpha$  of  $n$ .*

The rest of this section is dedicated to the proof of Lemma 2. We first introduce some additional notation and terminology.

We use  $2^r 3^s$  to denote a partition of  $n = 2r + 3s$  into  $r$  (possibly  $r = 0$ ) twos and  $s$  (possibly  $s = 0$ ) threes. A partition of  $n = 3k + 1$  into  $k$  threes and 1 one is denoted by  $13^k$ .

Suppose  $G = (C, I, E)$  is a split graph and  $H$  is a subgraph of  $G$ . Then a vertex of  $V(H) \cap C$  or  $V(H) \cap I$  is called a  $C$ -vertex or  $I$ -vertex of  $H$ , respectively. Analogously, we call a neighbor  $u$  of a vertex  $v \in V(G)$  a  $C$ -neighbor or  $I$ -neighbor of  $v$  if  $u \in C$  or  $u \in I$ , respectively. If  $|V(H)| = 3$ , we say that  $H$  is a  $T_i^c$  if  $|V(H) \cap C| = c$  and  $|V(H) \cap I| = i$ ; in the special case that  $c = 2$  and  $i = 1$  we add a bar (only) if  $T_1^2$  is a triangle, so we use  $\overline{T}_1^2$  instead of  $T_1^2$  if and only if the three vertices induce a triangle in  $G$ .

For proving Lemma 2 it is sufficient to prove the following result.

**Lemma 3.** *If a split graph  $G$  with  $n$  vertices is  $\alpha$ -decomposable for the canonical 2-primitive partition  $\alpha$  of  $n$  and for the canonical 3-primitive partition  $\alpha$  of  $n$ , then  $G$  is  $\alpha$ -decomposable for every 2-3-primitive partition  $\alpha$  of  $n$ .*

*Proof.* Let  $G = (C, I, E)$  be a split graph on  $n$  vertices, and assume that  $G$  is  $\alpha$ -decomposable for the canonical 2-primitive partition  $\alpha$  of  $n$  and for the canonical 3-primitive partition  $\alpha$  of  $n$ . First note that we may assume that  $n \geq 10$ ; if  $n < 10$  then the only possible 2-3-primitive partitions are the canonical 2-primitive and the canonical 3-primitive partitions. Secondly, note that  $G$  has a matching saturating at least  $|I| - 1$  vertices of  $I$  (and all vertices of  $I$  if  $n$  is even); since  $I$  is an independent set, this follows immediately from the hypothesis that  $G$  is  $\alpha$ -decomposable for the canonical 2-primitive partition  $\alpha$  of  $n$ . This also implies that  $|C| \geq |I| - 1$ .

**Definition 1.** *We say that  $G$  is (3, 3)-reducible if and only if it has the following property: If  $G$  is  $2^r 3^s$ -decomposable for some  $r \geq 0$  and  $s \geq 4$ , then it is also  $2^{r+3} 3^{s-2}$ -decomposable.*

*Similarly, we say that  $G$  is (1, 3)-reducible if and only if  $G$  has the following property: If  $G$  is  $13^k$ -decomposable for some  $k \geq 3$ , then it is also  $2^2 3^{k-1}$ -decomposable.*

Note that in the language of this definition, it is now sufficient to prove that  $G$  is both (3, 3)-reducible and (1, 3)-reducible. The following two claims establish these facts, and thus complete the proof of Lemma 3.

*Claim.*  $G$  is (3, 3)-reducible.

*Proof.* Suppose  $G$  has a  $2^r 3^s$ -decomposition  $\alpha$  with  $r \geq 0$  and  $s \geq 4$ . Then at least two of the 3-components in  $\alpha$  have at least two  $C$ -vertices, since  $|C| \geq |I| - 1$ .

It is obvious how to decompose the subgraph of  $G$  induced by the six vertices of two such 3-components into three 2-components.

*Claim.*  $G$  is  $(1, 3)$ -reducible.

*Proof.* Suppose  $G$  has a  $13^k$ -decomposition  $\alpha$  with  $k \geq 3$ . Then at least one of the 3-components in  $\alpha$  has at least two  $C$ -vertices, since  $|C| \geq |I| - 1$ . Let  $H$  denote such a 3-component, and let  $v$  denote the vertex of the 1-component in  $\alpha$ .

If  $v \in C$  it is clear how to decompose the subgraph of  $G$  induced by  $V(H) \cup \{v\}$  into two 2-components.

Next suppose  $v \in I$ . Clearly,  $v$  is not an isolated vertex since  $G$  is  $\alpha$ -decomposable for the canonical 2-primitive partition  $\alpha$  of  $n$ . Let  $u$  be a  $C$ -neighbor of  $v$ . If  $u$  is in a  $\overline{T}_1^2$  or  $T_0^3$  of  $\alpha$ , or if it is the vertex with degree 1 in a  $T_1^2$  of  $\alpha$ , then it is again clear how to decompose the subgraph of  $G$  induced by  $v$  and the vertices of the 3-component containing  $u$  into two 2-components.

If  $u$  is the vertex with degree 2 in a  $T_1^2$  of  $\alpha$ , we use that  $\alpha$  contains at least one other 3-component  $H'$  with at least two  $C$ -vertices, since  $|C| \geq |I| - 1$  and  $v \in I$ . In this case we can combine  $v$  with  $u$  and its  $I$ -neighbor in  $T_1^2$  into a 3-component, and we can decompose the subgraph of  $G$  induced by the remaining vertex of this  $T_1^2$  and the vertices of  $H'$  into two 2-components.

A similar transformation along a longer chain of 3-components can be used in the remaining case where  $u$  is the  $C$ -vertex of a  $T_2^1$ . In this case the existence of a matching that saturates at least  $|I| - 1$  vertices of  $I$  implies there is an alternating path  $P = v_1 v_2 \dots v_{2t}$  starting at  $v = v_1$  and terminating at a vertex  $w = v_{2t}$  in a  $\overline{T}_1^2$ ,  $T_1^2$  or  $T_0^3$ , in which each  $v_{2j}$  with  $1 \leq j < t$  is the  $C$ -vertex of a  $T_2^1$  and each  $v_{2j+1}$  with  $1 \leq j < t$  is an  $I$ -vertex adjacent to  $v_{2j}$  in the corresponding  $T_2^1$  for  $t - 1$  disjoint 3-components isomorphic to  $T_2^1$ . The chain of these  $t - 1$  copies of a  $T_2^1$  without the vertex  $v_{2t-1}$  together with the vertex  $v$  and the edges  $v_{2j-1} v_{2j}$  with  $1 \leq j < t$  can be transformed into  $t - 1$  new  $T_2^1$ s by swapping the edges of  $P$  (meaning that we include all edges  $v_{2j-1} v_{2j}$  with  $1 \leq j < t$  and remove all edges  $v_{2j} v_{2j+1}$  with  $1 \leq j < t$ ). The remaining vertex  $v_{2t-1}$  and the 3-component  $H_w$  containing  $w$  can be treated as before, yielding a decomposition of the subgraph of  $G$  induced by  $V(H_w) \cup \{v_{2t-1}\}$  into two 2-components in case  $w$  is not the vertex with degree 2 in a  $T_1^2$ ; otherwise we use again that  $\alpha$  contains at least one other 3-component  $H'$  with at least two  $C$ -vertices. In this case we can combine  $v_{2t-1}$  with  $w$  and its  $I$ -neighbor in  $T_1^2$  into a 3-component, and we can decompose the subgraph of  $G$  induced by the remaining vertex of this  $T_1^2$  and the vertices of  $H'$  into two 2-components.  $\square$

## 6 The Polynomial Time Result

From the previous sections we conclude that to prove Theorem 2 it now suffices to prove the following two lemmas. The first lemma can be proved by straightforward matching techniques, but for the second lemma we have to apply a more sophisticated technique closely related to matching theory. We leave the details for the full journal paper version, but indicate the main tool which can be found

in, e.g., Lovász & Plummer [7]: Let  $G' = (V', E')$  be an edge-weighted graph, and for every vertex  $v \in V'$  let  $d(v)$  be a non-negative integer. Then we can determine in polynomial time a maximum-weight subset  $F' \subseteq E'$  of the edges, such that in the graph  $(V', F')$  every vertex  $v$  has degree  $d(v)$ , or find out that no such set  $F'$  exists.

**Lemma 4.** *Let  $G = (V, E)$  be a split graph on  $n$  vertices, and let  $\alpha$  be the canonical 2-primitive partition of  $n$ . Then it can be decided in polynomial time whether  $G$  is  $\alpha$ -decomposable.*

*Proof.* This boils down to a bipartite matching problem. If  $n$  is even, we need to find a matching from the independent set  $I$  into the clique  $C$ . If  $n$  is odd, then we check all possibilities for the extra component with three vertices.  $\square$

**Lemma 5.** *Let  $G = (V, E)$  be a split graph on  $n$  vertices, and let  $\alpha$  be the canonical 3-primitive partition of  $n$ . Then it can be decided in polynomial time whether  $G$  is  $\alpha$ -decomposable.*

## 7 Conclusions

We have settled the complexity of recognizing fully decomposable split graphs. We feel that it might be very difficult to come up with other graph classes for which this problem is tractable. The algorithm of Barth, Baudon & Puech [1] for recognizing fully decomposable tripodes (trees with a single vertex of degree three, and all other vertices of degree one or two) is highly non-trivial. Unfortunately, many other graph classes contain graphs with a similar connectivity structure as tripodes (with respect to full decomposability); hence settling the problem for these classes would amount to generalizing the proof of [1].

Let us illustrate this claim for the class of co-graphs. Consider a tripode  $T$  that consists of a root and three paths with  $\ell_1$ ,  $\ell_2$ , and  $\ell_3$  vertices, respectively. We define a corresponding co-graph  $G(T)$  that consists of three independent cliques with  $\ell_1$ ,  $\ell_2$ , and  $\ell_3$  vertices, and a single vertex that is connected to all vertices in the cliques. It can be seen that the tripode  $T$  is fully decomposable if and only if the co-graph  $G(T)$  is fully decomposable. We pose the computational complexity of recognizing fully decomposable co-graphs as an open problem.

Furthermore, we are not aware of any natural NP-certificates or coNP-certificates for deciding full decomposability of general graphs. In fact, this problem might be located in one of the complexity classes above NP (see for instance Chapter 17 in Papadimitriou's book [8]). If the problem is hard, then the complexity class  $\text{DP}=\text{BH}_2$ , the second level of the Boolean Hierarchy, might perhaps be a reasonable guess.

Finally, we will formulate a conjecture that would imply that the problem is easy. Let us call a vector  $\alpha$  with positive integer components *balanced*, if  $k-1$  of these components are equal to each other, and the last component does not exceed the other components. We did not manage to construct a counter-example to the following bold conjecture.

*Conjecture 1.* An  $n$ -vertex graph  $G$  is fully decomposable, if and only if  $G$  is  $\alpha$ -decomposable for every balanced vector  $\alpha$  whose components add up to  $n$ .

If this conjecture turns out to be true (for which admittedly we do not have the slightest evidence), then this would yield an NP-certificate for fully decomposable graphs: There are only  $O(n)$  many balanced vectors  $\alpha$  whose components add up to the number  $n$  of vertices in a graph. The  $\alpha$ -decompositions for these  $O(n)$  vectors form a certificate of polynomial length that can easily be verified in polynomial time.

## References

1. Barth, D., Baudon, O., Puech, J.: Decomposable trees: A polynomial algorithm for tripodes. *Discrete Applied Mathematics* 119, 205–216 (2002)
2. Barth, D., Fournier, H.: A degree bound on decomposable trees. *Discrete Mathematics* 306, 469–477 (2006)
3. Dyer, M.E., Frieze, A.M.: On the complexity of partitioning graphs into connected subgraphs. *Discrete Applied Mathematics* 10, 139–153 (1985)
4. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco (1979)
5. Golubic, M.C.: *Algorithmic graph theory and perfect graphs*. Academic Press, New York (1980)
6. Horňák, M., Woźniak, M.: Arbitrarily vertex decomposable trees are of maximum degree at most six. *Opuscula Mathematica* 23, 49–62 (2003)
7. Lovász, L., Plummer, M.D.: *Matching Theory*. *Annals of Discrete Mathematics*, vol. 29. North-Holland, Amsterdam (1986)
8. Papadimitriou, C.H.: *Computational Complexity*. Addison-Wesley, Reading (1994)



# Feedback Vertex Set on Graphs of Low Cliquewidth\*

Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle

Department of Informatics, University of Bergen, Norway  
{buixuan,telle,vatshelle}@ii.uib.no

**Abstract.** The Feedback Vertex Set problem asks whether a graph contains  $q$  vertices meeting all its cycles. This is not a local property, in the sense that we cannot check if  $q$  vertices meet all cycles by looking only at their neighbors. Dynamic programming algorithms for problems based on non-local properties are usually more complicated. In this paper, given a graph  $G$  of cliquewidth  $cw$  and a  $cw$ -expression of  $G$ , we solve the Minimum Feedback Vertex Set problem in time  $O(n^2 2^{2cw^2 \log cw})$ . Our algorithm applies a non-standard dynamic programming on a so-called  $k$ -module decomposition of a graph, as defined by Rao [26], which is easily derivable from a  $k$ -expression of the graph. The related notion of module-width of a graph is tightly linked to both cliquewidth and nlc-width, and in this paper we give an alternative equivalent characterization of module-width.

## 1 Introduction

The problem of finding a minimum Feedback Vertex Set (FVS) in a graph, i.e. the smallest set of vertices whose removal results in a graph that has no cycles, has many applications, for example to optical networks [19], circuit testing, deadlock resolution, analyzing manufacturing processes and computational biology (see [8] and its bibliography). It is one of the classical NP-complete problems from the 1972 list of Karp [18] and has been extensively studied from many viewpoints, including linear programming [6], approximation algorithms [2,10,13,19], exact algorithms [11] and parameterized complexity [5,8,14,25].

The minimum FVS problem is 2-approximable in polynomial time [1]. The fastest exact algorithm has runtime  $O(1.7548^n)$  [11]. The fastest FPT (Fixed Parameter Tractable) algorithm when parameterized by the size  $q$  of the FVS has runtime  $O(5^q q n^2)$  [5]. These algorithmic results are quite strong, but are not useful for cases of input graphs having a large number of vertices  $n$ , and a large minimum FVS  $q$ , if we want the actual smallest FVS. For such cases we may instead hope that the input graph has a bounded width parameter. For example, if  $G$  is a planar graph of treewidth  $tw$  then Kloks et al [20] give a dynamic programming algorithm solving minimum FVS on  $G$  in time  $O(2^{O(tw \log tw)} n)$ . A similar algorithm can be devised also for non-planar  $G$  of treewidth  $tw$ , given with an

---

\* Supported by the Norwegian Research Council, project PARALGO.

optimal tree-decomposition, but it is an open problem if algorithms with runtime  $O(2^{O(tw)}n)$  exist for minimum FVS, even though such algorithms exist for a large variety of NP-hard problems. However, for minimum FVS it would require a small breakthrough to get such an algorithm. One reason for this is that FVS is not a locally checkable property, in the sense that if given  $q$  vertices we cannot check that they form an FVS simply by looking at the neighbors of these  $q$  vertices. In this paper we consider instead graphs of cliquewidth  $cw$ , that encompasses large classes of graphs of unbounded treewidth, and for which powerful algorithmic results are known. For instance, we have that any graph problem expressible in  $MSO_1$ -logic, as is the case with minimum FVS, is FPT when parameterized by cliquewidth (roughly, apply [17], then [23, Proposition 6.3], then [7]).

In this paper we will be interested in as low exponential dependency on  $cw$  as possible, and for this we need to use a specially designed dynamic programming algorithm. Dynamic programming based on decompositions having small cliquewidth are usually more complicated than dynamic programming on decompositions having small treewidth. The algorithm we give solves minimum FVS on a graph  $G$  of cliquewidth  $cw$  in time  $O(2^{2cw^2 \log cw} n^2)$ , when given a  $cw$ -expression of  $G$  which is a decomposition of the graph showing that it has cliquewidth  $cw$ . The only other problems for which  $O(\text{poly}(n)2^{\text{poly}(cw)})$  algorithms exist are for problems based on domination-type properties, like Dominating Set in [21] and a class of vertex partitioning problems as in [4].

Cliquewidth is related to the notion of nlc-width of a graph [9] with which it shares most properties but we have chosen to use cliquewidth in this paper simply because that notion is more famous. Our algorithm applies a non-standard dynamic programming on a so-called  $k$ -module decomposition of a graph, as defined by Rao [26], which is easily derivable from a  $k$ -expression of the graph. The related notion of module-width of a graph is tightly linked to both cliquewidth and nlc-width, and in this paper we give an alternative equivalent characterization of module-width. Our dynamic programming algorithm is non-standard in the sense that we index tables by the classes of one equivalence relation on the set of possible solutions, but store optimal solutions at these indices that are related to another equivalence relation which is a coarsening of the first one. We need to do this in order to achieve the stated runtime.

## 2 Framework

Let  $G$  be a graph with vertex set  $V(G)$  and edge set  $E(G)$ . Consider the following unifying decomposition framework for several decomposition schemes. A binary tree is a rooted tree where every internal node has exactly two children.

**Definition 1 (Decomposition Tree).** A rooted decomposition tree of a graph  $G$  is a pair  $(T, \delta)$  where  $T$  is a binary tree having  $n = |V(G)|$  leaves and  $\delta$  is a bijection between the vertices of  $G$  and the leaves of  $T$ .

Roughly, trees with their leaves in a bijection with the vertices of  $G$  are important for techniques like divide-and-conquer or dynamic programming since

they show how to “divide” the graph instance into several sub-instances and recurse. Clearly, any tree with the right number of leaves and a bijection can be considered as a decomposition tree. Then, a common technique to select those that are more suited for some task is to use an evaluating function.

**Definition 2 (Decomposition and Width Parameters).** Let  $G$  be a graph,  $f$  a set function over  $V(G)$ , and  $(T, \delta)$  a rooted decomposition tree of  $G$ . For every node  $u$  of  $T$ , let  $V_u$  denote the vertex subset of  $G$  induced by the leaves of the subtree of  $T$  rooted at  $u$ . The  $f$ -width of  $(T, \delta)$  is the maximum value of  $f(V_u)$ , taken over every node  $u$  of  $T$ . An *optimal  $f$ -decomposition* of  $G$  is a rooted decomposition tree of  $G$  having minimum  $f$ -width. The  $f$ -width of  $G$  is the  $f$ -width of an optimal  $f$ -decomposition of  $G$ .

If  $f$  is also required to be symmetric, namely that  $f(V_u) = f(V(G) \setminus V_u)$  for every  $V_u$ , then the above framework, up to unrooting the tree  $T$  and setting  $f(V(G)) = f(\emptyset) = -\infty$ , is equivalent to the one developed for the study of branch decomposition of symmetric and submodular functions (see, e.g., [23, Section 2] for a short and recent introduction). This includes the branch-width [27], rank-width [23], and boolean-width [4] decompositions of graphs. On the other hand, rooted decomposition trees as defined here can be used for situations where the symmetry does not occur, for instance with a branch-like decomposition of a submodular function that is not necessarily symmetric, a cliquewidth or NLC-width expression, or a so-called  $k$ -module decomposition as will be presented below.

For an efficient complexity analysis of the algorithm that will be described in Section 4 we will be interested in the following definition of  $f$ -width, so-called module-width in [22,26].

**Definition 3.** Let  $G$  be a graph and let  $X \subseteq V(G)$  be a vertex subset. A subset  $A \subseteq X$  is a *twin set* of  $X$  if, for every  $z \in V(G) \setminus X$  and pair of vertices  $x, y \in A$ , we have  $x$  adjacent to  $z$  if and only if  $y$  adjacent to  $z$ . A twin set  $A$  is a *twin class* of  $X$  if  $A$  is maximal. The set of all twin classes of  $X$  forms a partition of  $X$ , that we call the *twin class partition* of  $X$ .

**Definition 4 (Module-width).** The function  $\mu_G : 2^{V(G)} \rightarrow \mathbb{N}$  is defined such that  $\mu_G(X)$  is the number of twin classes of  $X$  in the graph  $G$ . The *module-width decompositions and parameters* of  $G$  refer to those of Definition 2 when  $f = \mu_G$ . The  $\mu_G$ -width of  $G$  will be called the *module-width* of  $G$  and denoted by  $\mu w(G)$ .

The terminology of module-width is according to the name given to an equivalent notion that was mentioned in [22, last two pages] and formalized in [26]. More precisely, Lanlignel and Rao defined so-called  $k$ -module decomposition, leading to the same parameter.

Clique-width and NLC-width expressions are constructions of a graph using logic operations. For a proper introduction to cliquewidth and NLC-width refer to [7,9]. The underlying graphs of cliquewidth and NLC-width expressions are rooted trees where every internal node has at most two children and where the

leaves are in a bijection with the vertices of the graph. This, up to contracting one child nodes, can be seen as a rooted decomposition tree. The clique-width  $cw(G)$  and the NLC-width  $nlc-w(G)$  of a graph  $G$  are parameters of  $G$  having powerful algorithmic properties. For instance, we have that any graph problem expressible in  $MSO_1$ -logic is FPT when parameterized by one of these two parameters (roughly, apply [17], then [23, Proposition 6.3], then [7]). They are closely linked to module-width by the following property.

**Theorem 1.** ([26]) *We have for any graph  $G$  that*

$$\mu w(G) \leq nlc-w(G) \leq cw(G) \leq 2\mu w(G).$$

We now give an alternative viewpoint of these module-width decompositions, that will link module-width to the so-called  $H$ -join decomposition framework [3] in an unexpected way.

**Definition 5.** Let  $H$  be a bipartite graph with color classes  $V_1$  and  $V_2$ , thus  $V(H) = V_1 \cup V_2$ . Let  $G$  be a graph and  $X \subseteq V(G)$  a subset of its vertices. We say that  $G$  is an  $H$ -join across the ordered cut  $(X, V(G) \setminus X)$  if there exists a partition of  $X$  with set of classes  $P$  and a partition of  $V(G) \setminus X$  with set of classes  $Q$ , and injective functions  $f_1 : P \rightarrow V_1$  and  $f_2 : Q \rightarrow V_2$ , such that for any  $x \in X$  and  $y \in V(G) \setminus X$  we have  $x$  adjacent to  $y$  in  $G$  if and only if  $x$  belongs to a class  $P_i$  of  $P$  and  $y$  to a class  $Q_j$  of  $Q$  with  $f_1(P_i)$  adjacent to  $f_2(Q_j)$  in  $H$ .

We will abusively refer to ordered cuts simply by cuts. Twins in a bipartite graph are vertices in the same color class having exactly the same neighborhood. A *twin contraction* is the deletion of a vertex when it has a twin. Notice that  $H$ -joins are insensitive to twin contractions: if  $H'$  is obtained from  $H$  by a twin contraction then  $G$  is an  $H$ -join across some cut if and only if  $G$  is an  $H'$ -join across the same cut. Note also that we do allow a twin-free bipartite graph to have one isolated vertex in each color class. We model the joining in module-width decompositions by using the following graph.

**Definition 6.** For a positive integer  $k$  we define a bipartite graph  $Y_k$  having for each integer  $i$  of  $\{1, 2, \dots, k\}$  a vertex  $a_i \in A$  and having for each subset  $S$  of  $\{1, 2, \dots, k\}$  a vertex  $b_S \in B$ , with  $V(Y_k) = A \cup B$ . This gives  $k$  vertices in  $A$  and  $2^k$  vertices in  $B$ . A vertex  $a_i$  is adjacent to a vertex  $b_S$  if and only if  $i \in S$ .

**Lemma 1.** *Let  $k$  be an integer, let  $H$  be a bipartite graph over color classes  $V_1 \cup V_2$  with  $|V_1| \leq k$ . Then, applying successive twin contractions in  $H$  until stability will always result in a graph that is isomorphic to an induced subgraph of  $Y_k$ .*

**Corollary 1.** *The function  $\mu_G$  of Definition [4] is exactly equal to the function  $\eta_G$  defined by:*

$$\eta_G(X) = \min\{k : G \text{ is a } Y_k\text{-join across the cut } (X, V(G) \setminus X)\}, \text{ for all } X \subseteq V(G).$$

### 3 Computing the Twin Classes

In the next section we will give a dynamic programming algorithm to solve the feedback vertex set problem on an input made by an  $n$ -vertex  $m$ -edge graph  $G$  and one of its rooted decomposition tree  $(T, \delta)$ . Note that the underlying graph of a clique-width expression of  $G$  is a rooted tree where each internal node having at most two children, and the leaves are in a bijection with the vertices of  $G$ . By contracting the internal nodes having one child, we will result in a rooted decomposition tree of  $G$ . Moreover, it can also be obtained from the proof of Theorem [1](#) that the module-width of this rooted decomposition tree is at most the clique-width of the clique-width expression. Consequently, if the input to our algorithm is the graph  $G$  and a clique-width  $k$  expression of  $G$ , we can transform them in a straightforward manner to an input made of  $G$  and one of its rooted decomposition tree of module-width at most  $k$ .

For every internal node  $u$  of  $T$  with  $V_u$  being the vertex subset of  $G$  induced by the subtree of  $T$  rooted at  $u$ , we will need to compute the twin classes of  $V_u$  as mentioned in the definition of  $\mu_G$  in Definition [4](#). For this, the algorithm given in [23](#) for transforming a rank decomposition into a cliquewidth expression can be used for a global runtime in  $O(n^2 2^{2rw(G)})$ . In this section, we will describe such a computation for every internal node  $u$  of  $T$ , with global runtime  $O(n^2)$ .

We will use the so-called partition refinement algorithmic technique (refer to, e.g., [15,24](#) for details).

A simple way to compute the twin class partition of  $V_u$  is to initialize  $\mathcal{Q} = (V_u)$  and, for every vertex  $z \in V(G) \setminus V_u$ , perform an one-to-one refinement of  $\mathcal{Q}$  using the neighborhood  $N(z)$  of  $z$  as pivot. The correctness follows directly from the definition of twin classes. This computation would have  $O(m)$  runtime for each internal node  $u$  of  $T$ , hence a global  $O(nm)$  runtime.

The main idea to reduce this runtime is to observe that, in the above operations, we can use  $N(z) \cap V_u$  as pivot instead of  $N(z)$  (for every  $z \in V(G) \setminus V_u$ ) without modifying the refined partition of each step. However, the sum over every possible  $V_u$  and  $z \in V(G) \setminus V_u$  of the value  $|N(z) \cap V_u|$  might still be large. We will observe a second fact. For a partition  $\mathcal{Q} = (Q_1, Q_2, \dots, Q_k)$  of  $X$  and a subset  $Y \subseteq X$ , we denote by  $\mathcal{Q}[Y]$  the partition of  $Y$  which results from removing all empty sets from  $(Q_1 \cap Y, Q_2 \cap Y, \dots, Q_k \cap Y)$ .

*Remark 1.* Let  $w$  be an internal node of  $T$  with children  $a$  and  $b$ . Let  $V_w, V_a$ , and  $V_b$  be the vertex subsets of  $G$  induced by the leaves of the subtrees of  $T$  rooted at  $w, a$ , and  $b$ , respectively. Let  $\mathcal{Q}_w = (Q_w(1), Q_w(2), \dots, Q_w(h_w))$  be the twin class partition of  $V_w$ . Then, initializing  $\mathcal{Q} = \mathcal{Q}_w[V_a]$  and refining  $\mathcal{Q}$  using  $N(z) \cap V_a$  as pivot for all  $z \in V_b$  will result to the twin class partition of  $V_a$ .

Basically, the algorithmic difference given by the remark is that we can now be restricted to  $z \in V_b$  instead of using all  $z \in V(G) \setminus V_a$  as before. The main point is that the sum over every possible  $V_a$  and  $z \in V_b$  of the value  $|N(z) \cap V_a|$  will be at most twice the value  $n + m$  (every edge of  $G$  appears at most twice in the sum). We now implement Remark [1](#).

First of all, the bottleneck of using  $N(z) \cap V_a$  as pivot will be that, unlike the case with  $N(z)$  which can be read simply in the adjacency list of  $G$ , we will need to compute  $N(z) \cap V_a$  for every possible  $V_a$  and  $z$ . We do this as a preprocessing step as follows.

We prepare the tree  $T$  as described in [16] so that afterwards we can, given two leaves  $x$  and  $y$  of  $T$ , compute the lowest common ancestor  $w$  of  $x$  and  $y$  in  $T$  in  $O(1)$  time. This can also be done in such a way that, if  $a$  and  $b$  denote the children of  $w$ , then we can in  $O(1)$  time decide whether  $x$  is a descendent of  $a$  or it is a descendent of  $b$ . Then, for every internal node  $w$  of the tree  $T$ , with children  $a$  and  $b$ , we initialize two tables  $N_w^{b \rightarrow a}$  and  $N_w^{a \rightarrow b}$  that will contain, for every vertex  $z$  in  $V_b$  (resp.  $V_a$ ), the neighborhood of  $z$  in  $V_a$  (resp.  $V_b$ ). Now, we scan through every edge  $xy$  of  $G$  and compute the lowest common ancestor  $w$  of  $x$  and  $y$ , as well as the children  $a$  and  $b$  of  $w$  such that  $x$  is a descendent of  $a$ , and finally add  $x$  to  $N_w^{b \rightarrow a}[y]$  and  $y$  to  $N_w^{a \rightarrow b}[x]$ . Clearly, after scanning all edges of  $G$ , we have that  $N_w^{b \rightarrow a}[z] = N(z) \cap V_a$  for all  $w, a, b$ , and  $z$ . This preprocessing takes  $O(n)$  time.

We come to the proper computation of the twin class partitions. The twin class partition associated to the root of  $T$  only has one class, which is  $V(G)$ . Suppose that we have computed the twin class partition  $\mathcal{Q}_w$  of an internal node  $w$  having children  $a$  and  $b$ . This partition  $\mathcal{Q}_w$  is stored in a double-linked list w.r.t. the data structure used for partition refinement. Basically, the following operations can operate directly on this data structure, if we allow ourselves to modify the double-linked list. However, the information on the twin classes of  $V_w$  would then be lost. For this reason, before continuing, we duplicate the data structure of  $\mathcal{Q}_w$  so that we store the twin classes of  $V_w$  in a private place of node  $w$ . Then, we can compute  $\mathcal{Q}_w[V_a]$  and  $\mathcal{Q}_w[V_b]$  simply by performing an one-to-two refinement of  $\mathcal{Q}_w$  using either  $V_a$  or  $V_b$  as pivot (cf.  $V_b = V_w \setminus V_a$ ). for each  $w$ . Duplication and refinement using  $V_a$  (or  $V_b$ ) as pivot take  $O(n)$  time for every node  $w$ , hence an  $O(n^2)$  global runtime.

We then initialize  $\mathcal{Q} = \mathcal{Q}_w[V_a]$  and, for every entry  $z$  of the table  $N_w^{b \rightarrow a}$ , refine  $\mathcal{Q}$  using  $N_w^{b \rightarrow a}[z]$  as pivot. As mentioned before, the main point of all these procedures is that the sum of the size of all possible pivots will now be at most twice the value  $n+m$ . Hence, the global runtime of this step is in  $O(n+m)$ . We deduce the following lemma, whose proof is straightforward. Recall that from the input of a cliquewidth expression of  $G$ , we can derive a rooted decomposition tree simply by contracting all internal nodes having one child in the underlying graph of the cliquewidth expression. The module-width of this decomposition tree is at most the cliquewidth of the expression.

**Lemma 2.** *Given a graph  $G$  and either  $(T, \delta)$  a rooted decomposition tree of  $G$ , or a clique-width expression tree of  $G$ . Then in  $O(n^2)$  global runtime we can compute and store, for every internal node  $u$  of  $T$  with  $V_u$  being the vertex subset of  $G$  induced by the leaves of the subtree of  $T$  rooted at  $u$ , the partition of  $V_u$  into its twin classes  $Q_u(1), Q_u(2), \dots, Q_u(h_u)$ .*

## 4 Solving the Feedback Vertex Set Problem

**Definition 7.** A *Feedback Vertex Set* of a graph  $G$  is a subset of vertices  $S$  with  $G[V(G) \setminus S]$  a forest. A *Forest Inducing Set (FI-set)* of a graph  $G$  is a subset of vertices  $S$  with  $G[S]$  a forest.

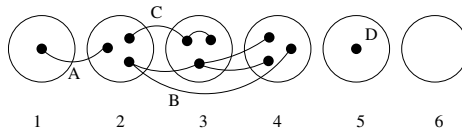
**Fact 41.** If  $S$  is a FI-set of maximum cardinality then  $V(G) - S$  is a Feedback Vertex Set of minimum cardinality.

We give dynamic programming algorithms that given a graph  $G$  and a rooted decomposition tree  $(T, \delta)$  of  $G$  will find the size of a minimum Feedback Vertex Set of  $G$ , by computing the size of a maximum FI-set in  $G$ . For a node  $a$  of  $T$ , let  $V_a$  be the vertices of  $G$  mapped to leaves in the subtree of  $T$  rooted at  $a$ . The runtime of the algorithm will be expressed as a function of  $\mu_G(V_a)$ , i.e. the number of twin-classes of such vertex subsets  $V_a$ .

### 4.1 Two Equivalence Relations on FI-Sets: by FI-Classes and by FI-Patterns

Let node  $a$  of  $T$  have  $h_a = \mu_G(V_a)$  twin-classes  $Q_a(1), \dots, Q_a(h_a)$ . We first consider how a FI-set  $X \subseteq V_a$  interacts with twin-classes. We characterize  $X$  by a 5-tuple that we call  $Pat_a(X)$  that records how the trees  $T(X)$  in the forest induced by the FI-set  $X$  interact with twin-classes. Note that  $T(X)$  may contain trees having only one vertex.

We first consider interaction of  $X$  on the set of pairs of twin-classes. Define  $Z(X)$  to be the pairs  $(i, j)$  such that there is no path in  $G[X]$  between a vertex in twin-class  $Q_a(i)$  and a vertex in twin-class  $Q_a(j)$ . Define  $S(X)$  to be the pairs  $(i, j)$  such that there is exactly one tree in  $T(X)$  having a path between a vertex in twin-class  $Q_a(i)$  and a vertex in twin-class  $Q_a(j)$ . Define  $W(X)$  to be the pairs  $(i, j)$  such that there are at least two trees in  $T(X)$  having a path between a vertex in twin-class  $Q_a(i)$  and a vertex in twin-class  $Q_a(j)$ . Note that  $Z(X), S(X), W(X)$  is a partition of the set of pairs of indices of twin-classes  $\{(i, j) : 1 \leq i \leq j \leq h_a\}$ , and thus given  $S(X), W(X)$  we can uniquely identify  $Z(X)$ . Define also  $\mathcal{P}(S(X))$  to be the partition of  $S(X)$  into distinct trees, i.e. with two pairs  $(i, j)$  and  $(i', j')$  belonging to the same class of  $\mathcal{P}(S(X))$  if and only if the same tree in  $T(X)$  has both a path between vertices in twin-classes  $Q_a(i)$  and  $Q_a(j)$  and a path between vertices in twin-classes  $Q_a(i')$  and  $Q_a(j')$ .



**Fig. 1.** In the above figure is an example containing 6 twin-classes named 1,2,...,6 and a FI-set  $X$  having 11 vertices.  $T(X)$  contains 4 trees named A,B,C,D. The FI-pattern of  $X$  will be the 5-tuple  $\langle S, W, S', W', P(S) \rangle$ , where  $S = \{12, 24, 34, 44, 33\}$ , and  $W = \{23\}$  and  $S' = \{1, 5\}$  and  $W' = \{2, 3, 4\}$  and  $P(S) = \{\{12\}, \{24, 34, 44\}, \{33\}\}$ .



Now we consider interaction of  $X$  on twin-classes. Define  $Z'(X), S'(X), W'(X)$  to be the partition of the set of indices  $\{1, 2, \dots, h_a\}$  of twin-classes such that if  $i \in Z'(X)$  there is no vertex in  $X$  belonging to  $Q_a(i)$ , and if  $i \in S'(X)$  there is exactly one vertex in  $X$  belonging to  $Q_a(i)$ , and if  $i \in W'(X)$  there are at least two vertices in  $X$  belonging to  $Q_a(i)$ .

**Definition 8.** For a FI-set  $X \subseteq V_a$  define the FI-pattern  $Pat_a(X)$  to be the 5-tuple  $\langle S(X), W(X), S'(X), W'(X), \mathcal{P}(S(X)) \rangle$ .

Our dynamic programming algorithm computing the size of a maximum FI-set in graph  $G$  will store a table  $Tab_a$  at each node  $a$  of the rooted decomposition tree  $T$  of  $G$ . For the computation of maximum FI-sets by a bottom-up traversal of  $T$  the following equivalence relation on FI-sets contained in  $V_a$  is clearly important.

**Definition 9.** For two FI-sets  $X, Y \subseteq V_a$  we define  $X \equiv_a Y$  if for any FI-set  $Z \subseteq V(G) \setminus V_a$  the set  $X \cup Z$  is a FI-set if and only if  $Y \cup Z$  is a FI-set. The equivalence classes of  $\equiv_a$  are called FI-classes and for a FI-set  $X \subseteq V_a$  we denote by  $FIclass_a(X)$  the FI-class containing  $X$ .

The table  $Tab_a$  will be indexed by FI-patterns, and the following lemma will be crucial for correctness of the algorithm.

**Lemma 3.** If  $Pat_a(X) = Pat_a(X')$  for two FI-sets  $X, X' \subseteq V_a$  then  $X \equiv_a X'$ , i.e.  $FIclass_a(X) = FIclass_a(X')$ .

*Proof.* Consider a FI-set  $Z \subseteq V(G) \setminus V_a$ . We show that if  $G[X \cup Z]$  contains a cycle then  $G[X' \cup Z]$  contains a cycle, and the statement in the lemma will follow by symmetry. Let  $C$  be a chordless cycle in  $G[X \cup Z]$  that has a minimum number of edges between a vertex of  $X$  and a vertex of  $Z$ . We call these crossing edges, and note that there are at least two crossing edges in  $C$ .

Let  $u_1v_1$  be a crossing edge of  $C$  with  $u_1 \in X, v_1 \in Z$ . Let the cycle  $C$  continue from  $v_1$  by a path  $P(v_1, v_2)$  in  $G[B]$  to vertex  $v_2 \in Z$  and then a crossing edge  $v_2u_2$ , then a path  $P(u_2, u_3)$  from  $u_2$  to  $u_3$  in  $G[X]$  and again a crossing edge  $u_3v_3$ , etc. We thus get a total ordering  $u_1v_1, u_2v_2, \dots, u_{2k}v_{2k}$  on all crossing edges of  $C$ , with  $u_i$ s in  $X$  and  $v_i$ s in  $Z$ .

In this way we get  $k$  paths of cycle  $C$  belonging to  $G[X]$ , i.e.  $k - 1$  paths  $P(u_{2i}, u_{2i+1})$  for  $1 \leq i \leq k - 1$ , and also the path  $P(u_{2k}, u_1)$ . Note that any such path will contain edges and vertices from a single tree of  $T(X)$ , even if such a tree may be trivial, i.e. with  $u_{2i} = u_{2i+1}$  or  $u_{2k} = u_1$ . Moreover, since the cycle  $C$  has the minimum number of crossing edges no tree of  $T(X)$  contains a vertex belonging to two of these  $k$  paths.

Consider now  $C_T = \{j : Q_a(j) \cap \{u_i : 1 \leq i \leq 2k\} \neq \emptyset\}$ , i.e. the indices of twin-classes containing a vertex involved in a crossing edge. Note that  $|Q_a(j) \cap \{u_i : 1 \leq i \leq 2k\}|$  can be at most two. Thus, to be able to exchange these  $k$  paths in  $G[X]$  by equivalent paths in  $G[X']$  we only need that the set of twin-classes containing respectively zero, one or at least two vertices of  $X$  are the same as



the set of twin-classes containing respectively zero, one or at least two vertices of  $X'$ , and this indeed holds since  $Pat_a(X) = Pat_a(X')$  implies that  $S'(X) = S'(X')$  and  $W'(X) = W'(X')$  and hence also  $Z'(X) = Z'(X')$ . Thus, since  $Pat_a(X) = Pat_a(X')$  we can exchange the set of  $k$  paths in  $G[X]$ , named  $P(u_{2i}, u_{2i+1})$  for  $1 \leq i \leq k-1$  and  $P(u_{2k}, u_1)$ , by  $k$  paths in  $G[X']$  named  $P(u'_{2i}, u'_{2i+1})$  for  $1 \leq i \leq k-1$  and  $P(u'_{2k}, u'_1)$ , in such a way that vertex  $u_i$ , for any  $1 \leq i \leq 2k$ , is in the same twin class as  $u'_i$ , and also such that all the new paths belong to distinct trees in  $T(Y)$ . The latter statement is guaranteed by the fact that  $\mathcal{P}(S(X)) = \mathcal{P}(S(X'))$  and  $W(X) = W(X')$ . Since any two vertices in the same twin class have the same neighbors in  $Z$  we have that also  $G[X' \cup Z]$  contains a cycle and the lemma follows.

**Corollary 2.** *The relation on FI-sets  $\{X \subseteq V_a : X \text{ is a FI-set}\}$  given by  $Pat_a(X) = Pat_a(Y)$  is an equivalence relation that is a refinement of  $\equiv_a$ .*

### 4.2 Tables and Algorithm

The table  $Tab_a$  at node  $a$  of the rooted decomposition tree  $T$  of  $G$ , is defined as follows:

**Definition 10.** *The table  $Tab_a$  will be indexed by FI-patterns. For a FI-pattern  $P_a$ , let  $FIclass_a(P_a)$  be the FI-class of the FI-sets having FI-pattern  $P_a$ , or let it be undefined if no such FI-set exists. Thus  $FIclass_a(P_a) = FIclass_a(X)$  for any FI-set  $X \subseteq V_a$  having  $Pat_a(X) = P_a$ . For a FI-class  $C$  let  $max(C)$  be the maximum cardinality of a FI-set in  $C$ . If  $FIclass_a(P_a)$  is undefined then define  $max(FIclass_a(P_a))$  to be  $-\infty$ . The table  $Tab_a$  is correct when it satisfies the following two conditions:*

1. *For any FI-pattern  $P_a$  we have  $Tab_a[P_a] \leq max(FIclass_a(P_a))$*
2. *For any FI-class  $C$  there is FI-pattern  $P_a$  with  $FIclass_a(P_a) = C$  and  $Tab_a[P_a] = max(C)$*

Whenever  $Tab_a[P_a] \geq 0$  we also store with this table index some arbitrary FI-set  $F_a$  with  $Pat_a(F_a) = P_a$ . We are now ready to describe the algorithm computing the cardinality of a maximum FI-set of  $G$ . Let us start by noting that based on the above definition, the maximum entry over all table entries at the root of  $T$  will correctly solve the problem.

The algorithm starts by initializing all table entries to  $-\infty$ .

At any leaf  $a$  of the tree  $T$  we have  $V_a = \{\delta(a)\}$  and set  $Tab_a[\emptyset, \emptyset, \emptyset, \emptyset] = 0$  (and store the empty FI-set  $F_a$ ) and  $Tab_a[\emptyset, \emptyset, \{1\}, \emptyset] = 1$  (and store the FI-set  $\{\delta(a)\}$  consisting of  $\delta(a)$  belonging to the twin-class  $Q_a(1)$ ).

In a bottom-up traversal of the tree  $T$ , when reaching an internal node  $w$  having children  $a$  and  $b$  we do the following:

For all pairs of patterns  $P_a, P_b$  from  $Tab_a, Tab_b$  with non-negative entries

Let  $F_a$  and  $F_b$  be the FI-sets stored with these indices

If  $G[F_a \cup F_b]$  is acyclic (i.e. if  $F_a \cup F_b$  is a FI-set)

Compute  $P_w = Pat_w(F_a \cup F_b)$

If  $Tab_w[P_w] = -\infty$  store  $F_a \cup F_b$  with this index

$Tab_w[P_w] = max(Tab_w[P_w], Tab_a[P_a] + Tab_b[P_b])$

### 4.3 Correctness and Runtime

We consider correctness of the algorithm and start by a lemma showing that the equivalence relation given by FI-classes are well-behaved with respect to the child-parent relation of the rooted decomposition tree.

**Lemma 4.** *Let  $w$  be an inner node of  $T$  with children  $a$  and  $b$ . If  $A \equiv_a A'$  and  $B \equiv_b B'$  then  $A \cup B \equiv_w A' \cup B'$ .*

**Lemma 5.** *Based on correct tables of children  $Tab_a, Tab_b$ , the algorithm will correctly update the parent table  $Tab_w$ .*

*Proof.* We first show that  $Tab_w$  will satisfy condition 1 in Definition 10. If after updating we have  $Tab_w[P_w] = k$  for some integer  $k$  then there must be a pair  $P_a, P_b$  with  $Tab_a[P_a] = k_a$  and  $Tab_b[P_b] = k_b$  with  $k = k_a + k_b$ , and since condition 1 holds for the children tables we have  $\max(FIclass_a(P_a)) \geq k_a$  and  $\max(FIclass_b(P_b)) \geq k_b$ . Thus, there exists FI-sets  $X_a \subseteq V_a$  and  $X_b \subseteq V_b$  with  $FIclass_a(X_a) = FIclass_a(P_a)$  and  $FIclass_b(X_b) = FIclass_b(P_b)$  and  $|X_a| \geq k_a$  and  $|X_b| \geq k_b$ . Moreover, we have stored FI-sets  $F_a$  and  $F_b$ , with  $FIclass_a(F_a) = FIclass_a(P_a)$  and  $FIclass_b(F_b) = FIclass_b(P_b)$ , by Lemma 3, and in the algorithm we have checked that  $F_a \cup F_b$  is a FI-set. We now apply Lemma 4 to conclude that also  $X_a \cup X_b$  is a FI-set. By Lemma 4 we also get the stronger statement that  $FIclass_w(F_a \cup F_b) = FIclass_w(X_a \cup X_b)$ . We have thus shown that if  $Tab_w[P_w] = k$  then there must exist a FI-set  $X_a \cup X_b$  with  $|X_a \cup X_b| \geq k$  and  $FIclass_w(X_a \cup X_b) = FIclass_w(P_w)$  so that condition 1 is satisfied.

We now show that  $Tab_w$  will satisfy condition 2 in Definition 10. Consider a FI-class  $C$  of  $\equiv_w$ . Let  $Z \subseteq V_w$  be a FI-set in this class. Note that  $Z_a = Z \cap V_a$  and  $Z_b = Z \cap V_b$  must be FI-sets. Since condition 2 holds for the children tables we therefore have  $P_a$  and  $P_b$  with  $Tab_a[P_a] \geq |Z_a|$  and  $Tab_b[P_b] \geq |Z_b|$  and  $FIclass_a(P_a) = FIclass_a(Z_a)$  and  $FIclass_b(P_b) = FIclass_b(Z_b)$ . When the algorithm considers the pair  $P_a, P_b$  we find stored FI-sets  $F_a$  and  $F_b$  which by Lemma 3 have the property that  $FIclass_a(F_a) = FIclass_a(Z_a)$  and  $FIclass_b(F_b) = FIclass_b(Z_b)$ . Therefore, applying Lemma 4 we get that since  $Z_a \cup Z_b$  is a FI-set then  $F_a \cup F_b$  is a FI-set. Moreover, we also get the stronger statement that  $C = FIclass_w(F_a \cup F_b) = FIclass_w(Z_a \cup Z_b)$ . We have thus shown that for any FI-class  $C$  containing a FI-set  $Z \subseteq V_w$  there will be a FI-pattern  $P_w = Pat_w(F_a \cup F_b)$  with  $FIclass_w(P_w) = C$  such that  $Tab_w[P_w] \geq |Z_a| + |Z_b| = |Z|$ . We conclude that also condition 2 will be satisfied.

**Theorem 2.** *Given either a rooted decomposition tree  $(T, \delta)$  of module-width  $k$  of a graph  $G$ , or a  $k$ -expression of a graph  $G$  of cliquewidth at most  $k$ , we can in  $O(2^{2k^2 \log^k n^2})$  steps solve the Minimum Feedback Vertex Set problem on  $G$ .*

*Proof.* Consider first the case of input being a rooted decomposition tree. By Lemma 2 we can compute twin classes for all nodes of the tree in time  $O(n^2)$ . Note that for any node  $a$  of the tree  $T$  the number of twin-classes of  $V_a$  is at most  $k$ . By Definition 10 and Lemma 5 the maximum value over all entries in

the table at the root of our dynamic programming algorithm will correctly solve the problem.

For the runtime, the bottleneck is the inner node update procedure which loops over all *pairs* of patterns  $P_a, P_b$ . The number of patterns is bounded by the number of choices for the 5-tuples  $S, W, S', W', \mathcal{P}(S)$ . An upper bound on the number of choices for the third and fourth components jointly (3-partitions of the  $k$  twin-classes) is  $3^k$ , while for the first and second components jointly (3-partitions of unordered pairs of twin-classes) it is  $3^{\frac{k^2+k}{2}}$ . The number of choices for the fifth component can be (loosely) upper bounded by the number of partitions of unordered pairs of indices of the  $k$  twin-classes. This gives an upper bound on the total number of patterns  $2^{k^2 \log k}$ . In the update procedure we spend for each pair of patterns time at most  $O(nk^2)$  to check if the union of two FI-sets are a FI-set and to compute the new pattern, making use of the fact that two vertices in the same twin class have the same neighbors across the cut. Since there are at most  $n$  inner nodes the runtime in the theorem follows.

Note that within the same runtime we could instead have taken as input a  $k$ -expression of a graph  $G$  of cliquewidth at most  $k$ . This since by Theorem [11](#) the module-width of  $G$  is no larger than the clique-width of  $G$ , and from the  $k$ -expression we easily derive a rooted decomposition tree of module-width at most  $k$ .

**Note:** After submitting this paper we have learned that Ganian and Hliněný have a manuscript [12](#) with an algorithm solving FVS in time  $O(rw^2|V(G)|^2 + 2^{5rw^2}rw^3|V(G)|)$  parameterized by the rankwidth  $rw$  of  $G$ , when a rank decomposition of width  $rw$  is given.

## References

1. Bafna, V., Berman, P., Fujito, T.: A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Math.* 12, 289–297 (1999)
2. Bar-Yehuda, R., Geiger, D., Naor, J., Roth, R.: Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference. *SIAM Journal on Computing* 27, 942–959 (1998)
3. Bui-Xuan, B.-M., Telle, J.A., Vatshelle, M.:  $H$ -join decomposable graphs and algorithms with runtime single exponential in rankwidth. *Discrete Applied Mathematics: special issue of GROW* (to appear)
4. Bui-Xuan, B.-M., Telle, J.A., Vatshelle, M.: Fast FPT algorithms for vertex subset and vertex partitioning problems using neighborhood unions, <http://arxiv.org/abs/0903.4796>
5. Chen, J., Fomin, F., Liu, Y., Lu, S., Villanger, Y.: Improved Algorithms for the Feedback Vertex Set Problems. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) *WADS 2007*. LNCS, vol. 4619, pp. 422–433. Springer, Heidelberg (2007)
6. Chudak, F., Goemans, M., Hochbaum, D., Williamson, D.: A primal-dual interpretation of two 2-approximation algorithms for the feedback vertex set problem in undirected graphs. *Operations Research Letters* 22, 111–118 (1998)
7. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique width. *Theory of Comp. Sys.* 33(2), 125–150 (2000)

8. Dehne, F., Fellows, M., Langston, M., Rosamond, F., Stevens, K.: An  $O(2^{O(k)}n^3)$  FPT Algorithm for the Undirected Feedback Vertex Set Problem. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 859–869. Springer, Heidelberg (2005)
9. Espelage, W., Gurski, F., Wanke, E.: How to Solve NP-hard Graph Problems on Clique-Width Bounded Graphs in Polynomial Time. In: Brandstädt, A., Van Bang Le (eds.) WG 2001. LNCS, vol. 2204, pp. 117–128. Springer, Heidelberg (2001)
10. Even, G., Naor, J., Schieber, B., Zosin, L.: Approximating minimum subset feedback sets in undirected graphs with applications. *SIAM J. Discrete Math.* 13, 255–267 (2000)
11. Fomin, F., Gaspers, S., Pyatkin, A., Razgon, I.: On the Minimum Feedback Vertex Set Problem: Exact and Enumeration Algorithms. *Algorithmica* 52, 293–307 (2008)
12. Ganian, R., Hliněný, P.: On Parse Trees and Myhill-Nerode-type Tools for handling Graphs of Bounded Rank-width,  
<http://www.fi.muni.cz/~hlineny/Research/papers/MNtools-dam3.pdf>
13. Goemans, M., Williamson, D.: Primal-dual approximation algorithms for feedback problems in planar graphs. *Combinatorica* 18(1), 37–59 (1998)
14. Guo, J., Gramm, J., Hüffner, F., Niedermeier, R., Wernicke, S.: Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *Journal of Computer and System Sciences* 72(8), 1386–1396 (2006)
15. Habib, M., Paul, C., Viennot, L.: Partition Refinement Techniques: An Interesting Algorithmic Tool Kit. *Int. J. of Foundations on Comp. Sci.* 10(2), 147–170 (1999)
16. Harel, D., Tarjan, R.: Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing* 13(2), 338–355 (1984)
17. Hliněný, P., Oum, S.: Finding branch-decompositions and rank-decompositions. *SIAM Journal on Computing* 38(3), 1012–1032 (2008)
18. Karp, R.: Reducibility among combinatorial problems. In: *Complexity of Computer Computations*, pp. 85–103 (1972)
19. Kleinberg, J., Kumar, A.: Wavelength conversion in optical networks. *Journal of Algorithms* 38, 25–50 (2001)
20. Kloks, T., Lee, C., Liu, J.: New Algorithms for  $k$ -Face Cover,  $k$ -Feedback Vertex Set, and  $k$ -Disjoint Cycles on Plane and Planar Graphs. In: Kučera, L. (ed.) WG 2002. LNCS, vol. 2573, pp. 282–295. Springer, Heidelberg (2002)
21. Kobler, D., Rotics, U.: Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Applied Mathematics* 126(2-3), 197–221 (2003)
22. Lanlignel, J.-M.: Autour de la décomposition en coupe. Ph. D. thesis, Université Montpellier II (2001)
23. Oum, S., Seymour, P.: Approximating clique-width and branch-width. *J. Combin. Theory Ser. B* 96(4), 514–528 (2006)
24. Paige, R., Tarjan, R.: Three partition refinement algorithms. *SIAM Journal on Computing* 16(6), 973–989 (1987)
25. Raman, V., Saurabh, S., Subramanian, C.: Faster fixed parameter tractable algorithms for finding feedback vertex sets. *ACM Trans. on Alg.* 2(3), 403–415 (2006)
26. Rao, M.: Clique-width of graphs defined by one-vertex extensions. *Discrete Mathematics* 308(24), 6157–6165 (2008)
27. Robertson, N., Seymour, P.: Graph minors X: Obstructions to tree-decomposition. *Journal on Combinatorial Theory Series B* 52, 153–190 (1991)

# Note on Decomposition of $K_{n,n}$ into $(0, j)$ -prisms

Sylwia Cichacz<sup>1,\*</sup>, Dalibor Fronček<sup>2</sup>, and Petr Kovář<sup>3,\*\*</sup>

<sup>1</sup> AGH University of Science and Technology

<sup>2</sup> University of Minnesota Duluth

<sup>3</sup> VŠB – Technical University of Ostrava

**Abstract.** R. Häggkvist proved that every 3-regular bipartite graph of order  $2n$  with no component isomorphic to the Heawood graph decomposes the complete bipartite graph  $K_{6n,6n}$ . In [2] the first two authors established a necessary and sufficient condition for the existence of a factorization of the complete bipartite graph  $K_{n,n}$  into certain families of 3-regular graphs of order  $2n$ . In this paper we tackle the problem of decompositions of  $K_{n,n}$  into 3-regular graphs some more. We will show that certain families of 3-regular graphs of order  $2n$  decompose the complete bipartite graph  $K_{\frac{3n}{2}, \frac{3n}{2}}$ .

## 1 Introduction

All graphs considered in this paper are simple, finite and undirected. We use standard terminology and notation of graph theory.

Graph decompositions have been widely studied in many different settings. We say that a graph  $B$  has a  $G$ -decomposition if there are subgraphs  $G_1, G_2, \dots, G_s$  of  $B$ , all isomorphic to  $G$ , such that each edge of  $B$  belongs to exactly one  $G_i$ . If each  $G_i$ ,  $i = 1, \dots, s$  contains all vertices of  $B$ , then we say that  $B$  has a  $G$ -factorization.

Recall that a *prism* is a graph of the form  $C_m \times P_2$ . As in [2] for  $j$  even let the  $(0, j)$ -prism (pronounced “oh-jay prism”) of order  $2n$  be a graph with two vertex disjoint cycles  $R_n^i = v_0^i, \dots, v_{n-1}^i$  for  $i = 1, 2$  of length  $n$  called *rims* and edges  $v_1^1 v_1^2, v_3^1 v_3^2, v_5^1 v_5^2, \dots$  and  $v_0^1 v_j^2, v_2^1 v_{2+j}^2, v_4^1 v_{4+j}^2, \dots$  called *spokes* of *type 0* and *type j*, respectively (see Fig. 1). It is easy to observe that an  $(0, j)$ -prism is a 3-regular graph and is isomorphic to an  $(0, -j)$ -prism,  $(j, 0)$ -prism and  $(-j, 0)$ -prism. In our terminology the usual prism is an  $(0, 0)$ -prism.

The problem of factorization of  $K_{n,n}$  into  $(0, j)$ -prisms was solved in [2]. In this paper we approach the decomposition problem of  $K_{n,n}$  into  $(0, j)$ -prisms. As in [6] by  $G[H]$  we denote the composition of graphs  $G$  and  $H$  which is obtained by replacing every vertex of  $G$  by a copy of  $H$  and every edge of  $G$  by the complete bipartite graph  $K_{|H|, |H|}$ . We say that  $G[H]$  arose from  $G$  by *blowing*

---

\* The work was supported by Fulbright Scholarship No 15072441.

\*\* Research for this article was partially supported by the institutional project MSM6198910027.

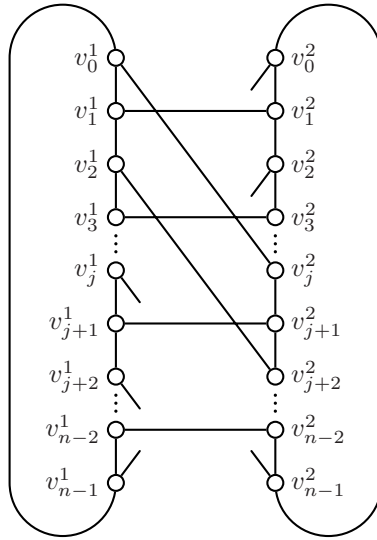


Fig. 1.  $(0, j)$ -prism

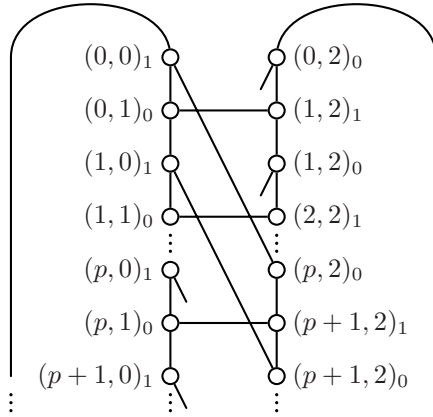
up by  $H$  and recall that  $\overline{K}_m$  is the complement of  $K_m$ , i.e., the graph consisting of  $m$  independent vertices.

A labeling of a graph  $G$  is a function from  $V(G)$  into a group  $\Gamma$ . A. Rosa [11] introduced several types of graph labelings as tools for decompositions of complete graphs. In this paper we will use a decomposition method based on certain vertex labeling.

**Definition 1.** Let  $G$  be a bipartite graph with  $k$  edges. Let  $V(G) = V_0 \cup V_1$ ,  $V_0 \cap V_1 = \emptyset$  and  $|V_0| \leq |V_1| \leq k$ . Let  $\lambda$  be an injection such that  $\lambda : V_i \rightarrow \{(u, v)_i : u \in \mathbb{Z}_a, v \in \mathbb{Z}_b, a \cdot b = k\}$  and  $i = 0, 1$ . We define the dimension of an edge  $x_0y_1$  with  $\lambda(x_0) = (u, v)_0$  and  $\lambda(y_1) = (t, z)_1$  as  $\dim(x_0y_1) = ((t - u) \pmod a, (z - v) \pmod b)$  for  $x_0 \in V_0$  and  $y_1 \in V_1$ .

Let  $p = \frac{j}{2}$  and  $b = 3$ . Notice that, if  $G$  is an  $(0, j)$ -prism of order  $2n$ , where  $n$  is even, then we can label vertices of  $G$  in such a way that  $R_n^1 = (0, 0)_1, (0, 1)_0, (1, 0)_1, (1, 1)_0, (2, 0)_1, (2, 1)_0, (3, 0)_1, \dots, (\frac{n}{2} - 1, 1)_0, (0, 0)_1, R_n^2 = (0, 2)_0, (1, 2)_1, (1, 2)_0, (2, 2)_1, (2, 2)_0, (3, 2)_1, \dots, (\frac{n}{2} - 1, 2)_0, (0, 2)_1, (0, 2)_0$  and  $(i + 1, 0)_1(i, 2)_0, (i, 1)_1(i + p, 2)_0 \in E(G)$ , where  $i \in \mathbb{Z}_{\frac{n}{2}}$  (see Fig. 2). Observe that such labeling implies that spokes have the dimension either  $(1, 1)$  or  $(-p, 1)$ . Notice that we can also label the end-vertices of the spokes as  $(i, 1)_0(i, 2)_1, (i, 2)_0(i + p + 1, 0)_1$ , where  $i \in \mathbb{Z}_{\frac{n}{2}}$  and then they have the dimension either  $(0, 1)$  or  $(p + 1, 1)$ .

Problems of decomposition of graphs into  $k$ -regular graphs were studied widely. R. Häggkvist [8] proved that every 3-regular bipartite graph of order  $2n$  with no component isomorphic to the Heawood graph decomposes the complete bipartite graph  $K_{6n, 6n}$ . In [2] it was proved that  $K_{n, n}$  can be factorized into



**Fig. 2.** Labeling of  $(0, j)$ -prism

$(0, j)$ -prisms of order  $2n$  if and only if  $n \equiv 0 \pmod{6}$ . It is natural to consider also a more general problem. In this paper we decompose complete bipartite graphs  $K_{k,k}$  into (non-spanning)  $(0, j)$ -prisms on  $2n$  vertices. It is obvious that for  $n \equiv 0 \pmod{6}$  we can decompose every graph  $K_{mn,mn}$  by first decomposing it into  $m^2$  copies of  $K_{n,n}$  and then factorizing each copy into the  $(0, j)$ -prisms. Hence, for  $(0, j)$ -prisms our construction for  $n \not\equiv 0 \pmod{6}$  gives stronger results than Häggkvist’s theorem. On the other hand, we notice that the obvious necessary conditions allow wider classes of complete bipartite graphs than just  $K_{mn,mn}$  for consideration. For if we want to decompose  $K_{k,k}$  into  $(0, j)$ -prisms of order  $2n$ , then it follows that  $k^2 \equiv 0 \pmod{3n}$ , because the number of edges of the  $(0, j)$ -prism is  $3n$ . Moreover, since an  $(0, j)$ -prism has to be bipartite in order to decompose  $K_{k,k}$ , it follows that  $n$  must be even and the  $(0, j)$ -prism has an even number of edges. Therefore,  $k$  must be even, which implies that  $k \equiv 0 \pmod{6}$ . However, these conditions may be in some cases satisfied even when  $k \neq mn$ . For instance, if  $n \equiv 0 \pmod{4}$ , then  $K_{\frac{3n}{2}, \frac{3n}{2}}$  satisfies the necessary conditions. In this paper we will deal with case  $n \equiv 0 \pmod{8}$  and decompose  $K_{\frac{3n}{2}, \frac{3n}{2}}$  into an *even* number of  $(0, j)$ -prisms.

The main idea of the proof is the following. First we decompose  $K_{\frac{n}{2}, \frac{n}{2}}$  into  $C_n$ , then we blow up  $K_{\frac{n}{2}, \frac{n}{2}}$  into  $K_{\frac{3n}{2}, \frac{3n}{2}} = K_{\frac{n}{2}, \frac{n}{2}}[\overline{K}_3]$  and each  $C_n$  into  $C_n[\overline{K}_3]$ . Then we “glue together” certain pairs of  $C_n[\overline{K}_3]$  and we decompose them into six copies of  $(0, j)$ -prisms.

The decomposition of  $K_{n_1, n_2}$  into cycles was completely solved by J.C. Bermond, C. Huang, and D. Sotteau [1] and [10].

**Theorem 2.**  $K_{k_1, k_2}$  can be decomposed into  $C_n$  if and only if  $n, k_1, k_2$  are all even,  $n$  divides  $k_1 k_2$  and both  $k_1, k_2 \geq \frac{n}{2}$ .

## 2 Decomposition

In order to prove our main result, we first need the following three lemmas.

**Lemma 3.** *Let  $G$  be an  $(0, 0)$ -prism of order  $2n$ , where  $n$  is even. Then  $K_{k_1, k_2}$  can be decomposed into  $G$  if  $9n$  divides  $k_1 k_2$ , both  $k_1, k_2 \geq \frac{3n}{2}$  and 6 divides both  $k_1$  and  $k_2$ .*

*Proof.* Since  $n, \frac{k_1}{3}, \frac{k_2}{3}$  are all even and  $\frac{k_1}{3}, \frac{k_2}{3} \geq \frac{n}{2}$ , we observe that  $C_n$  decomposes  $K_{\frac{k_1}{3}, \frac{k_2}{3}}$  by Theorem 2. By blowing up each cycle by  $\overline{K_3}$  we obtain decomposition of  $K_{k_1, k_2}$  into  $C_n[\overline{K_3}]$ . Denote vertices of  $C_n$  by  $0_1, 0_0, 1_1, 1_0, \dots, (\frac{n}{2} - 1)_1, (\frac{n}{2} - 1)_0, 0_1$ . By blowing up  $C_n$  by  $\overline{K_3}$  we obtain  $C_n[\overline{K_3}]$ . From each vertex  $j_i$  of  $C_n$  we obtain three vertices  $(j, 0)_i, (j, 1)_i, (j, 2)_i$  of  $C_n[\overline{K_3}]$ .

Let  $G$  be an  $(0, 0)$ -prism of order  $2n$ . Notice that we can find one copy of  $G$  in  $C_n[\overline{K_3}]$  in such a way that the rims are  $R_n^1 = (0, 0)_1, (0, 1)_0, (1, 0)_1, (1, 1)_0, (2, 0)_1, (2, 1)_0, (3, 0)_1, \dots, (\frac{n}{2} - 1, 1)_0, (0, 0)_1$ ,  $R_n^2 = (0, 2)_1, (0, 2)_0, (1, 2)_1, (1, 2)_0, (2, 2)_1, (2, 2)_0, (3, 2)_1, \dots, (\frac{n}{2} - 1, 2)_0, (0, 2)_1$  and the spokes are  $(i, 1)_0(i, 2)_1, (i, 2)_0(i + 1, 0)_1 \in E(G)$ , where  $i \in \mathbb{Z}_{\frac{n}{2}}$  (see Fig. 3).

Using Definition 1 we can see that between any two triples  $(i, j)_0$  and  $(i, k)_1$  for  $j, k \in \{0, 1, 2\}$  we have three edges of dimensions  $(0, 0), (0, 1), (0, 2)$ . Similarly, for any two triples  $(i, j)_0$  and  $(i + 1, k)_1$  for  $j, k \in \{0, 1, 2\}$  we have three edges of dimensions  $(1, 0), (1, 1), (1, 2)$ . If we now apply mappings  $\phi_m((a, b)_s) = (a, b + m)_s$  for  $m = 0, 1, 2$  and  $s = 0, 1$ , the image of every edge  $e$  will have the same dimension as  $e$  itself. Because the endvertices of all three edges with the same dimension between a fixed pair of triples are all different, one can observe that the isomorphisms  $\phi_m$  produce three edge-disjoint copies  $G_0, G_1, G_2$  in  $C_n[\overline{K_3}]$ , where  $G_m = \phi_m(G)$ . This technique is a slightly generalized form of a widely used labeling technique, first introduced by G. Ringel, A. S. Lladó and O. Serra [12]. Notice that such labeling implies that spokes have the dimension either  $(1, 1)$  or  $(0, 1)$ .  $\square$

**Lemma 4.** *Let  $n \equiv 0 \pmod{4}$  and  $G$  be an  $(0, 2)$ -prism of order  $2n$ . Then  $K_{k_1, k_2}$  can be decomposed into  $G$  if  $18n$  divides  $k_1 k_2$  and both  $k_1, k_2 \geq \frac{3n}{2}$  and  $k_1, k_2 \equiv 0 \pmod{12}$ .*

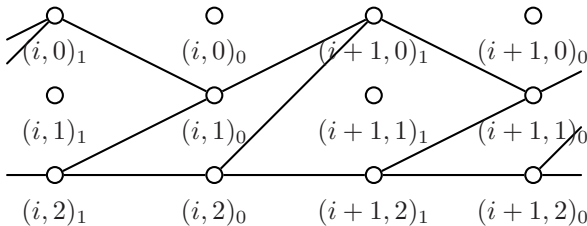


Fig. 3. One copy of the graph  $G$



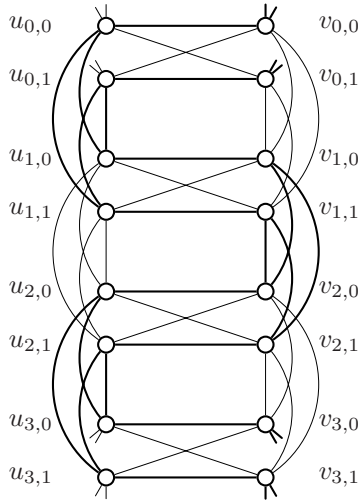


Fig. 4. Two copies of  $(0,2)$ -prisms in a segment of  $H[\overline{K}_2]$

*Proof.* For  $n = 4$  the graph  $G$  is isomorphic to an  $(0, 0)$ -prism, so we are done by Lemma 3. From now on assume  $n \geq 8$ . Let  $H$  be an  $(0, 0)$ -prism of order  $n$ . We know by Lemma 3 that  $H$  decomposes  $K_{\frac{n}{2}, \frac{n}{2}}$ . Denote the vertices of  $H$  so that the graph  $H$  has two rims  $R_{\frac{n}{2}}^1 = u_0, u_1, \dots, u_{n/2-1}, u_0$  and  $R_{\frac{n}{2}}^2 = v_0, v_1, \dots, v_{n/2-1}, v_0$  of length  $\frac{n}{2}$  and spokes  $u_0v_0, u_1v_1, \dots, u_{n/2-1}v_{n/2-1}$ . By blowing up each graph  $H$  by  $\overline{K}_2$  we obtain a decomposition of  $K_{k_1, k_2}$  into  $H[\overline{K}_2]$ . From each vertex  $u_j$  of  $H$  we obtain two vertices  $u_{j,0}$  and  $u_{j,1}$  of  $H[\overline{K}_2]$  and from each vertex  $v_j$  of  $H$  we obtain two vertices  $v_{j,0}$  and  $v_{j,1}$  of  $H[\overline{K}_2]$  (see Fig. 4).

Notice that we can find two copies  $G_1, G_2$  of  $G$  in  $C_n[\overline{K}_2]$  in such a way that for  $G_1$  we take the rims  $v_{0,0}, u_{0,0}, u_{1,1}, v_{1,1}, v_{2,0}, u_{2,0}, u_{3,1}, \dots, u_{n/2-1,1}, v_{n/2-1,1}, v_{0,0}$  and  $v_{0,1}, u_{0,1}, u_{1,0}, v_{1,0}, v_{2,1}, u_{2,1}, u_{3,0}, \dots, u_{n/2-1,0}, v_{n/2-1,0}, v_{0,1}$  and the spokes  $u_{0,0}u_{1,0}, u_{1,1}u_{0,1}, v_{1,1}v_{2,1}, v_{2,0}v_{1,0}, u_{2,0}u_{3,0}, u_{3,1}u_{2,1}, v_{3,1}v_{4,1}, v_{4,0}v_{3,0}, u_{4,0}u_{5,0}, \dots, v_{n/2-1,1}v_{0,1}$ . For  $G_2$  we let the rims be  $v_{0,0}, v_{1,0}, u_{1,1}, u_{2,1}, v_{2,0}, v_{3,0}, u_{3,1}, \dots, u_{n/2-1,1}, v_{n/2-1,0}, v_{0,0}$  and  $v_{0,1}, v_{1,1}, u_{1,0}, u_{2,0}, v_{2,1}, v_{3,1}, u_{3,0}, \dots, u_{n/2-1,0}, v_{n/2-1,1}, v_{0,1}$  and the spokes  $v_{0,0}v_{1,1}, v_{1,0}v_{0,1}, u_{1,1}u_{2,0}, u_{2,1}u_{1,0}, v_{2,0}v_{3,1}, v_{3,0}v_{2,1}, u_{3,1}u_{4,0}, u_{4,1}u_{3,0}, \dots, u_{n/2-1,1}u_{0,0}$  (see Fig. 4).  $\square$

In the proof of Theorem 6 we want to find a pair of  $(0, 0)$ -prisms with the property that we can remove every other spoke in each of them and mutually swap the two sets of spokes between the two prisms such that they become type  $j$  spokes and hence we obtain two  $(0, j)$ -prisms. Therefore we need to make sure that we can decompose  $K_{\frac{3n}{2}, \frac{3n}{2}}$  into unions of these pairs of  $(0, 0)$ -prisms.

Notice that we can write any integer  $z$  uniquely as  $z = a2^r$ , where  $a$  is odd and  $r \geq 0$ .

**Lemma 5.** *Let  $m \equiv 0 \pmod{4}$ ,  $c = a2^\alpha$ ,  $m = b2^\beta$  where  $a, b$  are odd,  $1 \leq \alpha < \beta$  and  $H$  be a 4-regular bipartite graph with bipartition  $X = \{x_0, x_1, \dots, x_{m-1}\}$ ,  $Y = \{y_0, y_1, \dots, y_{m-1}\}$  and edges  $x_i y_i, x_i y_{i+1}, x_i y_{i+c}, x_i y_{i+c+1}$  for some positive  $c \leq (m-2)/2$ , where the addition in subscripts is taken modulo  $m$ . Then  $K_{m,m}$  can be decomposed into  $H$ .*

*Proof.* *Claim 1* Let  $c = a2^\alpha, m = b2^\beta$  where  $a, b$  are odd and  $1 \leq \alpha < \beta$ . Then  $Z_m$  can be decomposed into a collection  $\mathcal{D}$  of  $m/4$  mutually disjoint sets  $D_q = \{f_q, f_q + 1, f_q + c, f_q + c + 1\}$  for  $q = 0, 1, \dots, m/4 - 1$ .

Denote  $g = \gcd(c, m)$ ,  $c' = c/g$ ,  $m' = m/g$  and notice that  $1 \leq \alpha < \beta$  implies that  $g$  and  $m'$  are both even.

Let  $M$  be the subgroup of  $Z_m$  generated by  $g$ . This is isomorphic to  $Z_{m'}$  where the isomorphism  $\phi : Z_{m'} \rightarrow Z_m$  is defined by  $\phi(i) = gi$ . Because  $c'$  and  $m'$  are relatively prime,  $c'$  is a generator of  $Z_{m'}$ . We can therefore write  $Z_{m'} = \{0, c', 2c', \dots, (m' - 1)c'\}$ . Obviously, because  $m'$  is even, we can decompose  $Z_{m'}$  into sets  $A_p = \{2pc', 2pc' + c'\}$  for  $p = 0, 1, \dots, m'/(4g) - 1$ . Using the isomorphism  $\phi$ , this gives corresponding subsets  $B_{0,p} = \{2pc, 2pc + c\}$  for  $p = 0, 1, \dots, m'/(4g) - 1$  of  $Z_m$ . Now for every coset  $i + M$  of  $Z_m$  we can define the sets  $B_{i,p} = \{2pc + i, 2pc + c + i\}$ , which obviously decompose  $i + M$ . Because  $g$  is even, we have an even number of cosets and can combine the sets from two consecutive cosets into desired quadruples of  $\mathcal{D}$ . In particular, we define  $C_{s,p} = B_{2s,p} \cup B_{2s+1,p} = \{2pc + 2s, 2pc + 2s + 1, 2pc + c + 2s, 2pc + c + 2s + 1\}$  for  $s = 0, 1, \dots, m'/4 - 1$ .

Because every  $q = 0, 1, \dots, m/4 - 1$  can be written uniquely as  $q = sg + p$ , where  $0 \leq s < m'/4$  and  $0 \leq p < g$ , we can set  $D_q = C_{s,p}$  for  $q = sg + p$ .

*Claim 2.* Under the assumptions above,  $K_{m,m}$  can be decomposed into  $H$ .

Define the *length* of an edge  $x_r y_s$  as  $r - s \pmod{m}$  and denote by  $H_0$  the factor of  $K_{m,m}$  isomorphic to  $H$  with edges of lengths  $0, 1, c, c + 1$ . Also denote by  $H_q$  a factor of  $K_{m,m}$  with edges of lengths  $f_q, f_q + 1, f_q + c, f_q + c + 1$ , where  $f_q \in D_q$ . More precisely, in  $H_q$  a vertex  $x_i$  will be adjacent to vertices  $y_j$  where  $j = f_q, f_q + 1, f_q + c, f_q + c + 1$ .

Now we can define mappings  $\psi : H \rightarrow H_q$  for  $q = 0, 1, \dots, m/4 - 1$  by  $\psi(x_i) = x_i, \psi(y_j) = y_{j+f_q}$ . It should be obvious that this mapping is an isomorphism between  $H$  and  $H_q$ . Because each factor  $H_q$  now contains edges of four different lengths belonging to the set  $D_q \in \mathcal{D}$  and according to Claim 1,  $\mathcal{D}$  decomposes  $Z_m$ , the collection of factors  $H_q$  for  $q = 0, 1, \dots, m/4 - 1$  forms a factorization of  $K_{m,m}$ . □

**Theorem 6.** *Let  $n \equiv 0 \pmod{8}$ ,  $j = a2^r$  and  $n = b2^s$ , for some positive integers  $a, b, r, s$ , where  $a, b$  are odd. If  $G$  is an  $(0, j)$ -prism of order  $2n$  and  $r < s$ , then  $G$  decomposes  $K_{\frac{3n}{2}, \frac{3n}{2}}$ .*

*Proof.* For  $j = 0$  or  $j = 2$  we are done by Lemma 3 or 4, respectively. From now on assume that  $j \geq 4$  and let  $p = \frac{j}{2}$ . Recall that it follows from the definition of an  $(0, j)$ -prism that  $j$  is always even.

As in Lemma 3 the graph  $K_{\frac{n}{2}, \frac{n}{2}}$  can be decomposed into  $C_n$  by Theorem 2. For  $(0, 0)$ -prisms the method is based on decomposition of  $C_n[\overline{K}_3]$  into three

prisms (see Lemma 3), but for general  $(0, 2p)$ -prisms we need to pair up two  $(0, 0)$ -prisms and swap half of their spokes (of type 0) so that they will be of type  $j = 2p$  in the other prism.

So we need two cycles  $C_n^1$  and  $C_n^2$ , which will together give us an appropriate collection of  $(0, 2p)$ -prisms. We denote the union  $C_n^1 \cup C_n^2$  of the appropriate cycles by  $H$ . Obviously,  $H$  is a bipartite 4-regular graph of order  $n$ . Let the partite sets be  $X = \{0_0, 1_0, \dots, (\frac{n}{2}-1)_0\}$  and  $Y = \{0_1, 1_1, \dots, (\frac{n}{2}-1)_1\}$ . If the neighbors of any vertex  $i_0 \in X$  in  $H$  are  $i_1, (i+1)_1, (i+d)_1, (i+d+1)_1$  for some even  $d \geq 2$ , then  $H$  consists of cycles  $C_n^1 = 0_1, 0_0, 1_1, 1_0, \dots, j_1, j_0, (j+1)_1, (j+1)_0, \dots, (\frac{n}{2})_1, (\frac{n}{2})_0$  and  $C_n^2 = d_1, 0_0, (d+1)_1, 1_0, \dots, (d+j)_1, j_0, (d+j+1)_1, (j+1)_0, \dots, (d-1)_1, (\frac{n}{2})_0$ . Now the graph  $K_{\frac{n}{2}, \frac{n}{2}}$  can be decomposed into  $H$  by Lemma 5.

As in Lemma 3 denote the vertices of  $C_n^1$  by  $0_1, 0_0, 1_1, 1_0, \dots, (\frac{n}{2}-1)_1, (\frac{n}{2}-1)_0, 0_1$ . By blowing up  $C_n^1$  by  $\overline{K}_3$  we obtain  $C_n^1[\overline{K}_3]$ . Recall that such labeling implies that after blowing up  $C_n^1$  into  $\overline{K}_3$  we can decompose  $C_n^1[\overline{K}_3]$  into three  $(0, 0)$ -prisms the spokes with the dimension either  $(1, 1)$  or  $(0, 1)$ . We consider the following cases:

*Case 1.  $p$  is odd.*

We want to “glue together” two  $C_n^1[\overline{K}_3]$  and  $C_n^2[\overline{K}_3]$  in such a way that we can swap spokes of type 0 and dimension  $(0, 1)$  between  $C_n^1$  and  $C_n^2$  to obtain spokes of dimension  $(-p, 1)$  that will be of type  $j = 2p$  in their new prisms.

It implies that we need edges  $p_1 0_0, (p+1)_1 1_0, (p+2)_1 2_0, \dots$  and  $x_1 0_0, (x+1)_1 1_0, (x+2)_1 2_0, \dots$  in the cycle  $C_n^2$ . Because these two matchings need to form the cycle  $C_n^2$  of length  $n$ , we must have  $px - x - p^2 + p + x \equiv 0 \pmod{\frac{n}{2}}$ .

Because  $x \equiv (p-1) \pmod{\frac{n}{2}}$  is a solution, we can set  $x = p-1$  and get  $C_n^2 = p_0, 1_1, (p+1)_0, 2_1, \dots, (p-1)_0, 0_1$ . Define a graph  $H$  as a union  $C_n^1 \cup C_n^2$ . By blowing up  $H$  by  $\overline{K}_3$  from each vertex  $j_i$  we obtain three vertices  $(j, 0)_i, (j, 1)_i, (j, 2)_i$ . Using that labeling we will show now that we can decompose the graph  $H[\overline{K}_3]$  into six copies of  $(0, 2p)$ -prisms.

Let  $G$  be an  $(0, 2p)$ -prism of order  $2n$ . Notice that we can find two edge-disjoint copies  $G_0, G_3$  of  $G$  in  $H[\overline{K}_3]$  in such a way that for  $G_0$  we define the rims  $R_n^{10} = (0, 0)_1, (0, 1)_0, (1, 0)_1, (1, 1)_0, (2, 0)_1, (2, 1)_0, (3, 0)_1, \dots, (\frac{n}{2}-1, 1)_0, (0, 0)_1, R_n^{20} = (0, 2)_1, (0, 2)_0, (1, 2)_1, (1, 2)_0, (2, 2)_1, (2, 2)_0, (3, 2)_1, \dots, (\frac{n}{2}-1, 2)_0, (0, 2)_1$  and the spokes  $(i+p, 1)_0(i, 2)_1, (i+1, 0)_1(i+1, 2)_0 \in E(G_0)$ , where  $i \in \mathbb{Z}_{\frac{n}{2}}$ . Whereas for  $G_3$  we define the rims  $R_n^{13} = (p-1, 0)_0, (0, 1)_1, (p, 0)_0, (1, 1)_1, (p+1, 0)_0, (2, 1)_1, (p+2, 0)_0, \dots, (p-2, 0)_0, (\frac{n}{2}-1, 1)_1, (p-1, 0)_0, R_n^{23} = (p-1, 2)_0, (0, 2)_1, (p, 2)_0, (1, 2)_1, (p+1, 2)_0, (2, 2)_1, (p+2, 2)_0, \dots, (p-2, 2)_0, (\frac{n}{2}-1, 2)_1(p-1, 2)_0$  and the spokes  $(i, 1)_0(i, 2)_1, (i+p-1, 2)_0(i, 0)_1 \in E(G_3)$ . We can obtain six edge-disjoint copies  $G_0, G_1, \dots, G_5$  in  $H[\overline{K}_3]$ , where  $G_m = \phi_m(G_0)$  and  $\phi_m((a, b)_i) = (a, b+m)_i$  and  $G_{3+m} = \phi_m(G_3)$  and  $\phi_{3+m}((a, b)_i) = (a, b+m)_i$  for  $m = 0, 1, 2$  and  $i = 0, 1$ .

*Case 2.  $p$  is even.*

In that case we want to “glue together” two  $C_n^1[\overline{K}_3]$  and  $C_n^2[\overline{K}_3]$  in such a way that we can swap spokes of type 0 and dimension  $(1, 1)$  obtaining spokes of dimension  $(p+1, 1)$  which are now of type  $j = 2p$ . It follows that  $C_n^2 = (p+1)_1, 1_0, (p+2)_1, 2_0, \dots, p_1, 0_0, (p+1)_1$ . Define a graph  $H$  as a union  $C_n^1 \cup$

$C_n^2$ . By blowing up  $H$  by  $\overline{K_3}$  from each vertex  $j_i$  we obtain three vertices  $(j, 0)_i, (j, 1)_i, (j, 2)_i$ . As in Case 1 we show that we can decompose the graph  $H[\overline{K_3}]$  into six copies of  $(0, 2p)$ -prisms. Let  $G$  be an  $(0, 2p)$ -prism of order  $2n$ . We can find two edge-disjoint copies  $G_0, G_3$  of  $G$  in  $H[\overline{K_3}]$  in such a way that  $G_0$  has the rims  $R_n^{10} = (0, 0)_1, (0, 1)_0, (1, 0)_1, (1, 1)_0, (2, 0)_1, (2, 1)_0, (3, 0)_1, \dots, (\frac{n}{2} - 1, 1)_0, (0, 0)_1, R_n^{20} = (0, 2)_1, (0, 2)_0, (1, 2)_1, (1, 2)_0, (2, 2)_1, (2, 2)_0, (3, 2)_1, \dots, (\frac{n}{2} - 1, 2)_0, (0, 2)_1$  and the spokes  $(i, 1)_0(i + p, 2)_1, (i, 2)_0(i + 1, 0)_1 \in E(G_0)$ , where  $i \in \mathbb{Z}_{\frac{n}{2}}$ . Whereas  $G_3$  has the rims  $R_n^{13} = (p, 0)_1, (0, 1)_0, (p + 1, 0)_1, (1, 1)_0, (p + 2, 0)_1, (2, 1)_0, (p + 3, 0)_1, \dots, (p - 1, 0)_1, (\frac{n}{2} - 1, 1)_0, (p, 0)_1, R_n^{23} = (p, 2)_1, (0, 2)_0, (p + 1, 2)_1, (1, 2)_0, (p + 2, 2)_1, (2, 2)_0, (p + 3, 2)_1, \dots, (p - 1, 2)_1, (\frac{n}{2} - 1, 2)_0, (p, 2)_1$  and the spokes  $(i, 1)_0(i + p, 2)_1, (i, 2)_0(i + 1, 0)_1 \in E(G_3)$ . We obtain six edge-disjoint copies  $G_0, G_1, \dots, G_5$  in  $H[\overline{K_3}]$ , where  $G_m = \phi_m(G_0)$  and  $\phi_m((a, b)_i) = (a, b + m)_i$  and  $G_{3+m} = \phi_m(G_3)$  and  $\phi_{3+m}((a, b)_i) = (a, b + m)_i$  for  $m = 0, 1, 2$  and  $i = 0, 1$ .  $\square$

Unfortunately, this technique cannot be used for the case when  $n \equiv 4 \pmod{8}$ , because we glue together *two*  $(0, 0)$ -prisms and swap spokes, whereas for  $n \equiv 4 \pmod{8}$  we want to decompose  $K_{\frac{3n}{2}, \frac{3n}{2}}$  into an *odd* number of  $(0, j)$ -prisms of order  $2n$ . Therefore, more powerful methods are needed to solve the case  $n \equiv 4 \pmod{8}$ . Also we remark here that we did not attempt to solve the cases when the necessary conditions are satisfied while the complete bipartite graph is *not*  $K_{\frac{3n}{2}, \frac{3n}{2}}$ . For these cases the labeling techniques we used are not suitable and methods borrowed from design theory need to be utilized instead.

## References

1. Bermond, J.C., Huang, C., Sotteau, D.: Balanced cycle and circuit designs: Even case. *Ars. Combinat.* 5, 293–318 (1978)
2. Cichacz, S., Fronček, D.: Factorization of  $K_{n,n}$  into  $(0, j)$ -prisms. *Information Processing Letters* 109, 932–934 (2009)
3. El-Zanati, S.I., Vanden Eynden, C.: Decompositions of  $K_{m,n}$  into cubes. *J. Comb. Designs* 4, 51–57 (1996)
4. Fronček, D.: Cyclic type decompositions of complete bipartite graphs into hypercubes. *Australasian Journal of Combinatorics* 25, 201–209 (2002)
5. Fronček, D.: Oberwolfach rectangular table negotiation problem. *Discr. Math.* 309, 501–504 (2009)
6. Fronček, D., Kovář, P., Kubesa, M.: Decomposition of complete graphs into  $C_m[2]$  (submitted)
7. Frucht, R.W., Gallian, J.A.: Labelling Prisms. *Ars. Combin.* 26, 69–82 (1988)
8. Häggkvist, R.: Decompositions of complete bipartite graphs. In: *Surveys in combinatorics*, Norwich. London Math. Soc. Lecture Note Ser., vol. 141, pp. 115–147. Cambridge Univ. Press, Cambridge (1989)
9. Piotrowski, W.-L.: The solution of the bipartite analogue of the Oberwolfach problem. *Discr. Math.* 97, 339–356 (1991)

10. Sotteau, D.: Decomposition of  $K_{m,n}$  ( $K_{m,n}^*$ ) into cycles (circuits) of length  $2k$ . J. Comb. Theory, Ser. B 30, 75–81 (1981)
11. Rosa, A.: On certain valuations of the vertices of a graph. In: Theory of Graphs (Internat. Symposium), Rome, July 1966, pp. 349–355. Gordon and Breach, N. Y. and Dunod Paris (1967)
12. Ringel, G., Lladó, A.S., Serra, O.: Decomposition of complete bipartite graphs into trees. DMAT Research Report 11/96, Univ. Politècnica de Catalunya

# Edge-Simple Circuits through 10 Ordered Vertices in Square Grids<sup>\*</sup>

David Coudert<sup>1</sup>, Frédéric Giroire<sup>1</sup>, and Ignasi Sau<sup>1,2</sup>

<sup>1</sup> Mascotte joint project - INRIA/CNRS/UNSA - Sophia Antipolis, France

<sup>2</sup> Graph Theory and Combinatorics group - Applied Mathematics IV Department of UPC - Barcelona, Spain

David.Coudert@sophia.inria.fr, Frederic.Giroire@sophia.inria.fr,  
Ignasi.Sau@sophia.inria.fr

**Abstract.** A *circuit* in a simple undirected graph  $G = (V, E)$  is a sequence of vertices  $\{v_1, v_2, \dots, v_{k+1}\}$  such that  $v_1 = v_{k+1}$  and  $\{v_i, v_{i+1}\} \in E$  for  $i = 1, \dots, k$ . A circuit  $C$  is said to be *edge-simple* if no edge of  $G$  is used twice in  $C$ . In this article we study the following problem: which is the largest integer  $k$  such that, given any subset of  $k$  ordered vertices of an infinite square grid, there exists an edge-simple circuit visiting the  $k$  vertices in the prescribed order? We prove that  $k = 10$ . To this end, we first provide a counterexample implying that  $k < 11$ . To show that  $k \geq 10$ , we introduce a methodology, based on the notion of core graph, to reduce drastically the number of possible vertex configurations, and then we test each one of the resulting configurations with an ILP solver.

**Keywords:** square grid, edge-simple circuit, prescribed vertices, ILP solver.

## 1 Introduction

A *circuit* in a simple undirected graph  $G = (V, E)$  is a sequence of vertices  $\{v_1, v_2, \dots, v_{k+1}\}$  such that  $v_1 = v_{k+1}$  and  $\{v_i, v_{i+1}\} \in E$  for  $i = 1, \dots, k$ . A circuit  $C$  is said to be *edge-simple* if no edge of  $G$  is used twice in  $C$ . An edge-simple circuit is also called *closed trail* in the literature. The existence of a circuit through a prescribed set of vertices or edges has been an important graph-theoretical question for many years [2, 3, 4, 7, 8, 9, 11, 12, 13, 16, 17, 20]. Typically, high connectivity is a powerful sufficient condition for the existence of such circuits. For instance, it is well known that in a  $k$ -vertex-connected graph any subset of  $k$  nodes [7] or any subset of  $k - 1$  independent edges [13] is included in a cycle. A circuit  $C$  is a *cycle* if no vertex of  $G$  is used twice in  $C$ , except for  $v_1 = v_{k+1}$ .

---

<sup>\*</sup> This work has been partially supported by European project IST FET AEOLUS, PACA region of France, Ministerio de Ciencia e Innovación, European Regional Development Fund under project MTM2008-06620-C03-01/MTM, and Catalan Research Council under project 2005SGR00256.

However, knowing specific properties of the graph often permits to prove much stronger results. In this article we focus on the existence of edge-simple circuits through specified vertices in the infinite square grid (or equivalently, a large enough torus), which is a widely studied 4-connected graph. In addition, we do not require the circuit only to visit a subset of vertices, but also to visit them in a prescribed order. It is clear that such a circuit in the square grid always exists for any ordered subset of 4 vertices. After thinking for a few minutes it is also easy to convince oneself that the same holds for 5 vertices. On the other hand, it seems intuitive to suspect that this property will not be true for an arbitrary large subset of ordered vertices of the square grid. Therefore, the following question arises: which is the largest integer  $k$  such that, given *any* subset of  $k$  ordered vertices of an infinite square grid, there exists an edge-simple circuit visiting the  $k$  vertices in the prescribed order? Here, we prove that  $k = 10$ .

To obtain this result, one has a priori to test the existence of an edge-simple circuit visiting  $k$  vertices in the prescribed order on the grid, for all possible placements and orderings of the  $k$  vertices. Since the number of possible placements and orderings is prohibitively large, we introduce a methodology, based on the notion of *core graph*, to reduce the number of configurations to be tested. We first provide some background and motivations for the problem in Section 2. We then show in Section 3 that checking the feasibility of a configuration on the grid is equivalent to checking its feasibility on an auxiliary graph, called *internal graph*. Then, in Section 4 we introduce the notion of *core graphs* to reduce drastically the number of internal graphs to be tested. In Section 5 we give a counterexample establishing the upper bound  $k \leq 10$ . In Section 6 we match this upper bound with an ILP solver to exhaustively test all the orderings for a small list of possible configurations that we obtained after applying the reductions of Sections 3 and 4. Finally, Section 7 concludes the article.

## 2 Background and Motivation

Connectivity is one of the cornerstone concepts of graph theory. Maybe the most archetypal results are Menger's classical theorems [6], which say that a graph is  $k$ -vertex-connected (resp.  $k$ -edge-connected) if and only if it contains  $k$  vertex-disjoint (resp. edge-disjoint) paths between any two vertices. There is a huge literature concerning extremal problems of cycles in  $k$ -connected graphs. For instance, it is well known that in a  $k$ -vertex-connected graph any subset of  $k$  nodes [7] or any subset of  $k-1$  independent edges [13] is included in a cycle. There are a number of works giving necessary or sufficient conditions for the existence of a cycle through a specified set of vertices in a general graph [16, 4, 8, 17].

Some stronger results have been given for specific classes of graphs, like 3-connected cubic graphs [9, 8]. For this class of graphs it is known that there exists a cycle through any 9 vertices, and that there exists a cycle which passes through any 10 given vertices if and only if the graph is not contractible to the Petersen graph [9] in such a way that each of the 10 vertices maps to a distinct vertex of the Petersen graph. If, in addition, the 3-connected cubic graph is planar,

then there exists a cycle through any 23 vertices [2]. Another example can be found in [11], where the authors provide necessary and sufficient conditions for a given graph embedded on the torus to contain edge-disjoint cycles satisfying prescribed topological properties.

*The disjoint paths problem.* Observe that, in a general (di)graph, the problem of deciding whether there exist edge-disjoint paths between given pairs of vertices is NP-complete [15] (even if the graph is a square grid [18]). When the number of pairs of vertices is bounded by a constant, the disjoint paths problem is polynomial in undirected graphs [21], NP-complete in directed graphs [19] (even with only two pairs of vertices [10]), and polynomial in symmetric directed graphs [14].

However, all these results do not take into account the *order* in which the cycle visits the prescribed set of nodes. This is a natural constraint, since for example in telecommunication networks it may be important to connect a subset of nodes in such a way that each node numbered  $i$  has capability to communicate only with the two nodes numbered  $i - 1$  and  $i + 1$  (modulo the cardinality of the subset of nodes). This could be the case, for instance, of the classical *token ring* networks defined by the standard *IEEE* 802.5. That is, there exists a whole class of problems to consider when the constraint on the order is introduced. In this article we study one of these problems in square grids.

When designing a telecommunication network, the fault tolerance is a crucial issue. Observe that the simplest network which is able to support any single link failure is an edge-simple circuit, and that is one of the main reasons why the study of such circuits is important. The study of the square grid is also natural, due among other reasons to its extensive use in parallel computing. In this context, it is interesting to know which is the largest integer  $k$  for which there always exists a circuit visiting any ordered subset of at most  $k$  nodes. Observe also that without taking into account the ordering, there exists a cycle (and thus, a circuit) visiting any subset of vertices of the square grid, since the square grid is a Hamiltonian graph.

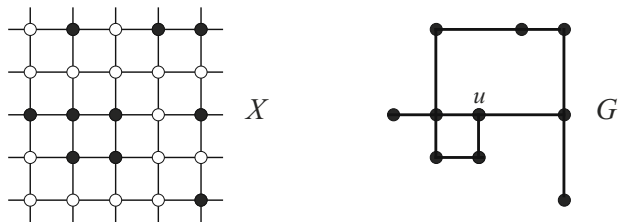
It is worth mentioning that the square grid is in some sense the common *skeleton* of planar graphs. Indeed it is well-known that every planar graph of branchwidth at least  $\ell$  contains an  $(\lfloor \ell/4 \rfloor \times \lfloor \ell/4 \rfloor)$ -grid as a minor [22]. Therefore, a square grid is *inside* every planar graph, and any edge-disjoint circuit in a minor of a graph can be easily transformed to an edge-disjoint circuit in the graph itself.

### 3 Preliminaries

In this section we introduce some definitions to be used throughout. We use standard graph terminology (see, for instance, [6]).

**Definition 1 (Configuration, Feasible Configuration).** A configuration  $X$  is a subset of vertices of the infinite square grid. A configuration  $X$  is feasible if, for any permutation  $\sigma$  of the vertices of  $X$ , there exists an edge-simple circuit





**Fig. 1.** A configuration  $X$  (defined by the full dots) and its corresponding internal graph  $G$

in the infinite square grid joining the vertices of  $X$  following the ordering given by  $\sigma$ .

**Definition 2 (Internal Graph, Internal and External Degree).** Given a subset

$X = \{u_1, \dots, u_n\}$  of nodes in the square grid, the internal graph  $G = (V, E)$  of  $X$  is the graph with  $V = \{v_1, \dots, v_n\}$ , and for  $u_i, u_j \in X$ ,  $\{v_i, v_j\} \in E$  if and only if  $u_i$  and  $u_j$  are on the same row (or column) and there is no other  $z \in X$  between  $u_i$  and  $u_j$  on that row (or column).

Given  $u \in X$ , the internal degree  $d_{in}(u)$  of  $u$  is the degree of  $u$  in the internal graph  $G$  of  $X$ , i.e.,  $d_G(u)$ . Similarly, the external degree of  $u \in X$  is  $d_{out}(u) = 4 - d_{in}(u)$ . A vertex  $u \in X$  is isolated if  $d_{in}(u) = 0$ .

For example, in Fig. 1, a configuration  $X$  in the square grid (defined by the full dots) and its corresponding internal graph  $G$  are depicted. The vertex labeled  $u$  satisfies  $d_{in}(u) = 3$  and  $d_{out}(u) = 1$ .

Since we deal with an infinite square grid, any two vertices of an internal graph  $G$  with external degree at least one can be connected with a path in the grid without using any edge of  $G$ . This is because a vertex that has external degree at least one has no neighbor in the internal graph along an infinite semirow or semicolumn of the grid. This fact can be modeled in the following way: given an internal graph  $G$ , we construct a (multi)graph  $\hat{G}$  from  $G$  by adding a new vertex  $\infty$  and, for each vertex  $u \in V(G)$ ,  $d_{out}(u)$  copies of the edge  $\{u, \infty\}$ .

**Definition 3 (Feasible internal graph).** An internal graph  $G$  is feasible if, for all the permutations  $\sigma$  of the vertices of  $G$ , there exists an edge-simple circuit in  $\hat{G}$  joining the vertices of  $G$  following the ordering given by  $\sigma$ .

The following lemma follows easily from the above definitions.

**Lemma 1.** A configuration  $X$  is feasible if its internal graph  $G$  is feasible.

Observe that the fact that  $G$  is feasible is a sufficient (but *not* necessary) condition for  $X$  to be feasible. Intuitively, the internal graph captures the most difficult case among all the configurations having the same internal graph.

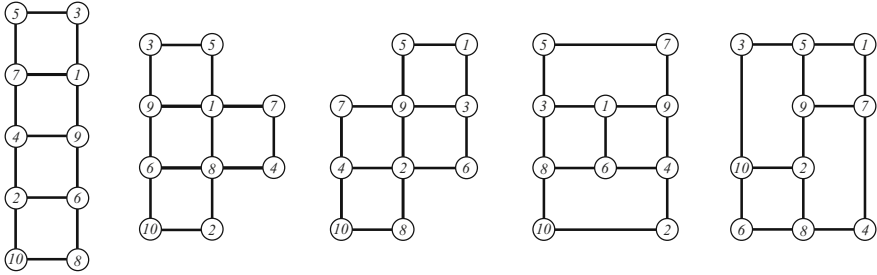


Fig. 2. Some feasible internal graphs on 10 vertices

Before getting into technical results, and in order to get familiar with the problem, the curious reader may verify that the internal graphs on 10 vertices depicted in Fig. 2 (together with a numbering of their vertices) are feasible. We shall see in Section 6 that this fact is not a coincidence, since any configuration on 10 vertices is feasible.

### 4 Reducing the Problem

We now prove several technical lemmata to be used in the sequel of the article. The objective is to reduce the number of configurations to be tested.

**Lemma 2.** *Any internal graph in which all vertices have external degree at least 2 is feasible.*

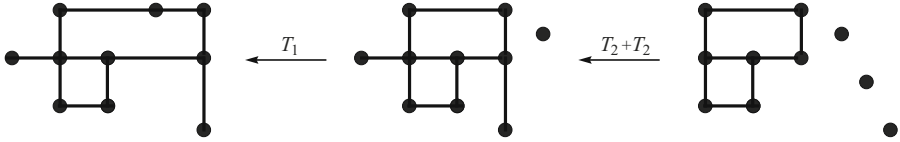
**Proof:** Let  $G$  be an internal graph in which all vertices have external degree at least 2, and assume that the vertices are ordered  $v_1, v_2, \dots, v_k$  by the permutation  $\sigma$ . Then the circuit  $\{v_1, \infty, v_2, \infty, v_3, \dots, v_{k-1}, \infty, v_k, \infty, v_1\}$  is a solution in  $G$ .  $\square$

**Lemma 3.** *If an internal graph  $G$  is feasible then any internal graph  $H$  that is a subgraph of  $G$  is feasible.*

**Proof:** Let  $G$  be a feasible internal graph, and let  $H$  be a subgraph of  $G$ . Assume first that  $|V(H)| = |V(G)|$ , and let  $v_1, \dots, v_k$  be an ordering of the vertices of  $H$ . Consider a solution  $C$  in  $G$  for the same ordering  $v_1, \dots, v_k$  of the vertices of  $G$ . A solution in  $H$  is obtained from  $C$  by replacing each  $\{u, v\} \in E(G) \setminus E(H)$  with the edges  $\{u, \infty\}, \{\infty, v\}$ . Otherwise, if  $|V(H)| = k < n = |V(G)|$ , given an ordering  $v_1, \dots, v_k$  of  $V(H)$ , consider a solution  $C$  in  $G$  for an ordering of  $V(G)$  that coincides with  $v_1, \dots, v_k$  when restricted to  $V(H)$ . Then the above replacement transforms  $C$  into a solution in  $H$ .  $\square$

The proofs of the three following lemmas can be found in 5. Two internal graphs  $G_1$  and  $G_2$  are said to be *equivalent* if  $G_2$  is feasible if and only if  $G_1$  is.

**Lemma 4.** *Any two isomorphic internal graphs  $G_1$  and  $G_2$  are equivalent.*



**Fig. 3.** We can restrict ourselves to core graphs. An arrow from a graph  $G$  to a graph  $H$  means that if  $G$  is feasible, so is  $H$  (due to either transformation  $T_1$  or transformation  $T_2$ ).

**Lemma 5.** *If an internal graph  $G$  is feasible, then any internal graph  $G'$  that can be obtained from  $G$  via the following transformation  $T_1$  is also feasible:*

- (1) Choose from  $G$  an isolated vertex  $u$  and an edge  $\{x, y\}$ .
- (2) Remove  $u$ , add a new vertex  $v$ , and replace the edge  $\{x, y\}$  with the edges  $\{x, v\}, \{v, y\}$ .

**Lemma 6.** *If an internal graph  $G$  is feasible, then any internal graph  $G'$  that can be obtained from  $G$  via the following transformation  $T_2$  is also feasible:*

- (1) Choose from  $G$  two vertices  $u$  and  $w$ , such that  $u$  is isolated and  $d_{in}(w) \leq 3$ .
- (2) Remove  $u$ , and add a new vertex  $v$  and the edge  $\{w, v\}$ .

Combining inductively Lemmas 5 and 6, we deduce that if  $G'$  is an internal graph obtained from a feasible graph  $G$  with a sequence of the transformations  $T_1$  and  $T_2$ , then  $G'$  is also feasible. In practice, this means that in any internal graph we can take the vertices that lie in the *middle* of a path and the vertices with internal degree one, and put them as isolated vertices. If the resulting graph is a feasible internal graph, then by Lemmas 5 and 6, so is the original one. In other words, we can restrict ourselves to internal graphs  $G$  whose connected components (except isolated vertices) have at least two vertices in each row and each column.

**Definition 4 (Core Graph,  $\ell$ -core Graph).** *An internal graph is a core graph if all its non-edgeless connected components have at least two vertices in each row and each column. A core graph  $G$  on  $k$  vertices is an  $\ell$ -core graph if  $G$  has  $k - \ell$  isolated vertices.*

Lemmas 5 and 6 imply that we can restrict ourselves to core graphs. For instance, consider the example of Fig. 3. The leftmost internal graph (which is the same example of Fig. 1) can be obtained by a sequence of the transformations  $T_1$  and  $T_2$ . Thus, to prove that the three internal graphs of Fig. 3 are feasible it is enough to prove it for the rightmost graph, which is a 7-core graph.

This simplification reduces the number of configurations dramatically. In particular, the above discussion together with Lemma 2 proves that all forests are feasible. Therefore, if we want to know if all the configurations on  $k$  vertices are feasible, it suffices to test all the core graphs on  $k$  vertices; this is the topic of Section 6 for  $k = 10$ . Summarizing,

**Proposition 1.** *If all the core graphs on  $k$  vertices are feasible, then all the configurations on  $k$  vertices are feasible.*

Note that if all the configurations on  $k$  vertices are feasible, then clearly so are all the configurations on  $k'$  vertices, for every  $k' < k$ .

We introduce a last criterium to deduce the feasibility of an internal graph on 10 vertices. The proof can be found in [5].

**Lemma 7.** *All the 10-core graphs on 10 vertices whose non-edgeless connected components can be obtained from a triple edge by subdividing edges are feasible.*

## 5 Upper Bound: $k < 11$

In this section we show an unfeasible counterexample proving that  $k < 11$ . For the sake of the presentation, we first describe a simple configuration showing that  $k < 12$ .

Given a set  $X = \{1, 2, 3, \dots, 12\}$  of ordered nodes in a square grid  $G$ , let  $X_e$  (resp.  $X_o$ ) be the subset of nodes of  $X$  with an even (resp. odd) number, and note that any path joining two consecutive vertices must go from  $X_o$  to  $X_e$ , or viceversa. Let  $X_o$  be the set displayed in Fig. 4. Then, regardless of the placement of  $X_e$ , we need at least 12 edges outgoing from the graph induced by  $X_o$  to route the 12 paths, but there are only 10 such edges (the thick edges in Fig. 4). So, this configuration is unfeasible for any placement of  $X_e$ .

Before providing the counterexample showing that  $k < 11$ , we need the following definition.

**Definition 5 (Internal Path).** *Given an internal graph  $G$ , a permutation  $\sigma$  of  $X$ , a solution  $C$  to the instance  $(G, \sigma)$ , and a subset  $S \subseteq X$ , an internal path in  $S$  is a subpath  $P$  of  $C$  linking two consecutive vertices of  $X$  according to  $\sigma$ , such that  $P$  is a subgraph of  $G[S]$ .*

Given a subset of vertices  $S \subseteq X$ , the *paths originating from  $S$*  are paths with at least one endpoint in  $S$ .

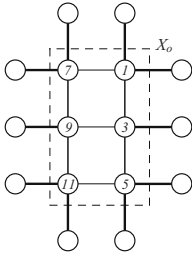
**Proposition 2.**  $k < 11$ .

**Proof:** Let  $(G, \sigma)$  be the internal graph on 11 vertices together with the ordering depicted in Fig. 5.

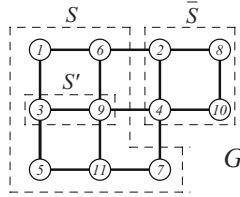
Suppose for the sake of contradiction that there exists a solution  $C$  to the instance  $(G, \sigma)$ . Let  $S = \{1, 6, 3, 9, 5, 11, 7\} \subseteq X$ , and let  $\bar{S} = \{2, 8, 4, 10\}$  (see Fig. 5(a)). Note that there are 12 edges outgoing from  $G[S]$  to the rest of the grid.

**Claim 1.**  $C$  contains exactly 1 internal path in  $S$ .

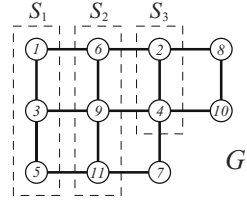
**Proof:** Suppose first that there is no internal path in  $S$ . Therefore, each path originating from  $S$  uses at least 2 edges outgoing from  $G[S]$ . Since  $|S| = 7$ , there



**Fig. 4.** Counterexample showing that  $k < 12$



(a)



(b)

**Fig. 5.** Counterexample  $(G, \sigma)$  of Proposition 2 showing that  $k < 11$ , together with the vertex sets defined in the proof

must be 14 edges in  $C$  outgoing from  $G[S]$  to the rest of the grid, but there are only 12. Therefore,  $C$  contains at least 1 internal path in  $S$ .

Suppose now that  $C$  contains at least 2 internal paths in  $S$ . Let  $S' = \{3, 9\}$  (see Fig. 5(a)), and note that there are 6 edges outgoing from  $G[S']$ . Note also that the only possible internal paths in  $S$  are  $5 \rightarrow 6$ ,  $6 \rightarrow 7$ , and  $11 \rightarrow 1$ , so any internal path in  $S$  must cross  $S'$ . Therefore, there can be at most 2 such internal paths, and those 2 paths use 4 edges outgoing from  $G[S']$ . Thus, only  $6 - 4 = 2$  outgoing edges from  $G[S']$  are left, which are not enough to route the 4 subpaths in  $C$  containing the vertices of  $S'$ . Therefore,  $C$  contains exactly 1 internal path in  $S$ .  $\square$

Claim 1 implies all the edges outgoing from  $G[S]$  are used by  $C$  to route paths originating at  $S$ . Let  $S_1 = \{1, 3, 5\}$  and  $S_3 = \{2, 4\}$  (see Fig. 5(b)).

**Claim 2.**  $C$  contains at least 2 internal paths from  $S_1$  to  $S_3$ .

**Proof:** Note that subgraph  $G[S_3]$  has 6 outgoing edges. Since all the edges outgoing from  $G[S]$  are used by  $C$ , exactly 3 paths go from  $S$  to  $\bar{S}$  in  $C$ . Clearly, the 4 subpaths in  $C$  containing the vertices of  $S_3$  use 4 outgoing edges from  $G[S_3]$ . Note that all paths from  $S$  to  $S_3$  are from  $S_1$ .

If there is no path in  $C$  from  $S_1$  to  $S_3$ , then the 3 paths from  $S$  to  $\bar{S}$  cross  $S_3$ , so no edge outgoing from  $G[S_3]$  would be left to route the paths originating from  $S_3$ , which is a contradiction.

If there is 1 path in  $C$  from  $S_1$  to  $S_3$ , then 2 paths from  $S$  to  $\bar{S}$  cross  $S_3$ , so altogether the 3 paths from  $S$  to  $\bar{S}$  use 5 out of the 6 outgoing edges from  $G[S_3]$ . However, 3 additional outgoing edges from  $G[S_3]$  would be needed to route the 3 remaining paths originating from  $S_3$ , which is a contradiction.  $\square$

Consider now  $S_2 = \{6, 9, 11\}$  (see Fig. 5(b)). The subgraph  $G[S_2]$  has 8 outgoing edges, and 6 of them are required in  $C$  to route the paths originating at  $S_2$ , so only 2 edges outgoing from  $G[S_2]$  are still available in  $C$ . But, by Claim 2,  $C$

contains at least 2 internal paths from  $S_1$  to  $S_3$  (which cross  $S_2$ ), hence using 4 outgoing edges from  $G[S_2]$ . The proposition follows.  $\square$

## 6 Lower Bound: $k \geq 10$

To show that  $k \geq 10$ , one has a priori to test all the configurations with 10 vertices on the grid are feasible. But, the number of such configurations is prohibitively big, as testing a single configuration may take a non-negligible (see discussion below). Hence we introduce a methodology, based on the notion of *core graph* (see the results of Section 4), to reduce the number of configurations to be tested.

A naïve strategy to generate all configurations is to consider all the possibilities of placing 10 points in the square grid. However, we showed in Proposition 1 that we only need to consider *core graphs* with 10 vertices (Definition 4). In addition, these core graphs can be considered modulo isomorphism (Lemma 4). It is clear that the smallest integer  $i$  such that an  $i$ -core on 10 vertices exists is 4, and in that case the non-edgeless connected component of the 4-core is a 4-cycle. Such a core is always feasible due to Lemma 2, because all the vertices have external degree at least 2. It is also easy to check that, due to the topology of the grid, a 5-core cannot exist. One can also verify that the only 6-core in which not all vertices have external degree at least 2 is a  $2 \times 3$ -grid. Therefore, it is enough to test this 6-core plus all the  $\ell$ -cores on 10 vertices, for  $\ell = 7, 8, 9, 10$ . The procedure to generate the core graphs to be tested is detailed in Algorithm 1. The complete code and some examples as well can be found at [1].

**Proposition 3.** *The feasibility of any configuration on 10 vertices follows from Algorithm 1.*

The proof can be found in [5].

*Remark 1.* In step 12 of Algorithm 1, we partition  $T_\ell$  into isomorphism classes. This step could take a non-negligible time if we just test if each pair of graphs are isomorphic. To deal with this problem, we first carry out a sieve according to the sorted degree sequence of the vertices and the sorted degree sequences of the neighbours of each vertex. That is, if two graphs do not have the same sequence of degrees and degrees of the neighbours of each vertex, we infer directly that these two graphs are not isomorphic. This sieve reduces the computation time considerably.

*Remark 2.* Observe that, due to Lemma 5, the internal graphs without at least 2 vertices per row and column could have been already removed from  $T_\ell$  after step 2. The reason why we kept those graphs until step 15 is that some graphs that do have at least 2 vertices per row and column are isomorphic to graphs without at least 2 vertices per row and column, so we can also remove them from  $T_\ell$ .

Table 1 summarizes the number of  $\ell$ -cores obtained while running Algorithm 1, for  $\ell \in \{6, 7, 8, 9, 10\}$ . The numbers given in the first row (initial number of internal graphs) follow from the introduction of internal graphs; without it, we would

**Algorithm 1.** Test configurations on 10 vertices

---

```

1: for  $\ell = 6$  to 10 do
2:   generate a list  $\mathcal{T}_\ell$  of all the internal graphs on  $\ell$  vertices in an  $(\lfloor \frac{\ell}{2} \rfloor \times \lfloor \frac{\ell}{2} \rfloor)$ -grid
   (modulo translations, symmetries, and compression of empty rows or columns)
3:   // INTERNAL DEGREE 1:
4:   for all  $G \in \mathcal{T}_\ell$  such that  $G$  has some vertex of internal degree 1 do
5:     remove  $G$  from  $\mathcal{T}_\ell$  {Lemma 6}
6:   // EXTERNAL DEGREE AT LEAST 2:
7:   for all  $G \in \mathcal{T}_\ell$  such that all the vertices of  $G$  have external degree at least 2
   do
8:     remove  $G$  from  $\mathcal{T}_\ell$  {Lemma 2}
9:   // ISOMORPHIC GRAPHS:
10:  partition  $\mathcal{T}_\ell$  into classes  $\mathcal{G}_1, \dots, \mathcal{G}_n$  of isomorphic graphs
11:  for  $i = 1$  to  $n$  do
12:    if there exists  $G \in \mathcal{G}_i$  without at least two vertices per row and column then
13:      remove from  $\mathcal{T}_\ell$  all the graphs in  $\mathcal{G}_i$  {Lemma 4 and Proposition 7}
14:    else
15:      remove from  $\mathcal{T}_\ell$  all the graphs in  $\mathcal{G}_i$  except one {Lemma 4}
16:  // SUBDIVISION OF TRIPLE EDGE:
17:  if  $\ell = 10$  then
18:    for all  $G \in \mathcal{T}_\ell$  such that  $G$  can be obtained from a triple edge by subdividing
    edges do
19:      remove  $G$  from  $\mathcal{T}_\ell$  {Lemma 7}
20:  // SUBGRAPHS:
21:  for each pair of graphs  $G, H \in \mathcal{T}_\ell$  such that  $H$  is a subgraph of  $G$  do
22:    remove  $H$  from  $\mathcal{T}_\ell$  {Lemma 3}
23:   $b_\ell \leftarrow 1$ 
24:  for each  $G \in \mathcal{T}_\ell$  do
25:     $G' \leftarrow G + (10 - \ell)$  isolated vertices
26:    for each permutation  $\sigma$  of the vertices of  $G'$  do
27:      test if  $(G', \sigma)$  is feasible using an LP solver
28:      if  $(G', \sigma)$  is not feasible then
29:         $b_\ell \leftarrow 0$ 
30:  if  $(b_6 \cdot b_7 \cdot b_8 \cdot b_9 \cdot b_{10}) == 1$  then
31:     $k = 10$ 
32:  else
33:     $k < 10$ 

```

---

have a much greater number of configurations to test. Note that the results of Section 4 induce an overall reduction from 4714 to 52 graphs.

**Testing the Feasibility of Core Graphs.** Recall that for each core graph  $G$  on 10 vertices,  $G$  is feasible if for any ordering of  $V(G)$  there is an edge-simple circuit visiting  $V(G)$  in the prescribed order. W.l.o.g. we can assign to one of the vertices of  $G$  the number 1 of the permutation (modulo cyclic permutations), and then for each core graph one has to test  $9! = 362.880$  possibilities.

**Table 1.** Number of  $\ell$ -core graphs on 10 vertices in Algorithm 1. A *single* graph is a graph with a line or column with only one vertex.

Graphs \ $\ell$	6	7	8	9	10	Total
Initial number of internal graphs	1	7	53	485	4166	4714
Number of isomorphisms	0	3	42	453	4051	4581
Number of subgraphs	0	0	5	10	58	73
Number of single graphs	0	2	6	22	74	104
Final number of internal graphs	1	2	4	10	35	52

For each core graph  $G$  and permutation  $\sigma$ , the problem we study can be easily formulated as an integer multicommodity flow problem in a graph with unitary capacity on the edges and so as an integer linear program (ILP). Indeed, the existence of an edge-simple circuit  $C_\sigma$  in a core graphs  $G$  is equivalent to the existence of  $k$  edge-disjoint paths in  $G$  between the pairs of vertices (or *commodities*)  $\{\sigma(1), \sigma(2)\}, \dots, \{\sigma(k-1), \sigma(k)\}, \{\sigma(k), \sigma(1)\}$ . Thus, a feasible solution of the ILP implies the existence of an edge simple circuit, and this feasibility can be quickly checked using an ILP solver (for instance, CPLEX).

In average, testing the  $9!$  permutations for each internal graph takes around 40 minutes on a PC with an Intel Core 2 Duo CPU 2.33GHz running Fedora 8 (see 1), so testing the 4714 internal graphs would take around 4 months and a half. Testing the 52 remaining graphs has taken just 35 hours and 37 minutes 2.

Running the ILP solver on the configurations given by Algorithm 1, we obtained that all  $\ell$ -cores are feasible for each  $\ell \in \{6, 7, 8, 9, 10\}$ . Therefore, combining Propositions 2 and 3 yields that

**Theorem 1.** *There exists an edge-simple circuit through any set of 10 ordered vertices of an infinite square grid.*

## 7 Concluding Remarks

In this article we showed that given any subset of 10 ordered vertices of an infinite square grid, there exists an edge-simple circuit visiting the 10 vertices in the prescribed order, and that the number 10 cannot be replaced by 11. To do so, we introduced a methodology to reduce the problem to a small number of configurations, which were then exhaustively tested using an ILP solver. The details about the implementation of our algorithm are available at 1. Finding a purely combinatorial proof of this result remains open.

Another avenue for further research could be to impose a bound on the size of the grid or torus, namely to consider an  $(n_1 \times n_2)$ -torus and to find the largest integer  $k(n_1, n_2)$  such that given any subset of  $k(n_1, n_2)$  ordered vertices in an  $(n_1 \times n_2)$ -torus, there exists an edge-simple circuit visiting the  $k(n_1, n_2)$  vertices in the prescribed order.

Another direction is to consider another graphs instead of the square grid, like triangular and hexagonal grids and, more generally, general planar graphs or graphs of bounded treewidth.



Finally, adding the constraint of the prescribed order to the classical problems concerning the existence of circuits (see related work in Section 2), creates a whole family of new problems to consider.

## References

1. <http://www-sop.inria.fr/members/Frederic.Giroire/circuit>
2. Aldred, R., Bau, S., Holton, D., McKay, B.: Cycles through 23 vertices in 3-connected cubic planar graphs. *Graphs and Combinatorics* 15, 373–376 (1999)
3. Bau, S., Holton, D.: Cycles containing 12 vertices in 3-connected cubic graphs. *J. Graph Theory* 15(4), 421–429 (1991)
4. Bondy, J., Lovász, L.: Cycles through specified vertices of a graph. *Combinatorica* 1, 117–140 (1981)
5. Coudert, D., Giroire, F., Sau, I.: Circuit visiting 10 ordered vertices in infinite grids. Technical Report RR-6910, INRIA (2009)
6. Diestel, R.: *Graph Theory*. Springer, Heidelberg (2005)
7. Dirac, G.: In abstrakten Graphen vorhandene vollständige 4-Graphen und ihre Unterteilungen. *Math. Nachr.* 22, 61–85 (1960)
8. Egawa, Y., Glas, R., Locke, S.: Cycles and paths through specified vertices in  $k$ -connected graphs. *J. Comb. Theory Ser. B* 52, 20–29 (1991)
9. Ellingham, M., Holton, D., Little, C.: Cycles through ten vertices in 3-connected cubic graphs. *Combinatorica* 4, 265–273 (1984)
10. Fortune, S., Hopcroft, J., Wyllie, J.: The directed subgraph homeomorphism problem. *Journal of Theoretical Computer Science* 10(2), 111–121 (1980)
11. Frank, A., Schrijver, A.: Edge-Disjoint Circuits in Graphs on the Torus. *J. Comb. Theory Ser. B* 55(1), 9–17 (1992)
12. Göring, F., Harant, J., Hexel, E., Tuzac, Z.: On short cycles through prescribed vertices of a graph. *Discrete Mathematics* 286(1-2), 67–74 (2004)
13. Häggkvist, R., Thomassen, C.: Circuits through specified edges. *Discrete Math.* 41, 29–34 (1982)
14. Jarry, A., Pérennes, S.: Disjoint paths in symmetric digraphs. *Discrete Applied Mathematics* 157(1), 90–97 (2009)
15. Karp, R.M.: On the complexity of combinatorial problems. *Networks* 5, 45–68 (1975)
16. Kawarabayashi, K.: Cycles through a prescribed vertex set in  $n$ -connected graphs. *J. Comb. Theory Ser. B* 90(2), 315–323 (2004)
17. Kelmans, A., Lomonosov, M.: When  $m$  vertices in a  $k$ -connected graph cannot be walked round along a simple cycle. *Discrete Math.* 38, 317–322 (1982)
18. Kramer, M., van Leeuwen, J.: Wire-routing is NP-complete. Technical Report RUU-CS-82-4, Department of Computer Science, University of Utrecht (1982)
19. LaPaugh, A.S., Rivest, R.L.: The subgraph homeomorphism problem. *J. Comput. Syst. Sci.* 20(2), 133–149 (1980)
20. Lovász, L.: On some connectivity properties of Eulerian graphs. *Acta Math. Acad. Sci. Hungar.* 28, 129–138 (1976)
21. Robertson, N., Seymour, P.: Graph minors XIII: the disjoint paths problem. *J. Comb. Theory Ser. B* 63, 65–110 (1995)
22. Robertson, N., Seymour, P., Thomas, R.: Quickly excluding a planar graph. *J. Comb. Theory Ser. B* 62(2), 323–348 (1994)

# Efficient Neighborhood Encoding for Interval Graphs and Permutation Graphs and $O(n)$ Breadth-First Search

Christophe Crespelle<sup>1</sup> and Philippe Gambette<sup>2,\*</sup>

<sup>1</sup> CNRS - Univ. Paris 6

`christophe.crespelle@lip6.fr`

<sup>2</sup> LIRMM, Univ. Montpellier 2 - CNRS

`gambette@lirmm.fr`

**Abstract.** In this paper we address the problem of designing  $O(n)$  space representations for permutation and interval graphs that provide the neighborhood of any vertex in  $O(d)$  time, where  $d$  is its degree. To that purpose, we introduce a new parameter, called linearity, that would solve the problem if bounded for the two classes. Surprisingly, we show that it is not. Nevertheless, we design representations with the desired property for the two classes, and we implement the Breadth-First Search algorithm in  $O(n)$  time for permutation graphs; thereby lowering the complexity of All Pairs Shortest Paths and Single Source Shortest Path problems for the class.

## 1 Introduction

Interval graphs are the intersection graphs of intervals of the real line, and permutation graphs are the intersection graphs of segments joining two parallel lines. They are extensively studied graph classes. One of the reasons is that they naturally appear in many contexts such as scheduling, genomics, phylogeny and archeology. On these two classes, a lot of usually NP-complete problems (e.g. coloring, maximum clique, domination) admit very efficient and elegant solutions (see [7,11]). These algorithms are based on geometric definitions of the two classes, which give rise to  $O(n)$  space representations providing adjacency information<sup>1</sup> between an arbitrary pair of vertices in  $O(1)$  time, while the adjacency matrix takes  $O(n^2)$  space, where  $n$  is the number of vertices in the graph.

Though it seems a natural question, the issue of designing  $O(n)$  space data structures providing the neighborhood of an arbitrary vertex in  $O(d)$  time, where  $d$  is its degree, has never been investigated for any of the two classes. For arbitrary graphs, the question of finding compact representations providing optimal time neighborhood queries is actually a practical issue [12]. The compactness of such representations allows to store the graph entirely in memory, and preserve the

---

\* This work was supported by the French ANR project ANR-06-BLAN-0148-01 (GRAAL).

<sup>1</sup> That is, answering the question "Is  $x$  adjacent to  $y$ ?"

complexity of algorithms using neighborhood queries. The conjunction of these two advantages has great impact on the running time of algorithms managing large amount of data.

Does there exist a  $O(n)$  space structure providing neighborhoods in  $O(d)$  time for interval and permutation graphs? There is a natural approach that one cannot avoid to consider. It consists in trying to extend some known results for subclasses of interval graphs or permutation graphs that are known to have very good properties with regard to neighborhood encoding. The proper (or unit) interval graphs are the subclass of interval graphs that admit a model whose intervals all have the same length. They are also characterized as the graphs admitting a linear order on their vertices such that the closed<sup>2</sup> neighborhood of each vertex is an interval [10]. Some compression techniques are based on this notion [11,12]: they try to find orders of the vertices that group the neighborhoods together, as much as possible. If one uses one single order on the vertices and allows the closed neighborhoods of the graph to be split in at most  $k$  intervals, the minimum value of  $k$  which make possible to encode the graph in this way is a known parameter called *closed contiguity*. Let us mention that [6,14] showed that deciding whether a graph has closed contiguity at most  $k$  is NP-complete for any fixed  $k \geq 2$ , and [5] gave an upper bound on the value of the parameter for arbitrary graphs. Another possible way of generalization is to use at most  $k$  orders on the vertices such that the closed neighborhood of a vertex is the union of one interval in each of the  $k$  orders. It gives rise to a new parameter, that we call *closed linearity*, which is always less or equal to the closed contiguity. Concerning the corresponding notions for open neighborhoods, it is known that bipartite permutation graphs have<sup>3</sup> *open contiguity 1* [3], or equivalently *open linearity 1*. Then, it seems unavoidable to ask whether interval and permutation graphs have one of the four parameters mentioned above bounded. Unfortunately, none of them is bounded for any of the two classes. We show that the linearity (closed or open) can be up to  $\Omega(\log n / \log \log n)$ . However, following another approach, we devise  $O(n)$  space data structures, for both interval graphs and permutation graphs, that provide neighborhoods in  $O(d)$  time, and that can be computed from an intersection model of the graph in  $O(n)$  time. This gives new possibilities for the applications dealing with big interval or permutation graphs on which neighborhood queries are needed.

The fact that the neighborhood representation question had not been risen before for interval and permutation graphs is even more surprising considering that, on arbitrary graphs, many algorithmic problems are efficiently solved thanks to adjacency lists, which provide  $O(d)$  time neighborhood queries. Part of the reason is that the structure induced by the intersection models of these two classes often allows to avoid addressing the problem. However, we believe

---

<sup>2</sup> The closed neighborhood is the classic neighborhood augmented with the vertex itself. We will refer to the classic neighborhood as the open neighborhood, in order to avoid confusion between the two notions.

<sup>3</sup> The graphs having open contiguity 1 are exactly biconvex graphs, which is a subclass of bipartite graphs that properly contains bipartite permutation graphs.

that the problem of efficiently managing neighborhood in interval and permutation graphs may lead to new algorithmic developments for the two classes by improving or simplifying some algorithms. As an illustration of the interest of considering this question, we show how to implement an  $O(n)$  time Breadth-First Search algorithm (BFS for short) for permutation graphs. This lowers the complexity of finding All Pair Shortest Paths and Single Source Shortest Paths in a non weighted graph of the class to respectively  $O(n^2)$  and  $O(n)$ .

**Outline of the Paper.** Section 2 gives some basic definitions and notations. In Section 3 we formally define the closed linearity and show that this parameter can be up to  $\Omega(\log n / \log \log n)$  for both interval graphs and permutation graphs. In Section 4, we design an  $O(n)$  space data structure providing the neighborhood of any vertex in  $O(d)$  time. Finally, in Section 5 we implement the BFS algorithm in  $O(n)$  time for permutation graphs.

## 2 Preliminaries

All graphs considered here will be finite, undirected, loopless and simple. In the following,  $G$  denotes for a graph,  $V$  for its vertex set and  $E$  for its edge set, we denote  $G = (V, E)$ . The set of subsets of  $V$  is denoted  $2^V$ . Throughout the paper,  $n$  stands for  $|V|$  and  $m$  for  $|E|$ . An edge between vertices  $x$  and  $y$  will be arbitrarily denoted  $xy$  or  $yx$ . The (open) neighborhood of  $x$  is denoted  $N(x)$  and the closed neighborhood  $N[x] = N(x) \cup \{x\}$ . For a rooted tree  $T$  and a vertex  $u \in T$ , we denote  $T(u)$  for the subtree of  $T$  rooted at  $u$ , and  $Anc_T(u)$  for the ancestors of  $u$  in  $T$  ( $u \in Anc_T(u)$ ). The depth of  $u$  in  $T$ , is the number of edges in the path from the root to  $u$  (the root has depth 0). The depth of  $T$ , denoted  $depth(T)$ , is the greatest depth of its leaves. The set of vertices at depth  $i$  in  $T$  will be denoted  $T^i$ . For a linear ordering  $\sigma$  on a set  $S$ , we denote  $min(\sigma)$  (resp.  $max(\sigma)$ ) for the first (resp. last) element of  $\sigma$ . For any  $s \in S$ , we denote  $\sigma(s)$  for the rank of  $s$  in  $\sigma$  ( $min(\sigma)$  has rank 1, and  $max(\sigma)$  rank  $|S|$ ), and for any  $i \in [1, |S|]$ , we denote  $\sigma^{-1}(i)$  for the element  $s \in S$  such that  $\sigma(s) = i$ . For  $s \in S$ ,  $s^-$  (resp.  $s^+$ ) denotes for the predecessor (resp. successor) of  $s$  in  $\sigma$ .  $\bar{\sigma}$  denotes for the reverse order of  $\sigma$ . The list  $L$  containing elements  $x_1, \dots, x_k$  is denoted  $L = [x_1, \dots, x_k]$ . For two lists  $L_1, L_2$  with  $L_1 = [x_1, \dots, x_k]$  and  $L_2 = [y_1, \dots, y_k]$ , we denote  $L_1.L_2$  for the concatenated list  $L_1.L_2 = [x_1, \dots, x_k, y_1, \dots, y_k]$ .

An interval model of a graph  $G$  is a set of intervals of the real line together with a one to one mapping onto the set of vertices of  $G$ , such that there is an edge between vertices  $x$  and  $y$  in  $G$  iff their corresponding intervals intersect (see Fig. 1(b)). An interval graph is a graph admitting such a model. The class remains the same if the intervals are required to be closed and to have integer endpoints between 1 and  $2n$ , all models considered in the following satisfy this restriction. Associating each vertex with the endpoints of its interval provides an efficient encoding of the graph that takes  $O(n)$  space and allows to answer adjacency queries between any pair of vertices in  $O(1)$  time.

A permutation model of a graph is a set of segments joining two given parallel lines along with a one to one mapping onto the set of vertices of  $G$ , such that there is an edge between vertices  $x$  and  $y$  in  $G$  iff their corresponding segments intersect (see Fig. [□\(c\)](#)). A graph  $G$  is a permutation graph iff it admits a permutation model. The class remains the same if the endpoints of the segments are required to be pairwise distinct. We denote  $\pi_1$  and  $\pi_2$  for the orders on the vertices induced by the order of their endpoints respectively on the first and second line. Vertices  $x$  and  $y$  are adjacent in  $G$  iff  $(\pi_1(y) - \pi_1(x)) \times (\pi_2(y) - \pi_2(x)) < 0$ ; that is iff  $x$  and  $y$  do not appear in the same relative order in the first and in the second linear ordering. Associating with each vertex  $x$  of  $G$  the couple  $(\pi_1(x), \pi_2(x))$  results in a  $O(n)$  representation of  $G$  providing adjacency in  $O(1)$  time. We will refer to  $\pi_1(x)$  and  $\pi_2(x)$  as the endpoints of  $x$ , identifying  $x$  and its corresponding segment.

### 3 Interval Graphs and Permutation Graphs Have Unbounded Closed Linearity

The aim of this section is to prove that interval graphs and permutation graphs have unbounded contiguity and linearity (both open and closed). Thanks to the relationships between the four parameters, we will derive the result from the case of closed linearity, for which we give a formal definition.

**Definition 1.** We call a closed  $p$ -line-model of a graph  $G = (V, E)$  a tuple  $(\sigma_1, \dots, \sigma_p)$  of linear orders on  $V$  such that  $\forall v \in V, \exists (I_1, \dots, I_p) \in (2^V)^p, (\forall i \in [1, p], I_i \text{ is an interval of } \sigma_i)$  and  $N[x] = \bigcup_{1 \leq i \leq p} I_i$ .

The closed linearity of  $G$ , denoted  $cl(G)$  is the minimum integer  $p$  such that there exists a closed  $p$ -line-model of  $G$ .

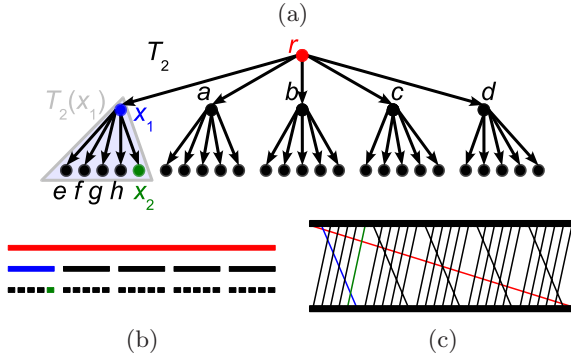
We now exhibit a family of graphs which shows that closed linearity is unbounded for interval and permutation graphs.

**Theorem 1.** For any  $k \in \mathbb{N}$ , there exists a graph  $G$  that is both an interval graph and a permutation graph, and such that  $cl(G) > k$ .

**Proof.** Consider the transitive closure of the rooted directed  $2k + 1$ -ary tree  $T_k$  of depth  $k$ , for  $k \geq 1$ . Let  $G_k$  be its underlying undirected graph.

We first prove that  $G_k$  is a permutation graph: for every internal node  $v$  of  $T_k$ , choose an arbitrary order  $\pi_v$  on its  $2k + 1$  children. Let  $\pi_1$  (resp.  $\pi_2$ ) be the order in which one discovers the nodes of  $T_k$  in a Depth-First Search that respects the orders  $\pi_v$  (resp.  $\overline{\pi}_v$ ) for all  $v \in T_k$ .  $(\pi_1, \overline{\pi}_2)$  is a permutation model of  $G_n$ , as illustrated for  $G_2$  in Fig. [□\(c\)](#). It is easy to see that  $G_k$  is also an interval graph: for any vertex  $v \in T$ , choose an interval included in its father's one and disjoint from its siblings' one, as in Fig. [□\(b\)](#).

We now prove that  $cl(G_k) > k$ . Assume for contradiction that  $G_k$  admits a closed  $k$ -line-model  $\mathcal{M}$ . We prove by recursion that for all  $i \in [0, k]$ , there is a vertex  $x_i$  at depth  $i$  in  $T_k$  such that for all  $u \in Anc_{T_k}(x_i) \setminus \{x_i\}$  and for all



**Fig. 1.** The interval model (b) and the permutation model (c) of the undirected underlying graph of the transitive closure of the rooted directed tree  $T_2$  (a). This graph has linearity strictly greater than 2, because two orders are not sufficient to represent the closed neighborhood  $\{r, x_1, x_2\}$  of  $x_2$ .

$y \in T_k(u)$ ,  $y$  is not next to  $u$  in any order of  $\mathcal{M}$ . This is trivially true for  $i = 0$  since the root is at depth 0 and has no strict ancestor. We now suppose this is true for some  $i \in [0, k - 1]$ , and prove it for  $i + 1$ . Since there are  $k$  orders in  $\mathcal{M}$  and since  $x_i$  has  $2k + 1$  children, there exists one child  $x_{i+1}$  of  $x_i$  such that for all  $y \in T_k(x_{i+1})$ ,  $y$  is not next to  $x_i$  in any order of  $\mathcal{M}$ . Then,  $x_{i+1}$  shows that the inductive hypothesis also holds for  $i + 1$ .

Consider the leaf  $x_k$  given by the statement above, proved by recursion. The closed neighborhood of  $x_k$  is exactly  $Anc_{T_k}(x_k)$ . Let  $\sigma$  be a linear order of  $\mathcal{M}$ . From the statement, no vertex of  $Anc_{T_k}(x_k)$  is next to another vertex of  $Anc_{T_k}(x_k)$  in  $\sigma$ . It follows that the interval associated to  $x_k$  in  $\sigma$  contains only one vertex. As there are  $k + 1$  vertices in the closed neighborhood of  $x_k$  and only  $k$  orders in  $\mathcal{M}$ , we get a contradiction. Thus,  $cl(G_k) > k$ .  $\square$

Since the graphs  $G_k$  used in the proof of Theorem  $\square$  have  $(2k + 1)^k$  vertices and since  $cl(G_k) > k$ , it follows that  $cl(G_k) = \Omega(\log n / \log \log n)$ . This lower bound also holds for open linearity, open contiguity and closed contiguity, which are respectively denoted  $ol(G)$ ,  $oc(G)$  and  $cc(G)$ . Indeed, one can obtain a closed  $p + 1$ -line-model from an open  $p$ -line-model by adding a linear order in which the interval associated to a vertex is reduced to the vertex itself. It follows that  $ol(G) \geq cl(G) - 1$ . In addition, if  $G$  has contiguity  $p$ , make  $p$  copies of a linear ordering  $\pi$  realizing this  $p$ , and, in each of the  $p$  copies, associate to all the vertices of  $G$  one of their at most  $p$  intervals in  $\pi$ . This results in a  $k$ -line-model of  $G$  and shows that  $cc(G) \geq cl(G)$  and  $oc(G) \geq ol(G) \geq cl(G) - 1$ . Thus, the four parameters have value  $\Omega(\log n / \log \log n)$  for  $G_n$ . Let us mention that there are examples showing that the contiguity (open and closed) can even be up to  $\Omega(\log n)$  for both interval and permutation graphs.

## 4 Encoding Neighborhoods in Interval Graphs and Permutation Graphs

Here, we aim at providing  $O(n)$  space representations of interval graphs and permutation graphs that allow to answer neighborhood queries on any vertex  $x$  in  $O(d)$  time, where  $d$  is the degree of  $x$ . Moreover, the structures we propose can be constructed in  $O(n)$  time from the interval or permutation model of the graph. We first show that, for interval graphs, the encoding problem we consider can be reduced to the same problem on permutation graphs. Then, we describe an encoding of permutation graphs satisfying the desired properties.

**Interval Graphs.** The neighborhood of a vertex  $x$  of an interval graph can be divided into three (not necessarily disjoint) parts: the subset  $L(x)$  of vertices whose interval left endpoint lies in the interval of  $x$ , the subset  $R(x)$  of vertices whose interval right endpoint lies in the interval of  $x$ , and the vertices whose interval either contains or is included in the one of  $x$ . Let  $\pi_1$  (resp.  $\pi_2$ ) be the order on the vertices of  $G$  obtained by sorting them with increasing interval left (resp. right) endpoints, breaking the ties in an arbitrary way. It is not difficult to see that  $L(x)$  is an interval  $I_1$  of  $\pi_1$  and  $R(x)$  is an interval  $I_2$  of  $\pi_2$ . Concerning the last part of the neighborhood, it is known that the containment relationship of intervals (i.e.  $I$  is in relation with  $J$  iff  $I \subseteq J$  or  $J \subseteq I$ ) is a permutation graph, denoted  $G'$ , whose permutation model is  $(\pi_1, \pi_2)$ . Therefore, a neighborhood query on  $x$  in  $G$  will be treated by augmenting the result of the query in  $G'$  with the vertices of  $I_1 \cup I_2$ .

Note that this encoding of interval graphs usually contains redundant information. During the query on  $x$ , a vertex whose interval is included in that of  $x$  will appear in the result of the query in  $G'$ , as well as in  $I_1$  and in  $I_2$ . This drawback can be avoided by simply parsing the output list to remove repetitions. Anyway, this redundancy only introduces a constant multiplicative factor and we still achieve expected time and space complexity, provided that there exists a structure achieving it for permutation graphs. We now concentrate on building such a structure.

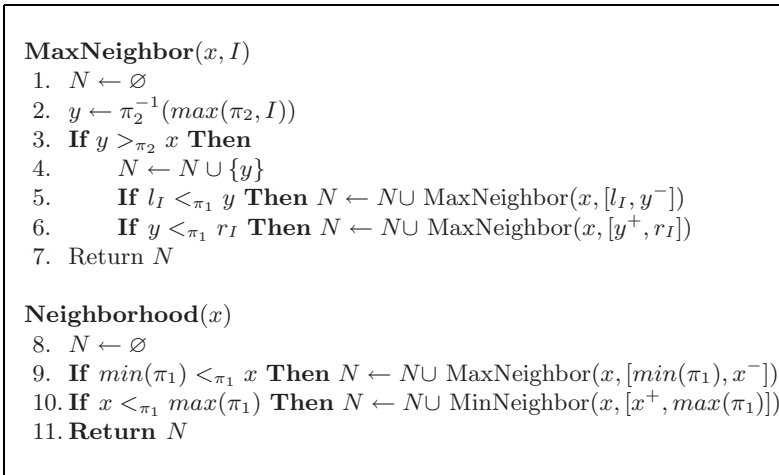
**Permutation Graphs.** The difficulty of encoding neighborhoods in permutation graphs comes from the fact that the neighborhood of a vertex can be spread everywhere in the two orders of the permutation model. Then, scanning the orders to collect the neighborhood of  $x$  may take up to  $O(n)$  time. However, we show that it is possible to extract the neighborhood of  $x$  from the permutation model, without scanning it, in  $O(d)$  time, where  $d$  is the degree of  $x$ . This can be achieved thanks to *augmented Cartesian trees*, introduced in [4], which is based on *Cartesian trees* [13] and constant time nearest common ancestor queries [8,9]. This structure provides, in constant time, the maximum, on any given interval, of an integer function  $f$  defined on a linear order; and it can be computed in  $O(n)$  time for a linear order on  $n$  elements. Notice that changing  $f$  for its opposite provides the same features for minimum queries. We denote  $\max(f, I)$  (resp.  $\min(f, I)$ ) for the maximum (resp. minimum) of function  $f$  over interval  $I$ .

We now detail the algorithm listing the neighborhood of a vertex (cf. Fig 2). We assume that the model  $(\pi_1, \pi_2)$  of the permutation graph  $G$  is given, as well as the augmented Cartesian tree of the function  $\pi_2$  on the linear order  $\pi_1$ , which provides the maximum of  $\pi_2$  on any interval of  $\pi_1$ . The prerequisites of Routine **MaxNeighbor** $(x, I)$  are: 1)  $x$  is a vertex of  $G$  and 2)  $I = [l_I, r_I]$  is a non-empty interval of  $\pi_1$  such that  $r_I <_{\pi_1} x$ . Lemma 1 below states that, when they are satisfied, the routine returns all the neighbors  $y$  of  $x$  that belong to  $I$ , and runs in  $O(d)$  time.

**Lemma 1.** *When its prerequisites are satisfied, Routine **MaxNeighbor** returns the neighbors of  $x$  belonging to the interval  $I$  of  $\pi_1$ , in  $O(d)$  time.*

**Proof.** The prerequisite 2) guarantees that  $r_I <_{\pi_1} x$ . Then, a vertex  $z \in I$  is adjacent to  $x$  iff  $z >_{\pi_2} x$ . Since the test of Line 3 occurs on the vertex  $y \in I$  having the greatest value for  $\pi_2(y)$ , it exactly determines whether  $x$  is adjacent to some vertex of  $I$ ; and in the positive,  $y$  is added to  $N$ . Then, the search recursively goes on in the two pieces made by the removal of  $x$  in  $I$ . Since these two pieces still satisfy the prerequisites and since the search do not forget any part of  $I$ , it discovers all the neighbors of  $x$  belonging to  $I$ .

Thanks to the augmented Cartesian tree, the maximum query of Line 2 takes constant time, and so does each recursive call to **MaxNeighbor**. Then the complexity of the routine is its number of recursive calls. Each vertex  $y \in N(x) \cap I$  discovered during a recursive call, then called a *discovering call*, has not been discovered before, since the neighbors discovered are excluded from the search. Then the number of discovering calls is  $O(d)$ . When a recursive call does not discover any neighbor of  $x$ , it does not call Routine **MaxNeighbor**. It follows that all calls are made by a discovering call, called its *parent call*. Since Routine



**Fig. 2.** Routines **MaxNeighbor** and **Neighborhood**.  $x$  is a vertex of  $G$  and  $I = [l_I, r_I]$  is an interval of  $\pi_1$ .



```

BuildTree( $\pi_1, \pi_2, \sigma$ )
1.  $x \leftarrow \min(\sigma)$ ;  $\text{ord}(x) \leftarrow 1$ 
2. initialize  $I$  and  $\Gamma$  with  $(x, x, x, x)$ 
3. For all  $y \in N(x)$  Do
4.    $\text{parent}(y) \leftarrow x$ ; color  $y$  in gray;  $\text{update}(\Gamma, y)$ 
5. While  $I \neq \Gamma$  Do
6.    $\Gamma_{\text{new}} \leftarrow \Gamma$ ;  $\text{Exam} \leftarrow \emptyset$ 
7.   If  $\alpha_1 < a_1$  Then PutInTree( $\alpha_1, a_1^-, \pi_1, Q_1^l, \text{Exam}$ )
8.   If  $\beta_1 > b_1$  Then PutInTree( $\beta_1, b_1^+, \pi_1, Q_1^r, \text{Exam}$ )
9.   If  $\alpha_2 < a_2$  Then PutInTree( $\alpha_2, a_2^-, \pi_2, Q_2^l, \text{Exam}$ )
10.  If  $\beta_2 > b_2$  Then PutInTree( $\beta_2, b_2^+, \pi_2, Q_2^r, \text{Exam}$ )
11.   $\text{AssignOrd}(Q_1^l, Q_1^r, Q_2^l, Q_2^r)$ ; color  $\text{Exam}$  in gray
12.   $I \leftarrow \Gamma$ ;  $\Gamma \leftarrow \Gamma_{\text{new}}$ 

```

**Fig. 3.** Routine **BuildTree**. Lists  $Q_1^l$ ,  $Q_1^r$ ,  $Q_2^l$ ,  $Q_2^r$  and  $\text{Exam}$  are local variables of the main loop, as well as  $\Gamma_{\text{new}}$  which is a quadruplet of integers.  $I$  and  $\Gamma$  are global variables.

**MaxNeighbor** contains at most two recursive calls during an execution, a discovering call is the parent of at most two non-discovering calls. Thus, the number of non-discovering calls is also  $O(d)$ , and so is the total running time of Routine **MaxNeighbor**.  $\square$

Similarly to **MaxNeighbor**, we can design a Routine **MinNeighbor** that discovers all the neighbors of  $x$  belonging to an interval  $I$  of  $\pi_1$  lying entirely to the right of  $x$ : simply replace **max** with **min** at Line 2 and reverse the inequality of Line 3. Then, from Lemma 1, it is clear that Routine **Neighborhood** discovers all the neighbors of  $x$  in  $O(d)$  time, as stated by Theorem 2.

**Theorem 2.** *Routine **Neighborhood** returns the neighborhood of  $x$  in  $O(d)$  time.*

## 5 Breadth-First Search of Permutation Graphs

The input of our algorithm is a permutation graph  $G$  given by its model  $(\pi_1, \pi_2)$  and a linear order  $\sigma$  on  $V$ . It computes the BFS tree  $T_\sigma$  resulting from the BFS of  $G$  with priority order  $\sigma$ , that is, the BFS starting with vertex  $\min(\sigma)$  and where the neighbors  $y$  of a vertex are examined with increasing  $\sigma(y)$ . Our algorithm does not discover the vertices in the same order as the standard BFS would do. Nevertheless, the tree  $T$  produced is the same, except that the children of a vertex of  $T$  are not ordered; which we fix, in a final step, by sorting all the children lists according to  $\sigma$ . The total complexity of the process is  $O(n)$ .

**The Algorithm.** Routine **BuildTree** (cf. Fig. 3) computes the BFS tree of the connected component of the first visited vertex. It can be applied iteratively on the first non-visited vertex in  $\sigma$  in order to get the complete BFS

forest. From now on, we concentrate on building a single tree  $T$ . In the description of the algorithm,  $parent(y)$  denotes for the parent of vertex  $y$  in  $T$ . Before the algorithm starts,  $parent(y)$  is initialized with  $\perp$ , and  $y$  is colored in white.  $\Gamma = (\alpha_1, \beta_1, \alpha_2, \beta_2)$  and  $I = (a_1, b_1, a_2, b_2)$  are quadruplets of vertices. We denote  $I_1$  for the interval  $[a_1, b_1]$  of  $\pi_1$  and  $I_2$  for  $[a_2, b_2]$  in  $\pi_2$ . Similarly,  $\Gamma_1 = [\alpha_1, \beta_1]$  in  $\pi_1$ , and  $\Gamma_2 = [\alpha_2, \beta_2]$  in  $\pi_2$ . Procedure  $update(X, y)$ , where  $y \in V$  and  $X = (l_1, r_1, l_2, r_2)$  is a quadruplet of vertices, executes the instructions  $l_j \leftarrow \min_{\pi_j}(l_j, y)$  and  $r_j \leftarrow \max_{\pi_j}(r_j, y)$ , for  $j \in \{1, 2\}$ . Function  $ord$  takes integer values assigned by Procedure  $AssignOrd$ . We denote  $<_{ord}$  the order defined by  $u <_{ord} v$  iff  $ord(u) < ord(v)$ . The arguments of  $AssignOrd$  are four lists (not necessarily disjoint), which are merged by the procedure into a single one  $Q_{glob}$  without repetition and sorted according to order  $<_{prior}$  (defined below), the first element of the list being the least one for  $<_{prior}$ . Then, for each vertex  $x$  in  $Q_{glob}$ , the procedure assigns  $ord(x)$  with the rank of  $x$  in  $Q_{glob}$ . For any  $i \geq 0$ , we denote  $\mathcal{O}^i$  for  $\{y \in T^i \mid ord(y) \text{ is defined}\}$ . Order  $<_{prior}$  is defined by  $u <_{prior} v$  iff  $ord(parent(u)) < ord(parent(v))$  or  $(ord(parent(u)) = ord(parent(v)) \text{ and } \sigma(u) < \sigma(v))$ . By convention, element  $\perp$  is the greatest for order  $<_{prior}$ .

Routine **BuildTree** builds  $T$  level by level from the root to the leaves. The  $i^{th}$  iteration of the main loop (starting at Line 5) builds  $T^{i+1}$  by parsing the vertices of  $(\Gamma_1 \setminus I_1) \cup (\Gamma_2 \setminus I_2)$ , thanks to the four calls to Routine **PutInTree** (cf. Fig. 4). This routine affects to the encountered white vertices their parent in  $T$  (Line 4). A vertex is colored gray (Line 11 of **BuildTree**) only when it has been placed correctly in  $T$ ; note that a vertex may be assigned a parent twice, once in  $\pi_1$  and once in  $\pi_2$ . The main loop stops when  $I = \Gamma$  (Line 5), that is, when the two intervals of  $\pi_1$  and  $\pi_2$  corresponding to the connected component  $C$  of  $min(\sigma)$  have been entirely parsed. Then, all the vertices have been assigned a parent, and the construction of  $T$  is over.

**Correctness.** It is straightforward that during all the algorithm,  $I_1 \subseteq \Gamma_1$  and  $I_2 \subseteq \Gamma_2$ . The key of the correctness of our algorithm is the following invariants. At the beginning of the  $i^{th}$  iteration of the main loop of **BuildTree** (Line 5),  $T$  has been built correctly until depth  $i$ , the set of gray vertices is exactly  $\bigcup_{0 \leq j \leq i} T^j$ , and the properties below hold.

1. The vertices of  $\bigcup_{0 \leq j < i} T^j$  have their two endpoints in  $I_1$  and  $I_2$ ;
2. The vertices of  $T^i$  have their two endpoints in  $\Gamma_1$  and  $\Gamma_2$ ;
3.  $I_1 \cup I_2$  contains all the vertices of  $\bigcup_{0 \leq j \leq i} T^j$  and no others;
4. The bounds of  $\Gamma_1$  and  $\Gamma_2$  belong to  $\bigcup_{0 \leq j \leq i} T^j$ .
5. The order  $<_{ord}$  restricted to  $\mathcal{O}^{i-1}$  is exactly the order of visit of the vertices of  $\mathcal{O}^{i-1}$  by the standard BFS algorithm.

During the main loop, the four calls to **PutInTree** parse the vertices of  $B = (\Gamma_1 \setminus I_1) \cup (\Gamma_2 \setminus I_2)$ . The white vertices of  $B$  are exactly the vertices of  $T^{i+1}$  and the gray ones are in  $T^i$ . **PutInTree** affects to the white vertices of  $B$  their parent in  $T$ , which is among the gray vertices of  $B$ . The vertices of  $T^i$  that have their two endpoints in  $I_1$  and  $I_2$  are leaves of  $T$ . Let us examine more precisely the

**PutInTree**( $u, v, \pi, Q, F$ )

1.  $p \leftarrow u; Q \leftarrow [p]$
2. **For**  $y$  from  $u$  to  $v$  in  $\pi$  **Do**
3.     **If**  $y$  is white and  $p <_{prior} \text{parent}(y)$  **Then**
4.          $\text{parent}(y) \leftarrow p; F \leftarrow F.[y]; \text{update}(\Gamma_{new}, y)$
5.     **If**  $y$  is gray and  $y <_{prior} p$  **Then**
6.          $p \leftarrow y; Q \leftarrow [p].Q$

**Fig. 4.** Routine **PutInTree**.  $\pi$  is an order,  $u$  and  $v$  two vertices, and  $Q$  and  $F$  are lists.  $\Gamma_{new}$  is a variable of Routine **BuildTree**.

first call to **PutInTree**, when  $\alpha_1 < a_1$ . From Invariant [1](#) and [4](#) (Inv. for short),  $\alpha_1 \in T^i$ . Let  $w$  be a white vertex of  $[\alpha_1, a_1^-]$ . Since all the vertices of  $I_1 \cup I_2$  are gray (Inv. [3](#)) and since  $w$  is not adjacent to  $\min(\sigma)$ , necessarily  $w <_{\pi_2} a_2$ . Moreover, since  $\alpha_1 \in T^i$ ,  $\alpha_1$  has an endpoint in  $I_2$  (Inv. [3](#)). It follows that  $w$  and  $\alpha_1$  are adjacent and  $w \in T^{i+1}$ . Similarly,  $w$  is adjacent to all the gray vertices  $z <_{\pi_1} w$  and to none of the gray vertices  $z >_{\pi_1} w$ . From Inv. [5](#) and the definition of  $<_{prior}$ , we have that, for  $u, v \in T^i$ ,  $u <_{prior} v$  iff  $u$  is visited before  $v$  by the standard BFS. Then, the test at Line 5 together with the affectation at Line 6 maintain variable  $p$  as the gray node of  $[\alpha_1, y]$  which is the first visited by the standard BFS, where  $y$  is the current vertex in the loop starting at Line 2. Thus, any white vertex  $w \in [\alpha_1, a_1^-]$  is assigned a parent which is the gray vertex of  $[\alpha_1, a_1^-] \cap N(w)$  being the first one visited by the standard BFS (remind that  $[w, a_1^-] \cap N(w) = \emptyset$ ). More generally, this is true for any white node discovered in any of the four calls to **PutInTree**. Note that a white vertex  $w$  may be seen in two different calls to **PutInTree**, once in  $\pi_1$  and once in  $\pi_2$ . In this case, the test at Line 3 guarantees that the second possible parent is affected if and only if it is visited by the standard BFS before the first one. Thus, during the main loop, any white vertex  $w$  of  $B$  is affected its correct parent in  $T$ . We showed that  $w \in T^{i+1}$ ; note that conversely, every vertex  $w \in T^{i+1}$  is white and has necessarily an endpoint in  $B$ : otherwise, it would either be adjacent to none of the vertices of  $\bigcup_{0 \leq j \leq i} T^j$ , or adjacent to  $\min(\sigma)$ ; both contradict the fact that  $w \in T^{i+1}$ . As a conclusion, the  $i^{th}$  iteration of the main loop properly computes level  $T^{i+1}$  and colors its vertices in gray (Line 11).

The fact that  $\Gamma_{new}$  is updated, at Line 4 of **PutInTree**, every time a white vertex (which will become gray at Line 11) is visited, together with the two affectations at Line 12 of **BuildTree**, imply that Inv. [1](#) to [4](#) are still true at the beginning of the next iteration. Finally, since order  $<_{prior}$  on set  $T^i$  is the order of visit of the vertices of  $T^i$  by the standard BFS, and since **AssignOrd** (Line 11) assigns the values of  $ord(v)$  to the vertices  $v \in T^i$  according to  $<_{prior}$ , it follows that Inv. [5](#) is also maintained during the loop.

**Complexity.** The total running time of our BFS algorithm is  $O(n)$ . First, we build the structure of Section 4 that allows to answer neighborhood queries in  $O(d)$  time. It takes  $O(n)$  time. Then, Routine `BuildTree` gives the BFS tree (except the order on the children) of the connected component  $C$  of the first vertex examined, in  $O(|C|)$  time. We repeat Routine `BuildTree`, starting from the first non-visited vertex in  $\sigma$ , until all the vertices of the graph have been visited. As the sets of vertices visited during each call to `BuildTree` are disjoint, the total running time of all the calls to `BuildTree` is  $O(n)$ . At last, we order the lists of children of the vertices in all the trees produced: set the lists of children of all the vertices in the forest to  $\emptyset$ , and scan  $\sigma$  placing each vertex in the list of its parent. At the end of the scan, all the children lists have been rebuilt and sorted according to  $\sigma$ . This process takes  $O(n)$  time.

Let us detail the  $O(|C|)$  time complexity of Routine `BuildTree`. Remind that  $B$  denotes for  $(I_1 \setminus I_1) \cup (I_2 \setminus I_2)$ . Thanks to the data structure of Section 4, at Line 3, we can get  $N(x)$  in  $O(|N(x)|)$  time, which is also the running time of the initialization loop. In `PutInTree`, all instructions take  $O(1)$  time, and the Routine runs in  $O(|\pi(u) - \pi(v)|)$  time. That is,  $O(|B|)$  time for the four calls of the main loop. Coloring *Exam* also takes  $O(|B|)$  time. The complexity of procedure *AssignOrd* is a crucial point. It is worth to note that any list  $Q'$  being one of its arguments is already sorted according to  $<_{prior}$ . This is a property of `PutInTree`, which produces  $Q'$ , guaranteed by the test  $y <_{prior} p$  at Line 5 and the affectations of Line 6. It follows that *AssignOrd* can be implemented to merge the four lists in a single one, sorted according to  $<_{prior}$ , in  $O(|Q'_1| + |Q'_1| + |Q'_2| + |Q'_2|) = O(|B|)$  time. Then, the running time of an iteration of the loop is  $O(|B|)$ , and since all the  $B$ 's considered until the end of the loop are pairwise disjoint, it follows that the main loop, as well as Routine `BuildTree`, runs in  $O(|C|)$  time.

## References

1. Boldi, P., Vigna, S.: The webgraph framework I: compression techniques. In: WWW 2004, pp. 595–602. ACM, New York (2004)
2. Boldi, P., Vigna, S.: Codes for the world wide web. *Internet Mathematics* 2(4), 407–429 (2005)
3. Brandstädt, A., Le, V.B., Spinrad, J.P.: *Graph Classes: a Survey*. SIAM Monographs on Discrete Mathematics and Applications (1999)
4. Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for geometry problems. In: STOC 1984, pp. 135–143 (1984)
5. Gavoille, C., Peleg, D.: The compactness of interval routing. *SIAM Journal on Discrete Mathematics* 12(4), 459–473 (1999)
6. Goldberg, P.W., Golumbic, M.C., Kaplan, H., Shamir, R.: Four strikes against physical mapping of DNA. *Journal of Computational Biology* 2(1), 139–152 (1995)
7. Golumbic, M.C.: *Algorithmic Graph Theory and Perfect Graphs*. *Annals of Discrete Mathematics*, 2nd edn., vol. 57. Elsevier, Amsterdam (2004)
8. Harel, D.: A linear time algorithm for the lowest common ancestors problem (extended abstract). In: FOCS 1980, pp. 308–319 (1980)

9. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.* 13(2), 338–355 (1984)
10. Roberts, F.S.: Representations of Indifference Relations. PhD thesis, Stanford University (1968)
11. Spinrad, J.P.: Efficient graph representations. *Fields Institute Monographs*, vol. 19. American Mathematical Society (2003)
12. Turan, G.: On the succinct representation of graphs. *Discr. Appl. Math.* 8, 289–294 (1984)
13. Vuillemin, J.: A unifying look at data structures. *Commun. ACM* 23(4), 229–239 (1980)
14. Wang, R., Lau, F.C.M., Zhao, Y.: Hamiltonicity of regular graphs and blocks of consecutive ones in symmetric matrices. *Discr. Appl. Math.* 155(17), 2312–2320 (2007)

# LPF Computation Revisited\*

Maxime Crochemore<sup>1,4</sup>, Lucian Ilie<sup>2,\*\*</sup>, Costas S. Iliopoulos<sup>1,5</sup>,  
Marcin Kubica<sup>3</sup>, Wojciech Rytter<sup>3,6,\*\*\*</sup>, and Tomasz Walen<sup>3</sup>

<sup>1</sup> Dept. of Computer Science, King's College London, London WC2R 2LS, UK

<sup>2</sup> Dept. of Computer Science, University of Western Ontario, London,  
Ontario, N6A 5B7, Canada

<sup>3</sup> Institute of Informatics, Warsaw University, ul. Banacha 2,  
02-097 Warszawa, Poland

<sup>4</sup> Université Paris-Est, France

<sup>5</sup> Digital Ecosystems & Business Intelligence Institute, Curtin University of  
Technology, Perth WA 6845, Australia

<sup>6</sup> Dept. of Math. and Informatics, Copernicus University, Torun, Poland

**Abstract.** We present efficient algorithms for storing past segments of a text. They are computed using two previously computed read-only arrays (SUF and LCP) composing the Suffix Array of the text. They compute the maximal length of the previous factor (subword) occurring at each position of the text in a table called LPF. This notion is central both in many conservative text compression techniques and in the most efficient algorithms for detecting motifs and repetitions occurring in a text.

The main results are: a linear-time algorithm that computes explicitly the permutation that transforms the LCP table into the LPF table; a time-space optimal computation of the LPF table; and an  $O(n \log n)$  strong in-place computation of the LPF table.

**Keywords:** longest previous factor, suffix array, Ziv-Lempel factorisation, text compression, detection of repetitions.

## 1 Longest Previous Factor

We consider a string  $y$  of length  $n$  on the alphabet  $A$ :  $y = y[0..n-1]$ . The problem is to compute the Longest Previous Factor table defined, for  $0 \leq i < n$ , by

$$\text{LPF}[i] = \max\{k \mid y[i..i+k-1] \text{ occurs at a position } j < i\}.$$

For example, the text  $y = \text{abaabababbabb}$  has the following LPF table.

position $i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$y[i]$	a	b	a	a	b	a	b	a	b	b	a	b	b	b
LPF $[i]$	0	0	1	3	2	4	3	2	1	4	3	2	2	1

\* Research supported in part by the Royal Society, UK.

\*\* Research supported in part by NSERC.

\*\*\* Supported by grant N206 004 32/0806 of the Polish Ministry of Science and Higher Education.

The problem may be regarded as an extension of the Ziv-Lempel factorisation (LZ77) of a string as defined in [20]. A string  $y$  is decomposed into factors, called phrases,  $u_0, u_1, \dots, u_k$ , for which  $y = u_0u_1 \cdots u_k$  and defined informally by:  $u_i = va$  where  $a$  is a letter and  $v$  is the longest segment of  $u_0u_1 \cdots u_i$  occurring both at position  $|u_0u_1 \cdots u_{i-1}|$  and before it in  $y$ . The factorisation is used in several adaptive compression methods which encode carefully re-occurrences of phrases by pointers or integers (see [18] or [19]). The factorisation yields more powerful compressors than the factorisation in [21] (called LZ78) where a phrase is an extension by a single letter of a previous phrase. But the LZ77 factorisation is more difficult to compute.

It is clear that the LZ77 factorisation comes readily when the LPF table is available, which is a remarkable application of the table (see [5]). This implies a linear-time solution for LZ77 factorisation on integer alphabets. Previous solutions using a Suffix Tree [17] or a Suffix Automaton [2] of the text not only run in time  $O(|y| \log |A|)$  but these data structures are more space-expensive than the Suffix Array.

A slight variant of string parsing, whose relation with the LZ77 factorisation is analysed in [1], plays an important role in String Algorithms. The intuitive reason is that, when processing a string on-line, the work done on an element of the factorisation can usually be skipped because already done on one of its previous occurrences. A typical application of this idea resides in algorithms to compute repetitions in strings (see [2][15][14]). For example, the algorithm in [14] reports all maximal repetitions (called runs) occurring in a string in  $O(|y| \log |A|)$  time. It runs in linear time if a Suffix Array is used instead of a Suffix Tree [4]. Indeed the technique seems to be the only technique that leads to linear-time algorithms independently of the alphabet size for this type of question.

Suffix Arrays provide an ideal data structure to solve many questions requiring an index on all the factors of a string. Introduced by Manber and Myers [16] the structure can be built in linear-time by different methods [10][12][13][8] for sorting the suffixes of the text possibly adding the method of [11] to compute the Longest Common Prefix table. The result holds if the text is drawn from an integer alphabet, that is, if the alphabet of the text can be sorted in linear time (otherwise the  $\Omega(n \log n)$  lower bound for sorting applies).

The notion of Longest Previous Factor has been introduced by Franek et al. as part of their concept of a Quasi Suffix Array (their  $\pi$  array is the LPF table) in [7], where the authors presented a direct computation running in  $O(n \log n)$  average time. A naive computation of the LPF table, either on the text itself or on its Suffix Array, as done by the algorithm LPF-NAIVE in Section [5], leads to quadratic effective running time on many inputs.

In this article we intensively use the Suffix Array of the text to be processed, and we consider only linear-time solutions. A graphical representation of the Suffix Array structure helps understand the design of the algorithms. The Longest Previous Factor is coined in [4], where a linear-time computation is described and applied to LZ77 factorisation. Another version running on-line on the Suffix Array of the text and requiring only  $O(\sqrt{n})$  extra memory space is shown in [5].

We improve on the previous results and show that the computation of the Longest Previous Factor table of a text from its Suffix Array can be implemented to run in linear time with only a constant amount of additional memory space. Thus, the method is time-space optimal.

The next section introduces the necessary material for the design of Longest Previous Factor computations. First, an algorithm similar to the one in [4] is described. The algorithm is regarded in Section 3 as using the Suffix Array like a sorting network. Section 4 shows how the computation can be done on-line on the Suffix Array and Section 5 describes the time-space optimal algorithm for constructing the table.

## 2 Using a Suffix Array

The Suffix Array of the text  $y$  is a data structure used for indexing its content. It comprises two tables that we denote **SUF** and **LCP** and that are defined as follows.

The table **SUF** stores the list of position on  $y$  associated with the sorted list of its suffixes in increasing lexicographic order. That is, the table is such that

$$y[\text{SUF}[0]..n-1] < y[\text{SUF}[1]..n-1] < \dots < y[\text{SUF}[n-1]..n-1].$$

Thus, indices on **SUF** are ranks of the suffixes in their sorted list.

The second table **LCP** is also indexed by the ranks of suffixes and stores the longest common prefixes between consecutive suffixes in the sorted list. Let  $\text{lcp}(i, j) =$  longest common prefix of  $y[i..n-1]$  and  $y[j..n-1]$ , for two positions  $i$  and  $j$  on  $y$ . Then,  $\text{LCP}[0] = 0$  and, for  $0 < r < n$ , we set

$$\text{LCP}[r] = |\text{lcp}(\text{SUF}[r-1], \text{SUF}[r])|.$$

(The actual Suffix Array contains indeed about  $n$  additional **LCP** values used for binary searching the suffixes.)

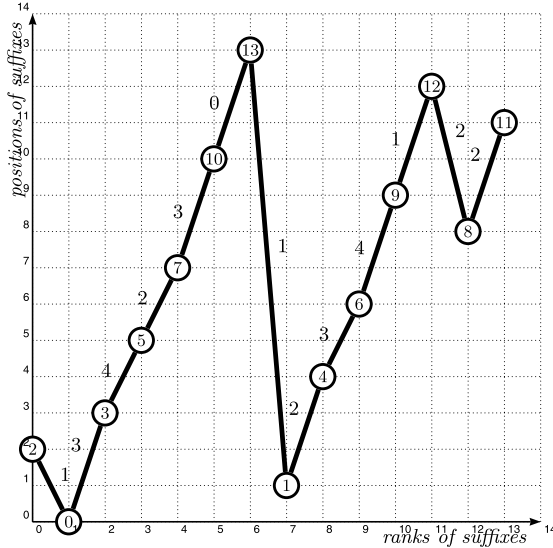
*Example 1.* For the text  $y = \text{abaabababbabbb}$  we get the Suffix Array:

rank $r$	0	1	2	3	4	5	6	7	8	9	10	11	12	13
<b>SUF</b> [ $r$ ]	2	0	3	5	7	10	13	1	4	6	9	12	8	11
<b>LCP</b> [ $r$ ]	0	1	3	4	2	3	0	1	2	3	4	1	2	2

The computation of the Suffix Array of  $y$  can be done in time  $O(n \log n)$  in the comparison model [16] (see [3,6,9]). If the text is on an alphabet of integers in the range  $[0, n^c]$  for some constant  $c$ , the Suffix Array can be built in time  $O(n)$  [10,12,13,8] (see also [3]).

*Graphic representation.* The Suffix Array of the text  $y$  has a nice graphic representation that helps understand the algorithms computing the **LPF** table. The abscissae axis refers to ranks of suffixes and the ordinates axis refers to their positions. The sorted list of suffixes is plotted by their positions, and consecutive positions are linked by an edge whose label is the associated **LCP** value. Figure 1 shows the Suffix Array representation for the text of Example 1.





**Fig. 1.** Graph of the permutation of positions on `abaabababbabbb` (Example 1) in the lexicographic order of its suffixes. Labels of edges are Longest Common Prefix lengths between consecutive suffixes.

*Simple algorithm.* The LCP table satisfies a simple property consequence of the lexicographic order: the LCP value between two positions at ranks  $r$  and  $t$ ,  $r < t$ , is the minimal value in  $\text{LCP}[r + 1..t]$ .

For a rank  $r$ , let us define  $\text{prev}[r]$  as the largest rank  $s$ ,  $s < r$ , for which  $\text{SUF}[s] < \text{SUF}[r]$  if it exists, and as  $-1$  otherwise. Let us also define the dual notion  $\text{next}[r]$  as the smallest rank  $t$ ,  $r < t$ , for which  $\text{SUF}[t] < \text{SUF}[r]$  if it exists, and as  $n$  otherwise. The above property implies that to compute  $\text{LPF}[\text{SUF}[r]]$  there is no need to look at ranks smaller than  $\text{prev}[r]$  or greater than  $\text{next}[r]$ , that is

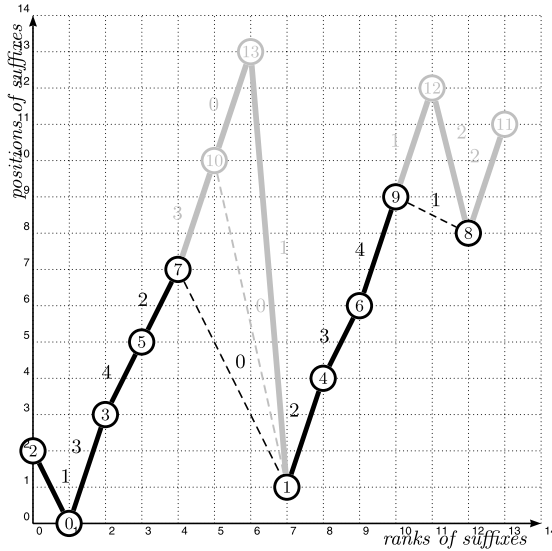
$$\text{LPF}[\text{SUF}[r]] = \max\{|\text{lcp}(\text{SUF}[r], \text{SUF}[\text{prev}[r]])|, |\text{lcp}(\text{SUF}[r], \text{SUF}[\text{next}[r]])|\} \quad (1)$$

where undefined LCP values are set to 0. In particular, if a position  $i$  is a “peak” at rank  $r$  in the graphic representation of the Suffix Array ( $\text{SUF}[r] = i$ ) we get

$$\text{LPF}[i] = \max\{\text{LCP}[r], \text{LCP}[r + 1]\}.$$

And the minimum of the two values is the LCP between positions at ranks  $r - 1$  and  $r + 1$  if defined. This gives the idea underlying the next algorithm.

Figure 2 illustrates a step of the algorithm when the input is the text of Example 1. The algorithm `LPF-SIMPLE` computes the two tables `prev` and `next`, which implement indeed a double-linked list of ranks. The algorithm runs in linear time but requires several extra arrays in addition to its input and output.



**Fig. 2.** Illustration of the run of the algorithm LPF-SIMPLE on the text `abaabababbabb` (see Figure 1) after processing positions 13, 12, 11, 10. Gray positions and gray edges are no longer considered. The next step is to process position 9 at rank 10. According to the weights of edges pending from 9, we have  $\text{LPF}[9] = \max\{4, 1\} = 4$ . Positions 6 and 8 will be linked, through the prev and next arrays on their ranks, with an edge of weight  $\min\{4, 1\} = 1$ , which is indeed  $\text{LPF}[8]$ .

Namely, the array `ISU`, inverse of `SUF`, that provides the rank of a position, and the two arrays `prev` and `next`.

`LPF-SIMPLE(SUF, LCP, n)`

```

1  LCP-copy ← LCP
2  LCP-copy[n] ← 0
3  ISU ← inverse SUF
4  for r ← 0 to n - 1 do
5      prev[r] ← r - 1
6      next[r] ← r + 1
7  for i ← n - 1 downto 0 do
8      r ← ISU[i]
9      LPF[i] ← max{LCP-copy[r], LCP-copy[next[r]]}
10     LCP-copy[next[r]] ← min{LCP-copy[r], LCP-copy[next[r]]}
11     if prev[r] ≥ 0 then
12         next[prev[r]] ← next[r]
13     if next[r] < n then
14         prev[next[r]] ← prev[r]
15  return LPF

```

**Proposition 1.** *The algorithm LPF-SIMPLE computes the LPF array of a text of length  $n$  from its Suffix Array in linear time and space. It requires  $3n + c$  integer cells in addition to its input and output.*

The array `next` can alternatively be pre-computed by the following procedure whose linear-time behaviour is an interesting exercise left to the reader. The procedure also computes the array  $\text{LPF}_>$  defined in [4] and that accounts for larger ranks only. It is defined, for  $r = 0, \dots, n - 1$ , by

$$\text{LPF}_>[r] = |\text{lcp}(\text{SUF}[r], \text{SUF}[\text{next}[r]])|.$$

`NEXT(SUF, LCP, n)`

```

1  next[n - 1] ← n
2  LPF_>[n - 1] ← 0
3  for r ← n - 2 downto 0 do
4      t ← r + 1
5      ℓ ← LCP[t]
6      while t < n and SUF[t] > SUF[r] do
7          ℓ ← min{ℓ, LPF_>[t]}
8          t ← next[t]
9      next[r] ← t
10     LPF_>[r] ← ℓ
11  return next, LPF_>
```

The computation of the dual `prev` and  $\text{LPF}_<$  arrays is done symmetrically. When both arrays  $\text{LPF}_<$  and  $\text{LPF}_>$  are available, the computation of the LPF table is an application of identity [1] that rewrites, for position  $i$  at rank  $r$ , as

$$\text{LPF}[i] = \max(\text{LPF}_>[r], \text{LPF}_<[r]).$$

### 3 Sorting Network

It has been noticed in [4] that the content of the LPF table is the same as that of the LCP table up to some permutation. A question arises then: what permutation is it? Does it depend on the text, its Suffix Array or maybe just its LCP array? It turns out that, for fixed SUF and LCP arrays, it does not depend on the actual text. It is possible to construct such a sorting network, whose shape depends only on the Suffix Array, that transforms the LCP array into the LPF table. This observation leads to the algorithm LPF-SORTING, producing LPF by permutating the elements of LCP.

In the algorithm LPF-SORTING below we assume that the table `next` introduced in the previous section has been pre-computed by the procedure `NEXT` (in which instructions related to  $\text{LPF}_>$  are useless and may be removed, as well as the parameter `LCP`). Table `next` is used to compute, for each position  $i$ , the next closest position `nextp[i]` in the Suffix Array that is smaller than  $i$ , that is,

$$\text{nextp}[\text{SUF}[r]] = \text{SUF}[\min\{t \mid t > r \text{ and } \text{SUF}[t] < \text{SUF}[r]\}].$$

Initially, the LPF table is a copy of the LCP array permuted according to the SUF table. The algorithm sorts the array by permuting its elements to get the LPF table.

LPF-SORTING(SUF, LCP,  $n$ )

```

1  SUF[ $n$ ]  $\leftarrow$   $-1$ 
2  for  $r \leftarrow 0$  to  $n - 1$  do
3      LPF[SUF[ $r$ ]]  $\leftarrow$  LCP[ $r$ ]
4      nextp[SUF[ $r$ ]]  $\leftarrow$  SUF[next[ $r$ ]]
5  for  $i \leftarrow n - 1$  downto 0 do
6      if (nextp[ $i$ ]  $\geq$  0) and (LPF[ $i$ ] < LPF[nextp[ $i$ ]]) then
7          EXCHANGE(LPF[ $i$ ], LPF[nextp[ $i$ ]])
8  return LPF
```

Note that the elements of LPF are exchanged (lines 6-7) only if they are not in increasing order. Which elements are compared, depends on values in nextp, and this in turn depends only on the Suffix Array. So, for a given Suffix Array, one can construct a sorting network implementing lines 5-7 of the algorithm LPF-SORTING. Hence, the following proposition holds:

**Proposition 2.** *For a given Suffix Array of a text, there exists a sorting network processing a sequence of  $n$  numbers in such a way that it transforms the LCP table into the LPF table. Moreover, the shape of the sorting network depends only on the Suffix Array, but not on its LCP table nor on the actual text.*

As it is done for any sorting procedure, instead of directly computing the LPF table, the algorithm LPF-SORTING can equivalently produce explicitly the permutation  $\pi$  to transform the LCP array into the LPF table, that is, the permutation that satisfies  $\text{LPF}[i] = \text{LCP}[\pi[i]]$ .

The algorithm LPF-SORTING uses only one integer array in addition to its input and output since the next table it uses is substituted for the nextp table. This yields the next statement.

**Proposition 3.** *The algorithm LPF-SORTING computes the LPF table of a text of length  $n$  from its Suffix Array in linear time and space. It requires  $n+c$  integer cells in addition to its input and output.*

## 4 On-Line Computation

Techniques of the previous sections to compute the LPF table are simple but space consuming. In this section and the next one we address this issue. We show that a computation on-line on the Suffix Array using a stack reduces the memory space to only  $O(\sqrt{n})$  for a text of length  $n$ .

The design of the on-line computation still relies on the property used for the algorithm of Section 2 and related to “peaks” (see lines 6-8 below). It relies additionally on another straightforward property that we describe now. Assume that a position  $\text{SUF}[r]$  at rank  $r$  satisfies  $\text{LCP}[r] \geq \text{LCP}[r + 1]$ . Then, no

position after it in the list can provide a larger LCP value and therefore we get  $\text{LPF}[\text{SUF}[r]] = \text{LCP}[r]$ . This is implemented in lines 10-11 of the algorithm LPF-ON-LINE.

Note that lines 15 to 18 may be removed from the algorithm LPF-ON-LINE if the Suffix Array can be extended to rank  $n$  and initialised to  $\text{SUF}[n] = -1$  and  $\text{LCP}[n] = 0$ . But we prefer the present design that is compatible with the algorithm of the next section.

```

LPF-ON-LINE(SUF, LCP, n)
1  EMPTYSTACK(S)
2  for r ← 0 to n - 1 do
3      r-lcp ← LCP[r]
4      while not EMPTY(S) do
5          (t, t-lcp) ← TOP(S)
6          if SUF[r] < SUF[t] then
7              LPF[SUF[t]] ← max{t-lcp, r-lcp}
8              r-lcp ← min{t-lcp, r-lcp}
9              POP(S)
10         elseif (SUF[r] > SUF[t]) and (r-lcp ≤ t-lcp) then
11             LPF[SUF[t]] ← t-lcp
12             POP(S)
13         else break
14     PUSH(S, (r, r-lcp))
15 while not EMPTY(S) do
16     (t, t-lcp) ← TOP(S)
17     LPF[SUF[t]] ← t-lcp
18     POP(S)
19 return LPF
    
```

*Stack size.* The extra memory space used by the algorithm LPF-ON-LINE to compute the LPF table of a text is occupied by the stack and a constant number of integer variables. To evaluate the total size required by the algorithm it is then important to determine the maximal size of the stack for a text of length  $n$ . It is proved in [5] that this quantity is  $O(\sqrt{n})$ .

For most values of  $n$  there are plenty of texts for which the stack reaches its maximal size. But if  $n$  is of the form  $k(k + 1)/2$ , that is, if it is the sum of the first  $k$  positive integers, then there is a unique string on the alphabet  $\{a, b\}$  (with  $a < b$ ) giving the maximal size stack. This word is  $aabab^2 \dots ab^{k-1}$  and the maximal stack size is  $k$ .

The next table shows maximal stack sizes for texts of lengths 4 to 19:

length $n$	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
max-stack-size	2	2	3	3	3	3	4	4	4	4	4	5	5	5	5	5	5	6	6

**Proposition 4.** *The algorithm LPF-ON-LINE computes the LPF array of a text of length  $n$  from its Suffix Array in linear time and  $O(\sqrt{n})$  space. It requires less than  $2\sqrt{2n} + c$  integer cells in addition to its input and output.*

## 5 Time-Space Optimal Implementation

In this section we show that the computation of the LPF table of a text can be implemented with only constant memory space in addition to the Suffix Array of the text and its LPF table. The underlying property used for this purpose is the  $O(\sqrt{n})$  stack size reported in the previous section. The property allows an implementation of the stack inside the LPF table for a sufficiently large part of the text. The rest of the computation for the remaining positions is done in a more time-expensive manner but for a small part of the text. This preserves the linear running time of the whole computation.

LPF-OPTIMAL(SUF, LCP,  $n$ )

- 1 ▷ it is assumed that  $n \geq 8$
- 2  $K \leftarrow \lfloor n - 2\sqrt{2n} \rfloor$
- 3 ▷ next three procedures share the same LPF table
- 4 LPF-ON-LINE(SUF, LCP,  $K$ )
- 5 LPF-NAIVE(SUF, LCP,  $K, n$ )
- 6 LPF-ANCHORED(SUF, LCP,  $K, n$ )
- 7 **return** LPF

It is rather clear that only constant extra memory space is required to implement the strategy retained by the algorithm LPF-OPTIMAL. The choice of the parameter  $K$  is a result of the previous section and is done to let enough space in the LPF array to implement the stack used by the algorithm LPF-ON-LINE.

Although not done here, the choice of the parameter  $K$  can be dynamic and done during the algorithm LPF-ON-LINE as  $n - 2k - 1$ , where  $k$  is the size of the stack just before executing line 15. This certainly reduces the actual running time but does not improve its asymptotic evaluation.

In the algorithm LPF-OPTIMAL, the stack of the procedure LPF-ON-LINE is implemented in the LPF table. Access to the table is done via the SUF array. Doing so, the stack is treated like a continuous space LPF[SUF[ $K \dots n - 1$ ]]. Elements are stored sequentially so that the elementary stack operations (empty, top, push, pop) are all executed in constant time. Therefore, the running time of the first step is  $O(n)$  (indeed  $O(K)$ ) as for the algorithm LPF-ON-LINE.

*Example 2.* The next table shows the content of the LPF array at two stages of the run of the procedure LPF-ON-LINE on Example 1: (i) immediately after processing rank 5 and (ii) at the end of the first step.

	rank $r$	0	1	2	3	4	5	6	7	8	9	10	11	12	13
(i)	LPF[SUF[ $r$ ]]	1		3	4					5	3	4	2	1	0
(ii)	LPF[SUF[ $r$ ]]	1	0	3	4	2	3	1				8	2	7	0

Row (i) shows that LPF values of positions 2, 3, 5 at respective ranks 0, 2, 3 have already been computed. The part LPF[SUF[8..13]] of the table stores the content of the stack:  $((1, 0), (4, 2), (5, 3))$ . Row (ii) shows that values have already been computed for positions at ranks 0 to 6. The content of the stack is  $((7, 0), (8, 2))$ . On this example,  $K$  could have been set dynamically to 5.

*Naive computation.* The second step of the algorithm LPF-OPTIMAL processes the Suffix Array from rank  $K$ . For each rank  $r$ , the values  $\text{prev}[r]$ ,  $\text{next}[r]$  and the corresponding LCP values are computed starting from  $r$  and going backward and forward, respectively. The code is given below for the sake of completeness but it contains no algorithmic high value.

The process takes  $O(n - K)$  time per rank, and hence the total running time of the step is  $O((n - K)^2)$ , which is  $O(n)$  since  $K = n - 2\sqrt{2n}$ .

LPF-NAIVE(SUF, LCP,  $K, n$ )

```

1  LCP[ $n$ ]  $\leftarrow$  0
2  for  $r \leftarrow K$  to  $n - 1$  do
3       $left \leftarrow$  LCP[ $r$ ]
4       $s \leftarrow r - 1$ 
5      while  $s \geq K$  and  $\text{SUF}[s] > \text{SUF}[r]$  do
6           $left \leftarrow \min\{left, \text{LCP}[s]\}$ 
7           $s \leftarrow s - 1$ 
8      if  $s = K - 1$  then
9           $left \leftarrow 0$ 
10      $t \leftarrow r + 1$ 
11      $right \leftarrow$  LCP[ $t$ ]
12     while  $t < n$  and  $\text{SUF}[t] > \text{SUF}[r]$  do
13          $t \leftarrow t + 1$ 
14          $right \leftarrow \min\{right, \text{LCP}[t]\}$ 
15     LPF[SUF[ $r$ ]]  $\leftarrow \max\{left, right\}$ 
16 return LPF

```

*Completing the computation.* The first two steps of the algorithm LPF-OPTIMAL process independently two segments of the Suffix Array. The last step consists in joining their results, which requires updating some LPF values. Indeed, for a rank  $r$ ,  $r < K$ ,  $\text{next}[r]$  can be in  $[K, n - 1]$ , which implies that the computation of LPF[SUF[ $r$ ]] might not be achieved. The same phenomenon happens for a rank in the second part, whose associated  $\text{prev}$  rank is smaller than  $K$ .

The next algorithm updates all LPF values and completes the whole computation. It assumes that the LPF calculation has been done independently on parts LPF[SUF[ $0 \dots K - 1$ ]] and LPF[SUF[ $K \dots n - 1$ ]] of the array, which is realised by the first two steps on the algorithm LPF-OPTIMAL.

The running time of this last step LPF-ANCHORED is obviously linear,  $O(n)$ , as are the other steps of the algorithm LPF-OPTIMAL.

The first conclusion of the section is the following statement.

**Theorem 1.** *The Longest Previous Factor table of a text of length  $n$  on an integer alphabet can be built from its Suffix Array in time  $O(n)$  (independently of the alphabet size) with a constant amount of extra memory space.*

```

LPF-ANCHORED(SUF, LCP,  $K$ ,  $n$ )
1   $s \leftarrow K - 1$ 
2   $t \leftarrow K$ 
3   $\ell \leftarrow \text{LCP}[K]$ 
4  while ( $s \geq 0$ ) and ( $t < n$ ) do
5       $\triangleright$  invariant:  $\ell = \min \text{LCP}[s + 1, \dots, r]$ 
6      if  $\text{SUF}[s] < \text{SUF}[t]$  then
7           $\text{LPF}[\text{SUF}[t]] \leftarrow \max\{\text{LPF}[\text{SUF}[t]], \ell\}$ 
8           $t \leftarrow t + 1$ 
9           $\ell \leftarrow \min\{\ell, \text{LCP}[t]\}$ 
10     else  $\text{LPF}[\text{SUF}[s]] \leftarrow \max\{\text{LPF}[\text{SUF}[s]], \ell\}$ 
11          $\ell \leftarrow \min\{\ell, \text{LCP}[s]\}$ 
12          $s \leftarrow s - 1$ 
13 return LPF

```

The algorithm LPF-OPTIMAL uses the Suffix Array of the input text in a read-only manner but does not use the LPF table in a write-only manner. If this last condition is to be satisfied, the question remains of whether there exists a linear-time LPF table construction running with constant extra space. We get this feature if the algorithm LPF-ANCHORED is applied recursively by dividing the Suffix Array into two equal parts. The running time becomes  $O(n \log n)$  in the model of computation allowing priority writes.

**Theorem 2.** *The Longest Previous Factor table of a text of length  $n$  on an integer alphabet can be built from its read-only Suffix Array in time  $O(n \log n)$  (independently of the alphabet size) with a constant amount of extra memory space and with a write-only output.*

Despite the use of the output as auxiliary storage in the ultimate linear-time algorithm, the series of algorithms described in the article provide a large range of efficient solutions that meet many practical needs.

## Acknowledgements

Authors warmly thank German Tischler for his careful inspection of the algorithms described in the article.

## References

1. Berstel, J., Savelli, A.: Crochemore factorization of Sturmian and other infinite words. In: Kráľovič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 157–166. Springer, Heidelberg (2006)
2. Crochemore, M.: Transducers and repetitions. *Theoretical Computer Science* 45(1), 63–86 (1986)
3. Crochemore, M., Hancart, C., Lecroq, T.: *Algorithms on Strings*, 392 p. Cambridge University Press, Cambridge (2007)



4. Crochemore, M., Ilie, L.: Computing Longest Previous Factor in linear time and applications. *Inf. Process. Lett.* 106(2), 75–80 (2008)
5. Crochemore, M., Ilie, L., Smyth, W.F.: A simple algorithm for computing the Lempel-Ziv factorization. In: Storer, J.A., Marcellin, M.W. (eds.) 18th Data Compression Conference, pp. 482–488. IEEE Computer Society, Los Alamitos (2008)
6. Crochemore, M., Rytter, W.: *Jewels of Stringology*, 310 p. World Scientific Publishing, Hong-Kong (2002)
7. Franek, F., Holub, J., Smyth, W.F., Xiao, X.: Computing quasi suffix arrays. *Journal of Automata, Languages and Combinatorics* 8(4), 593–606 (2003)
8. Nong, G., Zhang, S., Chan, W.H.: Linear Time Suffix Array Construction Using D-Critical Substrings. In: Kucherov, G., Ukkonen, E. (eds.) *Combinatorial Pattern Matching. LNCS*, vol. 5577, pp. 54–67. Springer, Heidelberg (2009)
9. Gusfield, D.: *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, 534 p. Cambridge University Press, Cambridge (1997)
10. Kärkkäinen, J., Sanders, P.: Simple linear work suffix array construction. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) *ICALP 2003. LNCS*, vol. 2719, pp. 943–955. Springer, Heidelberg (2003)
11. Kasai, T., Lee, G., Arimura, H., Arikawa, S., Park, K.: Linear-time longest-common-prefix computation in suffix arrays and its applications. In: Amir, A., Landau, G.M. (eds.) *CPM 2001. LNCS*, vol. 2089, pp. 181–192. Springer, Heidelberg (2001)
12. Kim, D.K., Sim, J.S., Park, H., Park, K.: Linear-time construction of suffix arrays. In: Baeza-Yates, R., Chávez, E., Crochemore, M. (eds.) *CPM 2003. LNCS*, vol. 2676, pp. 186–199. Springer, Heidelberg (2003)
13. Ko, P., Aluru, S.: Space efficient linear time construction of suffix arrays. In: Baeza-Yates, R., Chávez, E., Crochemore, M. (eds.) *CPM 2003. LNCS*, vol. 2676, pp. 200–210. Springer, Heidelberg (2003)
14. Kolpakov, R.M., Kucherov, G.: Finding maximal repetitions in a word in linear time. In: *FOCS*, pp. 596–604 (1999)
15. Main, M.G.: Detecting leftmost maximal periodicities. *Discret. Appl. Math.* 25, 145–153 (1989)
16. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. *SIAM J. Comput.* 22(5), 935–948 (1993)
17. Rodeh, M., Pratt, V.R., Even, S.: Linear algorithm for data compression via string matching. *J. ACM* 28(1), 16–24 (1981)
18. Storer, J., Szymanski, T.: Data compression via textual substitution. *J. ACM* 29(4), 928–951 (1982)
19. Witten, I., Moffat, A., Bell, T.: *Managing Gigabytes*. Van Nostrand Reinhold, New York (1994)
20. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory* 23(3), 337–343 (1977)
21. Ziv, J., Lempel, A.: Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory* 24(5), 530–536 (1978)

# Limiting Distribution for Distances in $k$ -Trees

Alexis Darrasse and Michèle Soria\*

LIP6 – UPMC Paris Universit as

**Abstract.** This paper examines the distances between vertices in a rooted  $k$ -tree, for a fixed  $k$ , by exhibiting a correspondence with a variety of trees that can be specified in terms of combinatorial specifications. Studying these trees via generating functions, we show a Rayleigh limiting distribution for expected distances between pairs of vertices in a random  $k$ -tree: in a  $k$ -tree on  $n$  vertices, the proportion of vertices at distance  $d = x\sqrt{n}$  from a random vertex is asymptotic to  $\frac{c_k^2 x}{\sqrt{n}} \exp(-\frac{c_k^2 x^2}{2})$ , where  $c_k = kH_k$ .

## 1 Introduction

This work takes place within the general framework of analyzing statistical properties of combinatorial structures: we evaluate distances between vertices in a graph structure named  $k$ -tree.

In graph theory, very important research is being done on  $k$ -trees, for their characterization [15,10] and from an algorithmic viewpoint, since many NP-complete problems can be solved linearly on  $k$ -trees [1]. The class of  $k$ -trees, together with many close families, have also been extensively studied as combinatorial structures for their enumeration [2,6,9,12].

Our interest in  $k$ -trees focuses on their graph structure and the behavior of parameters such as degree or distance. This work is a generalization of our study of planar 3-trees [3], and the results presented here can be easily extended to planar  $k$ -trees.

Here we are interested in quantifying the distribution of expected distances between two random vertices in a random  $k$ -tree, for a fixed  $k$ . Our main advantage for addressing this problem is a bijection between  $k$ -trees and a class  $\mathcal{K}$  which is specifiable in terms of combinatorial constructions and thus amenable to the powerful tool of generating functions which combines algebraic methods for constructing relevant power series and analytic methods for evaluating parameters of interest.

Our bijection extends works by Klawe et al. [8] and Ibarra [7] providing a one to one correspondence that exploits the recursive structure of  $k$ -cliques to transform the  $k$ -tree graph into a labeled tree structure. Moreover the parameter “ $k$ -tree distance to the root” transfers to a clearly identifiable parameter on subtrees of the tree structure, that can be precisely analysed by bivariate generating functions and leads to an asymptotic distribution that obeys a Rayleigh

---

\* Work partially supported by ANR contract GAMMA, n BLAN07-2-195422.

law. In order to deal with distances between random pairs of vertices we use a tree “rerooting” process that is also expressible in terms of generating functions and the same result of a Rayleigh asymptotic law still holds.

Though the tree parameter that we are studying is not a profile, its behavior is similar to profiles in simple varieties of trees, where Rayleigh distributions were first shown by Meir and Moon [11], see the recent book by Drmota [4] for an extensive review.

While the class  $\mathcal{K}$  that we consider here is a class of labeled trees corresponding to all  $k$ -trees on  $n$  vertices, the class of label-increasing trees of  $\mathcal{K}$  would also be of interest: it would correspond, with the same bijection, to  $k$ -trees whose labeling is constrained by their recursive construction. The corresponding generating functions satisfy differential rather than algebraic equations and we show in future work that the properties are similar to those of recursive trees [4].

In section 2, we set the bijective algorithm between  $k$ -trees and class  $\mathcal{K}$ , and derive the enumerative generating function for  $k$ -trees. Section 3 explains the algorithm to calculate distances (in the graph) to the root of a  $k$ -tree, on the corresponding tree-structure in  $\mathcal{K}$ . The corresponding equations on bivariate generating functions are established by a careful analysis, marking vertices at distance  $d$  to the root and working on the cumulative generating functions to get the proportion of vertices at distance  $d$ . Finally, evaluating coefficients by means of complex analysis, we obtain the limiting distribution of distances to the root. In section 4 we extend this result to distances between a random pair of vertices in a random  $k$ -tree: we give an algorithm to “reroot” a  $k$ -tree, and show, via the equality of their generating functions, that “rerooted”  $k$ -trees are in bijection with  $k$ -trees with a pointed vertex. Using the same technique as in section 3, we finally obtain a Rayleigh limiting distribution.

## 2 Structures: $k$ -Trees and Class $\mathcal{K}$

In this section we show a bijection between rooted  $k$ -trees and a simple variety of labeled trees, named  $\mathcal{K}$ , which is specifiable in terms of combinatorial constructions:  $\mathcal{K} = \mathcal{Z}^k \mathcal{T}$ , and  $\mathcal{T} = \text{Set}(\mathcal{Z} \times \mathcal{T}^k)$ . This representation of  $k$ -trees, which highlights the cliques and their relationships, will be helpful to study the distance between vertices.

*Inductive definition of  $k$ -trees.* A  $k$ -tree on  $n$  vertices is a graph defined inductively as follows: the complete graph on  $k$  vertices (a  $k$ -clique) is a  $k$ -tree (with  $n = k$ ), and if  $G$  is  $k$ -tree on  $n - 1$  vertices, then the graph resulting from adding a new vertex adjacent to the vertices of a  $k$ -clique of  $G$  is also a  $k$ -tree. This definition corresponds to graph theory trees when  $k = 1$ .

A *rooted  $k$ -tree* is a  $k$ -tree with a distinguished  $k$ -clique, together with a given permutation of the  $k$  vertices.

The  $\mathcal{K}$ -representation of a  $k$ -tree is a tree with black and white nodes: black nodes correspond to  $k+1$ -cliques and white nodes to  $k$ -cliques. Each black node is adjacent to the  $k + 1$  white nodes representing the  $k$ -cliques it contains. The size of a tree in  $\mathcal{K}$  is the number of black nodes it contains plus  $k$ .

*Remark 1.* In this paper we shall consistently use the term *vertex* to denote “points” of the graph ( $k$ -tree), and *node* to denote “points” of the tree (in  $\mathcal{K}$ ).

*Inductive definition of  $\mathcal{K}$ .* A tree  $T \in \mathcal{K}$  of size  $n$  is either reduced to its root, a white node with  $k$  vertices (size  $k$ ), or a tree  $T' \in \mathcal{K}$  of size  $n - 1$  in which we add a new black node adjacent to a white node of  $T$  and its  $k$  white sons.

## 2.1 Bijection

The transformation of a rooted  $k$ -tree  $G$  into a tree  $T \in \mathcal{K}$  is a two step process. First, create a tree called the completed clique-separator tree of  $G$ , whose nodes are cliques of  $G$ ; moreover the root of the  $k$ -tree is the root of the corresponding tree. Second, simplify the labels of the tree by encoding most of the information in the tree structure.

The completed clique-separator tree of a  $k$ -tree  $G$  is a bipartite graph, whose black nodes are the  $k+1$ -cliques of  $G$  (which are also the maximal cliques of  $G$ ), and whose white nodes are the  $k$ -cliques of  $G$  (which include the minimal separators of  $G$ ). A black node is adjacent to the  $k + 1$  white nodes it contains. This structure is very similar to the clique-separator graph Ibarra [7] defined for the larger class of chordal graphs.

**Proposition 1.** *The completed clique-separator tree of  $G$  is a tree.*

*Proof.* Following the inductive definition of  $k$ -trees it is easy to see that each new vertex adds one black node connected to one existing and  $k$  new white nodes.

**Proposition 2.** *There is a bijection between the class of rooted  $k$ -trees and  $\mathcal{K}$ .*

*Sketch of proof.* For a rooted  $k$ -tree, the completed clique-separator tree is a rooted tree; its root is the root of the  $k$ -tree (a  $k$ -clique presented as an ordered list of vertices).

We see that the only information carried by a black node is the label of the vertex not included in its father. We can thus simplify the labeling of black nodes by retaining only this one vertex.

The one-to-one correspondence between  $k$ -trees and trees in  $\mathcal{K}$  relies on an ordering of the sons of the black nodes. We proceed recursively from the root: consider a white node  $w$  with its ordered list of vertices  $(x_1, \dots, x_k)$ ; this node is adjacent to a set of black nodes and there is a natural order between the  $k$  sons of any of these black nodes. Let  $b$  be a black node: the  $i$ -th son  $s_i$  is the  $k$ -clique that does not contain  $x_i$ . The list of vertices in  $s_i$  is the same as in  $w$  except that  $x_i$  is missing and is replaced by  $b$ :  $(x_1, \dots, x_{i-1}, b, x_{i+1}, \dots, x_k)$ .

Now we only need  $n$  labels in the representation of a  $k$ -tree on  $n$  vertices; the list of vertices in each white node can be completely determined by the position of the node in the tree, the vertices in the black nodes and the list of vertices at the root (i.e.: the first white node). We do however keep the labels on the white nodes for clarity, in figure 1 for instance, they still appear, in grey color.

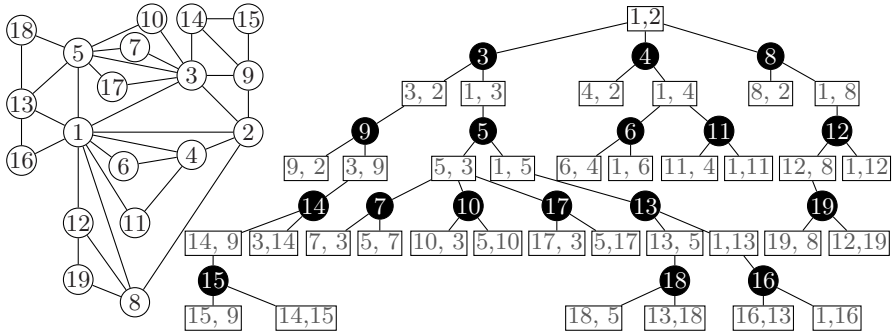


Fig. 1. A 2-tree and the corresponding tree in  $\mathcal{K}$

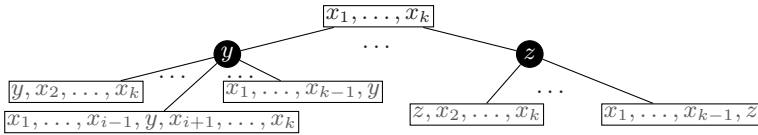


Fig. 2. The ordering of sons of black nodes

---

**Algorithm 1.** Tree representation

---

**Input:** a rooted  $k$ -tree  $G$  on  $n$  vertices

**Output:** a tree  $T$ , with a list of  $k$  vertices at the root

- 1: Create one white node for each  $k$ -clique of  $G$
  - 2: Create one black node for each  $k+1$ -clique of  $G$
  - 3: Put an edge between each black node and all  $k+1$   $k$ -cliques it contains  
 {At this point we have the completed clique-separator tree of  $G$ }
  - 4: Remove all vertices from each b. node except for the one not included in its father
  - 5: Propagate the order of the vertices in white nodes starting from the root
  - 6: Order the sons of b. nodes: the  $i$ -th does not containing the  $i$ -th vertex of the father
  - 7: Remove all vertices from each white node {The resulting tree is in  $\mathcal{K}$ }
- 

Reversing this algorithm is easy. The labels removed in steps 4 and 7 can be determined by the position of the nodes in the tree, and the list of  $k+1$ -cliques created in step 2 suffices to reconstruct the  $k$ -tree; no edge is missing: each one of them is between two vertices belonging to the same  $k+1$ -clique.

**2.2 Generating Function for  $\mathcal{K}$**

Algorithm □ transforms a rooted  $k$ -tree into a  $\mathcal{K}$  structure, composed of a tree-structure  $\mathcal{T}$  (that we call the *proper tree*), and a list of  $k$  vertices. A rooted  $k$ -tree on  $n$  vertices leads to a proper tree with  $n - k$  black nodes. The proper tree is

made of a white root, from which stems a set of black nodes; and each black node has a list of children which are  $k$  subtrees of the same type as the proper tree. Thus we get the following specification, with  $\mathcal{E}$  denoting white nodes and  $\mathcal{Z}$  denoting vertices of the original  $k$ -cliques:

$$\mathcal{K} = \mathcal{Z}^k \mathcal{T}, \quad \mathcal{T} = \mathcal{E} \times \text{Set}(\mathcal{Z} \times \mathcal{T}^k).$$

Using the symbolic method, see e.g. Flajolet and Sedgewick [5], we derive from this specification the functional equations on exponential generating functions,  $K(z) = \sum_n K_n \frac{z^n}{n!}$  and  $T(z) = \sum_n T_n \frac{z^n}{n!}$ , where  $K_n$  (resp.  $T_n$ ) is the number of trees of size  $n$  in  $\mathcal{K}$  (resp.  $\mathcal{T}$ ):

$$K(z) = z^k T(z), \quad T(z) = \exp(zT^k(z)). \tag{1}$$

These generating functions are extensively used in the rest of this paper, especially in a bivariate form. We first use them to compute the number of  $k$ -trees on  $n$  vertices (this is another proof of a well known result [2][12] ...]).

**Theorem 1.** *The number of  $k$ -trees on  $n + k$  vertices is  $\binom{n+k}{k} (kn + 1)^{n-2}$ .*

*Proof.* Let  $C_k(z) = kzT^k(z)$ . Then equation (1) becomes  $C_k(z) = kze^{C_k(z)}$  and  $T(z) = \exp(\frac{C_k(z)}{k})$ . Using the Lagrange-Bürmann theorem we thus get  $[z^n]T(z) = \frac{1}{n!} (kn + 1)^{n-1} = \frac{K_{n+k}}{(n+k)!}$ . Since  $\mathcal{K}_n$  is in bijection with the class of rooted  $k$ -trees on  $n$  vertices, we need to divide  $K_{n+k}$  by the number of possible roots,  $k!(kn + 1)$ , to obtain the number of  $k$ -trees.

### 3 Distance to the Root

This section deals with distances to the first vertex of the root of a  $k$ -tree. These distances can be easily marked on the corresponding tree structure, and by studying the resulting family of bivariate generating functions, we show that the proportion of vertices at distance  $d$  from the first vertex of the root is asymptotically Rayleigh distributed.

#### 3.1 Using the Tree to Calculate the Distance in the Graph

We use an algorithm similar to that of Proskurowski [13] to decorate each vertex with its (graph) distance to the root. Given a  $k$ -tree  $G$  and the corresponding tree  $T$  we start by assigning the distance 0 to the vertices of the root. Then, given a white node  $w$ , each of its black sons corresponds to a vertex at distance 1 plus the minimum of the distances of  $w$ 's vertices.

Note that this process can be applied recursively starting from the root since each vertex in a white node  $w$  is either a part of the root or in a black node on the path from  $w$  to the root.

**Algorithm 2.** Distances to the first vertex of the root in  $\mathcal{K}$

**Input:** a tree  $K \in \mathcal{K}$  (with root  $r$ )

**Output:** an association table (vertex, distance)

- 1: Create an empty table  $A$
- 2: **for all** vertices  $v$  in  $r$  **do**
- 3:   **if**  $v$  is the first vertex of  $r$  **then** Add  $(v, 0)$  to  $A$  **else** Add  $(v, 1)$  to  $A$
- 4:   **return**  $A \cup$  the result of Algorithm 3 on the proper tree of  $K$  and  $(0, 1, \dots, 1)$

*Remark 2.* The same process can be used to calculate the distance to any subset  $s = (x_{j_1}, \dots, x_{j_i})$  of  $i$  vertices in the root: instead of assigning distance 0 to all the vertices of the root, we set to 0 the distance of the  $i$  vertices in  $s$  and set to 1 the distance of the other  $k - i$  vertices of the root. Notice that the resulting decorated tree could have been obtained as a subtree of the distance tree in the original process.

**Algorithm 3.** Distances in  $\mathcal{T}$

**Input:** a tree  $T \in \mathcal{T}$  and  $k$  integers  $(a_i)_{i \in \{1, \dots, k\}}$

**Output:** an association table (vertex, distance)

- 1: Let  $d$  be  $1 + \min(a_1, \dots, a_k)$  and  $A$  an empty table
- 2: **for all** sons  $v$  of the root of  $T$  **do**
- 3:   Add  $(v, d)$  to  $A$
- 4:   **for all**  $i \in \{1, \dots, k\}$  **do**
- 5:      $A \leftarrow A \cup$  recursive call on the  $i$ -th son of  $v$  and  $(a_1, \dots, a_{i-1}, d, a_{i+1}, \dots, a_k)$
- 6: **return**  $A$

*Remark 3.* It is clear that if we shift, by a value  $d$ , all distances of a white node  $w$ 's vertices, then all the distances in the subtree under  $w$  will be consistently shifted by  $d$ .

**3.2 Bivariate Generating Functions**

In this section, we are interested in estimating  $K_{d,n,p}$ , the number of trees of size  $n$  in  $\mathcal{K}$  with  $p$  vertices at distance  $d$  from the first vertex of the root. Though it is possible to directly work on the combinatorial objects, we use a generating function framework that makes presentation easier. We thus define the bivariate generating functions  $K_d(z, u) = \sum_{n,p} K_{d,n,p} u^p \frac{z^n}{n!}$ , with  $K_d(z, 1) = K(z)$ , for all  $d$ . Differentiating  $K_d(z, u)$  with respect to  $u$  and setting  $u = 1$ , provides function

$$K'_d(z) \equiv \left. \frac{\partial}{\partial u} K_d(z, u) \right|_{u=1},$$

where the coefficient of  $\frac{z^n}{n!}$  represents the total number of vertices at distance  $d$  from the first vertex of the root in all trees of size  $n$  in  $\mathcal{K}$ . So that, in a random  $k$ -tree of size  $n$  the proportion of vertices at distance  $d$  from the vertex of the

root is  $\frac{1}{nK_n}K'_d(z)$ . The aim of this section is to give a closed form expression for  $K'_d(z)$  (see proposition [3](#)).

We mostly argument and calculate on the proper subtree  $T$  rather than on the whole structure  $K$  and the generating functions are very similar.

**Lemma 1.** *Let  $T_d(z, u)$  be the bivariate generating function with  $u$  marking vertices at distance  $d$  from the first vertex of the root,*

$$K_1(z, u) = z^k u^{k-1} T_1(z, u), \text{ and } \forall d \geq 2, K_d(z, u) = z^k T_d(z, u).$$

*Proof.* In the case  $d = 1$ , each of the  $k - 1$  other vertices of the root are at distance one from the first vertex, since they are in the same clique. Whereas for  $d \geq 2$  vertices in the root do not interfere.

*Case  $d = 1$ .* We begin with vertices at distance one to the first vertex of the root. In a second paragraph we shall be interested in vertices at distance one to a subset of  $i$  vertices in the root, in order to prepare the general study of vertices at distance  $d$ .

**Lemma 2.**  $T_1(z, u) = \exp(zuT(z))T_1^{k-1}(z, u)$ .

*Proof.* All sons of the root are at distance one, and for each of these black nodes, all but the first of its white children contain the first vertex of the root, so that the black nodes immediately below them are also at distance one. Hence the result for generating functions.

By differentiation we thus obtain the generating function for the total number of vertices at distance one, (also using the fact that  $T_1(z, 1) = T(z)$ ).

**Lemma 3.**  $T'_1(z) \equiv \frac{\partial}{\partial u} T_1(z, u) \Big|_{u=1} = \frac{zT^{k+1}(z)}{1-(k-1)zT^k(z)}$ .

*Remark 4.* If we want to count the distances to some other vertex in the root, the whole computation is the same: permuting the root's vertices brings us back to the initial problem.

The next problem is to count the number of vertices at distance one from a subset  $s$  of  $i$  vertices in the root. Extending the preceding notation ( $T_1(z, u) = T_{1,1}(z, u)$ ), let  $T_{1,i}(z, u)$  be the bivariate generating function for the number of vertices at distance one from any vertex in  $s$ .

**Lemma 4.**  $T_{1,i}(z, u) = \exp(zuT_{1,i-1}^i(z, u))T_{1,i}^{k-i}(z, u)$ .

*Proof.* The idea of the proof is the same as when  $i = 1$ : for each black node at distance one,  $k - i$  of its white children contain all the  $i$  vertices of the subset and the remaining  $i$  contain all but one of them. By symmetry, the corresponding bivariate generating functions do not depend on the position of the  $i$  vertices in the root.

Differentiating leads to the generating function for the number of vertices at distance one from  $i$  vertices of the root, (notice that for all  $i$ ,  $T_{1,i}(z, 1) = T(z)$ ).

**Lemma 5.**  $T'_{1,i}(z) \equiv \frac{\partial}{\partial u} T_{1,i}(z, u) \Big|_{u=1} = \frac{zT^k(z)}{1-(k-i)zT^k(z)} (T(z) + iT'_{1,i-1}(z))$ .

This recurrence can be solved and  $T'_{1,i}(z)$  is a rational function in  $z$  and  $T(z)$ .



*General case.* By remark [3](#), in order to calculate the number of vertices at distance two from the first vertex of the root, it suffices to find all the white nodes containing only vertices at distance one (from the first vertex of the root) and apply the previous process to count the vertices at distance one from any of the vertices of these white nodes. In terms of generating functions, this means that  $T_{2,1}(z, u) = \exp(zT_{1,k}(z, u)T_{2,1}^{k-1}(z, u))$  (notice that there is no occurrence of  $u$  outside  $T_{1,k}(z, u)$ ).

Applying the same argument recursively, we can treat the case of any distance  $d \geq 2$ , and obtain the following lemma.

**Lemma 6.** *The bivariate generating function  $T_{d,i}(z, u)$ , with  $u$  marking the vertices at distance  $d$  from a subset of  $i$  vertices of the root, satisfies, for  $d \geq 2$ :*

$$T_{d,i}(z, u) = \exp(zT_{d,i-1}^i(z, u)T_{d,i}^{k-i}(z, u)), \quad \text{for } i \geq 2, \quad \text{and}$$

$$T_{d,1}(z, u) = \exp(zT_{d-1,k}(z, u)T_{d,1}^{k-1}(z, u)).$$

By differentiating we obtain:

**Lemma 7.**  $T'_{d,i}(z) \equiv \frac{\partial}{\partial u} T_{d,i}(z, u) \Big|_{u=1} = \frac{izT^k(z)}{1-(k-i)zT^k(z)} T'_{d,i-1}(z)$ ,  
 for  $i \geq 1$  and  $d \geq 2$ , setting  $T'_{d,0}(z) = T'_{d-1,k}(z)$ .

**Lemma 8.**  $T'_{d,i}(z) = H(z)T'_{d-1,i}(z)$ , for  $d \geq 2$ ,  
 where  $H(z) = k!(zT^k(z))^k \prod_{i=1}^{k-1} \frac{1}{1-izT^k(z)}$ .

This recurrence is easy to expand, the only difficulty is that it does not extend to when  $d = 1$ . We can however calculate  $T'_{2,1}(z)$  which has the form of a rational function in  $z$  and  $T(z)$ .

**Lemma 9.**  $T'_{d,1}(z) \equiv \frac{\partial}{\partial u} T_{d,1}(z, u) \Big|_{u=1} = H^{d-2}(z)T'_{2,1}(z)$ .

Back to the whole structure  $\mathcal{K}$ , we have  $K_d(z, u) = z^k T_{d,1}(z, u)$ , so that

**Proposition 3.** *The exponential generating function counting the total number of vertices at distance  $d$  from the first vertex of the root in a rooted  $k$ -tree satisfies*

$$K'_d(z) \equiv \frac{\partial}{\partial u} K_d(z, u) \Big|_{u=1} = H^{d-2}(z)K'_2(z),$$

with  $K'_2(z) = z^k T'_{2,1}(z)$  and  $H(z) = k!(zT^k(z))^k \prod_{i=1}^{k-1} \frac{1}{1-izT^k(z)}$ .

### 3.3 Limiting Distribution

The number of vertices at distance  $d$  from the first vertex of the root in a  $k$ -tree of size  $n$  is obtained by estimating the coefficient of  $\frac{z^n}{n!}$  in  $K'_d(z)$  and normalizing by  $nK_n$ . We study the asymptotic of this quantity when  $n$  becomes large.

We first turn to the asymptotic estimation of coefficients of  $T(z)$ ,  $H(z)$  and  $H^d(z)$ , which relies on complex analysis.

**Proposition 4.** *Function  $T(z)$  is analytic at the origin, with radius of convergence  $\rho = \frac{1}{ke}$ , singular value  $\tau = e^{\frac{1}{k}}$ , and a square-root singular expansion:*

$$T(z) = \tau - \frac{\tau}{k} \sqrt{2(1 - z/\rho)} + O(1 - z/\rho).$$

*Proof.* Class  $\mathcal{T}$  is a simple variety of trees [11], and the result follows from the implicit function theorem. Moreover by singularity analysis  $T_n \sim \frac{\tau}{k\sqrt{2\pi}} \rho^{-n} n^{-\frac{3}{2}}$ .

**Lemma 10.**  *$H(z)$  is singular in  $\rho$ , with a square-root singular expansion*

$$H(z) = 1 - kH_k \sqrt{2(1 - z/\rho)} + O(1 - z/\rho), \text{ where } H_k = \sum_{i=1}^k \frac{1}{i}.$$

*Proof.* As seen before,  $H(z) = k!(zT^k(z))^k \prod_{i=1}^{k-1} \frac{1}{1-izT^k(z)}$ . For all  $k \in \mathbb{N}^*$  and  $i \in \{1, \dots, k-1\}$ , the term  $izT^k$  is singular in  $\rho$  and asymptotically equivalent to  $i\rho\tau^k = \frac{i}{k} < 1$ , so that no singularity comes from the cancellation of the denominators in the product. This product is shown to be equivalent, around the singularity  $\rho$ , to  $G(z) = \frac{k^{k-1}}{(k-1)!} \prod_{i=1}^{k-1} (1 - \frac{i\sqrt{2(1-z/\rho)}}{k-i})$ .  $H(z)$  is equivalent to  $(zT^k(z))^k G(z)$ , and the combination of square-root terms brings up a factor involving the  $k$ -th harmonic number  $H_k$ .

Evaluating the coefficient of  $z^n$  in  $H^d(z)$  depends on the values of  $d$ . The region where an interesting renormalization takes place is for  $d = x\sqrt{n}$ , as shown in the semi-large power theorem [5, Theorem IX.16].

**Proposition 5.** *For  $d = x\sqrt{n}$ , with  $x$  in any compact of  $\mathbb{R}_+^*$ ,*

$$[z^n]H^d(z) \sim \frac{kH_k x}{n\sqrt{2\pi}} e^{-\frac{k^2 H_k^2 x^2}{4}}.$$

*Proof.* This result can be obtained by using the saddle-point method or singularity analysis [5].

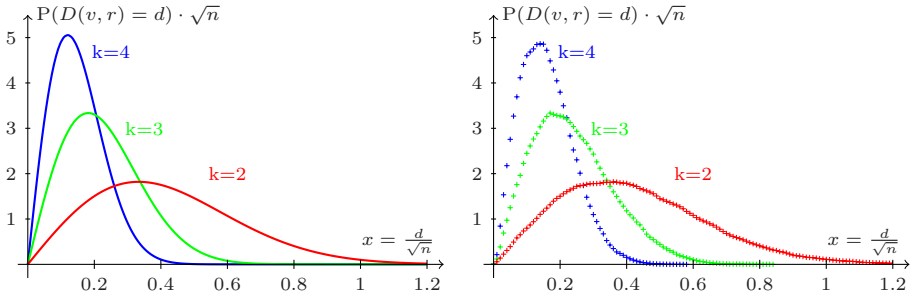
Back to the estimation of distances, we have  $K'_d(z) = H^{d-2}(z)K'_2(z)$ , where  $K'_2(z)$  is a rational function of  $z$  and  $T(z)$  with no pole in  $[0, \rho]$ , so that it contributes for a constant in the coefficient of  $z^n$ . Finally, to get the proportion of vertices at distance  $d$ , we normalize by  $nK_n$ , and obtain the following theorem.

**Theorem 2.** *In a  $k$ -tree on  $n$  vertices, the probability that a random vertex  $r$  is at distance  $d = x\sqrt{n}$  (with  $x$  in a compact of  $\mathbb{R}_+^*$ ) from the first vertex  $v$  of the root, satisfies, as  $n \rightarrow \infty$ , a local law of the Rayleigh type:*

$$\lim_{n \rightarrow \infty} \sqrt{n} \mathbb{P}(D(v, r) = \lfloor x\sqrt{n} \rfloor) = c_k^2 x e^{-\frac{(c_k x)^2}{2}}, \text{ with } c_k = k \sum_{i=1}^k \frac{1}{i}.$$

In parallel to proving the limiting distribution, we made a series of measures on random  $k$ -trees generated with a purpose-built Boltzmann sampler [1], and the experimental curves perfectly fit the theoretical results, as shown in figure [3].

<sup>1</sup> Available at <http://www-apr.lip6.fr/~darrasse/ktrees>



**Fig. 3.** Theoretical (on the left) and experimental (on the right) distributions of distances in  $k$ -trees, for  $k = 2, 3$  and  $4$ . The experimental curves come from measures on  $10^3$  random  $k$ -trees of size  $10^4 \pm 10\%$ .

### 4 Distances to a Random Vertex

To estimate the distance to a random vertex of a  $k$ -tree we use the same idea as Proskurowski [14]: “rerooting” the corresponding tree in order to move the vertex we are interested in to the root. In our case, we find an expression  $K^\circ(z)$  of the rerooted  $k$ -trees in terms of generating functions. This is proved by exhibiting a bijection between rerooted  $k$ -trees and pointed  $k$ -trees (that is  $k$ -trees with one pointed vertex, counted by  $K^\bullet(z)$ ).

The limiting distribution is obtained by using the same analytic tools as before and we finally prove that the distances to a random vertex exactly obey the same distribution as the previous case.

#### 4.1 Rerooting Process

Given a distinguished vertex  $v$  in a rooted  $k$ -tree  $G$  with root  $r$ , we want to associate another rooted  $k$ -tree  $G'$  having the same underlying  $k$ -tree as  $G$  and where  $v$  is the first vertex of the new root  $r'$ . Two cases appear: either  $v$  belongs to the list of vertices of  $r$ , and exchanging  $v$  with  $r$ 's first vertex suffices, or we need to find another  $k$ -clique of  $G$  to be the root of  $G'$ .

It is easier to work on the corresponding tree  $T \in \mathcal{K}$ . In  $T$ , we want to find a white node containing  $v$  as first vertex. There are many choices, but only one of them is always the closest to the root: the first son of  $v$ . This white node, named  $r'$ , will be the root of  $G'$ . The corresponding tree  $T'$  can be either constructed using the method of section 2, or obtained directly from  $T$  by pulling on  $r'$ .

Note that this process is reversible, since the mark allows to find the old root  $r$ .

*Remark 5.* The mark of the old root is always contained in the first subtree of one of the sons of the new root.

**Algorithm 4.** Rerooting

**Input:** a tree  $T \in \mathcal{K}$  (with root  $r$ ) and a vertex  $v$

**Output:** a tree in  $\mathcal{K}^\circ$ , with  $v$  its first root vertex

- 1: **if**  $v$  is in  $r$  **then**
- 2:   Put a mark on the first vertex of  $r$
- 3:   Exchange the first vertex in  $r$  with  $v$
- 4:   Apply steps 5 and 6 of Algorithm [4](#) to reorder the sons of black nodes
- 5:   **return** the resulting tree
- 6: **else** {in this case  $v$  is a black node}
- 7:   Let  $r'$  be the first son of  $v$  ( $r'$  is a white node and  $v$  its first vertex)
- 8:   Get the  $k$ -tree  $G$  corresponding to  $T$  by the inverse of Algorithm [4](#)
- 9:   Let  $G'$  be the same graph as  $G$ , with a new root:  $r'$
- 10:   Apply Algorithm [4](#) to  $G'$  to obtain a new tree  $T'$
- 11:   **return**  $T'$  with a mark on the white node corresponding to  $r$

**Theorem 3.** *The class of rerooted  $k$ -trees is counted by the generating function  $K^\circ(z) = kz^kT(z) + z^{k+1}T^k(z)T^\circ(z)$ , where  $T^\circ(z)$  counts the trees in  $\mathcal{T}$  with a mark on a white node.*

*Proof.* The two terms of the sum correspond to the two cases of the rerooting process. For the first one, we have  $k$  possibilities. For the second, remark [5](#) implies that the black son of the root containing the old root is described by  $zT^\circ(z)T^{k-1}(z)$  and the other black sons give the factor  $\exp(zT^k(z))$ :  
 $K^\circ(z) = kz^kT(z) + z^k \exp(zT^k(z)) zT^\circ(z)T^{k-1}(z) = kz^kT(z) + z^{k+1}T^k(z)T^\circ(z)$ .

**Theorem 4.** *There is a bijection between rerooted  $k$ -trees and pointed  $k$ -trees.*

*Proof.* We show that the generating functions for both classes are the same. We first need to express  $T^\circ(z)$ : adding a variable  $x$  to count white nodes for trees in  $\mathcal{T}$ , deriving with respect to  $x$  and setting  $x = 1$ , we get

$$T^\circ(z) = T(z) + k \exp(zT^k(z))zT^\circ(z)T^{k-1}(z) = T(z) + zkT^k(z)T^\circ(z),$$

which can also be obtained with a combinatorial argument: marking a white node consists either in marking the root or in choosing one of the sets of black nodes below it and marking in one of the  $k$  subtrees below this black node.

We now need to show the equality of generating functions  $T^\bullet(z) = zT^k(z)T^\circ(z)$ , where  $T^\bullet(z)$  counts trees in  $\mathcal{T}$  with a mark on a black node.

For that, we use the fact that a tree in  $\mathcal{T}$  with  $n$  black nodes contains  $kn + 1$  white nodes and that  $[z^n]T^\bullet(z) = n[z^n]T(z)$ . We thus have

$$\begin{aligned} [z^n]zT^k(z)T^\circ(z) &= \frac{1}{k}[z^n](T^\circ(z) - T(z)) = \frac{kn + 1}{k}[z^n]T(z) - \frac{1}{k}[z^n]T(z) \\ &= n[z^n]T(z) = [z^n]T^\bullet(z). \end{aligned}$$

Hence  $K^\bullet(z) = kz^kT(z) + z^kT^\bullet(z) = kz^kT(z) + z^{k+1}T^k(z)T^\circ(z) = K^\circ(z)$ .

### 4.2 Limiting Distribution

We now come to study the distances to the first vertex of the root in a tree of type  $\mathcal{K}^\circ$ , which are exactly the distances to the marked vertex of a tree of type  $\mathcal{K}^\bullet$ . This allows to show that the distance between two random points in a random  $k$ -tree follows a Rayleigh distribution.

With similar notations as in section 3 and a similar analysis of the recursive structure of the trees, but more involved computations, we obtain:

**Lemma 11.** *The b.g.f. of vertices at distance  $d$  in  $K^\circ$  can be expressed as*

$$K_1^\circ(z, u) = kz^k u^{k-1} T_{1,1}(z, u) + z^{k+1} u^k T_{1,1}^k(z, u) T^\circ(z),$$

$$K_d^\circ(z, u) = kz^k T_{d,1}(z, u) + z^{k+1} T_{d,1}^k(z, u) T_{d-1,k}^\circ(z, u).$$

Differentiating with respect to  $u$  gives

$$K_d^{\circ'}(z) = kz^k T^{k-1}(z) T'_{d,1}(z) + kz^{k+1} T^{k-1}(z) T'_{d,1}(z) T^\circ(z) + z^{k+1} T^k(z) T_{d-1,k}^{\circ'}(z).$$

In this function the dominant term is  $z^{k+1} T^k(z) T_{d-1,k}^{\circ'}(z)$  and has the same singular behavior as  $K_d^{\circ'}(z)$ , up to a factor  $n$ , corresponding to the choice of a random vertex.

This result agrees with the general fact that in a very large random tree the root tends to have the same properties as any random vertex. We thus get exactly the same asymptotic distribution, as in section 3.

**Theorem 5.** *Given a random  $k$ -tree  $G$  over  $n$  vertices, the distance between two random vertices  $v, w$  of  $G$  has mean value of order  $\sqrt{n}$  and is asymptotically Rayleigh distributed in the range  $x\sqrt{n}$ :*

$$\lim_{n \rightarrow \infty} \sqrt{n} \mathbb{P}(D(v, w) = \lfloor x\sqrt{n} \rfloor) = c_k^2 x e^{-\frac{(c_k x)^2}{2}}, \text{ with } c_k = k \sum_{i=1}^k \frac{1}{i}.$$

### References

1. Arnborg, S., Proskurowski, A.: Linear time algorithms for NP-hard problems restricted to partial  $k$ -trees. *Discrete Applied Mathematics* 23(1), 11–24 (1989)
2. Beineke, L.W., Pippert, R.E.: The number of labeled  $k$ -dimensional trees. *Journal of Combinatorial Theory* 6(2), 200–205 (1969)
3. Bodini, O., Darrasse, A., Soria, M.: Distances in random apollonian network structures. In: *FPSAC 2008. DMTCS Proceedings*, pp. 307–318 (2008)
4. Drmota, M.: *Random Trees: An Interplay between Combinatorics and Probability*. Springer, Heidelberg (2009)
5. Flajolet, P., Sedgewick, R.: *Analytic Combinatorics*. Cambridge University Press, Cambridge (2009)
6. Fowler, T., Gessel, I., Labelle, G., Leroux, P.: The specification of 2-trees. *Advances in Applied Mathematics* 28(2), 145–168 (2002)
7. Ibarra, L.: The clique-separator graph for chordal graphs and subclasses of chordal graphs. In: *Symposium on Discrete Mathematics, Nashville, TN* (2004)

8. Klawe, M.M., Corniel, D.G., Proskurowski, A.: Isomorphism testing in hookup classes. *SIAM Journal on Algebraic and Discrete Methods* 3(2), 260–274 (1982)
9. Labelle, G., Lamathe, C., Leroux, P.: Labelled and unlabelled enumeration of  $k$ -gonal 2-trees. *Journal of Combinatorial Theory, Series A* 106(2), 193–219 (2004)
10. Markenzon, L., Justel, C.M., Paciornik, N.: Subclasses of  $k$ -trees: Characterization and recognition. *Discrete Applied Mathematics* 154(5), 818–825 (2006)
11. Meir, A., Moon, J.W.: On the altitude of nodes in random trees. *Canadian Journal of Mathematics* 30, 997–1015 (1978)
12. Moon, J.W.: The number of labeled  $k$ -trees. *Journal of Combinatorial Theory* 6(2), 196–199 (1969)
13. Proskurowski, A.:  $K$ -trees: representation and distances. In: *Congressus Numerantium*, vol. 29, pp. 785–794. *Utilitas Mathematica* (1980)
14. Proskurowski, A.: Recursive graphs, recursive labelings and shortest paths. *SIAM Journal on Computing* 10(2), 391–397 (1981)
15. Rose, D.J.: On simple characterizations of  $k$ -trees. *Discrete Mathematics* 7(3-4), 317–322 (1974)

# Gray Code Compression<sup>\*</sup>

Darko Dimitrov<sup>1</sup>, Tomáš Dvořák<sup>2</sup>, Petr Gregor<sup>2</sup>, and Riste Škrekovski<sup>3</sup>

<sup>1</sup> Institut für Informatik, Freie Universität, Berlin, Germany

Darko.Dimitrov@inf.fu-berlin.de

<sup>2</sup> Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

{Tomas.Dvorak,Petr.Gregor}@mff.cuni.cz

<sup>3</sup> Department of Mathematics, University of Ljubljana, Ljubljana, Slovenia

**Abstract.** An  $n$ -bit (cyclic) Gray code is a (cyclic) sequence of all  $n$ -bit strings such that consecutive strings differ in a single bit. We describe an algorithm which for every positive integer  $n$  constructs an  $n$ -bit cyclic Gray code whose graph of transitions is the  $d$ -dimensional hypercube  $Q_d$  if  $n = 2^d$ , or a subgraph of  $Q_d$  if  $2^{d-1} < n < 2^d$ . This allows to compress sequences that follow this code so that only  $\Theta(\log \log n)$  bits per  $n$ -bit string are needed. The algorithm generates the transitional sequence of the code in a constant amortized time per one transition.

## 1 Introduction

An  $n$ -bit (cyclic) Gray code  $\mathbf{C}_n = (u_1, u_2, \dots, u_N)$  where  $N = 2^n$  is a (cyclic) sequence listing all  $n$ -bit strings, so that every two consecutive strings differ in exactly one bit. This corresponds to a Hamiltonian path (cycle) in the  $n$ -dimensional hypercube  $Q_n$ . A well-known example of such a code [3] is the reflected Gray code  $\mathbf{\Gamma}_n$  which may be defined recursively by

$$\mathbf{\Gamma}_1 = (0, 1), \quad \mathbf{\Gamma}_{n+1} = 0\mathbf{\Gamma}_n, 1\mathbf{\Gamma}_n^R \quad (1)$$

where  $b\mathbf{S}$  denotes the sequence  $\mathbf{S}$  with  $b \in \{0, 1\}$  prefixed to each string, and  $\mathbf{S}^R$  denotes the sequence  $\mathbf{S}$  in reverse order.

Gray codes are named after Frank Gray, who in 1953 patented the use of the reflected code  $\mathbf{\Gamma}_n$  for shaft encoders: a pattern representing the code, printed on a shaft, determines the angle of shaft rotation. Since then, considerable attention has been paid to the research on Gray codes satisfying certain additional properties, and applications have been found in such diverse areas as graphics and image processing, information retrieval or signal encoding [7]. Here we are particularly concerned with applications of Gray codes in the field of data compression [6, Section 4.2.1].

The *transitional sequence*  $\tau(\mathbf{C}_n) = [t_1, t_2, \dots, t_N]$  of a code  $\mathbf{C}_n$  lists the positions (called *transitions*)  $t_i \in [n] = \{1, 2, \dots, n\}$  for  $i \in [N]$  in which  $u_i$  and

---

<sup>\*</sup> Supported in part by the Czech-Slovenian bilateral grant MEB 080905, by the GAUK Grant 69408, by the Czech Science Foundation Grant 201/08/P298, and by AARS Research Program P-0297.

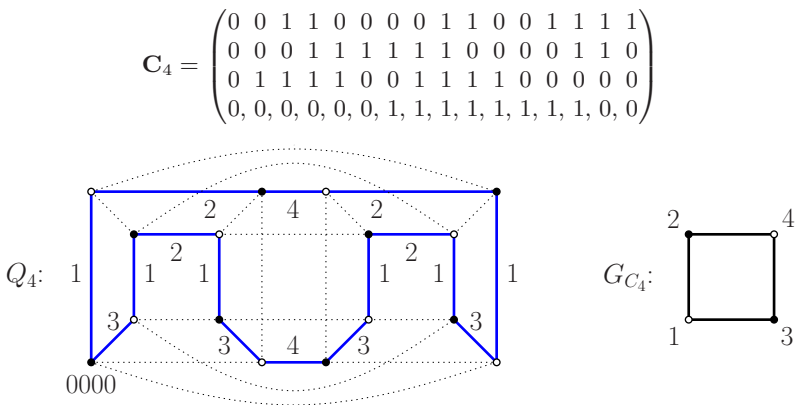
$u_{i+1}$  differ. For simplicity, the indices are always taken cyclically, thus  $u_{N+1}$  is identified with  $u_1$ . An (undirected) graph  $G_{C_n}$  induced by  $C_n$  (sometimes called the graph of transitions of  $C_n$ ) is defined by

$$V(G_{C_n}) = [n] \quad \text{and} \quad E(G_{C_n}) = \{t_i t_{i+1} \mid i \in [N]\}.$$

See Figure 1 for an illustration. Slater [8,9] and independently Bultena and Ruskey [1], motivated by applications of Gray codes, asked what graphs can be induced by (cyclic) Gray codes. For example, the star  $K_{1,n-1}$  is induced by the reflected Gray code  $\Gamma_n$  defined by (1).

The problem to characterize graphs which can be induced by (cyclic) Gray codes is still widely open. By computational search, Bultena and Ruskey [1] catalogued these graphs for  $n \leq 5$ , and Ernst and Wilmer [11] extended the list to  $n \leq 7$ . For general  $n$ , there are only some partial results, positive and negative.

Bultena and Ruskey [1] showed that every tree of diameter 4 can be induced by a cyclic Gray code. On the other hand, no tree of diameter 3 can be induced by such code. Also, they conjectured that all trees induced by cyclic Gray codes have diameter 2 or 4. This was disproved by Ernst and Wilmer [11] who introduced so called supercomposite Gray codes which induce trees of arbitrarily large diameter. Moreover, they answered two questions from [1] by showing that supercomposite Gray codes induce spanning trees of arbitrary 2-dimensional grids, and for a directed version of the problem, that there are cyclic Gray codes that induce digraphs with no bidirectional edge. Furthermore, Suparta and van Zanten [10] showed that the complete graph can also be induced by cyclic Gray codes, which solves a problem in [11]. Among many open problems posed in [1,8,9,10,11], it is particularly interesting whether paths and cycles can be induced by (cyclic) Gray codes.



**Fig. 1.** The cyclic Gray code  $C_4$ , the corresponding Hamiltonian cycle of  $Q_4$  and the graph  $G_{C_4}$  induced by the code  $C_4$ . The transitional sequence is  $\tau(C_4) = [3, 1, 2, 1, 3, 4, 3, 1, 2, 1, 3, 1, 2, 4, 2, 1]$ .



In this paper, for every positive integer  $n$  we construct an  $n$ -bit cyclic Gray code  $\mathbf{C}_n$  which induces the  $d$ -dimensional hypercube  $Q_d$  if  $n = 2^d$ , or a subgraph of  $Q_d$  if  $2^{d-1} < n < 2^d$ . More precisely, since the vertices of  $G_{\mathbf{C}_n}$  are labeled by the elements of  $[n]$ , we obtain the graph  $Q_d^*$  defined by

$$V(Q_d^*) = [2^d] \quad \text{and} \quad E(Q_d^*) = \{xy \mid \text{where } |x - y| = 2^i \text{ for some } 0 \leq i < d\}.$$

Clearly,  $Q_d^* \cong Q_d$  by the isomorphism that maps  $x \in [2^d]$  to the binary representation of  $x - 1$ .

We conclude the introduction with an explanation of the title of this paper. Note that every Gray code  $\mathbf{C}_n = (u_1, u_2, \dots, u_N)$  is uniquely determined by its first string  $u_1$  and the transitional sequence  $\tau(\mathbf{C}_n) = [t_1, t_2, \dots, t_N]$ . Since each transition is an integer from  $[n]$ , it may be encoded with  $d = \lceil \log_2 n \rceil$  bits. This provides a representation of  $\mathbf{C}_n$  with  $\Theta(\log n)$  bits per one  $n$ -bit string.

However, in case that  $\mathbf{C}_n$  induces a subgraph of  $Q_d^*$ , we may further explore the property that two consecutive transitions of  $\tau(\mathbf{C}_n)$  always form an edge of  $Q_d^*$ . Indeed, each transition  $t_{i+1}$ ,  $i \in [N - 1]$ , is then determined by the preceding transition  $t_i$  and by the edge  $t_i t_{i+1} \in E(Q_d^*)$ , which may be represented by its *direction*

$$d(t_i t_{i+1}) = j \text{ such that } |t_i - t_{i+1}| = 2^j.$$

Consequently, the code  $\mathbf{C}_n$  may be represented by the sequence

$$u_1, t_1, d(t_1 t_2), d(t_2 t_3), \dots, d(t_{N-1} t_N).$$

Since edges of  $Q_d^*$  occur only in  $d$  directions, each  $d(t_i t_{i+1})$  for  $i \in [N - 1]$  may be encoded with  $\lceil \log_2 d \rceil$  bits. Hence we obtain a representation of  $\mathbf{C}_n$  which requires only  $\Theta(\log \log n)$  bits on the average to represent one  $n$ -bit string of the code, which outperforms the  $\Theta(\log n)$  bits obtained above.

## 2 Preliminaries

For the rest of the paper, all Gray codes are cyclic. Let  $\mathbf{C}_n = (u_1, u_2, \dots, u_N)$  be a Gray code where  $n$  denotes the dimension of the code and  $N = 2^n$ , and let  $\tau(\mathbf{C}_n) = [t_1, t_2, \dots, t_N]$  be the transitional sequence of  $\mathbf{C}_n$ . We deal with  $\mathbf{C}_n$  as with a Hamiltonian cycle of the  $n$ -dimensional hypercube  $Q_n$ , which is the graph with  $V(Q_n) = \{0, 1\}^n$  and  $uv \in E(Q_n)$  if and only if  $u$  and  $v$  differ in exactly one coordinate. For a vertex  $v \in V(Q_n)$  let  $Q_n - v$  denote the graph obtained by removing  $v$  and all its incident edges from  $Q_n$ .

Let  $e_i$  denote the vertex of  $Q_n$  with 1 exactly in the  $i$ -th coordinate for  $i \in [n]$ . Thus  $u_i \oplus u_{i+1} = e_{t_i}$  for every  $i \in [N]$  where  $\oplus$  denotes the (coordinatewise) addition modulo 2. Moreover, let  $e_{ij} = e_i \oplus e_j$  for distinct  $i, j \in [n]$ . The elements of  $[n]$  are called *directions*.

Let  $\mathbf{C}_n^R = (u_N, \dots, u_2, u_1)$  denote the Gray code  $\mathbf{C}_n$  in reverse order. Similarly, for any path  $P = (v_1, v_2, \dots, v_m)$  of  $Q_n$ , let  $P^R = (v_m, \dots, v_2, v_1)$  denote the reverse of  $P$ . The notion of transitional sequences and induced graphs can be naturally extended to paths as follows. We define  $\tau(P) = [p_1, p_2, \dots, p_{m-1}]$

where  $p_i$  for  $i \in [m - 1]$  is the coordinate in which  $v_i$  and  $v_{i+1}$  differ, and the (undirected) graph  $G_P$  induced by  $P$  is

$$V(G_P) = [n] \quad \text{and} \quad E(G_P) = \{p_i p_{i+1} \mid i \in [m - 2]\}.$$

Note that for cycles, the transitional sequence is considered to be cyclic, whereas for paths it is not.

Let  $T = [t_1, t_2, \dots, t_m]$  be a (cyclic) transitional sequence of a path  $(u_1, u_2, \dots, u_{m+1})$  (resp. of a cycle  $(u_1, u_2, \dots, u_m)$ ). We say that  $T$  contains a *segment*  $S = [s_1, s_2, \dots, s_k]$  if there exists  $j \in [m - k]$  (resp.  $j \in [m]$ ) such that

$$s_i = t_{i+j-1} \text{ for all } i \in [k].$$

Furthermore, if  $k$  is even, we say that  $S$  is *centered at a vertex*  $u_{j+k/2}$ . For example,  $\tau(\mathbf{C}_4)$  on Figure 1 contains a segment  $[2, 1, 3, 1]$  centered at  $u_1 = 0000$ .

We say that a direction  $t$  is *repeating* in a transitional sequence  $T$ , if  $T$  contains a segment  $[t, x, t]$  for some  $x$ .

Let  $\pi : [n] \rightarrow [n]$  be a permutation and  $w = (w_1 w_2 \dots w_n) \in \{0, 1\}^n$  be a vector called *translation*. It is well known that the mapping  $\varrho : V(Q_n) \rightarrow V(Q_n)$  given by

$$\varrho(u_1 u_2 \dots u_n) = (v_1 v_2 \dots v_n) \text{ such that } v_i = u_{\pi(i)} \oplus w_i \text{ for every } i \in [n] \quad (2)$$

is an automorphism of  $Q_n$ . Moreover, for every automorphism  $\varrho$  of  $Q_n$  there exist unique  $\pi$  and  $w$  such that  $\varrho$  is given by (2). That is, every hypercube automorphism is composed of a unique permutation of coordinates and a unique translation. The translation determines where the vertex  $\mathbf{0} = (00 \dots 0)$  is mapped, i.e.  $\varrho(\mathbf{0}) = w$ .

The hypercube  $Q_n$  may be expressed as a Cartesian product  $Q_n = Q_k \square Q_{n-k}$  for  $1 \leq k < n$ . Every vertex  $v \in V(Q_n)$  is then represented as a pair  $v = (v_1, v_2)$  where  $v_1 \in V(Q_k)$  and  $v_2 \in V(Q_{n-k})$ . The subgraph of  $Q_n$  induced on vertices  $(v_1, v_2)$  for all  $v_1 \in V(Q_k)$  and fixed  $v_2 \in V(Q_{n-k})$  is called a *subcube* and denoted by  $Q_k(v_2)$ . Clearly,  $Q_k(v_2)$  is isomorphic to  $Q_k$ . Thus,  $Q_n$  may be viewed as  $Q_{n-k}$  in which every vertex  $v_2 \in V(Q_{n-k})$  corresponds to the subcube  $Q_k(v_2)$  and every edge  $v_2 v_3 \in E(Q_{n-k})$  corresponds to the collection of edges  $(v_1, v_2)(v_1, v_3)$  for all  $v_1 \in V(Q_k)$ .

In particular, the graph  $Q_{d+1}^*$  defined in the previous section can be decomposed into two subcubes denoted by  $Q_d^A$  and  $Q_d^B$  induced on the sets  $A = \{1, 2, \dots, n\}$  and  $B = \{n + 1, n + 2, \dots, 2n\}$ . Note that by the definition, every vertex  $i \in A$  of  $Q_d^A$  is joined in  $Q_{d+1}^*$  only with the vertex  $n + i \in B$  of  $Q_d^B$ .

Let  $G_{C_n}$  be the graph induced by the Gray code  $\mathbf{C}_n$ . A transition  $t_j$  where  $j \in [N]$  is *critical* for  $G_{C_n}$  if at least one of the edges  $t_{j-1} t_j, t_j t_{j+1} \in E(G_{C_n})$  is induced by no other pair of consecutive transitions in  $\tau(\mathbf{C}_n)$ , i.e.  $E(G_{C_n}) \neq \{t_i t_{i+1} \mid i \in [N] \setminus \{j - 1, j\}\}$ . If we view the cycle  $\mathbf{C}_n$  in  $Q_n$  as a path  $\mathbf{P}_n$ , then  $\tau(\mathbf{C}_n) = [\tau(\mathbf{P}_n), t_N]$ . Thus, if  $t_N$  is not critical for  $G_{C_n}$ , we obtain that  $G_{P_n} = G_{C_n}$ .

### 3 Inducing the Hypercube

In this section, we construct an  $n$ -bit Gray code  $\mathbf{C}_n$  for  $n = 2^d$  that induces the hypercube  $Q_d^*$ . The following lemma shows that under certain conditions, we may modify a Gray code so that the induced  $Q_d^*$  is preserved, and at the same time, a given segment of its transitional sequence is replaced by a new prescribed one.

**Lemma 3.1.** *Let  $\mathbf{C}$  be an  $n$ -bit Gray code with  $G_C = Q_d^*$ ,  $d > 1$ , such that  $\tau(\mathbf{C}_n)$  contains a segment  $[a, b, a, c]$  where  $a, b, c$  are pairwise distinct and  $n = 2^d$ . Let  $S$  be a segment  $[x, y, x, z]$  or  $[z, x, y, x]$  where  $x, y, z$  are pairwise distinct and  $xy, xz \in E(G_C)$ , and let  $v$  be a vertex of  $Q_n$ . Then, there exists a Gray code  $\mathbf{B}$  such that  $G_B = Q_d^*$ , each occurrence of  $[a, b, a, c]$  in  $\tau(\mathbf{C})$  is replaced by  $S$  in  $\tau(\mathbf{B})$ , and one of them is centered at the vertex  $v$ .*

*Proof.* We assume that  $S = [x, y, x, z]$ , otherwise we proceed with  $S^R$  and obtain  $\mathbf{B}^R$ , so by changing the direction we get  $\mathbf{B}$ . Assume that one occurrence of  $[a, b, a, c]$  in  $S$  is centered at a vertex  $u \in V(Q_n)$ . Since  $ab, ac \in E(G_C)$  and  $G_C = Q_d^*$ , we can extend the mapping  $\pi(a) = x, \pi(b) = y, \pi(c) = z$  to a permutation  $\pi : [n] \rightarrow [n]$  such that  $\pi$  is an automorphism of  $G_C$ . Consider the automorphism  $\rho$  of  $Q_n$  given by (2) with the permutation  $\pi$  and a translation vector  $w = (w_1 w_2 \cdots w_n) \in \{0, 1\}^n$  such that  $w_i = u_{\pi(i)} \oplus v_i$  for all  $i \in [n]$ .

It follows directly by (2) that  $\rho(u) = v$ , and furthermore,  $\rho$  maps the subsequence  $(u \oplus e_{ab}, u \oplus e_b, u, u \oplus e_a, u \oplus e_{ac})$  of the code  $\mathbf{C}$  to

$$\rho(u \oplus e_{ab}, u \oplus e_b, u, u \oplus e_a, u \oplus e_{ac}) = (v \oplus e_{xy}, v \oplus e_y, v, v \oplus e_x, v \oplus e_{xz}).$$

Hence, for the  $n$ -bit Gray code  $\mathbf{B} = \rho(\mathbf{C})$ , each occurrence of  $[a, b, a, c]$  in  $\tau(\mathbf{C})$  is replaced by  $S$  in  $\tau(\mathbf{B})$ , and one of them is centered at the vertex  $v$ . Moreover, for every  $p, q \in [n]$ ,

$$pq \in E(G_B) \text{ if and only if } \pi^{-1}(p)\pi^{-1}(q) \in E(G_C) \text{ if and only if } pq \in E(G_C).$$

The first equivalence holds by the definition of  $\rho$ , the latter holds since  $\pi$  is an automorphism of  $G_C$ . It follows that also  $\mathbf{B}$  induces  $G_C = Q_d^*$ . This establishes the lemma. □

Now we state one of our main results. Note that the last part of the following theorem (on repeating directions) is only needed in the next section for a general dimension  $n$ .

**Theorem 3.1.** *For every integer  $d \geq 1$ , there exists an  $n$ -bit cyclic Gray code  $\mathbf{C}_n$ ,  $n = 2^d$ , such that  $G_{C_n} = Q_d^*$ . Moreover, for  $d > 1$  and  $\tau(\mathbf{C}_n) = [T, t_N]$ , it holds that the transition  $t_N$  is not critical for  $G_{C_n}$ ,  $T$  contains two disjoint occurrences of some segment  $[a, b, a, c]$ , and every direction from  $[n - 1]$  is repeating in  $T$ .*

*Proof.* We argue by induction on  $d$ . For  $d = 1$  the statement is trivial. For  $d = 2$  consider the 4-bit Gray code  $\mathbf{C}_4$  given on Figure 1. Observe that  $G_{C_4} = Q_2^*$  and

for  $\tau(\mathbf{C}_4) = [T, t_N]$ , the transition  $t_N$  is not critical for  $G_{C_4}$ ,  $T$  contains two disjoint occurrences of the segment  $[1, 2, 1, 3]$ , and  $T$  contains segments  $[1, 2, 1]$ ,  $[2, 4, 2]$ , and  $[3, 4, 3]$ , so the directions 1, 2, and 3 are repeating in  $T$ .

Now we assume that the statement holds for  $d > 1$  and we prove it for  $d + 1$ . Recall that  $n = 2^d$  and  $N = 2^n$ .

The idea of the proof is as follows. We view  $Q_{2n}$  as a Cartesian product  $Q_{2n} = Q_n \square Q_n$ . First, we interconnect the copies  $(0^n, u)$  of the vertex  $0^n$  in all subcubes  $Q_n(u)$  for  $u \in V(Q_n)$  by a path  $P$  which induces  $Q_d^B$  on vertices  $B = \{n + 1, \dots, 2n\}$ . Then, in each subcube  $Q_n(u)$  we find a Hamiltonian path  $R(u)$  of  $Q_n(u) - (0^n, u)$  which induces  $Q_d^A$  on vertices  $A = \{1, \dots, n\}$ . Moreover, by Lemma B.1 we can choose the path  $R(u)$  so that  $R(u)$  joins prescribed neighbors of  $(0^n, u)$ , and its first and last edge are of prescribed directions. This assures that we can interconnect these paths together into a Hamiltonian cycle of  $Q_{2n}$ , and when we do so, the newly induced edges are only between  $i \in V(Q_d^A)$  and  $n + i \in V(Q_d^B)$ . See Figure 2 for an illustration. Note that the bold (green) paths  $R(u)$ 's are connected by dash-dotted (red) edges between the subcubes  $Q_n(u)$ 's, and the dashed (blue) path  $P$  is connected with  $R(u_1)$  and  $R(u_N)$  by the purple edges.

By the induction hypothesis, let  $\mathbf{C}_n = (u_1, u_2, \dots, u_N)$  be an  $n$ -bit Gray code such that  $G_{C_n} = Q_d^*$  and for  $\tau(\mathbf{C}_n) = [T, t_N]$ ,  $t_N$  is not critical for  $G_{C_n}$ ,  $T$  contains two disjoint occurrences of some segment  $S = [a, b, a, c]$ , one centered at a vertex  $u$ , and every direction from  $[n - 1]$  is repeating in  $T$ .

First, we interconnect the copies of the vertex  $0^n$  in each subcube  $Q_n(u_i)$  by a path

$$P = (0^n, u_1), (0^n, u_2), \dots, (0^n, u_N). \tag{3}$$

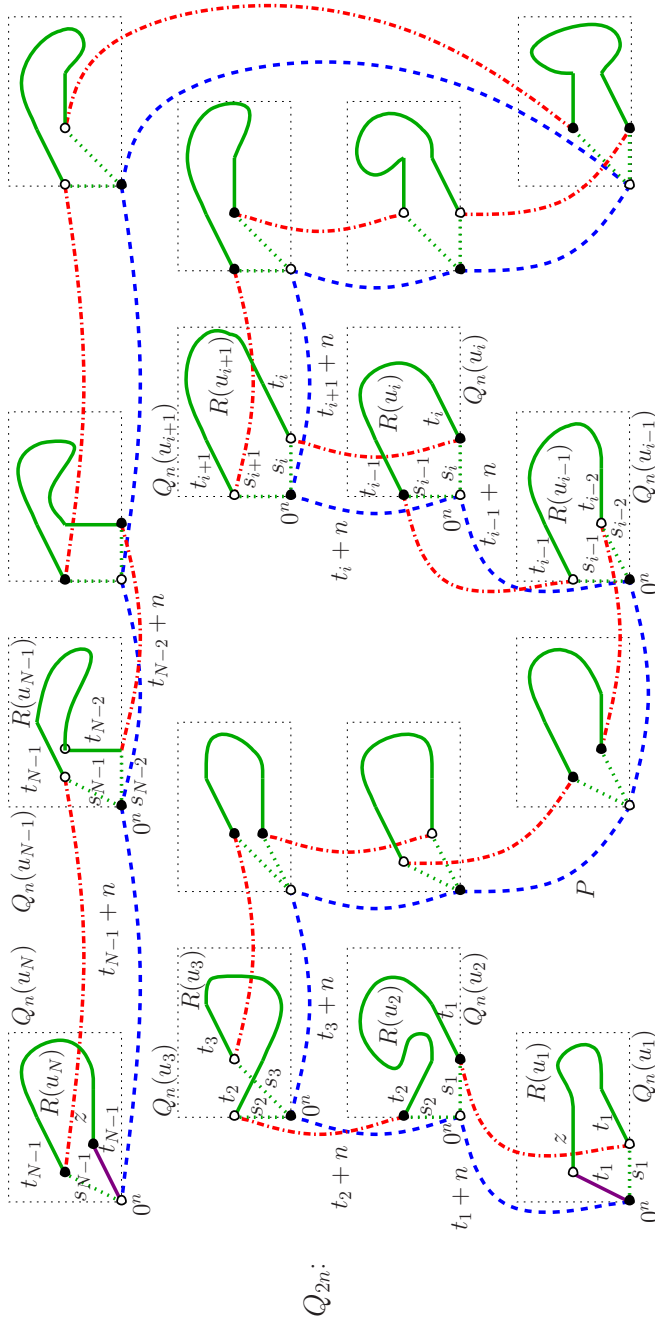
Since  $P$  will be a part of  $\mathbf{C}_{2n}$ ,  $T$  contains two disjoint occurrences of  $S = [a, b, a, c]$ , and every direction of  $[n - 1]$  is repeating in  $T$ , it follows that  $\tau(\mathbf{C}_{2n})$  will contain two disjoint occurrences of  $[a + n, b + n, a + n, c + n]$ , and every direction from  $\{n + 1, n + 2, \dots, 2n - 1\}$  will be repeating in  $\tau(\mathbf{C}_{2n})$ .

Second, we find a sequence which determines the endvertices of the paths  $R(u)$ 's, see Figure 2 for illustration. We claim that there is  $\sigma(\mathbf{C}_n) = [s_1, s_2, \dots, s_{N-1}]$  such that

- (a)  $t_i s_i \in E(G_{C_n})$  for every  $1 \leq i < N$ , and
- (b) precisely one of  $t_i = s_{i-1}$  and  $s_i = t_{i-1}$  holds for every  $1 < i < N$ .

Such a sequence can be found as follows. Note that  $\deg_{G_{C_n}}(t_i) = d \geq 2$  for every  $i \in [n]$ . For  $i = 1$ , we choose  $s_i$  arbitrarily such that  $t_i s_i \in E(G_{C_n})$ . Now assume  $1 < i < N$ . If  $t_i = s_{i-1}$ , then we choose  $s_i$  such that  $s_i \neq t_{i-1}$  and  $t_i s_i \in E(G_{C_n})$ . If  $t_i \neq s_{i-1}$ , then we put  $s_i = t_{i-1}$  and observe that  $t_i s_i \in E(G_{C_n})$  since  $t_{i-1} t_i \in E(G_{C_n})$ . Thus both (a) and (b) hold.

The sequence  $\sigma(\mathbf{C}_n)$  determines the endvertices of paths  $R(u_i)$  as described below. Note that from (a) and (b) we have that  $s_{i-1} s_i \in E(G_{C_n})$  for every  $1 < i < N$ . In each subcube  $Q_n(u_i)$  we find a Hamiltonian path  $R(u_i)$  of  $Q_n(u_i) - (0^n, u_i)$  as follows:



**Fig. 2.** The example for  $d = 2$  illustrating the construction of the code  $C_{2n}$  in  $Q_{2n}$ . For the transitional sequence  $\tau(C_4) = [3, 1, 2, 1, 3, 4, 3, 1, 2, 1, 3, 1, 2, 1, 3, 1, 2, 4, 2, 1, 3, 1, 2, 4, 2, 1, 2, 4, 2, 1, 2, 4, 3, 4]$ .

- (i) For  $i = 1$  we apply Lemma 3.1 for a vertex  $v = 0^n$  and a segment  $S = [z, t_1, s_1, t_1]$  where  $z \neq s_1$  such that  $t_1 z \in E(G_{C_n})$ . Let  $\mathbf{B}$  be the obtained Gray code containing  $S$  centered at  $v$ . By removing  $v$  from  $\mathbf{B}$  we get a Hamiltonian path  $R(u_1)$  of  $Q_n(u_1) - (0^n, u_1)$

$$R(u_1) = (e_{s_1}, u_1), (e_{t_1 s_1}, u_1), \dots, (e_{t_1 z}, u_1), (e_{t_1}, u_1). \tag{4}$$

- (ii) For  $1 < i < N$  we proceed similarly, but we apply Lemma 3.1 for  $v = 0^n$  and  $S = [t_{i-1}, s_{i-1}, s_i, t_i]$ . Note that by (a) and (b), the conditions of the lemma are satisfied. Again, let  $\mathbf{B}$  be the obtained Gray code containing  $S$  centered at  $v$ . By removing  $v$  from  $\mathbf{B}$  we get a Hamiltonian path  $R(u_i)$  of  $Q_n(u_i) - (0^n, u_i)$

$$R(u_i) = (e_{s_i}, u_i), (e_{s_i t_i}, u_i), \dots, (e_{s_{i-1} t_{i-1}}, u_i), (e_{s_{i-1}}, u_i). \tag{5}$$

- (iii) For  $i = N$  we apply Lemma 3.1 for  $v = 0^n$  and  $S = [t_{N-1}, s_{N-1}, t_{N-1}, z]$  where  $z \neq s_{N-1}$  and  $t_{N-1} z \in E(G_{C_n})$ . Similarly as above, we get a Hamiltonian path  $R(u_N)$  of  $Q_n(u_N) - (0^n, u_N)$

$$R(u_N) = (e_{t_{N-1}}, u_N), (e_{z t_{N-1}}, u_N), \dots, (e_{s_{N-1} t_{N-1}}, u_N), (e_{s_{N-1}}, u_N). \tag{6}$$

Observe that the following sequence is a  $2n$ -bit Gray code since the endvertices of consecutive subpaths given by (3)–(6) are adjacent:

$$\mathbf{C}_{2n} = P, R(u_N), R(u_{N-1}), \dots, R(u_2), R(u_1).$$

Next, we verify that  $\mathbf{C}_{2n}$  induces  $Q_{d+1}^*$ . We have

$$\begin{aligned} \tau(\mathbf{C}_{2n}) = & [\tau(P), t_{N-1}, \tau(R(u_N)), t_{N-1} + n, \tau(R(u_{N-1})), t_{N-2} + n, \\ & \dots, t_2 + n, \tau(R(u_2)), t_1 + n, \tau(R(u_1)), t_1]. \end{aligned}$$

Since  $t_N$  is not critical for  $G_{C_n}$ , we have by (3) that  $\tau(P)$  induces the subcube  $Q_d^B \cong Q_d^*$  of  $G_{C_{2n}}$  on vertices  $B = \{n + 1, n + 2, \dots, 2n\}$ . Furthermore, no other edge is induced between two vertices of  $B$  since  $\tau(\mathbf{C}_{2n})$  contains no two consecutive transitions from  $B$  other than those in  $\tau(P)$ .

Moreover, we show that  $\tau(R(u_i))$  for every  $i \in [N]$  induces the subcube  $Q_d^A \cong Q_d^*$  of  $G_{C_{2n}}$  on vertices  $A = \{1, 2, \dots, n\}$ . This follows from the fact that in each of the cases (i)–(iii) above,  $G_B = Q_d^*$  and  $\tau(\mathbf{B})$  contains two occurrences of the segment  $S$ . In addition, no other edge is induced between two vertices of  $A$  since  $\tau(\mathbf{C}_{2n})$  contains no two consecutive transitions from  $A$  other than those in  $\tau(R(u_i))$  for some  $i \in [N]$ .

Finally, observe by (3)–(6) that the remaining edges of  $G_{C_{2n}}$  are joining vertices  $i$  and  $n + i$  for some  $i \in [n]$ , and for every  $i \in [n]$  there exists such edge since  $\tau(\mathbf{C}_n)$  contains all  $i \in [n]$ . Altogether, we obtain that  $G_{C_{2n}} = Q_{d+1}^*$ .

To conclude the proof, it remains to verify the second part of the statement. Let  $\tau(\mathbf{C}_{2n}) = [t'_1, \dots, t'_{N2}] = [T', t'_{N2}]$ . Since  $t'_{N2-1} = z$ ,  $t'_{N2} = t_1$ , and  $t'_1 = t_1 + n$ , observe that the transition  $t'_{N2}$  is not critical for  $G_{C_{2n}}$ , because the edge

$zt_1 \in E(Q_d^B)$  is induced by  $\tau(R(u_i))$  for any  $i \in [n]$ , and the edge of  $G_{C_{2n}}$  joining  $t_1$  and  $t_1 + n$  is induced also by transitions  $t'_{N^2-N} = t_1$  and  $t'_{N^2-N+1} = t_1 + n$ .

Furthermore,  $T'$  contains  $\tau(P)$ . Consequently,  $T'$  contains two disjoint occurrences of a segment  $[a + n, b + n, a + n, c + n]$ , and every direction from  $\{n + 1, n + 2, \dots, 2n - 1\}$  is repeating in  $T'$ . In addition,  $T'$  contains the segments  $[t_1, t_1 + n, t_1], [t_2, t_2 + n, t_2], \dots, [t_{N-1}, t_{N-1} + n, t_{N-1}]$ . Hence, the directions  $D = \{t_1, \dots, t_{N-1}\}$  are repeating in  $T'$ . Clearly  $D = [n]$  since every direction from  $[n]$  appears at least twice in  $\tau(C_n) = [t_1, \dots, t_{N-1}, t_N]$ . Therefore, every direction from  $[2n - 1]$  is repeating in  $T'$ .  $\square$

### 4 General Dimension

In this section, we generalize Theorem 3.1 to an arbitrary dimension  $n$ . More precisely, we construct a Gray code inducing a subgraph of  $Q_d^*$  for the smallest  $d$  possible.

**Theorem 4.1.** *For every integer  $n \geq 1$ , there exists an  $n$ -bit cyclic Gray code  $C_n$  such that  $G_{C_n} \subseteq Q_{\lceil \log_2 n \rceil}^*$ . Moreover, if  $n \geq 4$  and  $n = 2^d + k$  where  $0 \leq k \leq 2^d - 2$ , then every direction from  $\{k + 1, \dots, 2^d - 1\}$  is repeating in  $\tau(C_n)$ .*

*Proof.* We argue by induction on  $k$ . By Theorem 3.1, the statement holds if  $n = 2^d$  for some integer  $d$ , i. e. for  $k = 0$ . If  $n = 1$  or  $n = 3$ , observe that the reflected codes  $\Gamma_1 = (0, 1)$  and  $\Gamma_3 = (000, 001, 011, 010, 110, 111, 101, 100)$  from (II) induce a subgraph of  $Q_0^*$  and  $Q_2^*$ , respectively.

Now we have  $n = 2^d + k \geq 5$  where  $d > 1$  and  $1 \leq k < 2^d$ , so  $\lceil \log_2 n \rceil = d + 1$ . By the induction hypothesis, there is an  $(n - 1)$ -bit Gray code  $C_{n-1}$  inducing a subgraph of  $Q_{d+1}^*$  such that every direction from  $D = \{k, \dots, 2^d - 1\}$  is repeating in  $\tau(C_{n-1})$ . That is, for every  $t \in D$  the transitional sequence  $\tau(C_{n-1}) = [t_1, \dots, t_{N/2}]$  where  $N = 2^n$  contains a segment  $[t, x, t]$  for some  $x \in [n - 1]$ . We may assume that

$$t_{N/2-1} = k, t_{N/2} = x, t_1 = k, \tag{7}$$

otherwise we shift the code  $C_{n-1}$  so that the segment  $[k, x, k]$  appears at this position.

We define the Gray code  $C_n$  schematically as in (III),

$$C_n = 0C_{n-1}, 1C_{n-1}^R. \tag{8}$$

From (7) and (8) it follows that

$$\tau(C_n) = [k = t_1, \dots, t_{N/2-1} = k, n, t_{N/2-1} = k, \dots, t_1 = k, n].$$

Hence, for the graph  $G_{C_n}$  induced by  $C_n$  we have that

$$E(G_{C_n}) \subseteq E(G_{C_{n-1}}) \cup \{kn\}.$$

Consequently,  $G_{C_n} \subseteq Q_{d+1}^*$  since  $G_{C_{n-1}} \subseteq Q_{d+1}^*$  and  $kn \in E(Q_{d+1}^*)$  because  $n - k = 2^d$ .

It remains to verify the second part of the statement. Observe that if  $S = [s, x, s]$  and  $T = [t, y, t]$  are segments of  $\tau(\mathbf{C}_{n-1})$  for some  $x, y \in [n - 1]$  and distinct repeating transitions  $s, t \in D$ , then  $S$  and  $T$  must be disjoint. Therefore, since every direction from  $D$  is repeating in  $\tau(\mathbf{C}_{n-1})$  and by (7), it follows that every direction from  $D \setminus \{k\}$  is repeating in  $[t_1, \dots, t_{N/2} - 1]$ , which is a segment of  $\tau(\mathbf{C}_n)$ .  $\square$

## 5 Concluding Remarks

In this paper we have described a construction of a cyclic  $n$ -bit Gray code whose graph of transitions is the  $d$ -dimensional hypercube  $Q_d$  if  $n = 2^d$ , or a subgraph of  $Q_d$  if  $2^{d-1} < n < 2^d$ .

Note that the proofs of Theorems 3.1 and 4.1 actually provide a description of an algorithm which, given a positive integer  $n$ , constructs a transitional sequence of an  $n$ -bit code with the desired property. Following the inductive construction described in both proofs, the running time  $T(n)$  of the algorithm may be expressed as

$$T(n) = \begin{cases} T(n/2) + O(2^n) & \text{if } n = 2^d \text{ and } d > 2, \\ T(n - 1) + O(2^n) & \text{if } 2^{d-1} < n < 2^d \text{ and } d > 2, \\ O(1) & \text{if } n \leq 4. \end{cases}$$

Consequently, the time complexity of our construction is bounded by  $O(N)$ , where  $N = 2^n$  is the output size, i. e. only constant amortized time is required per one element of the output sequence. However, it is well-known [3] that the reflected Gray code  $\Gamma_n$  may be generated *looplessly* in the sense that time to find the next transition is constant even in the worst case. Is there a loopless construction algorithm for a Gray code inducing a subgraph of  $Q_d$ ?

As mentioned in the introduction, our variant of Gray code allows for a more space-saving representation compared to Gray codes in general. This suggests that it may be reasonable to inspect other data compression applications where Gray codes are traditionally used.

In particular, consider the problem of compressing a sequence of  $n$ -bit strings which arises in context of compressing bitmap indices of large databases. There are several efficient methods developed for this purpose [12] whose compression rate may be improved by reordering the input sequence so that the sum of Hamming distances of consecutive strings is minimized [4]. Unfortunately, this problem is known to be NP-complete [2]. In the special case when the sequence contains all  $n$ -bit strings, the optimal solution is provided by an  $n$ -bit Gray code. This suggests a heuristics for this problem [4]: sort the strings in the order given by a Gray code. We suggest that it is conceivable to employ our variant of Gray code for that purpose.

Then, it would be necessary to devise an efficient algorithm for sorting the strings in the order given by our variant of Gray code. It is well-known that



sorting by the  $\Gamma_n$  code may be performed in  $O(mn)$  time [5]. Is such a time complexity achievable for sorting by a Gray code inducing a subgraph of  $Q_d$ ?

## References

1. Bultena, B., Ruskey, F.: Transition restricted Gray codes. *Electron. J. Combin.* 3, #R11 (1996)
2. Ernvall, J., Katajainen, J., Penttonen, M.: NP-completeness of the Hamming salesman problem *BIT* 25, 289–292 (1985)
3. Knuth, D.E.: *The Art of Computer Programming. Fascicle 2: Generating All Tuples and Permutations*, vol. 4. Addison-Wesley Professional, Reading (2005)
4. Pinar, A., Tao, T., Ferhatosmanoglu, H.: Compressing bitmap indices by data reorganization. In: *Proc. 21st Int. Conf. Data Engineering (ICDE 2005)*, pp. 310–321. IEEE Computer Society, Los Alamitos (2005)
5. Richards, D.: Data compression and Gray-code sorting. *Inform. Process. Lett.* 22, 201–205 (1986)
6. Salomon, D.: *Data Compression: The Complete Reference*, 4th edn. Springer, Berlin (2006)
7. Savage, C.: A survey of combinatorial Gray codes. *SIAM Rev.* 39, 605–629 (1997)
8. Slater, P.J.: Open problem. In: *Proc. 10th Southeastern Conf. on Combinatorics, Graph Theory, and Computing*, Congress. Utilitas Math., Winnipeg, vol. XXIV, pp. 918–919 (1979)
9. Slater, P.J.: Research Problems 109 and 110. *Discrete Math.* 76, 293–294 (1989)
10. Suparta, I.N., van Zanten, A.J.: A construction of Gray codes inducing complete graphs. *Discrete Math.* 308, 4124–4132 (2008)
11. Wilmer, E.L., Ernst, M.D.: Graphs induced by Gray codes. *Discrete Math.* 257, 585–598 (2002)
12. Wu, K., Otoo, E.J., Shoshani, A.: Optimizing bitmap indices with efficient compression. *ACM Trans. Database Syst.* 31, 1–38 (2006)

# Embedded Trees and the Support of the ISE\*

Michael Drmota

Institute of Discrete Mathematics and Geometry, TU Wien,  
Wiedner Hauptstr. 8–10, A-1040 Wien, Austria

[michael.drmota@tuwien.ac.at](mailto:michael.drmota@tuwien.ac.at)

<http://www.dmg.tuwien.ac.at/drmota/>

**Abstract.** Embedded trees are labelled rooted trees, where the root has zero label and where the labels of adjacent vertices differ by  $\pm 1$ . Recently it was proved by Chassaing and Schaeffer, and Janson and Marckert that the distribution of the maximum and minimum label are closely related to the support of the density of the integrated superbrownian excursion (ISE). The purpose of this paper is make this probabilistic limiting relation more explicit by using a generating function approach due to Bouttier, Di Francesco, and Guitter that is based on properties of Jacobi's  $\theta$ -functions. In particular we derive an integral representation of the joint distribution function of the supremum and infimum of the support of the ISE in terms of the Weierstrass  $\wp$ -function.

## 1 Introduction

A *planted plane tree* is a rooted ordered tree, which means that all successors of a node have a left-to-right order. It is a classical result that the number  $p_n$  of planted plane trees with  $n$  edges equals the Catalan number

$$p_n = \frac{1}{n+1} \binom{2n}{n}.$$

An *embedded tree* (with increments  $\pm 1$ ) is a planted plane tree, where the vertices are labelled by integers such that the root has label 0 and labels of adjacent vertices differ by  $\pm 1$  (see Figure [1](#)). By construction the number  $q$  of different embedded trees (with increments  $\pm 1$ ) is given by

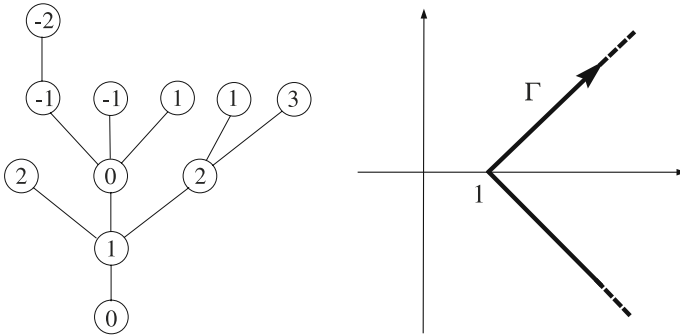
$$q_n = 2^n p_n = \frac{2^n}{n+1} \binom{2n}{n}.$$

In what follows we assume that every embedded tree (with  $n$  edges) is equally likely. Of course, in this random setting every parameter on embedded trees becomes a random variable.

Let  $X_n(j)$  denote the number of vertices with label  $j$  in a (random) embedded tree of size  $n$ . The sequence  $(X_n(j))_{j \in \mathbb{Z}}$  is then the *label profile*, and let  $X_n(t)$ ,

---

\* This research was supported by the Austrian Science Foundation FWF, Project S9604.



**Fig. 1.** Embedded tree and contour of integration  $\Gamma$

$t \in \mathbb{R}$ , be the the linearly interpolated (random) function. Recently, Bousquet-Mélou and Janson [3] proved that

$$\left(n^{-3/4}X_n(n^{1/4}t), -\infty < t < \infty\right) \xrightarrow{d} (f_{\text{ISE}}(t), -\infty < t < \infty), \quad (1)$$

where  $\xrightarrow{d}$  denotes weak convergence in the space  $C_0(\mathbb{R})$  and the stochastic process  $(f_{\text{ISE}}(t), -\infty < t < \infty)$  is the density of the integrated superbrownian excursion (ISE). Recall that the ISE is a random measure which can be seen – despite a scaling factor  $\sqrt{2}$  – as the occupation measure of the head of the Brownian snake (see Chassaing and Schaeffer [6], Janson and Marckert [9], and Bousquet-Mélou and Janson [3]).

One interesting feature of the ISE is that the support of its density  $[L_{\text{ISE}}, R_{\text{ISE}}]$  is (almost surely) a finite interval. By (I) it is clear that the largest label  $M_n$  and the smallest label  $m_n$  of a random embedded tree with  $n$  edges is related to  $R_{\text{ISE}}$  and  $L_{\text{ISE}}$ :

$$\frac{M_n}{n^{1/4}} \xrightarrow{d} R_{\text{ISE}} \quad \text{and} \quad \frac{m_n}{n^{1/4}} \xrightarrow{d} L_{\text{ISE}}.$$

We also have

$$\frac{M_n - m_n}{n^{1/4}} \xrightarrow{d} R_{\text{ISE}} - L_{\text{ISE}}.$$

Note that  $R_{\text{ISE}}$  and  $-L_{\text{ISE}}$  have the same distribution but they are not independent.

By using the relation between  $M_n$  and  $R_{\text{ISE}}$  and asymptotics of generating functions Bousquet-Mélou [2] proved a remarkable integral representation of the tail distribution function  $G(\lambda) = \mathbb{P}\{R_{\text{ISE}} > \lambda\}$ :

$$G(\lambda) = \frac{12}{i\sqrt{\pi}} \int_{\Gamma} \frac{v^5 e^{v^4}}{\sinh^2(\lambda v)} dv, \quad (2)$$

where

$$\Gamma = \{1 - te^{-i\pi/4}, t \in (-\infty, 0]\} \cup \{1 + te^{i\pi/4}, t \in [0, \infty)\}, \quad (3)$$

see Figure 1. In [7] one can also find a relation for the Laplace transform of the function  $x^{-\frac{3}{2}}\mathbb{P}\{R_{\text{ISE}} > x^{\frac{1}{4}}\}$  which is given by

$$\int_0^\infty x^{-\frac{3}{2}}\mathbb{P}\{R_{\text{ISE}} > x^{\frac{1}{4}}\} e^{-sx} dx = \frac{6\sqrt{\pi s}}{(\sinh((s/2)^{1/4}))^2}$$

and representations for the moments

$$\mathbb{E}(R_{\text{ISE}}^r) = \frac{24\sqrt{\pi}\Gamma(r+1)\zeta(r-1)}{2^r\Gamma((r-2)/4)},$$

for  $\Re(r) > -4$ , where the right hand side has to be analytically continued at the points  $-3, -2, -1, 2$ .

The purpose of this paper is to extend the result [2] by Bousquet-Mélou. We will provide integral representations for the joint distribution of  $L_{\text{ISE}}$  and  $R_{\text{ISE}}$  and also for the length  $R_{\text{ISE}} - L_{\text{ISE}}$  of the support of the ISE. In the proof we use an explicit representation of the corresponding generating function of embedded trees in terms of  $\theta$ -functions (see [5]) and use asymptotics of these generating functions, where Eisenstein series the Weierstrass  $\wp$ -function appear.

The structure of the paper is as follows. In Section 2 we give precise statements of our results. The proof is then divided into two major parts. First we discuss combinatorics on embedded trees (Section 3) and derive then the asymptotic results in Section 4.

## 2 Results

As above let  $M_n$  and  $m_n$  denote the maximum and minimum labels in embedded trees of size  $n$ , respectively. In order to formulate our main result we need the notion of the Weierstrass  $\wp$ -function

$$\wp(z; \tau) = \frac{1}{z^2} + \sum_{(m_1, m_2) \in \mathbb{Z}^2 \setminus \{(0,0)\}} \left( \frac{1}{(z - m_1\tau - m_2)^2} - \frac{1}{(m_1\tau + m_2)^2} \right),$$

where  $\tau$  and  $z$  are complex variables with  $\Im(\tau) > 0$  and  $z \notin \mathbb{Z} + \tau\mathbb{Z}$ . The  $\wp$ -function – considered as a function in  $z$  – is an elliptic function that has periods 1 and  $\tau$ . It is analytic in  $\tau$  and meromorphic in  $z$  with double poles on the lattice points  $\mathbb{Z} + \tau\mathbb{Z}$ ; for details we refer to [10].

**Theorem 1.** *The distribution function*

$$F(\lambda_1, \lambda_2) = \mathbb{P}\{R_{\text{ISE}} \leq \lambda_1, -L_{\text{ISE}} \leq \lambda_2\}$$

of the limit

$$\left( \frac{M_n}{n^{1/4}}, \frac{-m_n}{n^{1/4}} \right) \xrightarrow{d} (R_{\text{ISE}}, -L_{\text{ISE}})$$

is given by

$$F(\lambda_1, \lambda_2) = \frac{20}{3i\pi^{5/2}} \int_\Gamma \wp \left( -\frac{i\lambda_1}{\pi}v; -\frac{i(\lambda_1 + \lambda_2)}{\pi}v \right) v^5 e^{\frac{5}{9}v^4} dv.$$

Similarly we obtain an integral representation for the length of the support of the ISE.

**Theorem 2.** *The distribution function*

$$H(\lambda) = \mathbb{P}\{R_{\text{ISE}} - L_{\text{ISE}} \leq \lambda\}$$

of the limit

$$\frac{M_n - m_n}{n^{1/4}} \xrightarrow{d} R_{\text{ISE}} - L_{\text{ISE}}$$

is given by

$$H(\lambda) = -\frac{20}{3\pi^{7/3}} \int_0^\lambda \int_\Gamma \frac{\partial}{\partial \tau} \wp \left( -\frac{is}{\pi}v; -\frac{i\lambda}{\pi}v \right) v^6 e^{\frac{5}{9}v^4} dv ds.$$

There is almost no literature on explicit results on the support  $[L_{\text{ISE}}, R_{\text{ISE}}]$  of the ISE. Besides the aforementioned results on  $R_{\text{ISE}}$  the expected values

$$\mathbb{E}(-R_{\text{ISE}}L_{\text{ISE}}) = -3\sqrt{2\pi} + 2\sqrt{2\pi} \int_1^\infty \int_1^\infty \frac{(u+1)}{\sqrt{t^3-1}\sqrt{u^3-1}(u+\sqrt{u^2+u+1})} du dt$$

and

$$\mathbb{E}(\min\{R_{\text{ISE}}, -L_{\text{ISE}}\}) = 6\sqrt{2\pi} \left( 1 - \frac{1}{8} \left( \int_1^\infty \frac{du}{\sqrt{u^3-1}} \right)^2 \right)$$

have been computed by Delmas [7].

### 3 Combinatorics

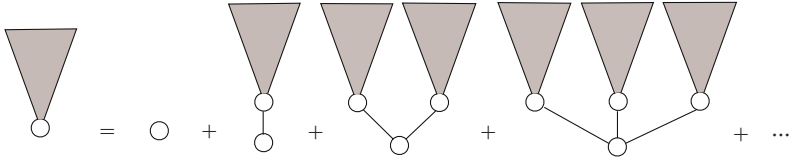
Let  $P(t)$  denote the generating function of planted plane trees, where the exponent of  $t$  counts the number of edges. Then by using the combinatorial decomposition – namely that all subtrees of the root are again planted plane trees, see Figure 2 – we obtain the relation

$$P(t) = 1 + tP(t) + t^2P(t)^2 + t^3P(t)^3 + \dots = \frac{1}{1 - tP(t)}$$

and consequently

$$P(t) = \frac{1 - \sqrt{1 - 4t}}{2t} = \sum_{n \geq 0} \frac{1}{n+1} \binom{2n}{n} t^n.$$

It is also very easy to count embedded trees without any restriction with the help of generating functions. Let  $R(t)$  denote the generating function of embedded trees, where the exponent of  $t$  counts the number of edges. Furthermore let  $R_n(t)$ ,  $n \in \mathbb{Z}$ , be the generating function of embedded trees, where we assume



**Fig. 2.** Recursion for planted plane trees

that the root is labelled by  $n$  (and labels of adjacent vertices differ by  $\pm 1$ ). Then by using the same decomposition as above we have

$$R_n(t) = \frac{1}{1 - t(R_{n-1}(t) + R_{n+1}(t))}. \tag{4}$$

Since there are no restrictions on the embedded trees we have  $R_n(t) = R_0(t) = R(t)$  for all  $n \in \mathbb{Z}$  leading to the relation

$$R(t) = \frac{1}{1 - 2tR(t)}$$

and to the explicit representation

$$R(t) = \frac{1 - \sqrt{1 - 8t}}{4t} = \sum_{n \geq 0} \frac{2^n}{n + 1} \binom{2n}{n} t^n.$$

The situation becomes more interesting if we just consider embedded trees, where all labels are non-negative. Let  $R_n^{[0]}(t)$  be the generating functions of those embedded trees, where the root has label  $n$ . By definition  $R_n^{[0]}(t) = 0$  if  $n < 0$ . However, we have the same recurrence relation as above:

$$R_n^{[0]}(t) = \frac{1}{1 - t(R_{n-1}^{[0]}(t) + R_{n+1}^{[0]}(t))}, \quad (n \geq 0). \tag{5}$$

Interestingly, this system of equations has an explicit solution of the form

$$R_n^{[0]}(t) = R(t) \frac{u_n u_{n+4}}{u_{n+1} u_{n+3}},$$

where

$$u_n = u_n(t) = Z(t)^{\frac{n+1}{2}} - Z(t)^{-\frac{n+1}{2}}$$

and

$$Z(t) = \frac{1 - (1 - 8t)^{1/4}}{1 + (1 - 8t)^{1/4}}$$

is the solution of the equation

$$Z + \frac{1}{Z} + 2 = \frac{1}{tR(t)}$$

that is analytic at  $t = 0$ . This miraculous relation was observed by Bouttier, Di Francesco, and Guitter [4]. In fact this explicit solution was used by Bousquet-Mélou [2] to obtain the integral representation for (2).

In another paper Bouttier, Di Francesco, and Guitter [5] considered the class of embedded trees, where all labels are bounded between 0 and  $L$ , where  $L$  is a non-negative integer. Let  $R_n^{[0,L]}(t)$  be the generating functions of those embedded trees, where the root has label  $n$ . By definition  $R_n^{[0,L]}(t) = 0$  if  $n < 0$  or  $n > L$ . We have the same recurrence relation as above:

$$R_n^{[0,L]}(t) = \frac{1}{1 - t(R_{n-1}^{[0,L]}(t) + R_{n+1}^{[0,L]}(t))}, \quad (0 \leq n \leq L). \tag{6}$$

Interestingly there is an explicit solution of this system of equation in terms of the Jacobi theta function

$$\theta_1(u; q) = 2i \sin(\pi u) \prod_{j \geq 1} (1 - 2q^j \cos(2\pi u) + q^{2j}). \tag{7}$$

First let  $q = q(t)$  be determined by the equation

$$t = \frac{\theta_1\left(\frac{1}{L+6}, q\right)^4 \theta_1\left(\frac{4}{L+6}, q\right)}{\theta_1\left(\frac{2}{L+6}, q\right)^5} \tag{8}$$

Then we have (see [5])

$$R_n^{[0,L]}(t) = \frac{\theta_1\left(\frac{2}{L+6}, q\right)^3}{\theta_1\left(\frac{1}{L+6}, q\right)^2 \theta_1\left(\frac{4}{L+6}, q\right)} \frac{\theta_1\left(\frac{n+1}{L+6}, q\right) \theta_1\left(\frac{n+5}{L+6}, q\right)}{\theta_1\left(\frac{n+2}{L+6}, q\right) \theta_1\left(\frac{n+4}{L+6}, q\right)}. \tag{9}$$

### 4 Asymptotic Analysis

In [5] the generating functions  $R_n^{[0,L]}(t)$  have been analyzed by considering so-called scaling limits which can be interpreted in terms of potentials and characteristic lengths etc. For our purpose we have to be more precise, since we are interested in asymptotics of the coefficients. Nevertheless, we use – more or less – the same of scaling as in [5].

By shifting labels from 0 to  $j$  it follows that

$$\mathbb{P}\{M_n \leq k, m_n \geq -j\} = \frac{[t^n] R_j^{[0,j+k]}(t)}{\frac{2^n}{n+1} \binom{2n}{n}}. \tag{10}$$

Thus, in order to prove Theorem 1 we need asymptotics of the coefficient  $[t^n] R_j^{[0,j+k]}(t)$ . Note that it is not necessary to prove asymptotics in the full range of parameters. In particular, we will set  $j \sim \lambda_1 n^{1/4}$  and  $k \sim \lambda_2 n^{1/4}$  for positive real numbers  $\lambda_1, \lambda_2$ .

We use Cauchy’s formula

$$[t^n] R_j^{[0,j+k]}(t) = \frac{1}{2\pi i} \int_{\gamma} R_j^{[0,j+k]}(t) t^{-n-1} dt,$$

where  $\gamma$  is a certain contour of winding number +1 around the origin, contained in the analyticity region of  $R_j(t)^{[0,j+k]}$ . In this case we will use a path of integration  $\gamma$  of the form  $\gamma = \gamma_1 \cup \gamma_2 \cup \gamma_3 \cup \gamma_4$ , where  $x_0 = \frac{1}{8}$ ,  $c > 0$ ,

$$\begin{aligned} \gamma_1 &= \left\{ x = x_0 \left( 1 - c \frac{i + n^{1/4} - s}{n} \right) : 0 \leq s \leq n^{1/4} \right\}, \\ \gamma_2 &= \left\{ x = x_0 \left( 1 - c \frac{1}{n} e^{-i\varphi} \right) : -\frac{\pi}{2} \leq \varphi \leq \frac{\pi}{2} \right\}, \\ \gamma_3 &= \left\{ x = x_0 \left( 1 + c \frac{i + s}{n} \right) : 0 \leq s \leq n^{1/4} \right\}, \end{aligned}$$

and  $\gamma_4$  is a circular arc centred at the origin and making  $\gamma$  a closed curve (see also Figure 3). Note that  $\gamma_1 \cup \gamma_2 \cup \gamma_3$  constitutes a so-called Hankel contour that appears in Hankel’s integral representation of  $1/\Gamma(s)$ .

By the relation (8),  $t$  and  $q$  are related. We will first study this relation for  $t \in \gamma_1 \cup \gamma_2 \cup \gamma_3$ . For this purpose we use the following asymptotic property.

**Lemma 1.** *Suppose that  $q = e^{2\pi i\tau}$  and  $L$  satisfy  $|1 - q| \geq c/L$  for some constant  $c > 0$ . Then*

$$\frac{\theta_1\left(\frac{1}{L+6}, q\right)^4 \theta_1\left(\frac{4}{L+6}, q\right)}{\theta_1\left(\frac{2}{L+6}, q\right)^5} = \frac{1}{8} \left( 1 - \frac{25}{(L+6)^4} G_4(\tau) + O\left(\frac{1}{L^6|1-q|^6}\right) \right), \quad (11)$$

where  $G_4(\tau)$  denotes the Eisenstein series

$$G_4(\tau) = \sum_{(m_1, m_2) \in \mathbb{Z}^2 \setminus \{(0,0)\}} \frac{1}{(m_1 + m_2\tau)^4}.$$

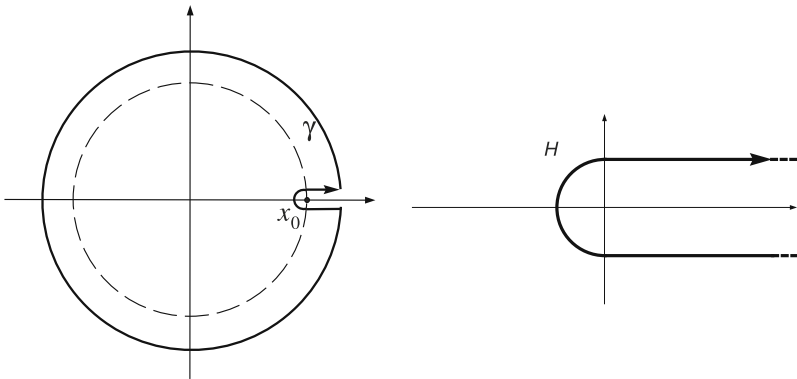


Fig. 3. Contour of integration and Hankel contour



*Proof.* By using the Taylor series expansions for  $\sin(\pi u)$  and  $\cos(2\pi u)$  we can represent  $\theta_1(u, q)$  as

$$\theta_1(u, q) = 2\pi i u \exp\left(\left(-\frac{\pi^2}{6} + 4\pi^2 \sum_{j \geq 1} \frac{q^j}{(1-q^j)^2}\right)u^2 + \left(-\frac{\pi^4}{180} - \frac{4\pi^4}{3} \sum_{j \geq 1} \frac{q^j}{(1-q^j)^2} - 8\pi^4 \sum_{j \geq 1} \frac{q^{2j}}{(1-q^j)^4}\right)u^4 + O\left(\frac{u^6}{|1-q|^6}\right)\right).$$

This gives

$$\frac{\theta_1\left(\frac{1}{L+6}, q\right)^4 \theta_1\left(\frac{4}{L+6}, q\right)}{\theta_1\left(\frac{2}{L+6}, q\right)^5} = \frac{1}{8} \left(1 - \frac{100\pi^4}{(L+6)^4} \left(\frac{1}{180} + \frac{4}{3}S_1 + 8S_2\right) + O\left(\frac{1}{L^6|1-q|^6}\right)\right),$$

where  $S_1$  and  $S_2$  abbreviate

$$S_1 = \sum_{j \geq 1} \frac{q^j}{(1-q^j)^2} \quad \text{and} \quad S_2 = \sum_{j \geq 1} \frac{q^{2j}}{(1-q^j)^4}$$

By using the notation  $\sigma_\ell(n) = \sum_{d|n} d^\ell$  we have

$$\frac{4}{3}S_1 + 8S_2 = \frac{4}{3} \sum_{j,k} k^3 q^{jk} = \frac{4}{3} \sum_{n \geq 1} \sigma_3(n) q^n.$$

Since

$$G_4(\tau) = \sum_{(m_1, m_2) \in \mathbb{Z}^2 \setminus \{(0,0)\}} \frac{1}{(m_1 + m_2\tau)^4} = \frac{\pi^4}{45} + \frac{16\pi^4}{3} \sum_{n \geq 1} \sigma_3(n) e^{2\pi i n \tau}.$$

it follows that

$$\frac{\pi^4}{180} + \frac{4}{3}\pi^4 S_1 + 8\pi^4 S_2 = \frac{1}{4}G_4(\tau). \tag{12}$$

This proves □.

Alternatively to the previous proof we could have used the relation

$$G_4(\tau) = \frac{5}{8} \left(\frac{\theta_1''(0, q)}{\theta_1'(0, q)}\right)^2 + \frac{3}{8} \frac{\theta_1^{(5)}(0, q)}{\theta_1'(0, q)},$$

where  $'$  denotes the derivative with respect to the first variable  $u$ .

Next we discuss the behaviour of  $G_4(\tau)$  if  $\tau$  is close to 0.

**Lemma 2.** *We have uniformly for  $\tau \rightarrow 0$  with  $\varepsilon \leq \arg(\tau) \leq \pi - \varepsilon$  (for any  $\varepsilon > 0$ )*

$$G_4(\tau) = \frac{\pi^4}{45} \tau^{-4} + O(\tau^{-3}). \tag{13}$$

*Proof.* We just study the behaviour of  $S_2$ , since it constitutes the asymptotic leading term in the representation (12). Set  $q = e^{-x}$ . Then by using the representation

$$S_2 = S_2(x) = \frac{1}{6} \sum_{n \geq 1} (\sigma_3(n) - \sigma_1(n)) e^{-nx}$$

we obtain that the Mellin transform of  $S_2(x)$  (see 8) is given by

$$F(s) = \int_0^\infty S_2(x) x^{s-1} dx = \frac{\Gamma(s)}{6} \zeta(s) (\zeta(s-3) - \zeta(s-1)).$$

for complex  $s$  with  $\Re(s) > 4$ . By taking the inverse Mellin transform (and shifting the line of integration to the left and taking into account the residue at  $s = 4$ ) one gets directly

$$S_2(x) = \zeta(4) x^{-4} + O(x^{-3}) = \frac{\pi^4}{90} x^{-4} + O(x^{-3})$$

which is uniform for  $x \rightarrow 0$  when  $|\arg(x)| \leq \pi - \varepsilon$  for any  $\varepsilon > 0$  (see again 8). Finally by setting  $\tau = -x/(2\pi i)$  and by using the relation  $G_4(\tau) \sim 32\pi^4 S_2$  we obtain (13).

We now assume that  $L \sim j + k = (\lambda_1 + \lambda_2)n^{1/4}$  for some positive constants  $\lambda_1$  and  $\lambda_2$ . Furthermore it is convenient to introduce a new variable

$$w = \frac{1}{2\pi} (\log(1/q))^{-1} = \frac{i}{\tau}.$$

Now suppose that  $t$  varies in  $\gamma_1 \cup \gamma_2 \cup \gamma_3$  (with  $x_0 = \frac{1}{8}$ ). If we write  $t = \frac{1}{8} \left(1 - \frac{w'}{n}\right)$  then  $w'$  varies in  $-H'$ , where  $H'$  is a Hankel contour cut at real part  $n^{1/4}$ . For simplicity we neglect this cut for a moment. With the help of the asymptotic relations of Lemmas 1 and 2 we have

$$w' = \frac{5\pi^4}{9(\lambda_1 + \lambda_2)^4} w^4 + O(w^2).$$

Hence  $w$  varies on a contour coming from  $+e^{i\pi/4}\infty$ , cutting the real axis at some positive value and leaving to  $+e^{-i\pi/4}\infty$  (compare with Figure 4). Hence, without loss of generality we can assume that  $w$  varies on  $\hat{\Gamma}$ , where  $\Gamma$  is defined in 3 and  $\hat{\cdot}$  denotes the time reversed contour.

The next goal is to determine the asymptotic behaviour of  $R_j^{[0, j+k]}(t)$  for  $t \in \gamma_1 \cup \gamma_2 \cup \gamma_3$ . For this purpose we will use the following property.

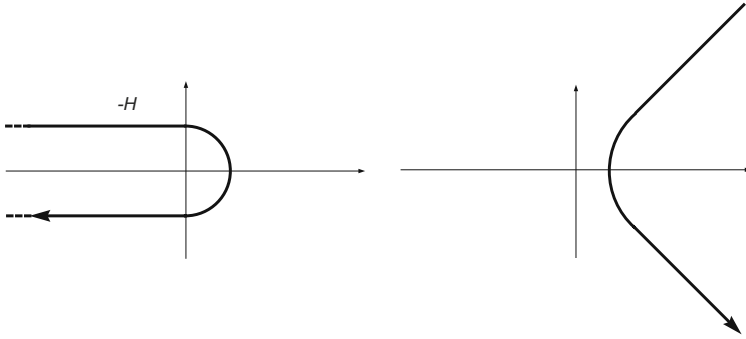


Fig. 4. Negative Hankel contour and contour of integration of  $w$

**Lemma 3.** *Suppose that  $q = e^{2\pi i\tau}$  and  $L$  satisfy  $|1 - q| \geq c/L$  for some constant  $c > 0$ . Then*

$$R_j^{[0,L]}(t) = 2 \left( 1 - \frac{3}{(L+6)^2} \wp \left( \frac{j+1}{L+6}; \tau \right) + O \left( \frac{1}{L^4 |1 - q|^4} \right) \right) \quad (14)$$

uniformly for  $\varepsilon \leq j/L \leq 1 - \varepsilon$  (for any  $\varepsilon > 0$ ), where  $\wp(z; \tau)$  denotes the Weierstrass  $\wp$ -function

$$\wp(z; \tau) = \frac{1}{z^2} + \sum_{(m_1, m_2) \in \mathbb{Z}^2 \setminus \{(0,0)\}} \left( \frac{1}{(z - m_1\tau - m_2)^2} - \frac{1}{(m_1\tau + m_2)^2} \right)$$

*Proof.* By proceeding as in the proof of Lemma [1](#) we obtain

$$\frac{\theta_1 \left( \frac{2}{L+6}, q \right)^3}{\theta_1 \left( \frac{1}{L+6}, q \right)^2 \theta_1 \left( \frac{4}{L+6}, q \right)} = 2 \left( 1 - \frac{1}{(L+6)^2} \frac{\theta_1'''(0, q)}{\theta_1'(0, q)} + O \left( \frac{1}{L^4 |1 - q|^4} \right) \right).$$

Furthermore we have for  $u = (j + 1)/(L + 6)$  (and uniformly for  $\varepsilon \leq u \leq 1 - \varepsilon$ )

$$\frac{\theta_1 \left( \frac{j+1}{L+6}, q \right) \theta_1 \left( \frac{j+5}{L+6}, q \right)}{\theta_1 \left( \frac{j+2}{L+6}, q \right) \theta_1 \left( \frac{j+4}{L+6}, q \right)} = 1 + \frac{3}{(L+6)^2} \left( \frac{\theta_1''(u, q)}{\theta_1(u, q)} - \left( \frac{\theta_1'(u, q)}{\theta_1(u, q)} \right)^2 \right) + O \left( \frac{1}{L^4 |1 - q|^4} \right)$$

Finally, by using the relation (see [1](#))

$$\frac{\theta_1'''(0, q)}{3\theta_1'(0, q)} - \frac{\theta_1''(u, q)}{\theta_1(u, q)} + \left( \frac{\theta_1'(u, q)}{\theta_1(u, q)} \right)^2 = \wp(u; \tau)$$

we obtain the asymptotic relation [\(14\)](#).

We are now ready to prove Theorem [1](#). We set  $j + 3 = \lambda_1 n^{1/4}$ ,  $k + 3 = \lambda_2 n^{1/4}$  and  $L + 6 = (j + 3) + (k + 3) = (\lambda_1 + \lambda_2) n^{1/4}$ . As mentioned above we use

Cauchy’s formula. For technical reasons we apply it to  $R_j^{[0,j+k]}(t) - 2$  instead of  $R_j^{[0,j+k]}(t)$ . Of course, if  $n > 0$  we have

$$\begin{aligned}
 [t^n] R_j^{[0,j+k]}(t) &= [t^n] \left( R_j^{[0,j+k]}(t) - 2 \right) = \frac{1}{2\pi i} \int_{\gamma} \left( R_j^{[0,j+k]}(t) - 2 \right) t^{-n-1} dt \\
 &= \frac{1}{2\pi i} \int_{\gamma_1 \cup \gamma_2 \cup \gamma_3} \left( R_j^{[0,j+k]}(t) - 2 \right) t^{-n-1} dt + \frac{1}{2\pi i} \int_{\gamma_4} \left( R_j^{[0,j+k]}(t) - 2 \right) t^{-n-1} dt
 \end{aligned}$$

We will focus on the contribution coming from the contour  $\gamma_1 \cup \gamma_2 \cup \gamma_3$ . Namely if  $t \in \gamma_4$  then  $|t| \sim \frac{1}{8} (1 + cn^{-3/4})$  (for some  $c > 0$ ) whereas  $R_j^{[0,j+k]}(t)$  stays bounded (note that Lemma 3 still applies). Hence

$$\int_{\gamma_4} \left( R_j^{[0,j+k]}(t) - 2 \right) t^{-n-1} dt = O \left( 8^n e^{-cn^{1/4}} \right)$$

which is negligible compared to the normalization  $\frac{2^n}{n+1} \binom{2n}{n} \sim 8^n n^{-3/2} / \sqrt{\pi}$ .

For  $t \in \gamma_1 \cup \gamma_2 \cup \gamma_3$  we use the substitution

$$t = \frac{1}{8} \left( 1 - \frac{5\pi^4 w^4}{9L^4} \right),$$

where  $w = i/\tau$  now varies on a contour that we can deform (due to analyticity) to  $\hat{\Gamma}$ . Note, however, that we have to cut  $\hat{\Gamma}$  to a finite contour  $\hat{\Gamma}'$ , since  $t \in \gamma_1 \cup \gamma_2 \cup \gamma_3$  implies that  $w = O(n^{1/16})$ . In this range we use the approximations

$$\begin{aligned}
 R_j^{[0,j+k]}(t) - 2 &= -\frac{6}{(\lambda_1 + \lambda_2)^2 \sqrt{n}} \wp \left( \frac{\lambda_1}{\lambda_1 + \lambda_2}; \frac{i}{w} \right) + O \left( \frac{w^4}{n} \right), \\
 t^{-n-1} &= 8^{n+1} \exp \left( \frac{5}{9} \frac{\pi^4}{(\lambda_1 + \lambda_2)^4} w^4 + O \left( \frac{w^6}{\sqrt{n}} \right) \right),
 \end{aligned}$$

and the substitution

$$dt = -\frac{20}{8 \cdot 9} \frac{\pi^4}{(\lambda_1 + \lambda_2)^4 n} w^3 dw$$

that lead to the integral

$$\begin{aligned}
 \frac{1}{2\pi i} \int_{\gamma_1 \cup \gamma_2 \cup \gamma_3} \left( R_j^{[0,j+k]}(t) - 2 \right) t^{-n-1} dt &= \frac{\pi^4}{2\pi i} \frac{8^n}{n^{3/2}} \frac{40}{3(\lambda_1 + \lambda_2)^6} \\
 &\times \int_{\hat{\Gamma}'} \left( \wp \left( \frac{\lambda_1}{\lambda_1 + \lambda_2}; \frac{i}{w} \right) + O \left( \frac{w^4 + w^6}{\sqrt{n}} \right) \right) \exp \left( \frac{5}{9} \frac{\pi^4}{(\lambda_1 + \lambda_2)^4} w^4 \right) w^3 dw
 \end{aligned}$$

At this point we can neglect the error terms and extend the cutted path of integration  $\hat{\Gamma}'$  to the infinite path  $\hat{\Gamma}$ . Furthermore, we substitute  $v = \pi w / (\lambda_1 + \lambda_2)$ , use the relation  $\wp(z; -1/\tau) = \tau^2 \wp(z\tau; \tau)$  and obtain (after reversing  $\hat{\Gamma}$  to  $\Gamma$  and deforming  $(\pi / (\lambda_1 + \lambda_2))\Gamma$  to  $\Gamma$ )

$$[t^n] R_j^{[0,j+k]}(t) \sim \frac{1}{\pi^3 i} \frac{8^n}{n^{3/2}} \frac{20}{3} \int_{\Gamma} \wp \left( -\frac{i\lambda_1}{\pi} v; -\frac{i(\lambda_1 + \lambda_2)}{\pi} v \right) v^5 e^{\frac{5}{9} v^4} dv.$$

Since  $\frac{2^n}{n+1} \binom{2n}{n} \sim 8^n n^{-3/2} / \sqrt{\pi}$  we finally derive the proposed result of Theorem [1](#).

Theorem [2](#) can be deduced from Theorem [1](#) and by applying the following property. Suppose that  $F(\lambda_1, \lambda_2)$  is the distribution function of a non-negative random vector  $(X_1, X_2)$  which has density  $\frac{\partial^2}{\partial \lambda_1 \partial \lambda_2} F(\lambda_1, \lambda_2)$ . Then the distribution function  $H(\lambda)$  of the sum  $Y = X_1 + X_2$  is given by

$$H(\lambda) = \int_0^\lambda \int_0^{\lambda-\lambda_1} \frac{\partial^2}{\partial \lambda_1 \partial \lambda_2} F(\lambda_1, \lambda_2) d\lambda_2 d\lambda_1 = \int_0^\lambda \frac{\partial}{\partial \lambda_1} F(\lambda_1, \lambda - \lambda_1) d\lambda_1$$

This directly implies

$$H(\lambda) = -\frac{20}{3\pi^{7/3}} \int_0^\lambda \int_\Gamma \left( \frac{\partial}{\partial \tau} \wp \left( -\frac{i\lambda_1}{\pi} v; -\frac{i\lambda}{\pi} v \right) + \frac{\partial}{\partial z} \wp \left( -\frac{i\lambda_1}{\pi} v; -\frac{i\lambda}{\pi} v \right) \right) v^6 e^{\frac{5}{9}v^4} dv d\lambda_1.$$

Finally, since  $\wp(\tau - z; \tau) = \wp(z; \tau)$  we have  $\frac{\partial}{\partial \tau} \wp(\tau - z; \tau) + \frac{\partial}{\partial z} \wp(\tau - z; \tau) = \frac{\partial}{\partial \tau} \wp(z; \tau)$  and consequently (by setting  $s = \lambda - \lambda_1$ ) we derive the proposed representation for  $H(\lambda)$  that is given in Theorem [2](#). Note that it is not possible to interchange the integrals, since the  $\wp$ -function is singular at  $z = 0$  and  $z = \tau$ .

*Acknowledgement.* The author is grateful to Mireille Bousquet-Mélou for several helpful comments.

## References

1. Batman, H.: Higher transcendental functions, vol. II. McGraw-Hill, New York (1953)
2. Bousquet-Mélou, M.: Limit laws for embedded trees: applications to the integrated super-Brownian excursion. *Random Structures Algorithms* 29, 475–523 (2006)
3. Bousquet-Mélou, M., Janson, S.: The density of the ISE and local limit laws for embedded trees. *Ann. Appl. Probab.* 16, 1597–1632 (2006)
4. Bouttier, J., Di Francesco, P., Guitter, E.: Geodesic distance in planar graphs. *Nucl. Phys., B* 663, 535–567 (2003)
5. Bouttier, J., Di Francesco, P., Guitter, E.: Random trees between two walls: exact partition function. *J. Phys. A: Math. Gen.* 36, 12349–12366 (2003)
6. Chassaing, P., Schaeffer, G.: Random planar lattices and integrated superBrownian excursion. *Probab. Theory Relat. Fields* 128, 161–212 (2004)
7. Delmas, J.-F.: Computation of moments for the length of the one dimensional ISE support. *Electron. J. Probab.* 8, 15, Paper no. 17 (2003)
8. Flajolet, P., Gourdon, X., Dumas, P.: Mellin transforms and asymptotics: harmonic sums. *Theoretical Computer Science* 144, 3–58 (1995)
9. Janson, S., Marckert, J.-F.: Convergence of discrete snakes. *J. Theor. Probab.* 18, 615–645 (2005)
10. Koecher, M., Krieg, A.: *Elliptische Funktionen und Modulformen*. Springer, Berlin (1998)

# Combinatorial Models for Cooperation Networks<sup>\*</sup>

Michael Drmota, Bernhard Gittenberger, and Reinhard Kutzelnigg

Institute of Discrete Mathematics and Geometry, TU Wien,  
Wiedner Hauptstr. 8–10, A-1040 Wien, Austria

michael.drmota@tuwien.ac.at,  
bernhard.gittenberger@tuwien.ac.at,  
reinhard.kutzelnigg@tuwien.ac.at  
<http://www.dmg.tuwien.ac.at/>

**Abstract.** We analyze special random network models – so-called *thickened trees* – which are constructed by random trees where the nodes are replaced by local clusters. These objects serve as models for random real world networks. It is shown that under a symmetry condition for the cluster sets a local-global principle for the degree distribution holds: the degrees given locally through the choice of the cluster sets directly affect the global degree distribution of the network. Furthermore, we show a superposition property when using clusters with different properties while building a thickened tree.

## 1 Introduction

There has been substantial interest in random graph models where vertices are added to the graph successively and are connected to several already existing nodes according to some given law. The so-called Albert-Barabási model (see [1]) joins a new node to an existing one with probability proportional to the degree. This is called preferential attachment and the motivation for introducing such schemes is to model various *real-word graphs* like the Internet or social networks.

It turns out that the preferential attachment rule of the Albert-Barabási model is not defined in an unambiguous way. One rigorous approach is due to [3]. They introduced a random multigraph which is built of random forests which are then formed into multigraphs by partitioning the vertex set and identifying vertices in the same block of the partition. It was shown in [4] that the graphs are *scale free*, that is, the degree distribution of these graphs satisfies asymptotically a *power law*. In particular, the fraction of vertices with degree  $d$  is asymptotically proportional to  $d^{-3}$ . Furthermore, the model fulfills the preferential attachment rule given by Albert and Barabási.

---

<sup>\*</sup> This research was supported by EU FP6-NEST-Adventure Programme, Contract n° 028875 (NEMO) and by the Austrian Science Foundation FWF, Project S9604.

## 2 Thickened Trees

The concept of thickened trees was introduced in [5]. The starting point was to construct a model of scale free graphs which is locally clustered, but the global structure is tree-like. This fits with observations from real world networks. In particular the design of this model was motivated to describe cooperation networks, where one usually has small groups with a strong interaction and some connections to other groups. Of course, there might be circles in a cooperation network but they are usually rare so that we neglect them.

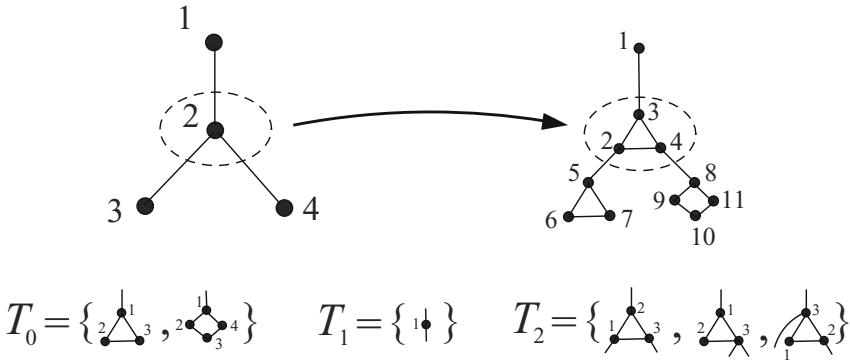
The idea how to get such a graph is to start with a scale free tree and then “thicken” the tree by substituting the nodes by clusters. Then the initial tree causes the global tree-like structure while the inserted clusters cause the local, highly clustered structure. However, the clusters are not produced by an evolution process. Nevertheless we think that our model has several advantages and can be used to explain several properties that are observed in practice:

- There is large flexibility in choosing the structure of local clusters and, thus, it can be adapted to the situation.
- The model is feasible for an analytic treatment.
- It can be used to study (analytically) the influence of local changes of the network to the global behavior.

In the following, we present a brief explanation how the model is constructed: We first introduce an evolution process that leads to a *plane oriented recursive tree* (PORT). Recall that a recursive tree is a labeled rooted tree, where the labels of all paths leading away from the root are strictly increasing. Furthermore, a rooted tree is plane oriented if the successors of every node have a left-to-right order. The process starts with the root that is labeled with 1. Next, we attach a node labeled with 2 to the root. We continue attaching the node with label 3, however there are three possibilities, two possibilities to attach at the root, and one to attach at the other vertex. Thus, with probability  $2/3$ , we attach to the root. We proceed in a similar way. Inductively at step  $j$  a new node with label  $j$  is attached to any previous node of out-degree  $k$  with probability proportional to  $k + 1$ . Thus we are in the framework of the Albert-Barabási model. Note that if a node  $v$  has out-degree  $k$ , then there are exactly  $k + 1$  ways of attaching a new node to  $v$ . Each of this  $k + 1$  options is leading to a different plane tree, since the left-to-right order of child nodes is significant and corresponds to a different embedding into the plane.

Observe that the evolution process described above generates each PORT of given size with the same probability. Thus, it is possible to analyze them by a usual counting procedure, in particular by a generating function approach. Because of the recursive structure, it is easy to see that the generating function  $y(z) = \sum_{n \geq 1} y_n \frac{z^n}{n!}$ , where  $y_n$  denotes number of PORTs of size  $n$ , satisfies the differential equation

$$y'(z) = 1 + y(z) + y(z)^2 + \dots = \frac{1}{1 - y(z)}.$$



**Fig. 1.** A simple example of a thickened PORT. The original tree has only nodes of out-degree 0,1, or 2. So the choice of all sets  $\mathcal{T}_k$  with  $k > 2$  is not relevant for the thickening process. For example, the node with label 2 is cut along the circular dashed line. Since it has out-degree 2, we choose one of the three graphs in  $\mathcal{T}_2$  (here the first one was chosen) and glue it into the corresponding space. Applying the same procedure to all nodes and relabeling afterwards yields the graph on the right hand side, a thickened PORT.

More generally, using the variable  $x_k$  to mark a node of out-degree  $k$ , we get the generating function  $y(z, x_0, x_1, x_2, \dots)$  fulfilling

$$y'(z, x_0, x_1, x_2, \dots) = \sum_{k \geq 0} x_k y(z, x_0, x_1, x_2, \dots)^k,$$

where  $'$  denotes differentiation with respect to  $z$ . Following this approach, it is for instance straightforward to give the probability generating functions of nodes of out-degree  $d$  in a random PORT of size  $n$ , see [5]. Thus it can be shown that PORTs are scale free trees, that is, the degree distribution follows asymptotically a power law. More precisely, the degree distribution is given by

$$\lim_{n \rightarrow \infty} p_n(d) = \frac{4}{d(d+1)(d+2)} \sim \frac{4}{d^3},$$

where  $p_n(d)$  denotes the probability that a random node in random PORT of size  $n$  has degree  $d$  (see [8] or [9]).

Now, we introduce a substitution process creating random graphs that have a global tree structure that is governed by plane oriented recursive trees. For every  $k \geq 0$  let  $\mathcal{T}_k$  denote a non-empty set of labeled graphs with *half edges* attached to their nodes in such a way that each graph receives in total  $k + 1$  half edges. The half edges are also ordered from 0 to  $k$ . Now consider the following random process. Take a tree  $T$  according to the PORT-model. Then we substitute every node  $v$  of out-degree  $k$  in the following way: cut  $v$  and one half of each edge incident with  $v$ . Then take a randomly chosen graph  $G$  of  $\mathcal{T}_k$  and *glue* the  $k + 1$  half edges of  $G$  to those left in  $T$  by the cutting of  $v$  respecting the given order,



that is, the half edge coming from the predecessor of  $v$  is glued to the 0th half edge<sup>1</sup> of  $G$  and the 1st, 2nd, ...,  $k$ th successor of  $v$  is attached to the 1st, 2nd, ...,  $k$ th half edge of  $G$ , respectively. Further we relabel all nodes in the new graph  $G = G(T)$  in a way that is consistent with the original labeling, i.e. the order relations among the labels are preserved, see Fig. 1. We denote the graphs that are obtained by this process *thickened trees* or more precisely *thickened PORTs*.

### 3 Generating Functions for Thickened PORTs and a Local-Global-Principle

Consider the formal solution  $y = y(z, x_0, x_1, x_2, \dots)$  of the differential equation

$$y' = \sum_{k \geq 0} x_k y^k,$$

where  $'$  denotes differentiation with respect to  $z$ . Then  $y = y(z, x_0, x_1, x_2, \dots)$  can be considered as a power series in  $z, x_0, x_1, \dots$ . By construction the coefficient

$$[z^n x_0^{k_0} x_1^{k_1} \dots] y(z, x_0, x_1, x_2, \dots)$$

is exactly the number of PORTs  $T$  of size  $n$  and with  $k_j$  nodes of out-degree  $j$  ( $j \geq 0$ ).

For every  $k \geq 0$  let  $\mathcal{T}_k$  denote a non-empty set of labeled graphs with  $k + 1$  additional *half edges*  $\tilde{e}_0, \tilde{e}_1, \dots, \tilde{e}_k$ . Furthermore, let

$$t_k(z) = \sum_{G \in \mathcal{T}_k} \frac{z^{|G|}}{|G|!}$$

denote the exponential generating function of these graphs.

The generating function of the numbers  $g_n$  of thickened trees with  $n$  vertices,  $g(z) = \sum_{n \geq 1} g_n \frac{z^n}{n!}$ , is then

$$g(z) = y(z, t_0(z)/z, t_1(z)/z, \dots).$$

We are interested in the number  $N_d(G)$  of nodes of degree  $d$  in a graph  $G$ . Therefore we consider the bivariate generating function

$$t_k^{(d)}(z, u) = \sum_{G \in \mathcal{T}_k} \frac{z^{|G|}}{|G|!} u^{N_d(G)},$$

Here the the half-edges  $\tilde{e}_0, \dots, \tilde{e}_k$  contribute to the node degrees as well. Then the generating function

$$g(z, u) = y(z, t_0^{(d)}(z, u)/z, t_1^{(d)}(z, u)/z, \dots)$$

encodes the distribution of nodes of degree  $d$  of thickened trees.

---

<sup>1</sup> If we substitute the root node, we disregard this 0th half edge, since there exists no predecessor.

Set

$$T_d(z, y, u) = \frac{1}{z} \sum_{k \geq 0} t_k^{(d)}(z, u) y^k.$$

Then the following results were shown in [5]:

**Lemma 1.** *Set*

$$G_d(z, y, u) = \int_0^y \frac{dt}{T_d(z, t, u)}.$$

Then  $g(z, u)$  satisfies the functional equation

$$G_d(z, g, u) = z.$$

**Theorem 1.** *Let  $\mathcal{T}_k$  be substitution sets (as described above) such that the equation*

$$X = \int_0^1 \frac{dt}{T_d(X, t, 1)}$$

has a unique positive solution  $X = \rho$  in the region of convergence of  $T_d(z, y, u)$  and that  $T_d(z, y, u)$  can be represented as

$$T_d(z, y, u) = \frac{C_0(z, y) + C_1(z, y)(1 - y)^{r'} y^{d+\alpha}(u - 1) + \mathcal{O}\left((1 - y)^{r'}(u - 1)^2\right)}{(1 - y)^r}, \tag{1}$$

where  $r'$  and  $r$  are real numbers with  $0 < r' \leq r$ ,  $\alpha$  is an integer,  $C_0(z, y)$  and  $C_1(z, y)$  are power series that contain  $z = \rho$  and  $y = 1$  in their regions of convergence and that satisfy  $C_i(\rho, y) \neq 0$  for  $i = 0, 1$  and  $0 \leq y \leq 1$ . Moreover, assume that the  $\mathcal{O}(\cdot)$ -term is uniform in a neighborhood of  $z = \rho$  and  $y = 1$ .

Let  $p_n(d)$  denote the probability that a random node in a thickened PORT of size  $n$  has degree  $d$ . Then the limits

$$\lim_{n \rightarrow \infty} p_n(d) =: p(d)$$

exist and we have, as  $d \rightarrow \infty$ ,

$$p(d) \sim \frac{C}{d^{r+r'+1}}.$$

Furthermore, for every  $d \geq 0$  let  $X_n^{(d)}$  denote the number of nodes of degree  $d$  in a random thickened PORT of size  $n$ . Then  $X_n^{(d)}$  satisfies a central limit theorem

$$\frac{X_n^{(d)} - \mathbb{E} X_n^{(d)}}{\sqrt{\mathbb{V} X_n^{(d)}}} \xrightarrow{d} N(0, 1),$$

where  $\mathbb{E} X_n^{(d)}$  and  $\mathbb{V} X_n^{(d)}$  are both asymptotically proportional to  $n$ .

The above theorem shows that thickened trees, where the generating function  $T_d(z, y, u)$  has the form (II), are scale free. Furthermore, the tail of the degree distribution has the order  $r + r' + 1$ . The parameters  $r$  and  $r'$  just depend on the structure of the cluster sets. Thus, a local change of the model, that is, a modification of the cluster sets, changes the global degree distribution.

In what follows we will show that the (quite technical) condition (III) is satisfied in very general situations. Furthermore we focus on the question how the structure of the cluster sets influences the parameters  $r$  and  $r'$ . In fact, we will formulate a proper *local-global-principle* for the order of the degree distribution.

We start with the following

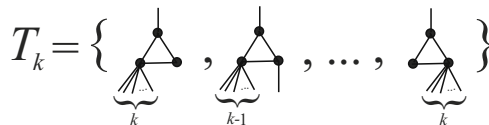
**Definition 1.** *Let  $\mathcal{M}$  be a set of graphs, where some vertices are marked and some or all of the marked vertices have (outgoing) half edges attached to them. Additionally we assume that one vertex is attached to a distinguished (ingoing) half edge. We say that  $\mathcal{M}$  satisfies the symmetry condition if the following property holds. Suppose that  $G \in \mathcal{M}$  has  $k'$  marked vertices and  $k$  (outgoing) half edges. Then every graph of that kind, where  $k$  (outgoing) half edges are attached to these  $k'$  marked vertices in an arbitrary way is also contained in  $\mathcal{M}$ .*

Note that the above definition assumes marked nodes in the graphs whereas the graphs in the cluster sets  $\mathcal{T}_k$  do not have marked nodes. In order to apply Definition I we mark nodes in the graphs of  $\mathcal{T}_k$  according to the following scheme. First, partition  $\mathcal{T}_k$  into isomorphism classes. Now consider one particular isomorphism class and mark in each graph all those nodes, for which another graph of the class exists which has an outgoing half edge attached to the corresponding node.

*Example 1.* In [5] thickened trees with cluster set as shown in Figure 2 where studied. Here the sets  $\mathcal{T}_k$  satisfy the symmetry condition. Since each  $\mathcal{T}_k$  with  $k \geq 2$  contains a triangular graph where both end vertices of the bottom are incident to a half edge, the two bottom vertices have to be marked in every graph of  $\mathcal{T}_k$ . The symmetry condition requires now, that all triangular graphs where the half edges are attached to the two bottom vertices (not necessarily involving both of them) are elements of  $\mathcal{T}_k$ , which is indeed true.

We will first show that the condition (III) is satisfied for a very special cluster set.

**Theorem 2.** *Consider a family of thickened trees such that all cluster sets  $\mathcal{T}_k$ ,  $k \geq k_0$  sufficiently large, contain only isomorphic copies of one graph (the same*



**Fig. 2.** Cluster sets containing all triangular graphs where the outgoing half edges are separated from the ingoing half edge

for all  $k \geq k_0$ ) of size  $m$  and  $k'$  marked vertices and satisfy the symmetry condition. Then we have

$$T_d(z, y, u) = \frac{z^{m-1}}{m!} \frac{1 + C(y)(1-y)(u-1) + \mathcal{O}((1-y)(u-1)^2)}{(1-y)^{k'}}, \quad (2)$$

where

$$C(y) = c_{k'-1} + c_{k'-2}(1-y) + \dots + c_0(1-y)^{k'-1}$$

is a polynomial of degree  $k' - 1$  and in particular  $c_{k'-1} \neq 0$ .

**Corollary 1.** *Suppose that the conditions of Theorem 2 are satisfied. Then (by applying Theorem 1) we obtain for the resulting thickened PORT family (with  $r = k'$ ,  $r' = 1$ )*

$$p(d) = \lim_{n \rightarrow \infty} p_n(d) \sim \frac{C}{d^{k'+2}}.$$

This result means that a tail behavior of the form  $C d^{-3}$  for usual PORTs is changed into a behavior of this form. The difference in the exponent equals  $k' - 1$  which can be seen as the *additional degree of freedom* we have when we choose the  $k$  half edges among the  $k'$  marked vertices.

Overall, this means that (under the symmetry condition or by assuming (II)) the local structure of the clusters determine in a relatively simple way the global behavior of the degree distribution. This can be seen as a *local-global-principle*.

*Proof.* For  $k \geq k_0$  there are  $k'$  marked vertices for attaching a half edge to the graph in  $\mathcal{T}_k$  and by the symmetry condition every distribution of the half edges among those  $k'$  places must lead to a graph in  $\mathcal{T}_k$ . Hence we obtain  $\binom{k+k'-1}{k'-1}$  possible configurations. Since this implies that

$$t_k^{(d)}(z, 1) = \binom{k+k'-1}{k'-1} \frac{z^m}{m!}.$$

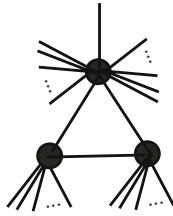
Thus

$$T_d(z, y, u) = \frac{z^{m-1}}{m!} \frac{1 + \mathcal{O}((1-y)(u-1))}{(1-y)^{k'}}.$$

Now let  $a_k^{(d)}(u) = m! t_k^{(d)}(z, u) / z^m$ . Since the coefficient of  $u$  in  $a_k^{(d)}(u)$  is the number of configurations where exactly one vertex has degree  $d$  is of order  $k^{k'-2}$  we surely have  $c_{k'-1} \neq 0$ . Similar arguments show that all the coefficients of the higher powers of  $u$  in  $a_k^{(d)}(u)$  are polynomials in  $k$  of degree less than  $k'$ . This implies the shape of the error term in (2) and completes the proof.

*Remark 1.* Theorem 2 applies to the triangular cluster set of Example 1, where we have  $k' = 2$  and consequently a tail of resulting degree distribution of the form  $p(d) \sim C k^{-4}$ . This is in accordance with the exact result (see 5)

$$p(d) = \frac{12}{(d-1)d(d+1)(d+2)} \quad (d \geq 4).$$



**Fig. 3.** Triangular cluster where the vertex with the ingoing half edge has two marks and the two other vertices just one

Another example (which is also discussed in [5]) is again based on triangular graphs, however, we distinguish now between left half edges and right half edges that can be additionally attached to the vertex with the ingoing half edge, see Figure 3. This distinction is necessary by the interpretation of PORTs as plane (or ordered) trees. Formally, such a cluster set can be handled by giving two marks to this vertex, that is,  $k' = 2 + 1 + 1 = 4$ . Obviously, Theorem 2 applies in such a situation, too. Hence, the tail of the resulting degree distribution is of the form  $p(d) \sim C d^{-6}$ . This is again in accordance with the exact result (see [5])

$$p(d) = \frac{1600}{(d - 1)d(d + 1)(d + 2)(d + 3)(d + 4)} \quad (d \geq 4).$$

### 4 Inserting Clusters of Different Type

Theorem 2 only applies if the cluster set is of a very simple form. We will next investigate what happens if we choose cluster sets containing clusters of different type, that is, of different degrees of freedom for attaching half edges. In order to understand in which way different tail behaviors caused by classes of clusters compete, it suffices to consider the case of two different types. Since  $t_k^{(d)}(z, u)$  is the generating function counting all clusters with respect to size and number of nodes of degree  $d$ , adding another class of clusters to the cluster sets results in adding the corresponding generating function to the first one. The same holds for the functions  $T_d(z, y, u)$ . Hence the problem reduces to an analysis of what happens if two behaviors of the forms (I) from Theorem 2 are added.

**Theorem 3.** *Let  $T_k$  be cluster sets (as described above) such that the equation*

$$X = \int_0^1 \frac{dt}{T_d(X, t, 1)}$$

*has a unique positive solution  $X = \rho$  in the region of convergence of  $T_d(z, y, u)$  and that  $T_d(z, y, u)$  can be represented as*

$$T_d(z, y, u) = \frac{C_0(z, y) + C_1(z, y)(1 - y)^{r'_1}y^{d+\alpha}(u - 1) + \mathcal{O}\left((1 - y)^{r'_1}(u - 1)^2\right)}{(1 - y)^{r_1}} + \frac{C_2(z, y) + C_3(z, y)(1 - y)^{r'_2}y^{d+\beta}(u - 1) + \mathcal{O}\left((1 - y)^{r'_2}(u - 1)^2\right)}{(1 - y)^{r_2}},$$

where  $r'_i$  and  $r_i$ ,  $i = 1, 2$ , are real numbers with  $0 < r'_i \leq r_i$  and  $r_1 > r_2$ ;  $\alpha$  and  $\beta$  are integers,  $C_i(z, y)$ ,  $i = 1, 2, 3, 4$ , are power series that contain  $z = \rho$  and  $y = 1$  in their regions of convergence and that satisfy  $C_i(\rho, y) \neq 0$  for  $i = 1, 3$  and  $0 \leq y \leq 1$  as well as  $C_0(\rho, y) + C_2(\rho, y)(1 - y)^{r_1-r_2} \neq 0$  for  $0 \leq y \leq 1$ . Moreover, assume that the  $\mathcal{O}(\cdot)$ -terms are uniform in a neighborhood of  $z = \rho$  and  $y = 1$ .

Let  $p_n(d)$  denote the probability that a random node in a thickened PORT of size  $n$  has degree  $d$ . Then the limits

$$\lim_{n \rightarrow \infty} p_n(d) =: p(d)$$

exist and we have, as  $d \rightarrow \infty$ ,

$$p(d) \sim \frac{C}{d^{\min\{r_1+r'_1+1, 2r_1-r_2+r'_2+1\}}}.$$

Furthermore, for every  $d \geq 0$  let  $X_n^{(d)}$  denote the number of nodes of degree  $d$  in a random thickened PORT of size  $n$ . Then  $X_n^{(d)}$  satisfies a central limit theorem

$$\frac{X_n^{(d)} - \mathbb{E} X_n^{(d)}}{\sqrt{\mathbb{V} X_n^{(d)}}} \xrightarrow{d} N(0, 1),$$

where  $\mathbb{E} X_n^{(d)}$  and  $\mathbb{V} X_n^{(d)}$  are both asymptotically proportional to  $n$ .

*Remark 2.* This theorem is – in some sense – a superposition principle for cluster sets. Note that (if  $r_2 \leq r_1$ )

$$\min\{r_1 + r'_1 + 1, 2r_1 - r_2 + r'_2 + 1\} = 1 + 2 \max\{r_1, r_2\} - \max\{r_1 - r'_1, r_2 - r'_2\}.$$

Hence, the resulting exponent in the tail of the degree distribution is determined by the behavior  $r_j$  and the differences  $r_j - r'_j$ . For those (basic) cluster sets which are covered in Theorem 2 we actually have  $r'_j = 1$ . Consequently, we obtain  $r_1 + 2 = \max\{r_1, r_2\} + 1$  as the resulting exponent. This means that if we interpret  $r_1$  as the degree of freedom to select half edges then the maximum degree of freedom is responsible for the exponent in the degree distribution.

This extends the above formulated *local-global-principle*.

*Proof.* We start by inspecting the generating function  $g(z)$  of all thickened PORTs. For simplicity we assume that the substitution sets  $\mathcal{T}_k$  are of a form that  $g_n > 0$  for sufficiently large  $n \geq n_0$ , that is, we exclude, for example, the case that the number of nodes of graphs in  $\mathcal{T}_k$  are all congruent to 1 modulo

some integer  $m > 1$ .<sup>2</sup> Then it follows that  $|g(z)| < g(|z|)$  if  $z$  is not contained in the positive real line.

We first observe that  $\rho > 0$  is the only singularity on the circle of convergence  $|z| \leq \rho$  and that  $g(\rho) = 1$ , that is,  $g(z)$  is convergent at  $z = \rho$ . First it is clear that  $g(z)$  can be analytically continued starting with  $g(0) = 0$  and the functional equation  $G_d(z, g(z), 1) = z$ . However, if  $g(z_0) \neq 1$  for some  $z_0$  contained in the region of convergence of  $g(z)$  then we have

$$\begin{aligned} \left(\frac{\partial}{\partial y} G_d\right)(z_0, g(z_0), 1) &= \frac{1}{T_d(z_0, g(z_0), 1)} \\ &= \frac{(1 - g(z_0))^{r_1}}{C_0(z_0, g(z_0)) + C_2(z_0, g(z_0))(1 - g(z_0))^{r_1 - r_2}} \neq 0. \end{aligned}$$

Thus, we can continue analytically with help of the implicit function theorem. Hence, if  $g(z)$  has a singularity  $\rho$  and if  $g(\rho)$  is convergent, then  $g(\rho) = 1$ . Since  $g(z)$  is monotone and analytic it certainly reaches a value with  $g(\rho) = 1$  where it has to be singular. Further,  $\rho$  is characterized by the equation  $G_d(\rho, 1, 1) = \rho$ .

Next we characterize the kind of singularity of  $g(z)$  at  $z = \rho$ . By Lemma 1 we have

$$\begin{aligned} z &= \int_0^g \frac{(1-t)^{r_1}}{C_0(z, t) + C_2(z, t)(1-t)^{r_1-r_2}} dt \\ &= \int_0^1 \frac{(1-t)^{r_1}}{C_0(z, t) + C_2(z, t)(1-t)^{r_1-r_2}} dt - \int_g^1 \frac{(1-t)^{r_1}}{C_0(z, t) + C_2(z, t)(1-t)^{r_1-r_2}} dt \\ &=: G(z) - H(z, g) \end{aligned} \tag{3}$$

Hence, by expanding

$$\frac{1}{C_0(z, t) + C_2(z, t)(1-t)^{r_1-r_2}} = c_0(z) + c_1(z)(1-t) + c_2(z)(1-t)^2 + \dots$$

we get

$$G(z) - z = c_0(z)(1 - g(z))^{r_1+1} (1 + \mathcal{O}(|1 - g(z)|))$$

which is equivalent to

$$\left(\frac{G(z) - z}{c_0(z)}\right)^{1/(r_1+1)} = (1 - g(z)) (1 + \mathcal{O}(|1 - g(z)|)). \tag{4}$$

Since  $G(\rho) = \rho$  and  $C_0(z, y)$  is analytic in  $z$  we can represent  $(G(z) - z)/c_0(z) = K(z)(1 - z/\rho)$ . Furthermore, we can invert relation (4) and obtain

$$g(z) = 1 - K(z)^{1/(r_1+1)} \left(1 - \frac{z}{\rho}\right)^{1/(r_1+1)} + \mathcal{O}\left(\left|1 - \frac{z}{\rho}\right|^{2/(r_1+1)}\right). \tag{5}$$

---

<sup>2</sup> We call this the aperiodic case. In the periodic case we have to deal with  $m$  singularities on the boundary of the circle of convergence of  $g(z)$  which are all of the same kind.

Since there are no other singularities on the circle  $|z| \leq \rho$  and  $g(z)$  can be analytically continued to a larger range (despite at the point  $z = \rho$ ) it follows from [6] that

$$g_n \sim K(\rho)^{1/(r_1+1)} \frac{\rho^{-n} n^{-\frac{r_1+2}{r_1+1}}}{-\Gamma\left(-\frac{1}{r_1+1}\right)}.$$

Next we determine the asymptotics of the average value  $\mathbb{E} X_n^{(d)}$ . Set  $S(z) = \frac{\partial}{\partial u} g(z, 1)$ . Then it follows from Lemma 1 that

$$\begin{aligned} S(z) = & - \frac{C_0(z, g(z)) + C_2(z, g(z))(1 - g(z))^{r_1-r_2}}{(1 - g(z))^{r_1}} \\ & \times \left[ \int_0^{g(z)} \frac{C_1(z, t)}{(C_0(z, t) + C_2(z, t)(1 - t)^{r_1-r_2})^2} (1 - t)^{r_1+r'_1} t^{d+\alpha} dt \right. \\ & \left. + \int_0^{g(z)} \frac{C_3(z, t)}{(C_0(z, t) + C_2(z, t)(1 - t)^{r_1-r_2})^2} (1 - t)^{2r_1-r_2+r'_2} t^{d+\beta} dt \right] \end{aligned}$$

By (5) and a decomposition of the integral as in (3) we can transform this to

$$\begin{aligned} S(z) = & \frac{1}{K(z)^{\frac{r_1}{r_1+1}} (1 - z/\rho)^{\frac{r_1}{r_1+1}}} \left[ \int_0^1 \tilde{C}_0(z, t)(1 - t)^{r_1+r'_1} t^{d+\alpha} dt \right. \\ & \left. + \int_0^1 \tilde{C}_1(z, t)(1 - t)^{r_2+r'_2} t^{d+\beta} dt \right] \\ & + \frac{1}{K(z)^{\frac{r_2}{r_1+1}} (1 - z/\rho)^{\frac{r_2}{r_1+1}}} \left[ \int_0^1 \tilde{C}_2(z, t)(1 - t)^{r_1+r'_1} t^{d+\alpha} dt \right. \\ & \left. + \int_0^1 \tilde{C}_3(z, t)(1 - t)^{2r_1-r_2+r'_2} t^{d+\beta} dt \right] + \mathcal{O}(1), \end{aligned}$$

where  $\tilde{C}_i(z, t)$  are analytic functions. This proves that

$$\begin{aligned} \mathbb{E} X_n^{(d)} = & n \cdot \frac{r_1 + 1}{K(\rho)} \int_0^1 \left[ \int_0^1 \tilde{C}_0(z, t)(1 - t)^{r_1+r'_1} t^{d+\alpha} dt \right. \\ & \left. + \int_0^1 \tilde{C}_1(z, t)(1 - t)^{2r_1-r_2+r'_2} t^{d+\beta} dt \right] + \mathcal{O}\left(n^{\frac{r_2+1}{r_1+1}}\right). \end{aligned}$$

Thus, the limit  $p(d) = \lim_{n \rightarrow \infty} \mathbb{E} X_n^{(d)} / n$  exists and is asymptotically given by

$$\begin{aligned} p(d) = & \frac{r_1 + 1}{K(\rho)} \left[ \int_0^1 \tilde{C}_0(z, t)(1 - t)^{r_1+r'_1} t^{d+\alpha} dt \right. \\ & \left. + \int_0^1 \tilde{C}_1(z, t)(1 - t)^{2r_1-r_2+r'_2} t^{d+\beta} dt \right] \\ \sim & \frac{C'}{d^{r_1+r'_1+1}} + \frac{C''}{d^{2r_1-r_2+r'_2+1}} \end{aligned}$$



for some constants  $C', C'' > 0$ .

Finally the proof that the limiting distribution is normal follows from the results and methods in [5,7] and [5].

*Example 2.* Let us continue the example where the cluster set  $\mathcal{T}_k$  all triangular graphs depicted in Figure 2 and the graph consisting of one single node. That means that – when building the thickened tree – in each substitution step we may substitute a node by a triangular graph or leave the node unchanged (no local clustering in this particular place). Then Theorem 3 says that the exponent  $-4$  in the tail  $p(d) \sim C d^{-4}$  of the degree distribution remains, but the constant  $C$  changes.

## References

1. Albert, R., Barabási, A.-L.: Statistical mechanics of complex networks. *Rev. Modern Phys.* 74(1), 47–97 (2002)
2. Bergeron, F., Flajolet, P., Salvy, B.: Varieties of increasing trees. In: Raoult, J.-C. (ed.) CAAP 1992. LNCS, vol. 581, pp. 24–48. Springer, Heidelberg (1992)
3. Bollobás, B., Riordan, O.: The diameter of a scale-free random graph. *Combinatorica* 24(1), 5–34 (2004)
4. Bollobás, B., Riordan, O., Spencer, J., Tusnády, G.: The degree sequence of a scale-free random graph process. *Random Structures Algorithms* 18(3), 279–290 (2001)
5. Drmota, M., Gittenberger, B., Panholzer, A.: The degree distribution of thickened trees. In: DMTCS Proceedings AI, pp. 149–162 (2008)
6. Flajolet, P., Odlyzko, A.M.: Singularity analysis of generating functions. *SIAM Journal on Discrete Mathematics* 3, 216–240 (1990)
7. Flajolet, P., Soria, M.: General combinatorial schemas: Gaussian limit distributions and exponential tails. *Discrete Math.* 114(1-3), 159–180 (1993)
8. Kuba, M., Panholzer, A.: On the degree distribution of the nodes in increasing trees. *J. Combin. Theory Ser. A* 114(4), 597–618 (2007)
9. Mahmoud, H.M., Smythe, R.T., Szymański, J.: On the structure of random plane-oriented recursive trees and their branches. *Random Structures Algorithms* 4(2), 151–176 (1993)
10. Pittel, B.: Note on the heights of random recursive trees and random  $m$ -ary search trees. *Random Structures Algorithms* 5(2), 337–347 (1994)

# Polar Permutation Graphs<sup>\*</sup>

Tinaz Ekim<sup>1</sup>, Pinar Heggernes<sup>2</sup>, and Daniel Meister<sup>2</sup>

<sup>1</sup> Industrial Engineering Department, Boğaziçi University, Istanbul, Turkey  
tinaz.ekim@boun.edu.tr

<sup>2</sup> Department of Informatics, University of Bergen, Norway  
pinar.heggernes@ii.uib.no, daniel.meister@ii.uib.no

**Abstract.** Polar graphs generalise bipartite, cobipartite, split graphs, and they constitute a special type of matrix partitions. A graph is polar if its vertex set can be partitioned into two, such that one part induces a complete multipartite graph and the other part induces a disjoint union of complete graphs. Deciding whether a given arbitrary graph is polar, is an NP-complete problem. Here we show that for permutation graphs this problem can be solved in polynomial time. The result is surprising, as related problems like achromatic number and cochromatic number are NP-complete on permutation graphs. We give a polynomial-time algorithm for recognising graphs that are both permutation and polar. Prior to our result, polarity has been resolved only for chordal graphs and cographs.

## 1 Introduction

Many graph problems can be formulated as finding a partition of the vertices such that various parts satisfy certain properties internally, and at the same time certain other properties are satisfied regarding the interaction between these parts. Examples of such problems are the broad variety of colouring and homomorphism problems, and matrix partition. The latter was posed by Feder et al. [13], and it asks for a partition of the vertex set of a graph into subsets  $A_1, \dots, A_k$ , such that each subset is a clique, an independent set or of arbitrary type, and pairs of subsets are completely adjacent, completely non-adjacent or can be connected arbitrarily, depending on a given pattern. If the pattern says that we partition into only cliques and independent sets, and  $A_i, A_j$  should be completely adjacent for  $A_i, A_j$  independent sets, completely non-adjacent for  $A_i, A_j$  cliques and there is no restriction for the two other cases, we get exactly the polar graphs.

Polar graphs were defined already in 1985 by Tyshkevich and Chernyak [25]. A graph is *polar* if its vertex set can be partitioned into  $A$  and  $B$  such that  $A$  induces a complete multipartite graph and  $B$  induces a cluster graph, i.e., a disjoint union of complete graphs. Such a partition is called *polar*. As a polar partition into  $A$  and  $B$  implies that  $A$  induces a  $\overline{P_3}$ -free graph and  $B$  induces a  $P_3$ -free graph, polar graphs are self-complementary, and they contain the well-known

---

<sup>\*</sup> This work is supported by the Research Council of Norway.

classes of split graphs, bipartite graphs, and cobipartite graphs. If  $A$  is simply an independent set, then the graph (and the partition) is called *monopolar*. In addition to fitting into the matrix partition problem [13] described above, polar partitions can be seen as generalised colourings [5].

Recognising polar graphs is an NP-complete problem [6]. Notice, however, that “admitting a polar partition” can be expressed in monadic second order logic without using edge set quantification, and hence polar graphs of bounded treewidth or bounded clique-width can be recognised in polynomial time, by the results of [11,7] and [8]. Consequently, it is of interest to find out where the boundary goes between subclasses of polar graphs that are recognisable in polynomial time and those whose recognition is intractable. When it comes to graph classes of unbounded treewidth and clique-width whose intersection with polar graphs can be recognised in polynomial time, so far we know only of chordal graphs [10,15].

In this paper we prove that it can be decided in polynomial time whether a given permutation graph is polar, and we give an  $O(n + m^4)$ -time algorithm for this. Permutation graphs are a well-studied graph class with a large number of theoretical applications [16,4], and they do not have bounded treewidth or clique-width [17]. Although many NP-complete problems become tractable on permutation graphs, well-known colouring problems, like cochromatic number [27,14] and achromatic number [2], are NP-complete on this graph class. Our result is obtained in two steps. First we give an algorithm for recognising monopolar permutation graphs. This algorithm is then used as a subroutine for the recognition of polar permutation graphs.

Other results on polynomial-time recognisable subclasses of polar graphs include [24] which studies polar partitions where the size of each independent set and clique is bounded, [15,10] which give forbidden subgraph characterisations and a recognition algorithm for polar chordal graphs, and [12] which gives similar results for polar cographs. In addition, [20] and [11] give respectively a forbidden subgraph characterisation and a polynomial-time recognition algorithm for bipartite graphs whose line graphs are polar. Finally, [9] gives a polynomial-time recognition algorithm for monopolar claw-free graphs. Another research direction is to study which NP-complete problems become tractable on polar graphs. For example, [21] gives polynomial-time algorithms for finding a minimum maximal independent set in some subclasses of polar graphs. This problem remains NP-hard in polar graphs admitting a polar partition where the size of every independent set is at most one and the size of every clique is at most two.

## 2 Definitions and Notation

Our input graphs are simple, finite, and undirected. Only in Section 3, we will use directed graphs (*digraphs*) as auxiliary tools.

Let  $G$  be a graph. We denote its vertex set by  $V(G)$  and its edge set by  $E(G)$ . An edge between vertices  $u$  and  $v$  is denoted by  $uv$ . If  $uv$  is an edge of  $G$  then  $u$  and  $v$  are *adjacent* in  $G$ . For a vertex  $x$  of  $G$ , the *neighbourhood* of  $x$ , denoted

as  $N_G(x)$ , is the set of vertices that are adjacent to  $x$ . Let  $X$  be a set of vertices of  $G$ . The subgraph of  $G$  induced by  $X$  is denoted as  $G[X]$  and defined as the graph on vertex set  $X$  and edge set the set of edges of  $G$  that join only vertices in  $X$ . By  $G \setminus X$ , we denote the graph  $G[V(G) \setminus X]$ . A graph is called *complete* if every pair of vertices is adjacent. A set  $X \subseteq V(G)$  is called a *clique* if  $G[X]$  is complete, and it is called an *independent set* if  $G[X]$  has no edges.

The *disjoint union* of two graphs  $G$  and  $H$  is the graph on vertex set  $V(G) \cup V(H)$  and edge set  $E(G) \cup E(H)$ ; the disjoint union of more than two graphs is defined analogously. The *complement* of  $G$ , denoted as  $\overline{G}$ , is the graph on vertex set  $V(G)$  and edge set  $\{uv \mid u, v \in V(G) \text{ and } u \neq v\} \setminus E(G)$ . A *multipartite graph* is the complement of the disjoint union of complete graphs. Equivalently, the vertex set of a multipartite graph admits a unique partition into maximal independent sets.

For a given graph  $G$ , a partition  $(A, B)$  of  $V(G)$ , where  $A$  or  $B$  can also be empty, is called *polar* if  $G[A]$  is a complete multipartite graph and  $G[B]$  is a disjoint union of complete graphs. Equivalently,  $(A, B)$  is a polar partition for  $G$  if and only if neither  $\overline{G}[A]$  nor  $G[B]$  contains an induced path on 3 vertices as an induced subgraph. Note that  $(A, B)$  is a polar partition for  $G$  if and only if  $(B, A)$  is a polar partition for  $\overline{G}$ . A polar partition  $(A, B)$  for  $G$  is called *monopolar* if  $A$  is an independent set in  $G$ . We say that a polar partition  $(A, B)$  for  $G$  is *B-maximal* if there is no polar partition  $(A', B')$  for  $G$  with  $B \subset B'$ . If there is a vertex  $u$  in  $A$  without a neighbour in  $B$  then  $(A \setminus \{u\}, B \cup \{u\})$  is also a polar partition. Hence, for  $(A, B)$  a *B-maximal* polar partition for a graph  $G$ , every vertex in  $A$  has a neighbour in  $B$ .

Let  $n \geq 1$  and  $\pi$  be a permutation over  $\{1, \dots, n\}$ , i.e., a bijection between  $\{1, \dots, n\}$  and  $\{1, \dots, n\}$ . We will denote  $\pi$  equivalently as a *permutation sequence*  $(\pi(1), \dots, \pi(n))$ . The *inversion graph* of  $\pi$  has vertex set  $\{1, \dots, n\}$  and two vertices  $u, v$  are adjacent if  $(u - v)(\pi^{-1}(u) - \pi^{-1}(v)) < 0$ . A graph is a *permutation graph* if it is isomorphic to the inversion graph of a permutation sequence [16, 4]. Permutation graphs can be recognised in linear time [22]. Permutation graphs also have a geometric intersection model: for two horizontal lines, mark  $n$  points on each line, assign to each point on the upper line a point on the lower line, and connect the two points by a line segment. The corresponding graph has a vertex for every line segment and two vertices are adjacent if the corresponding line segments cross. This representation is called a *permutation diagram*. A graph is a permutation graph if and only if it has a permutation diagram. It is important to note that every induced subgraph of a permutation graph is a permutation graph. For our purposes, we assume that a permutation graph is given as a permutation sequence and equal to the defined inversion graph. Every permutation graph with permutation sequence  $\pi$  has a permutation diagram  $\mathcal{D}$  in which the endpoints of the line segments on the lower line appear in the same order as they appear in  $\pi$ . For such pairs  $(\mathcal{D}, \pi)$ , we say that  $\mathcal{D}$  corresponds to  $\pi$ . For convenience reasons, sometimes we will not distinguish between vertices of the graph and line segments in the permutation diagram; however, the meaning will always be clear.

### 3 Monopolar Permutation Graphs

In this section, we give a polynomial-time algorithm for recognising monopolar permutation graphs, that certifies a positive answer by outputting a monopolar partition. This algorithm will be the main subroutine of our polar permutation graph recognition algorithm in the next section. In fact the subroutine presented in this section solves a more general problem: given a permutation graph  $G$  and a set  $R \subseteq V(G)$ , does  $G$  have a monopolar partition  $(A, B)$  such that  $A \subseteq R$ . By choosing  $R = V(G)$  we get exactly monopolar permutation graph recognition.

Let  $G$  be a permutation graph with permutation sequence  $\pi$  and corresponding permutation diagram  $\mathcal{D}$ . A *scanline* in  $\mathcal{D}$  is a pair  $(a, e)$  where  $a, e \in \{0.5, 1.5, \dots, n + \frac{1}{2}\}$  [3]. Scanlines can be interpreted as additional line segments in the permutation diagram, that partition the vertex set into three sets: vertices to the left of the scanline, vertices to the right of the scanline and vertices that intersect with the scanline. Formally, for  $s = (a, e)$  a scanline of  $G$ :

- $L(s) =_{\text{def}} \{x \in V(G) \mid x < a \text{ and } \pi^{-1}(x) < e\}$
- $R(s) =_{\text{def}} \{x \in V(G) \mid a < x \text{ and } e < \pi^{-1}(x)\}$
- $\text{int}(s) =_{\text{def}} V(G) \setminus (L(x) \cup R(x))$ .

The scanlines  $(\frac{1}{2}, \frac{1}{2})$  and  $(n + \frac{1}{2}, n + \frac{1}{2})$  play a particular role; they are denoted by the special symbols  $\mathbf{0}$  and  $\mathbf{1}$ . Note that  $L(\mathbf{0}) = \text{int}(\mathbf{0}) = \emptyset$  and  $R(\mathbf{1}) = \text{int}(\mathbf{1}) = \emptyset$  and  $R(\mathbf{0}) = L(\mathbf{1}) = V(G)$ . A scanline  $s = (a, e)$  is called *minimal separation line*<sup>1</sup> if  $1 < e < n$  and  $a - \frac{1}{2}, \pi(e - \frac{1}{2}), a + \frac{1}{2}, \pi(e + \frac{1}{2}) \notin \text{int}(s)$  or if  $s = \mathbf{0}$  or if  $s = \mathbf{1}$ . We will see later that the number of minimal separation lines of a permutation graph can be much smaller than the number of its scanlines. For two scanlines  $s = (a, e)$  and  $s' = (a', e')$ , we write  $s < s'$  if  $a \leq a'$  and  $e \leq e'$  and  $s \neq s'$ .

We solve the monopolar permutation graph recognition problem by deciding whether an auxiliary digraph has a directed path between two specified vertices. This auxiliary digraph is defined as follows. Let  $G$  be a permutation graph with permutation sequence  $\pi$  and permutation diagram  $\mathcal{D}$ , and let  $R \subseteq V(G)$ . By  $\text{aux}(\mathcal{D}, R)$ , we denote the digraph with the following vertices and arcs:

- $\text{aux}(\mathcal{D}, R)$  has a vertex for every minimal separation line of  $G$
- for two minimal separation lines  $s$  and  $s'$  with  $s < s'$ , there is an arc from the corresponding vertex of  $s$  to the corresponding vertex of  $s'$  in  $\text{aux}(\mathcal{D}, R)$  if  $G[\text{int}(s) \cup \text{int}(s') \cup (R(s) \cap L(s'))]$  has a monopolar partition  $(I, C)$  such that  $\text{int}(s) \cup \text{int}(s') \subseteq I \subseteq R$  and  $C$  is a clique in  $G$ .

Observe that the properties of the partition  $(I, C)$  imply that  $G[\text{int}(s) \cup \text{int}(s') \cup (R(s) \cap L(s'))]$  is a split graph. Observe also that  $\text{aux}(\mathcal{D}, R)$  is acyclic, which follows from the partial order  $<$  on the minimal separation lines. We denote the vertices of  $\text{aux}(\mathcal{D}, R)$  that correspond to  $\mathbf{0}$  and  $\mathbf{1}$  as respectively  $\mathbf{0}$  and  $\mathbf{1}$ . Note that  $\mathbf{0}$  has no incoming arc and  $\mathbf{1}$  has no outgoing arc in  $\text{aux}(\mathcal{D}, R)$ .

---

<sup>1</sup> Minimal separation lines correspond exactly to minimal separators [23].

**Lemma 1.** *Let  $G$  be a permutation graph with permutation sequence  $\pi$  and corresponding permutation diagram  $\mathcal{D}$ . Let  $R \subseteq V(G)$ . There is a  $\mathbf{0}, \mathbf{1}$ -path in  $\text{aux}(\mathcal{D}, R)$  if and only if  $G$  has a monopolar partition  $(A, B)$  with  $A \subseteq R$ .*

*Proof.* Let  $P$  be a  $\mathbf{0}, \mathbf{1}$ -path in  $\text{aux}(\mathcal{D}, R)$  and  $s_0 < \dots < s_r$  the minimal separation lines corresponding to the vertices on  $P$ . There is a monopolar partition  $(I_i, C_i)$  for  $G[\text{int}(s_{i-1}) \cup \text{int}(s_i) \cup (R(s_{i-1}) \cap L(s_i))]$  for every  $1 \leq i \leq r$  such that  $\text{int}(s_{i-1}) \cup \text{int}(s_i) \subseteq I_i \subseteq R$  and  $C_i$  is a clique of  $G$ . It can be shown that  $(I_1 \cup \dots \cup I_r, C_1 \cup \dots \cup C_r)$  is a monopolar partition for  $G$ . For the converse, let  $(A, B)$  be a monopolar partition for  $G$  with  $A \subseteq R$ . Let  $C_1, \dots, C_r$  be the sets of vertices that induce the connected components of  $G[B]$ . We can assume that  $\min C_1 < \dots < \min C_r$ . It follows that  $\min C_1 \leq \max C_1 < \dots < \min C_r \leq \max C_r$ . For every  $1 \leq i \leq r - 1$ , there is a minimal separation line  $s_i$  of  $G$  with  $C_i \subseteq L(s_i)$  and  $C_{i+1} \subseteq R(s_i)$  [3,23]. Let  $s_0 =_{\text{def}} \mathbf{0}$  and  $s_r =_{\text{def}} \mathbf{1}$ . There is a vertex of  $\text{aux}(\mathcal{D}, R)$  for every minimal separation line  $s_0, \dots, s_r$ , and these vertices form a  $\mathbf{0}, \mathbf{1}$ -path in  $\text{aux}(\mathcal{D}, R)$ .  $\square$

Note that a  $\mathbf{0}, \mathbf{1}$ -path in  $\text{aux}(\mathcal{D}, R)$  does not correspond to a specific monopolar partition for  $G$  but can represent many partitions. The main reason is that the same vertex can belong to a clique or to the independent set in different monopolar partitions.

**Theorem 1.** *There is an  $\mathcal{O}(n + m^3)$ -time algorithm that given a permutation graph  $G$  and a set  $R \subseteq V(G)$ , decides whether there is a monopolar partition  $(A, B)$  for  $G$  with  $A \subseteq R$ .*

*Proof.* For a connected input graph  $G$  with sequence  $\pi$  and corresponding diagram  $\mathcal{D}$  and  $R \subseteq V(G)$ , construct the auxiliary digraph  $\text{aux}(\mathcal{D}, R)$  and check whether  $\text{aux}(\mathcal{D}, R)$  has a  $\mathbf{0}, \mathbf{1}$ -path; accept if and only if a  $\mathbf{0}, \mathbf{1}$ -path exists. Correctness follows from Lemma 1. The running time mainly depends on the construction of  $\text{aux}(\mathcal{D}, R)$ . The minimal separation lines of  $G$  can be listed in linear time [23]. For every pair of minimal separation lines, checking whether it defines a monopolar partition of the special type for the induced subgraph can be done in linear time, for instance by applying the degree-sequence recognition algorithm for split graphs [18] and a preselection for the independent set and clique vertices, see also [19]. Thus, the construction of  $\text{aux}(\mathcal{D}, R)$  can be done in time  $\mathcal{O}(m^3)$ . A path from  $\mathbf{0}$  to  $\mathbf{1}$  in  $\text{aux}(\mathcal{D}, R)$  can be found in time linear in the size of  $\text{aux}(\mathcal{D}, R)$ . For disconnected input graphs, a graph has a monopolar partition that respects the restriction  $R$  if and only if each connected component has a monopolar partition respecting  $R$ .  $\square$

It is not difficult to modify the algorithm in the proof of Theorem 1 such that it also outputs a monopolar partition. This can be achieved for instance by assigning to every arc of the auxiliary digraph the monopolar partition that certifies the existence of this arc, following the proof of Lemma 1.

## 4 Polar Permutation Graphs

In the previous section, we have given an algorithm for recognising monopolar permutation graphs. This is a subclass of polar permutation graphs. In this section, we consider the complementary recognition problem, for polar permutation graphs that are not monopolar. We will devise an algorithm that, given a permutation graph, deletes a set of vertices with certain properties and checks whether the remaining graph has a monopolar partition of specific properties. The main task is to characterise the sets of vertices that can be deleted and to define the properties of the desired monopolar partitions.

Let  $G$  be a permutation graph with permutation sequence  $\pi$  and corresponding permutation diagram  $\mathcal{D}$ . Informally, a trapezoid in  $\mathcal{D}$  is defined by a line segment on the upper and lower horizontal line of  $\mathcal{D}$  and partitions the vertex set of  $G$  into three subsets. To describe such a trapezoid, it suffices to give the endpoints of the line segments. Formally, a *trapezoid* in  $\mathcal{D}$  is a pair  $((a, e), (a', e'))$  with  $1 \leq a \leq e \leq |V(G)|$  and  $1 \leq a' \leq e' \leq |V(G)|$ . Trapezoids are the main substructures in permutation diagrams that we consider in this section. We define four sets for trapezoids, in analogy to scanlines. Let  $\mathbf{S} = ((a, e), (a', e'))$  be a trapezoid in  $\mathcal{D}$ . We define for  $\mathbf{S}$ :

- *left side*:  $L(\mathbf{S}) =_{\text{def}} \{x \in V(G) \mid x < a \text{ and } \pi^{-1}(x) < a'\}$
- *right side*:  $R(\mathbf{S}) =_{\text{def}} \{x \in V(G) \mid e < x \text{ and } e' < \pi^{-1}(x)\}$
- *containment*:  $\text{con}(\mathbf{S}) =_{\text{def}} \{x \in V(G) \mid a \leq x \leq e \text{ and } a' \leq \pi^{-1}(x) \leq e'\}$
- *intersection*:  $\text{int}(\mathbf{S}) =_{\text{def}} V(G) \setminus (L(\mathbf{S}) \cup R(\mathbf{S}))$ .

Note that  $\text{con}(\mathbf{S}) \subseteq \text{int}(\mathbf{S})$ . For  $X \subseteq V(G)$ , the  $X$ -trapezoid in  $\mathcal{D}$  is the trapezoid  $\mathbf{T}$  with  $\mathbf{T} = ((\min X, \max X), (\min \pi^{-1}(X), \max \pi^{-1}(X)))$ . Note that  $X \subseteq \text{con}(\mathbf{T})$ . Informally, the  $X$ -trapezoid is the smallest trapezoid that contains  $X$ . Note that not every trapezoid is an  $X$ -trapezoid for some set  $X$ . Our main interest is in trapezoids with special properties. Let  $G$  have a polar partition  $(A, B)$  and let  $Y \subseteq V(G)$ . A trapezoid  $\mathbf{T}$  in  $\mathcal{D}$  is called *centre trapezoid for  $(A, B)$  in  $\mathcal{D}$  (with  $Y$  in-cliqued)* if the following conditions are satisfied:

- 1)  $G[\text{int}(\mathbf{T})]$  has a polar partition  $(A', C)$  with
  - $C$  is a clique in  $G$  (and  $Y \subseteq C$ ) and  $C \subseteq \text{con}(\mathbf{T})$
  - $\text{int}(\mathbf{T}) \setminus \text{con}(\mathbf{T}) \subseteq A' \subseteq A$
- 2) one of the following two cases holds:
  - $A \cap (L(\mathbf{T}) \cup R(\mathbf{T}))$  is an independent set in  $G$
  - $A \cap L(\mathbf{T}) \neq \emptyset$  and  $A \cap R(\mathbf{T}) = \emptyset$  and there is a vertex  $v \in \text{int}(\mathbf{T}) \setminus \text{con}(\mathbf{T})$  with  $N_G(v) \cap L(\mathbf{T}) \subseteq A$  such that  $\{v\} \cup (A \cap L(\mathbf{T})) \setminus N_G(v)$  is an independent set in  $G$ .

We show that such centre trapezoids indeed exist. A graph that is polar but not monopolar is called *multipolar*.

**Lemma 2.** *Let  $G$  be a permutation graph with permutation sequence  $\pi$  and corresponding permutation diagram  $\mathcal{D}$ . Let  $G$  be multipolar and  $(A, B)$  a  $B$ -maximal polar partition for  $G$ . There are a trapezoid  $\mathbf{T}$  in  $\mathcal{D}$  and a clique  $X$  of  $G$  with  $X \subseteq B$  and  $1 \leq |X| \leq 2$  such that  $\mathbf{T}$  is the  $X$ -trapezoid in  $\mathcal{D}$  and  $\mathbf{T}$  is a centre trapezoid for  $(A, B)$  in  $\mathcal{D}$  with  $X$  in-cliqued.*



*Proof.* Let the connected components of  $G[B]$  be induced by the sets  $C_1, \dots, C_r$ . We can assume that  $\min C_1 < \dots < \min C_r$ . For every  $x \in A$ , denote by  $\alpha(x)$  and  $\omega(x)$  the respectively smallest and largest index  $i$  with  $C_i$  containing a neighbour of  $x$ .

As the first case, assume that there are adjacent vertices  $u, v \in A$  with  $\omega(u) < \alpha(v)$ . Note that  $uv \in E(G)$  implies  $\alpha(v) = \omega(u) + 1$ . Let  $C =_{\text{def}} C_{\alpha(v)}$  and let  $\mathbf{T}$  be the  $C$ -trapezoid in  $\mathcal{D}$ . Note that  $u \in L(\mathbf{T})$  and therefore  $v \in \text{int}(\mathbf{T}) \setminus \text{con}(\mathbf{T})$  by the properties of permutation diagrams. Furthermore,  $L(\mathbf{T}) \cap N_G(v) \subseteq A$  by the definition of  $\alpha(v)$  and  $C \subseteq \text{con}(\mathbf{T})$ . For the first subcase, assume that  $A \subseteq L(\mathbf{T}) \cup \text{int}(\mathbf{T})$ , i.e.,  $A \cap R(\mathbf{T}) = \emptyset$ . A vertex in  $A \cap L(\mathbf{T})$  is either adjacent or non-adjacent to  $v$ . The vertices in  $A \cap L(\mathbf{T})$  that are non-adjacent to  $v$  are in the same maximal independent set as  $v$  in  $G[A]$ . Hence, they are pairwise non-adjacent, which means that  $\{v\} \cup (A \cap L(\mathbf{T})) \setminus N_G(v)$  is an independent set in  $G$ . For the second subcase, let  $A \cap R(\mathbf{T})$  be non-empty. This means that both  $A \cap L(\mathbf{T})$  and  $A \cap R(\mathbf{T})$  are non-empty. Since vertices from  $L(\mathbf{T})$  and  $R(\mathbf{T})$  are pairwise non-adjacent, the definition of complete multipartite graphs implies that  $A \cap (L(\mathbf{T}) \cup R(\mathbf{T}))$  is an independent set in  $G$ . Hence, in both subcases,  $\mathbf{T}$  satisfies the second condition of the definition of centre trapezoid.

As the second case, assume that for all pairs  $u, v \in A$  of adjacent vertices,  $\alpha(u) \leq \alpha(v) \leq \omega(u)$  or  $\alpha(v) \leq \alpha(u) \leq \omega(v)$ . Let  $u, v \in A$  be a pair of adjacent vertices such that the intersection  $\{\alpha(u), \dots, \omega(u)\} \cap \{\alpha(v), \dots, \omega(v)\}$  is of smallest size. We can assume that  $\alpha(u) \leq \alpha(v)$ . Let  $C =_{\text{def}} C_{\alpha(v)}$  and let  $\mathbf{T}$  be the  $C$ -trapezoid in  $\mathcal{D}$ . For the first subcase, let  $\omega(v) \leq \omega(u)$ . Then,  $\alpha(u) \leq \alpha(v) \leq \omega(v) \leq \omega(u)$  with our assumptions, which implies  $\alpha(x) \leq \alpha(v) \leq \omega(v) \leq \omega(x)$  for all vertices  $x \in A \cap N_G(v)$ . This follows from the choice of the pair  $u, v$  as of smallest intersection size. Hence,  $A \cap N_G(v) \subseteq \text{int}(\mathbf{T})$ , and by the properties of complete multipartite graphs,  $A \cap (L(\mathbf{T}) \cup R(\mathbf{T}))$  is an independent set in  $G$ . For the second subcase, let  $\omega(u) < \omega(v)$ . For every vertex  $x \in A \cap L(\mathbf{T})$ ,  $\omega(x) < \alpha(v)$ , which means by the assumptions of the case that  $(A \cap L(\mathbf{T})) \cap N_G(v) = \emptyset$ . Hence,  $\{v\} \cup (A \cap L(\mathbf{T}))$  is an independent set in  $G$ . In the following, we distinguish between the cases  $A \cap L(\mathbf{T})$  non-empty and empty. First, let  $A \cap L(\mathbf{T})$  be non-empty. The vertices in  $A \cap L(\mathbf{T})$  and  $A \cap R(\mathbf{T})$  are pairwise non-adjacent, which implies that  $\{v\} \cup (A \cap L(\mathbf{T})) \cup (A \cap R(\mathbf{T}))$  is an independent set in  $G$ . Second, let  $A \cap L(\mathbf{T})$  be empty. Let  $A \cap R(\mathbf{T})$  be non-empty and let  $w \in A \cap R(\mathbf{T})$ . Since  $u \in \text{int}(\mathbf{T})$ ,  $w \neq u$ . Suppose that  $uw \in E(G)$ . Then,  $\alpha(v) < \alpha(w)$  and  $\alpha(v) \leq \omega(u) < \omega(v)$  yield a contradiction to the intersection size of  $u, v$ . Hence,  $u$  and  $w$  are non-adjacent, which implies that  $\{u\} \cup (A \cap R(\mathbf{T})) = \{u\} \cup (A \cap L(\mathbf{T})) \cup (A \cap R(\mathbf{T}))$  is an independent set in  $G$ . Hence, in both subcases,  $A \cap (L(\mathbf{T}) \cup R(\mathbf{T}))$  is an independent set in  $G$ , which shows that  $\mathbf{T}$  satisfies the second condition of the definition of centre trapezoids.

It remains to check the first condition of the definition of centre trapezoids. We can consider the two cases above simultaneously. Let  $C$  and  $\mathbf{T}$  be as defined above. First, we show that  $(\text{int}(\mathbf{T}) \setminus C, C)$  is a polar partition for  $G[\text{int}(\mathbf{T})]$ . Since no edge of  $G$  joins vertices from different cliques among  $C_1, \dots, C_r$ ,  $B \cap \text{int}(\mathbf{T}) = C$ . Hence,  $\text{int}(\mathbf{T}) \setminus B = \text{int}(\mathbf{T}) \setminus C \subseteq A$ . By the definition of  $C$ -trapezoid,



$C \subseteq \text{con}(\mathbf{T})$ , which also implies that  $\text{int}(\mathbf{T}) \setminus \text{con}(\mathbf{T}) \subseteq \text{int}(\mathbf{T}) \setminus C \subseteq A$ . To complete the proof, observe that  $\mathbf{T}$  is the  $\{x, y\}$ -trapezoid and a centre trapezoid for  $(A, B)$  in  $\mathcal{D}$  with  $\{x, y\}$  in-cliqued for  $x =_{\text{def}} \min C$  and  $y =_{\text{def}} \max C$ .  $\square$

Lemma 2 is the main tool of our algorithm. Informally, the algorithm removes a trapezoid from the graph and checks whether the remaining subgraph is monopolar. The main problem then is to combine the independent set of the monopolar partition with the complete multipartite graph of the polar partition of the removed trapezoid. Not every monopolar partition is suitable. The next lemma will be useful for choosing a suitable monopolar partition. Let  $G$  be a permutation graph with permutation sequence  $\pi$  and corresponding permutation diagram  $\mathcal{D}$ . Let  $\mathbf{T} = ((a, e), (a', e'))$  be a trapezoid in  $\mathcal{D}$  and let  $S \subseteq \text{int}(\mathbf{T}) \setminus \text{con}(\mathbf{T})$ . A vertex  $x \in S$  is *left-endpoint close to  $\mathbf{T}$  among the vertices in  $S$*  if  $x < a$  and  $x \geq y$  for all vertices  $y \in S$  or if  $\pi^{-1}(x) < a'$  and  $\pi^{-1}(x) \geq \pi^{-1}(y)$  for all vertices  $y \in S$ . *Right-endpoint close vertex* is defined symmetrically. Note that there are at most two left-endpoint close and at most two right-endpoint close vertices for every trapezoid and chosen set.

**Lemma 3.** *Let  $G$  be a permutation graph with permutation sequence  $\pi$  and corresponding permutation diagram  $\mathcal{D}$ . Let  $G$  be multipolar and  $(A, B)$  a  $B$ -maximal polar partition for  $G$ . Let  $\mathbf{T}$  be a centre trapezoid for  $(A, B)$  in  $\mathcal{D}$ .*

1) *Let  $A \cap (\text{L}(\mathbf{T}) \cup \text{R}(\mathbf{T}))$  be an independent set in  $G$  and let  $A \cap \text{con}(\mathbf{T})$  be non-empty.*

*Let  $x$  be an arbitrary vertex in  $A \cap \text{con}(\mathbf{T})$ . Then,  $\{x\} \cup (A \cap (\text{L}(\mathbf{T}) \cup \text{R}(\mathbf{T})))$  is an independent set in  $G$ .*

2) *Let  $A \cap (\text{L}(\mathbf{T}) \cup \text{R}(\mathbf{T}))$  be an independent set in  $G$  and let  $A \cap \text{con}(\mathbf{T})$  be empty.*

*If there is  $x \in A \cap \text{int}(\mathbf{T})$  such that  $\{x\} \cup (A \cap (\text{L}(\mathbf{T}) \cup \text{R}(\mathbf{T})))$  is an independent set in  $G$  then  $x$  can be chosen as left-endpoint close or right-endpoint close to  $\mathbf{T}$  among the vertices in  $\text{int}(\mathbf{T}) \setminus \text{con}(\mathbf{T})$ .*

3) *Let  $A \cap (\text{L}(\mathbf{T}) \cup \text{R}(\mathbf{T}))$  not be an independent set in  $G$ .*

*Let  $A \cap \text{L}(\mathbf{T}) \neq \emptyset$  and  $A \cap \text{R}(\mathbf{T}) = \emptyset$  and let there be a vertex  $x \in \text{int}(\mathbf{T}) \setminus \text{con}(\mathbf{T})$  with  $N_G(x) \cap \text{L}(\mathbf{T}) \subseteq A$  such that  $(A \cap \text{L}(\mathbf{T})) \setminus N_G(x)$  is an independent set in  $G$ . Then,  $x$  can be chosen as left-endpoint close to  $\mathbf{T}$  among the vertices in  $\text{int}(\mathbf{T}) \setminus \text{con}(\mathbf{T})$ .*

We are ready to give the final algorithm. Our algorithm for recognising polar permutation graphs is called **Polar-Permutation-graphs** and given in Figure 4. If the input graph is polar, the algorithm outputs a polar partition, thus provides a certificate.

**Theorem 2.** *Algorithm Polar-Permutation-graphs recognises polar permutation graphs in  $\mathcal{O}(n + m^4)$  time.*

*Proof.* We only prove correctness. Let  $G$  be the input graph with permutation sequence  $\pi$  and corresponding permutation diagram  $\mathcal{D}$ . We first show that every

**Algorithm Polar-Permutation-graphs****Input** permutation graph  $G$  with sequence  $\pi$  and corresponding diagram  $\mathcal{D}$ **Output** answer ‘yes’ or ‘no’, and if ‘yes’ then a polar partition for  $G$ 

```

begin
1  if  $G$  is monopolar then
2    compute a monopolar partition  $(A, B)$  for  $G$ ; return ‘yes’ and  $(A, B)$ 
3  end if;
4  for  $X$  a clique of  $G$  of size 1 or 2 do
5    let  $\mathbf{T}$  be the  $X$ -trapezoid in  $\mathcal{D}$ ;
6    if  $G[\text{int}(\mathbf{T})]$  is polar and has polar partition  $(A', C)$  such that
7       $\text{int}(\mathbf{T}) \setminus \text{con}(\mathbf{T}) \subseteq A'$  and  $C$  is a clique of  $G$  and
8      maximal with  $X \subseteq C \subseteq \text{con}(\mathbf{T})$  then
9      let  $(A', C)$  be the computed polar partition for  $G[\text{int}(\mathbf{T})]$ ;
10     if  $A' \cap \text{con}(\mathbf{T}) \neq \emptyset$  then
11       let  $x \in A' \cap \text{con}(\mathbf{T})$ ;
12       if  $G \setminus \text{int}(\mathbf{T})$  has monopolar partition  $(A, B)$  with every vertex
13         in  $A$  is adjacent to every vertex in  $A' \cap N_G(x)$  and non-adjacent
14         to every vertex in  $A' \setminus N_G(x)$  then
15         let  $(A, B)$  be the computed monopolar partition for  $G \setminus \text{int}(\mathbf{T})$ ;
16         return ‘yes’ and  $(A \cup A', B \cup C)$ 
17       end if
18     else
19       if  $G \setminus \text{int}(\mathbf{T})$  has monopolar partition  $(A, B)$  with
20         every vertex in  $A$  is adjacent to every vertex in  $A'$  then
21         let  $(A, B)$  be the computed monopolar partition for  $G \setminus \text{int}(\mathbf{T})$ ;
22         return ‘yes’ and  $(A \cup A', B \cup C)$ 
23       end if;
24       let  $L$  and  $R$  be the sets of respectively left-endpoint close and
25         right-endpoint close vertices for  $\mathbf{T}$  among the vertices in
26          $\text{int}(\mathbf{T}) \setminus \text{con}(\mathbf{T})$ ;
27       if there are vertex  $x \in L \cup R$  and monopolar partition  $(A, B)$  for
28          $G \setminus \text{int}(\mathbf{T})$  with every vertex in  $A$  is adjacent to every vertex in
29          $A' \cap N_G(x)$  and non-adjacent to every vertex in  $A' \setminus N_G(x)$  then
30         let  $(A, B)$  be the computed monopolar partition for  $G \setminus \text{int}(\mathbf{T})$ ;
31         return ‘yes’ and  $(A \cup A', B \cup C)$ 
32       end if;
33       if there are vertex  $x \in L$  and monopolar partition  $(A, B)$  for
34          $G \setminus (\text{int}(\mathbf{T}) \cup (N_G(x) \cap L(\mathbf{T})))$  with  $A \cap R(\mathbf{T}) = \emptyset$  and  $\{x\} \cup A$ 
35         an independent set in  $G$  and  $G[A \cup A' \cup (N_G(x) \cap L(\mathbf{T}))]$ 
36         complete multipartite then
37         let  $(A, B)$  be the computed monopolar partition for  $G \setminus \text{int}(\mathbf{T})$ ;
38         return ‘yes’ and  $(A \cup A' \cup (N_G(x) \cap L(\mathbf{T})), B \cup C)$ 
39     end if end if end if end for;
40  return ‘no’
end.

```

**Fig. 1.** The polar permutation graph recognition algorithm

‘yes’ answer (in lines 2, 16, 22, 31, 38) is correct and the output partition is a proper polar partition for  $G$ . So, let the answer of the algorithm on input  $G, \pi, \mathcal{D}$  be ‘yes’. It is a simple check that the output vertex partition is indeed a partition of  $V(G)$ . If the answer is output in line 2 then  $G$  is monopolar, thus polar, and the output partition is a polar partition for  $G$ . We consider the four other cases. We consider the **for** loop during its last execution. Let  $\mathbf{T}$  be the trapezoid defined in line 5 and let  $(A', C)$  be the polar partition for  $G[\text{int}(\mathbf{T})]$  chosen in line 9. Note that  $(A', C)$  has the properties of lines 6–8. In particular, no vertex from  $C$  has a neighbour in  $G \setminus \text{int}(\mathbf{T})$ , which follows directly from  $C \subseteq \text{con}(\mathbf{T})$  and the properties of permutation diagrams. Hence, since  $C$  is a clique in  $G$  and  $G[B]$  is an induced subgraph of  $G \setminus \text{int}(\mathbf{T})$  and the disjoint union of complete graphs, for each of the four cases for  $B$ ,  $G[B \cup C]$  is the disjoint union of complete graphs. It remains to show for each of the four cases that the first component of the output vertex partition induces a complete multipartite graph in  $G$ . The case is clear for the output in line 38 by the condition in lines 34–36. Assume that the answer is output in line 16, let  $(A, B)$  be the monopolar partition for  $G \setminus \text{int}(\mathbf{T})$  in line 15 and let  $x$  be the vertex chosen in line 11. Since  $A$  and  $A' \setminus N_G(x)$  are independent sets in  $G$ ,  $A \cup (A' \setminus N_G(x))$  is an independent set in  $G$  due to the conditions in lines 12–14. And by the properties of complete multipartite graphs and the adjacency condition in lines 12–14, every vertex in  $A \cup (A' \setminus N_G(x))$  is adjacent to every vertex in  $A' \cap N_G(x)$ . Hence,  $G[A \cup A']$  is complete multipartite. The cases for the output in lines 22 and 31 follow similarly. Thus, the output partition is a polar partition for  $G$ , and  $G$  is polar.

For the converse, let  $G$  be polar. We show that **Polar-Permutation-graphs** answers ‘yes’. If  $G$  is monopolar then **Polar-Permutation-graphs** returns answer ‘yes’ in line 2. Let  $G$  not be monopolar, and let  $(P, Q)$  be a  $Q$ -maximal polar partition for  $G$ . According to Lemma 2, there are a trapezoid  $\mathbf{T}$  in  $\mathcal{D}$  and a clique  $X$  of  $G$  with  $X \subseteq Q$  and  $1 \leq |X| \leq 2$  such that  $\mathbf{T}$  is the  $X$ -trapezoid in  $\mathcal{D}$  and  $\mathbf{T}$  is a centre trapezoid for  $(P, Q)$  in  $\mathcal{D}$  with  $X$  in-cliqued. Note that  $X$  and thus  $\mathbf{T}$  can be chosen by **Polar-Permutation-graphs** in lines 4 and 5. Let  $(A', C)$  be a polar partition for  $G[\text{int}(\mathbf{T})]$  as defined in condition 1 of the definition of centre trapezoids; such a partition exists by assumption. Without loss of generality, we can assume that every vertex in  $A' \cap \text{con}(\mathbf{T})$  is non-adjacent to some vertex in  $C$ . This partition satisfies the conditions in lines 6–8, and **Polar-Permutation-graphs** continues execution in line 9. Observe for the following arguments that  $P \cap \text{int}(\mathbf{T}) = A'$ : if there is  $u \in (P \cap \text{int}(\mathbf{T})) \setminus A'$  then  $u \in C$ . By the definition of  $(A', C)$  according to condition 1 of the definition of centre trapezoid,  $u \in \text{con}(\mathbf{T})$ , which means that  $u$  is not adjacent to any vertex in  $Q \setminus \text{int}(\mathbf{T})$ . Then,  $(P \setminus \{u\}, Q \cup \{u\})$  is a polar partition for  $G$ , which contradicts the choice of  $(P, Q)$  as  $Q$ -maximal. Then,  $P \cap (\text{L}(\mathbf{T}) \cup \text{R}(\mathbf{T})) = P \setminus A'$ . We distinguish between the two cases in condition 2 of the definition of centre trapezoids. As the first main case, let  $P \setminus A'$  be an independent set in  $G$ . Then,  $G \setminus \text{int}(\mathbf{T})$  is monopolar with monopolar partition  $(P \setminus A', Q \setminus C)$ . As a first subcase, let  $P \cap \text{con}(\mathbf{T}) = A' \cap \text{con}(\mathbf{T})$  be non-empty (line 10). For any vertex  $x \in A' \cap \text{con}(\mathbf{T})$ ,  $\{x\} \cup (P \setminus A')$  is an independent set in  $G$  and particularly in  $G[P]$ . With the

properties of complete multipartite graphs, all vertices in  $P \setminus A'$  are adjacent to all vertices in  $A' \cap N_G(x)$  and non-adjacent to all vertices in  $A' \setminus N_G(x)$ . Then, the conditions in lines 12–14 are satisfied by partition  $(P \setminus A', Q \setminus C)$ , and the algorithm accepts. As a second subcase, let  $A' \cap \text{con}(\mathbf{T})$  be empty. If all vertices in  $P \setminus A'$  are adjacent to all vertices in  $A'$  then **Polar-Permutation-graphs** accepts in line 22. Let there be a vertex  $y \in P \setminus A'$  that is non-adjacent to some vertex  $x \in A'$ . Then,  $x$  and  $y$  are in the same maximal independent set of  $G[P]$ , which implies that  $\{x\} \cup (P \setminus A')$  is an independent set of  $G$ . Due to Lemma 3, there is a vertex  $z$  in  $L \cup R$  of lines 24–26 such that  $\{z\} \cup (P \setminus A')$  is an independent set of  $G$ . Analogous to the first subcase, **Polar-Permutation-graphs** accepts in line 31. As the second main case, let  $P \setminus A'$  not be an independent set of  $G$ . According to condition 2 of the definition of centre trapezoids,  $P \cap L(\mathbf{T}) \neq \emptyset$  and  $P \cap R(\mathbf{T}) = \emptyset$  and there is a vertex  $v \in A' \setminus \text{con}(\mathbf{T})$  with  $N_G(v) \cap L(\mathbf{T}) \subseteq P$  and  $\{v\} \cup (P \cap L(\mathbf{T})) \setminus N_G(v)$  is an independent set of  $G$ . Then,  $((P \setminus A') \setminus N_G(v), Q \setminus C)$  is a monopolar partition for  $G \setminus (\text{int}(\mathbf{T}) \cup (N_G(v) \cap L(\mathbf{T})))$ . Due to Lemma 3, we can choose  $v$  from  $L$ . Then, **Polar-Permutation-graphs** accepts in line 38.  $\square$

## 5 Concluding Remarks and Open Problems

The main time consuming operation in the presented algorithm is the construction of the auxiliary digraph. One possibility to improve on the running time is to bound the number of arcs in the auxiliary digraph and to avoid testing all possible pairs of vertices.

Permutation graphs are both comparability and cocomparability graphs. An interesting question is whether polar comparability graphs, or equivalently polar cocomparability graphs, can be recognised in polynomial time. Another question is whether in permutation graphs that are not polar, a maximum induced polar subgraph can be computed in polynomial time.

## Acknowledgement

The authors would like to thank Daniel Lokshtanov for useful comments.

## References

1. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. *Journal of Algorithms* 12, 308–340 (1991)
2. Bodlaender, H.: Chromatic number is NP-complete for cographs and interval graphs. *Information Processing Letters* 31, 135–138 (1989)
3. Bodlaender, H.L., Kloks, T., Kratsch, D.: Treewidth and pathwidth of permutation graphs. *SIAM Journal on Discrete Mathematics* 8, 606–616 (1995)
4. Brandstädt, A., Le, V.B., Spinrad, J.P.: *Graph classes: a survey*. SIAM, Philadelphia (1999)
5. Brown, J.I., Corneil, D.G.: On generalized graph colorings. *Journal of Graph Theory* 11, 87–99 (1987)

6. Chernyak, Z.A., Chernyak, A.A.: About recognizing  $(\alpha, \beta)$ -classes of polar graphs. *Discrete Mathematics* 62, 133–138 (1986)
7. Courcelle, B.: The monadic second-order logic of graphs. III. Tree-decompositions, minors and complexity issues. *RAIRO Inform. Theor. Appl.* 26, 257–286 (1992)
8. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems* 33, 125–150 (2000)
9. Ekim, T.: Polarity of claw-free graphs (submitted)
10. Ekim, T., Hell, P., Stacho, J., de Werra, D.: Polarity of chordal graphs. *Discrete Applied Mathematics* 156, 2469–2479 (2008)
11. Ekim, T., Huang, J.: Recognizing line-polar bipartite graphs (manuscript)
12. Ekim, T., Mahadev, N.V.R., de Werra, D.: Polar cographs. *Discrete Applied Mathematics* 156, 1652–1660 (2008)
13. Feder, T., Hell, P., Klein, S., Motwani, R.: Complexity of graph partition problems. In: *Proceedings of STOC 1999*, pp. 464–472 (1999)
14. Fomin, F.V., Kratsch, D., Novelli, J.-C.: Approximating minimum cocolorings. *Information Processing Letters* 84, 285–290 (2002)
15. Gagarin, A.V.: Chordal  $(1, \beta)$ -polar graphs. *Vestsi Nats. Akad. Navuk Belarusi Ser. Fiz.-Mat. Navuk* 143, 115–118 (1999)
16. Golumbic, M.C.: *Algorithmic Graph Theory and Perfect Graphs*. *Annals of Discrete Mathematics*, vol. 57. North-Holland, Amsterdam (2004)
17. Golumbic, M.C., Rotics, U.: On the Clique-Width of Some Perfect Graph Classes. *International Journal of Foundations of Computer Science* 11, 423–443 (2000)
18. Hammer, P.L., Simeone, B.: The splittance of a graph. *Combinatorica* 1, 275–284 (1981)
19. Hell, P., Klein, S., Nogueira, L.T., Protti, F.: Partitioning chordal graphs into independent sets and cliques. *Discrete Applied Mathematics* 141, 185–194 (2004)
20. Huang, J., Xu, B.: A forbidden subgraph characterization of line-polar bipartite graphs (submitted)
21. Lozin, V.V., Mosca, R.: Polar graphs and maximal independent sets. *Discrete Mathematics* 306, 2901–2908 (2006)
22. McConnell, R.M., Spinrad, J.P.: Modular decomposition and transitive orientation. *Discrete Mathematics* 201, 189–241 (1999)
23. Meister, D.: Computing Treewidth and Minimum Fill-In for Permutation Graphs in Linear Time. In: Kratsch, D. (ed.) *WG 2005*. LNCS, vol. 3787, pp. 91–102. Springer, Heidelberg (2005)
24. Melnikov, O.I., Kozhich, P.P.: Algorithms for recognizing the polarity of a graph with bounded parameters. *Vestsi Akad. Navuk BSSR Ser. Fiz.-Mat. Navuk* 6, 50–54 (1985)
25. Tyshkevich, R.I., Chernyak, A.A.: Decompositions of Graphs. *Cybernetics and System Analysis* 21, 231–242 (1985)
26. Tyshkevich, R.I., Chernyak, A.A.: Algorithms for the canonical decomposition of a graph and recognizing polarity. *Izvestia Akad. Nauk BSSR, ser. Fiz. Mat. Nauk* 6, 16–23 (1985) (in Russian)
27. Wagner, K.: Monotonic coverings of finite sets. *Elektronische Informationsverarbeitung und Kybernetik* 20, 633–639 (1984)

# A New Algorithm for Efficient Pattern Matching with Swaps

Matteo Campanelli<sup>1</sup>, Domenico Cantone<sup>2</sup>, and Simone Faro<sup>2</sup>

<sup>1</sup> Università di Catania, Scuola Superiore di Catania  
Via San Nullo 5/i, I-95123 Catania, Italy

<sup>2</sup> Università di Catania, Dipartimento di Matematica e Informatica  
Viale Andrea Doria 6, I-95125 Catania, Italy  
macampanelli@ssc.unict.it, cantone@dmi.unict.it, faro@dmi.unict.it

**Abstract.** The Pattern Matching problem with Swaps consists in finding all occurrences of a pattern  $P$  in a text  $T$ , when disjoint local swaps in the pattern are allowed.

In this paper, we present a new efficient algorithm for the Swap Matching problem with short patterns. In particular, we devise a  $\mathcal{O}(nm^2)$  general algorithm, named BACKWARD-CROSS-SAMPLING, and show an efficient implementation of it, based on bit-parallelism, which achieves  $\mathcal{O}(nm)$  worst-case time and  $\mathcal{O}(\sigma)$ -space complexity, with patterns whose length  $m$  is comparable to the word-size of the target machine ( $n$  and  $\sigma$  are respectively the size of the text and of the alphabet).

From an extensive comparison with some of the most recent and effective algorithms for the swap matching problem, it turns out that our algorithm is very flexible and achieves very good results in practice.

**Keywords:** pattern matching with swaps, nonstandard pattern matching, combinatorial algorithms on words, design and analysis of algorithms.

## 1 Introduction

The *Pattern Matching problem with Swaps* (Swap Matching problem, for short) is a well-studied variant of the classic Pattern Matching problem. It consists in finding all occurrences, up to character swaps, of a pattern  $P$  of length  $m$  in a text  $T$  of length  $n$ , with  $P$  and  $T$  sequences of characters drawn from a same finite character set  $\Sigma$  of size  $\sigma$ . More precisely, the pattern is said to *swap-match the text at a given location  $j$*  if adjacent pattern characters can be swapped, if necessary, so as to make it identical to the substring of the text ending (or, equivalently, starting) at location  $j$ . All swaps are constrained to be disjoint, i.e., each character can be involved in at most one swap. Moreover, identical adjacent characters are not allowed to be swapped.

This problem is of relevance in practical applications such as text and music retrieval, data mining, and network security, and many others. Following [5], we also mention a particularly important application of the swap matching problem

in biological computing, specifically in the process of translation in molecular biology, with the genetic triplets (otherwise called *codons*). In such application one wants to detect the possible positions of the start and stop codons of an mRNA in a biological sequence and find hints as to where the flanking regions are relative to the translated mRNA region.

The swap matching problem was introduced in 1995 as one of the open problems in nonstandard string matching [10]. The first nontrivial result was reported by Amir *et al.* [1], who provided a  $\mathcal{O}(nm^{\frac{1}{3}} \log m)$ -time in the case of alphabet sets of size 2, showing also that the case of alphabets of size exceeding 2 can be reduced to that of size 2 with a  $\mathcal{O}(\log^2 \sigma)$ -time overhead (subsequently reduced to  $\mathcal{O}(\log \sigma)$  in the journal version [2]). Amir *et al.* [4] studied some rather restrictive cases in which a  $\mathcal{O}(m \log^2 m)$ -time algorithm can be obtained. More recently, Amir *et al.* [3] solved the swap matching problem in  $\mathcal{O}(n \log m \log \sigma)$ -time. We observe that the above solutions are all based on the fast Fourier transform (FFT) technique.

In 2008 the first attempt to provide an efficient solution to the swap matching problem without using the FFT technique has been presented by Iliopoulos and Rahman in [9]. They introduced a new graph-theoretic approach to model the problem and devised an efficient algorithm, based on bit parallelism, which runs in  $\mathcal{O}((n + m) \log m)$ -time, provided that the pattern size is comparable to the word size in the target machine.

More recently, in 2009, Cantone and Faro [7] presented a first approach for solving the swap matching problem with short patterns in linear time. More precisely, they devised a simple algorithm, named CROSS-SAMPLING, which, though characterized by a  $\mathcal{O}(nm)$  worst-case time complexity, admits an efficient implementation based on bit-parallelism, achieving  $\mathcal{O}(n)$  worst-case time and  $\mathcal{O}(\sigma)$  space complexity for short patterns fitting in few machine words.

In this paper, we present a new efficient algorithm for solving the swap matching problem. In particular, we provide a  $\mathcal{O}(nm^2)$  general algorithm, named BACKWARD-CROSS-SAMPLING algorithm, which inherits much the same iterative structure of the CROSS-SAMPLING algorithm, but is based on a right-to-left scan of the text, giving better results in practice. We will also describe an efficient implementation of the algorithm, characterized by a  $\mathcal{O}(nm)$  worst-case time and  $\mathcal{O}(\sigma)$ -space complexity, for patterns of length comparable to the word size of the target machine.

The rest of the paper is organized as follows. In Section 2 we recall some preliminary definitions. Section 3 describes the CROSS-SAMPLING algorithm and its bit-parallel variant. In Section 4 we present the BACKWARD-CROSS-SAMPLING algorithm for the swap matching problem and then, in Section 5, we illustrate an efficient implementation of it based on bit-parallelism. Results of an extensive experimental comparison under various conditions with the most efficient algorithms present in the literature are reported in Section 6. Finally, we will briefly draw our conclusions in Section 7.

## 2 Notions and Basic Definitions

A string  $P$  of length  $m \geq 0$  is represented as a finite array  $P[0..m-1]$ . In such a case we also write  $\text{length}(P) = m$ . In particular, for  $m = 0$  we obtain the empty string, denoted by  $\varepsilon$ . We denote by  $P[i]$  the  $(i+1)$ -st character of  $P$ , for  $0 \leq i < \text{length}(P)$ . Likewise, we denote by  $P[i..j]$  the substring of  $P$  contained between the  $(i+1)$ -st and the  $(j+1)$ -st characters of  $P$ , for  $0 \leq i \leq j < \text{length}(P)$ . A  $h$ -substring of a string  $S$  is a substring of  $S$  of length  $h$ . For any two strings  $P$  and  $P'$ , we say that  $P'$  is a suffix of  $P$  if  $P' = P[i.. \text{length}(P) - 1]$ , for some  $0 \leq i < \text{length}(P)$ . Similarly, we say that  $P'$  is a prefix of  $P$  if  $P' = P[0..i-1]$ , for some  $0 \leq i \leq \text{length}(P)$ . We denote by  $P_i$  the nonempty prefix  $P[0..i]$  of  $P$  of length  $i+1$ , for  $0 \leq i < m$ . If  $i < 0$ , we convene that  $P_i$  is the empty string  $\varepsilon$ . Moreover we say that  $P'$  is a proper prefix (suffix) of  $P$  if  $P'$  is a prefix (suffix) of  $P$  and  $|P'| < |P|$ . Finally, we write  $P.P'$  to denote the concatenation of  $P$  and  $P'$ .

**Definition 1.** A swap permutation for a string  $P$  of length  $m$  is a permutation  $\pi : \{0, \dots, m-1\} \rightarrow \{0, \dots, m-1\}$  such that:

- (a) if  $\pi(i) = j$  then  $\pi(j) = i$  (characters at positions  $i$  and  $j$  are swapped);
- (b) for all  $i$ ,  $\pi(i) \in \{i-1, i, i+1\}$  (only adjacent characters are swapped);
- (c) if  $\pi(i) \neq i$  then  $P[\pi(i)] \neq P[i]$  (identical characters are not swapped).

For a given string  $P$  and a swap permutation  $\pi$  for  $P$ , we write  $\pi(P)$  to denote the swapped version of  $P$ , namely  $\pi(P) = P[\pi(0)].P[\pi(1)].\dots.P[\pi(m-1)]$ .

**Definition 2.** Given a text  $T$  of length  $n$  and a pattern  $P$  of length  $m$ ,  $P$  is said to swap-match (or to have a swapped occurrence) at location  $j \geq m-1$  of  $T$  if there exists a swap permutation  $\pi$  of  $P$  such that  $\pi(P)$  matches  $T$  at location  $j$ , i.e.,  $\pi(P) = T[j-m+1..j]$ . In such a case we write  $P \propto T_j$ .

**Definition 3 (Pattern Matching Problem with Swaps).** Given a text  $T$  of length  $n$  and a pattern  $P$  of length  $m$ , find all locations  $j \in \{m-1, \dots, n-1\}$  such that  $P$  swap-matches with  $T$  at location  $j$ , i.e.,  $P \propto T_j$ .

The following elementary result will be used later.

**Lemma 1** ([\[7\]](#)). Let  $P$  and  $R$  be strings of length  $m$  over an alphabet  $\Sigma$  and suppose that there exists a swap permutation  $\pi$  such that  $\pi(P) = R$ . Then  $\pi$  is unique. ■

**Corollary 1.** Given a text  $T$  of length  $n$  and a pattern  $P$  of length  $m$ , if  $P \propto T_j$ , for a given position  $j \in \{m-1, \dots, n-1\}$ , then there exists a unique swapped occurrence of  $P$  in  $T$  ending at position  $j$ . ■



### 3 The Cross-Sampling Algorithm

The CROSS-SAMPLING algorithm [7] computes the swap occurrences of all prefixes of a pattern  $P$  (of length  $m$ ) in continuously increasing prefixes of a text  $T$  (of length  $n$ ), using a dynamic programming approach. More precisely, during its  $(j + 1)$ -st iteration, for  $j = 0, 1, \dots, n - 1$ , we establish whether  $P_i \propto T_j$ , for each  $i = 0, 1, \dots, m - 1$ , by exploiting information gathered during previous iterations. To this end, if we put

$$\begin{aligned} \lambda_j &= \begin{cases} \{0\} & \text{if } P[0] = T[j] \\ \emptyset & \text{otherwise,} \end{cases} && \text{for } 0 \leq j \leq n - 1 \\ \mathcal{S}_j &= \{0 \leq i \leq m - 1 : P_i \propto T_j\}, && \text{for } 0 \leq j \leq n - 1 \\ \mathcal{S}'_j &= \{0 \leq i < m - 1 : P_{i-1} \propto T_{j-1} \text{ and } P[i] = T[j + 1]\} && \text{for } 1 \leq j \leq n - 1. \end{aligned}$$

then the following recurrences hold:

$$\begin{aligned} \mathcal{S}_{j+1} &= \{i \leq m - 1 : ((i - 1) \in \mathcal{S}_j \text{ and } P[i] = T[j + 1]) \text{ or} \\ &\quad ((i - 1) \in \mathcal{S}'_j \text{ and } P[i] = T[j])\} \cup \lambda_{j+1} \\ \mathcal{S}'_{j+1} &= \{i < m - 1 : (i - 1) \in \mathcal{S}_j \text{ and } P[i] = T[j + 2]\} \cup \lambda_{j+2}, \end{aligned} \tag{1}$$

where the base cases are given by  $\mathcal{S}_0 = \lambda_0$  and  $\mathcal{S}'_0 = \lambda_1$ .

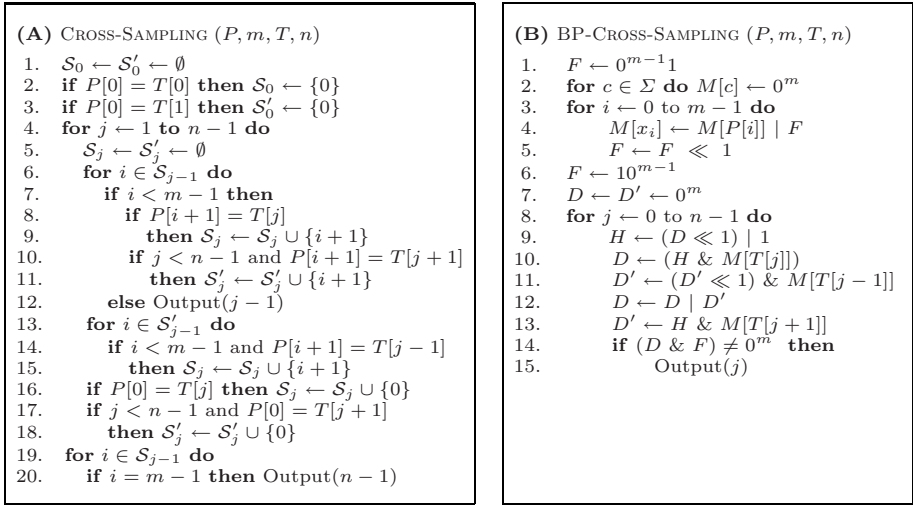
Such relations allow one to compute the sets  $\mathcal{S}_j$  and  $\mathcal{S}'_j$  in an iterative fashion, where  $\mathcal{S}_{j+1}$  is computed in terms of both  $\mathcal{S}_j$  and  $\mathcal{S}'_j$ , whereas  $\mathcal{S}'_{j+1}$  needs only  $\mathcal{S}_j$  for its computation. The resulting dependency graph has a doubly crossed structure, from which the name of the algorithm of Fig. 1, CROSS-SAMPLING, for the swap matching problem. Plainly, the time complexity of the CROSS-SAMPLING algorithm is  $\mathcal{O}(nm)$ .

[7] presents also an efficient implementation of the CROSS-SAMPLING algorithm based on the bit-parallelism technique [6], called BP-CROSS-SAMPLING algorithm. We recall that the bit-parallelism technique takes advantage of the intrinsic parallelism of the bit operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor of at most  $w$ , where  $w$  is the number of bits in the computer word.

The BP-CROSS-SAMPLING algorithm uses a representation of the sets  $\mathcal{S}_j$  and  $\mathcal{S}'_j$  as lists of  $m$  bits,  $D_j$  and  $D'_j$  respectively ( $m$  is the length of the pattern). The  $i$ -th bit of  $D_j$  is set to 1 if  $i \in \mathcal{S}_j$ , i.e., if  $P_i \propto T_j$ , whereas the  $i$ -th bit of  $D'_j$  is set to 1 if  $i \in \mathcal{S}'_j$ , i.e., if  $P_{i-1} \propto T_{j-1}$  and  $P[i] = T[j + 1]$ . The remaining bits are set to 0. Notice that if  $m \leq w$ , each list fits completely in a single computer word, whereas if  $m > w$  we need  $\lceil m/w \rceil$  computer words to represent each of the sets  $\mathcal{S}_j$  and  $\mathcal{S}'_j$ .

For each character  $c$  of the alphabet  $\Sigma$ , the algorithm maintains a bit mask  $M[c]$ , whose  $i$ -th bit is set to 1 if  $P[i] = c$ .

The bit vectors are initialized to  $0^m$ . Then the algorithm scans the text from left to right and, for each position  $j \geq 0$ , it computes the bit vector  $D_j$  in terms of  $D_{j-1}$  and  $D'_{j-1}$ , by performing the following bitwise operations:



**Fig. 1.** (A) The CROSS-SAMPLING algorithm for solving the swap matching problem. (B) The BP-CROSS-SAMPLING algorithm based on bit-parallelism.

$$\begin{array}{ll}
 D_j \leftarrow D_{j-1} \ll 1 & S_j = \{i : (i - 1) \in S_{j-1}\} \\
 D_j \leftarrow D_j \mid 1 & S_j = S_j \cup \{0\} \\
 D_j \leftarrow D_j \& M[T[j]] & S_j = S_j \setminus \{i : P[i] \neq T[j]\} \\
 D_j \leftarrow D_j \mid H^1 & S_j = S_j \cup \{i : (i - 1) \in S'_{j-1} \wedge P[i] = T[j - 1]\},
 \end{array}$$

where  $H^1 = ((D'_{j-1} \ll 1) \& M[T[j - 1]])$ .

Similarly, the bit vector  $D'_j$  is computed in the  $j$ -th iteration of the algorithm in terms of  $D_{j-1}$ , by performing the following bitwise operations:

$$\begin{array}{ll}
 D'_j \leftarrow D_{j-1} \ll 1 & S'_j = \{i : (i - 1) \in S_{j-1}\} \\
 D'_j \leftarrow D'_j \mid 1 & S'_j = S'_j \cup \{0\} \\
 D'_j \leftarrow D'_j \& M[T[j + 1]] & S'_j = S'_j \setminus \{i : P[i] \neq T[j + 1]\}.
 \end{array}$$

During the  $j$ -th iteration of the algorithm, if the leftmost bit of  $D_j$  is set to 1, i.e. if  $(D_j \& 10^{m-1}) \neq 0^m$ , a swap match is reported at position  $j$ .

The code of the BP-CROSS-SAMPLING algorithm is shown in Fig. 1(B). It achieves a  $\mathcal{O}(\lceil mn/w \rceil)$  worst-case time complexity and requires  $\mathcal{O}(\sigma \lceil m/w \rceil)$  extra space, where  $\sigma$  is the size of the alphabet. If  $m \leq w$ , then the algorithm requires  $\mathcal{O}(n)$ -time and  $\mathcal{O}(\sigma)$  extra space.

## 4 The Backward-Cross-Sampling Algorithm

In this section we present a new practical algorithm for solving the swap matching problem, called BACKWARD-CROSS-SAMPLING.

The new algorithm inherits from the CROSS-SAMPLING algorithm the same doubly crossed structure in its iterative computation. However, it searches for

all occurrences of the pattern in the text by scanning characters from right to left, as in the Backward DAWG Matching (BDM) algorithm for the exact single pattern matching problem [8].

The BDM algorithm processes the pattern by constructing a *directed acyclic word graph* (DAWG) of the reversed pattern. The text is processed in windows of size  $m$ , which are searched for the longest prefix of the pattern from right to left by means of the DAWG. At the end of each search phase, either a longest prefix or a match is found. If no match is found, the window is shifted to the start position of the longest prefix, otherwise it is shifted to the start position of the second longest prefix.

As in the BDM algorithm, the BACKWARD-CROSS-SAMPLING algorithm processes the text in windows of size  $m$ . Each attempt is identified by the last position,  $j$ , of the current window of the text. The window is searched for the longest prefix of the pattern which has a swapped occurrence ending at position  $j$  of the text. At the end of each attempt, a new value of  $j$  is computed by performing a safe shift to the right of the current window in such a way to left-align the current window of the text with the longest prefix matched in the previous attempt.

To this end, for any given position  $j$  in the text  $T$ , we let  $S_j^h$  denote the set of the integral values  $i$  such that the  $h$ -substring of  $P$  ending at position  $i$  has a swapped occurrence ending at position  $j$  of the text  $T$ . More formally, we have

$$S_j^h =_{\text{Def}} \{h - 1 \leq i \leq m - 1 : P[i - h + 1 .. i] \propto T_j\},$$

for  $0 \leq j < n$  and  $0 \leq h \leq m$ .

If  $h - 1 \in S_j^h$ , then there is a swapped occurrence of the prefix of the pattern of length  $h$ , i.e.,  $P[0 .. h - 1] \propto T_j$ . In addition, it turns out that  $P$  has a swapped occurrence at location  $j$  of  $T$  if and only if  $S_j^m \neq \emptyset$ . Indeed, if  $S_j^m \neq \emptyset$  then  $S_j^m = \{m - 1\}$ , for any given position  $j$  in the text.

The sets  $S_j^h$  can be computed efficiently by a dynamic programming algorithm, by exploiting the following very elementary property.

**Lemma 2.** *Let  $T$  and  $P$  be a text of length  $n$  and a pattern of length  $m$ , respectively. Then, for each  $0 \leq j < n$ ,  $0 \leq h \leq m$ , and  $h - 1 \leq i < m$  we have that  $P[i - h + 1 .. i] \propto T_j$  if and only if one of the following two facts holds*

- $P[i - h + 2 .. i] \propto T_j$  and  $P[i - h + 1] = T[j - h + 1]$ ;
- $P[i - h + 3 .. i] \propto T_j$ ,  $P[i - h + 1] = T[j - h + 2]$ , and  $P[i - h + 2] = T[j - h + 1]$ . ■

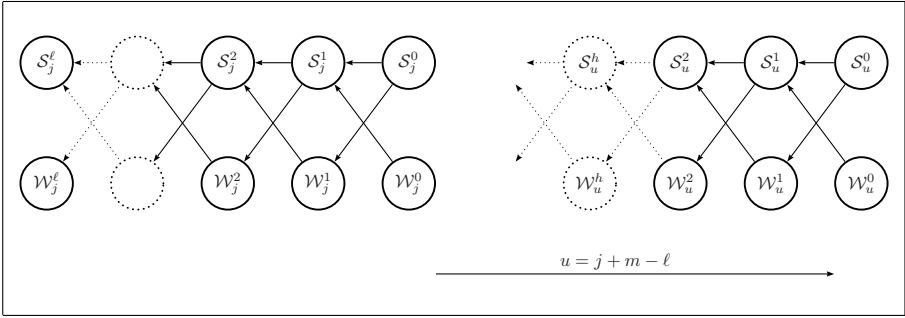
Let us denote by  $\mathcal{W}_j^h$ , for  $0 \leq j < n$  and  $0 \leq h < m$ , the collection of all values  $i$  such that  $P[i - h + 1] = T[j - h]$  and the  $(h - 1)$ -substring ending at position  $i$  of  $P$  has a swapped occurrence ending at location  $j$  of the text  $T$ .

More formally

$$\mathcal{W}_j^h =_{\text{Def}} \{h \leq i < m - 1 : P[i - h + 2 .. i] \propto T_j \text{ and } P[i - h + 1] = T[j - h]\}.$$

For any given position  $j$  in the text, the base case for  $h = 0$  is given by

$$S_j^0 = \{i : 0 \leq i < m\} \quad \text{and} \quad \mathcal{W}_j^0 = \{0 \leq i < m - 1 : P[i + 1] = T[j]\}. \quad (2)$$



**Fig. 2.** A graphic representation of the iterative pattern for computing sets  $\mathcal{S}_j^h$  and  $\mathcal{W}_j^h$  for increasing values of  $h$ . A first attempt, starting at position  $j$  of the text, ends with  $h = \ell$ . The subsequent attempt starts at position  $u = j + m - \ell$ .

Additionally, Lemma 2 justifies the following recursive definitions of the sets  $\mathcal{S}_j^{h+1}$  and  $\mathcal{W}_j^{h+1}$  in terms of  $\mathcal{S}_j^h$  and  $\mathcal{W}_j^h$ , for  $0 \leq j < n$  and  $0 \leq h < m$ :

$$\begin{aligned} \mathcal{S}_j^{h+1} &= \{h - 1 \leq i \leq m - 1 : (i \in \mathcal{S}_j^h \text{ and } P[i - h] = T[j - h]) \text{ or} \\ &\quad (i \in \mathcal{W}_j^h \text{ and } P[i - h] = T[j - h + 1])\} \quad (3) \\ \mathcal{W}_j^{h+1} &= \{h \leq i \leq m - 1 : i \in \mathcal{S}_j^h \text{ and } P[i - h] = T[j - h - 1]\}. \end{aligned}$$

Such relations, coupled with the initial conditions (2), allow one to compute the sets  $\mathcal{S}_j^h$  and  $\mathcal{W}_j^h$  in an iterative fashion as shown in Fig. 2.

The code of the BACKWARD-CROSS-SAMPLING algorithm is shown in Fig. 3(A). For any attempt at position  $j$  of the text, we denote by  $\ell$  the length of the longest prefix matched in the current attempt. Then the algorithm starts its computation with  $j = m - 1$  and  $\ell = 0$ . During each attempt, the window of the text is scanned from right to left, for  $h = 1$  to  $m$ . If, for a given value of  $h$ , the algorithm states that element  $(h - 1) \in \mathcal{S}_j^h$  then  $\ell$  is updated to value  $h$ .

The algorithm is not able to remember the characters read in previous iterations. Thus, an attempt ends successfully when  $h$  reaches the value  $m$  (a match is found), or unsuccessfully when both sets  $\mathcal{S}_j^h$  and  $\mathcal{W}_j^h$  are empty. In any case, at the end of each attempt, the start position of the window, i.e., position  $j - m + 1$  in the text, can be shifted to the start position of the longest proper prefix detected during the backward scan. Thus the window is advanced  $m - \ell$  positions to the right. Observe that since  $\ell < m$ , we plainly have that  $m - \ell > 0$ .

Moreover, in order to avoid accessing the text character of position  $j - h + 1 = n$ , when  $j = n - 1$  and  $h = 0$ , the algorithm benefits of the introduction of a sentinel character at the end of the text.

To compute the worst-case time complexity of the algorithm, preliminarily we observe that, since the algorithm does not remember the length of the prefix matched in previous attempts, each character of the text is processed at most  $m$  times during the searching phase. Thus the while-cycle of line 7 is executed  $\mathcal{O}(nm)$  times. The for-cycles of line 9 and line 14 are executed  $|\mathcal{S}_j^h|$  and  $|\mathcal{W}_j^h|$  times, respectively. However, according to Lemma 1, for each position  $j$  of the

<p><b>(A)</b> BACKWARD-CROSS-SAMPLING (<math>P, m, T, n</math>)</p> <ol style="list-style-type: none"> <li>1. <math>T[n] \leftarrow P[0]</math></li> <li>2. <math>j \leftarrow m - 1</math></li> <li>3. <b>while</b> <math>j &lt; n</math> <b>do</b></li> <li>4.   <math>h \leftarrow 0</math></li> <li>5.   <math>\mathcal{S}_j^0 \leftarrow \{i : 0 \leq i &lt; m\}</math></li> <li>6.   <math>\mathcal{W}_j^0 \leftarrow \{0 \leq i &lt; m - 1 : P[i + 1] = T[j]\}</math></li> <li>7.   <b>while</b> <math>h &lt; m</math> and <math>\mathcal{S}_j^h \cup \mathcal{W}_j^h \neq \emptyset</math> <b>do</b></li> <li>8.     <b>if</b> <math>(h - 1) \in \mathcal{S}_j^h</math> <b>then</b> <math>\ell \leftarrow h</math></li> <li>9.     <b>for each</b> <math>i \in \mathcal{S}_j^h</math> <b>do</b></li> <li>10.       <b>if</b> <math>i \geq h</math> and <math>P[i - h] = T[j - h]</math></li> <li>11.         <b>then</b> <math>\mathcal{S}_j^{h+1} \leftarrow \mathcal{S}_j^{h+1} \cup \{i\}</math></li> <li>12.       <b>if</b> <math>i &gt; h</math> and <math>P[i - h] = T[j - h - 1]</math></li> <li>13.         <b>then</b> <math>\mathcal{W}_j^{h+1} \leftarrow \mathcal{W}_j^{h+1} \cup \{i\}</math></li> <li>14.     <b>for each</b> <math>i \in \mathcal{W}_j^h</math> <b>do</b></li> <li>15.       <b>if</b> <math>i \geq h</math> and <math>P[i - h] = T[j - h + 1]</math></li> <li>16.         <b>then</b> <math>\mathcal{S}_j^{h+1} \leftarrow \mathcal{S}_j^{h+1} \cup \{i\}</math></li> <li>17.     <math>h \leftarrow h + 1</math></li> <li>18.   <b>if</b> <math>(h - 1) \in \mathcal{S}_j^h</math> <b>then</b> Output(<math>j</math>)</li> <li>19.   <math>j \leftarrow j + m - \ell</math></li> </ol>	<p><b>(B)</b> BP-BACKWARD-CROSS-SAMPLING (<math>P, m, T, n</math>)</p> <ol style="list-style-type: none"> <li>1. <math>F \leftarrow 10^{m-1}</math></li> <li>2. <b>for</b> <math>c \in \Sigma</math> <b>do</b> <math>M[c] \leftarrow 0^m</math></li> <li>3. <b>for</b> <math>i \leftarrow 0</math> to <math>m - 1</math> <b>do</b></li> <li>4.   <math>M[P[i]] \leftarrow M[P[i]] \mid F</math></li> <li>5.   <math>F \leftarrow F \gg 1</math></li> <li>6. <math>T[n] \leftarrow P[0]</math></li> <li>7. <math>j \leftarrow m - 1</math></li> <li>8. <math>F \leftarrow 10^{m-1}</math></li> <li>9. <b>while</b> <math>j &lt; n</math> <b>do</b></li> <li>10.   <math>h \leftarrow 1, \ell \leftarrow 0</math></li> <li>11.   <math>D \leftarrow M[T[j]]</math></li> <li>12.   <math>D \leftarrow D \mid (M[T[j + 1]] \&amp; (M[T[j]] \ll 1))</math></li> <li>13.   <math>C \leftarrow M[T[j - 1]]</math></li> <li>14.   <b>while</b> <math>h &lt; m</math> and <math>(D \mid C) \neq 0</math> <b>do</b></li> <li>15.     <b>if</b> <math>F \&amp; D \neq 0</math> <b>then</b> <math>\ell \leftarrow h</math></li> <li>16.     <math>H \leftarrow (C \ll 1) \&amp; M[T[j - h + 1]]</math></li> <li>17.     <math>C \leftarrow (D \ll 1) \&amp; M[T[j - h - 1]]</math></li> <li>18.     <math>D \leftarrow (D \ll 1) \&amp; M[T[j - h]]</math></li> <li>19.     <math>D \leftarrow D \mid H</math></li> <li>20.     <math>h \leftarrow h + 1</math></li> <li>21.   <b>if</b> <math>D \neq 0</math> <b>then</b> Output(<math>j</math>)</li> <li>22.   <math>j \leftarrow j + m - \ell</math></li> </ol>
--	--

**Fig. 3.** (A) The BACKWARD-CROSS-SAMPLING algorithm for the swap matching problem. (B) The BP-BACKWARD-CROSS-SAMPLING algorithm (based on bit-parallelism).

text we can report only a single swapped occurrence of the substring  $P[i - h + 1 \dots i]$  in  $T_j$ , for each  $h - 1 \leq i < m$ , which implies that  $|\mathcal{S}_j^h| \leq m$  and  $|\mathcal{W}_j^h| < m$ .

Therefore the BACKWARD-CROSS-SAMPLING algorithm has a  $\mathcal{O}(nm^2)$ -time complexity and requires  $\mathcal{O}(m)$  extra space to represent the sets  $\mathcal{S}_j^h$  and  $\mathcal{W}_j^h$ .

## 5 The BP-Backward-Cross-Sampling Algorithm

In this section we present a practical implementation of the BACKWARD-CROSS-SAMPLING algorithm based on the bit-parallelism technique [6]. The resulting algorithm works as the BNDM (Backward Nondeterministic DAWG Match) algorithm [11], which is a bit-parallel implementation of the BDM algorithm, where the simulation of a nondeterministic automaton takes place by updating the state vector much as in the Shift-And algorithm [6].

In the bit-parallel variant of the BACKWARD-CROSS-SAMPLING algorithm, the sets  $\mathcal{S}_j^h$  and  $\mathcal{W}_j^h$  are represented as lists of  $m$  bits,  $D_j^h$  and  $C_j^h$  respectively.

The  $(i - h + 1)$ -th bit of  $D_j^h$  is set to 1 if  $i \in \mathcal{S}_j^h$ , i.e., if  $P[i - h + 1 \dots i] \propto T_j$ , whereas the  $(i - h + 1)$ -th bit of  $C_j^h$  is set to 1 if  $i \in \mathcal{W}_j^h$ , i.e., if  $P[i - h + 2 \dots i] \propto T_j$  and  $P[i - h + 1] = T[j - h]$ . All remaining bits are set to 0. Notice that if  $m \leq w$ , each bit vector fits in a single computer word, whereas if  $m > w$  we need  $\lceil m/w \rceil$  computer words to represent each of the sets  $\mathcal{S}_j^h$  and  $\mathcal{W}_j^h$ .

For each character  $c$  of the alphabet  $\Sigma$ , the algorithm maintains a bit mask  $M[c]$  whose  $i$ -th bit is set to 1 if  $P[i] = c$ .

As in the BACKWARD-CROSS-SAMPLING algorithm, the text is processed in windows of size  $m$ , identified by the last position  $j$ , and the first attempt starts at position  $j = m - 1$ . For any searching attempt at location  $j$  of the text, the bit vectors  $D_j^1$  and  $C_j^1$  are initialized to  $M[T[j]] \mid (M[T[j+1]] \& (M[T[j]] \ll 1))$  and  $M[T[j-1]]$ , respectively, according to the base cases shown in (2) and recursive expressions shown in (3). Then the current window of the text, i.e.  $T[j-m+1..j]$ , is scanned from right to left, by reading character  $T[j-h+1]$ , for increasing values of  $h$ . Namely, for each value of  $h > 1$ , the bit vector  $D_j^{h+1}$  is computed in terms of  $D_j^h$  and  $C_j^h$ , by performing the following bitwise operations:

- (a)  $D_j^{h+1} \leftarrow (D_j^h \ll 1) \& M[T[j-h]]$ ,
- (b)  $D_j^{h+1} \leftarrow D_j^{h+1} \mid ((C_j^h \ll 1) \& M[T[j-h+1]])$ .

Concerning (a), by a left shift of  $D_j^h$ , all elements of  $\mathcal{S}_j^h$  are added to the set  $\mathcal{S}_j^{h+1}$ . Then, by performing a bitwise and with the mask  $M[T[j-h]]$ , all elements  $i$  such that  $P[i-h] \neq T[j-h]$  are removed from  $\mathcal{S}_j^{h+1}$ . Similarly, the bit operations in (b) have the effect to add to  $\mathcal{S}_j^{h+1}$  all elements  $i$  in  $\mathcal{W}_j^h$  such that  $P[i-h] = T[j-h+1]$ . Formally, we have the following correspondence:

- (a')  $\mathcal{S}_j^{h+1} \leftarrow \mathcal{S}_j^h \setminus \{i \in \mathcal{S}_j^h : P[i-h] \neq T[j-h]\}$ ,
- (b')  $\mathcal{S}_j^{h+1} \leftarrow \mathcal{S}_j^{h+1} \cup \mathcal{W}_j^h \setminus \{i \in \mathcal{W}_j^h : P[i-h] \neq T[j-h+1]\}$ .

Similarly, the bit vector  $C_j^{h+1}$  is computed in terms of  $D_j^h$ , by performing the following bitwise operations:

- (c)  $C_j^{h+1} \leftarrow (D_j^h \ll 1) \& M[T[j-h-1]]$

which have the effect to add to the set  $\mathcal{W}_j^{h+1}$  all elements of the set  $\mathcal{S}_j^h$  (by shifting  $D_j^h$  to the left by one position) and to remove all elements  $i$  such  $P[i] \neq T[j-h-1]$  holds (by a bitwise and with the mask  $M[T[j-h-1]]$ ).

More formally, we have the following symbolic correspondence:

- (c')  $\mathcal{W}_j^{h+1} \leftarrow \mathcal{S}_j^h \setminus \{i \in \mathcal{S}_j^h : P[i-h] \neq T[j-h-1]\}$ .

As in the BACKWARD-CROSS-SAMPLING algorithm, an attempt ends when  $h = m$  or  $(D_j^h \mid C_j^h) = 0$ . If  $h = m$  and  $D_j^h \neq 0$ , a swap match at position  $j$  of the text is reported. In any case, if  $h < m$  is the largest value such that  $D_j^h \neq 0$ , then a prefix of the pattern, of length  $\ell = h$ , which has a swapped occurrence ending at position  $j$  of the text, has been found. Thus a safe shift of  $m - \ell$  position to the right can take place.

In practice, we can use just two vectors to implement the sets  $D_j^h$  and  $C_j^h$ . Thus, during the  $h$ -th iteration of the algorithm at a given location  $j$  of the text, vector  $D_j^h$  is transformed into vector  $D_j^{h+1}$  and vector  $C_j^h$  is transformed into vector  $C_j^{h+1}$ . The resulting BP-BACKWARD-CROSS-SAMPLING algorithm is shown in Fig. 3(B). It achieves a  $\mathcal{O}(\lceil nm^2/w \rceil)$  worst-case time complexity and requires  $\mathcal{O}(\sigma \lceil m/w \rceil)$  extra space, where  $\sigma$  is the alphabet size. If the length of the pattern is  $m \leq w$ , then the algorithm finds all swapped matches in  $\mathcal{O}(nm)$  time and  $\mathcal{O}(\sigma)$  extra space.

## 6 Experimental Results

Next we present experimental data which allow to compare under various conditions the following string matching algorithms in terms of their running times:

- ILIOPOULOS-RAHMAN algorithm (IR)
- CROSS-SAMPLING algorithm (CS)
- BP-CROSS-SAMPLING algorithm (BPCS)
- BACKWARD-CROSS-SAMPLING algorithm (BCS)
- BP-BACKWARD-CROSS-SAMPLING algorithm (BPBCS)

We have chosen to exclude from our experimental comparison the Naive algorithm and all algorithms based on the FFT technique, since the overhead of such algorithms is quite high, resulting in very bad performances.

All algorithms have been implemented in the C programming language and were used to search for the same strings in large fixed text buffers on a PC with Intel Pentium M processor of 1.7GHz and a memory of 512Mb. In particular, all algorithms have been tested on three  $\text{Rand}\sigma$  problems, for  $\sigma = 8, 32,$  and  $128,$  on a genome, on a protein sequence, and on a natural language text buffer, with patterns of length  $m = 4, 8, 12, 16, 20, 24, 28, 32.$  In the tables below, running times have been expressed in hundredths of seconds and the best results are bold-faced.

### Running Times for Random Problems

In the case of random texts, the algorithms have been tested on three  $\text{Rand}\sigma$  problems. Each  $\text{Rand}\sigma$  problem consists in searching a set of 400 random patterns of a given length in a 4Mb random text over a common alphabet of size  $\sigma,$  with a uniform character distribution.

Running times for a  $\text{Rand}8$  problem

$m$	4	8	12	16	20	24	28	32
IR	<b>3.450</b>	3.420	3.420	3.440	3.580	3.560	3.520	3.560
CS	66.670	67.210	67.230	67.590	67.850	68.280	68.670	69.060
BPCS	3.960	3.900	3.890	3.900	3.920	3.900	3.930	3.910
BCS	62.130	41.160	33.700	29.480	26.750	24.870	23.700	22.450
BPBCS	4.140	<b>2.000</b>	<b>1.850</b>	<b>1.180</b>	<b>1.110</b>	<b>1.000</b>	<b>0.910</b>	<b>0.800</b>

Running times for a  $\text{Rand}32$  problem

$m$	4	8	12	16	20	24	28	32
IR	2.920	2.950	2.930	2.940	2.940	2.930	2.950	2.950
CS	60.030	59.760	59.740	59.710	59.610	59.580	59.350	59.200
BPCS	3.030	3.050	3.040	3.080	3.040	3.060	3.080	3.060
BCS	46.200	29.050	23.750	20.540	18.640	17.380	16.180	15.660
BPBCS	<b>2.650</b>	<b>1.930</b>	<b>1.050</b>	<b>1.000</b>	<b>0.820</b>	<b>0.600</b>	<b>0.380</b>	<b>0.240</b>

Running times for a  $\text{Rand}128$  problem

$m$	4	8	12	16	20	24	28	32
IR	3.550	3.610	3.610	3.590	3.630	3.650	3.660	3.640
CS	59.910	59.610	59.460	59.390	59.380	59.130	59.180	59.130
BPCS	3.120	3.150	3.160	3.160	3.140	3.110	3.130	3.130
BCS	42.720	25.750	20.130	17.720	15.950	14.650	13.990	13.310
BPBCS	<b>2.000</b>	<b>1.040</b>	<b>0.960</b>	<b>0.750</b>	<b>0.580</b>	<b>0.390</b>	<b>0.250</b>	<b>0.180</b>

The experimental results show that the BPBCS algorithm obtains the best run-time performance in most cases. In particular, for very short patterns and small alphabets, our algorithm is second only to the IR algorithm. We notice that IR, CS, and BPCS show a linear behavior, whereas BCS and BPBCS are characterized by a decreasing trend. Observe moreover that, in the case of small alphabets and pattern longer than 16 characters, the BPBCS algorithm is at least three times faster than BPCS and IR. Such a relation increases to thirty times for large alphabets.

### Running Times for Real World Problems

The tests on real world problems have been performed on a genome sequence, on a protein sequence, and on a natural language text buffer. The genome used is a sequence of 4,638,690 base pairs of *Escherichia coli*, taken from the file E.coli of the Large Canterbury Corpus<sup>1</sup>. The protein sequence used in the tests is a 2.4Mb file with 22 different characters from the human genome. Finally, as natural language text buffer we used the file world192.txt (The CIA World Fact Book) from the Large Canterbury Corpus, which contains 2,473,400 characters drawn from an alphabet of 93 different characters.

Running times for a genome sequence ( $\sigma = 4$ )

$m$	4	8	12	16	20	24	28	32
IR	<b>3.070</b>	<b>3.060</b>	3.070	3.080	3.100	3.150	3.150	3.100
CS	83.020	79.930	79.760	79.380	79.350	79.430	79.500	79.460
BPCS	6.820	3.950	3.910	3.920	3.930	3.920	3.930	3.940
BCS	102.410	67.010	55.480	49.050	45.250	42.290	40.260	38.650
BPBCS	10.170	3.930	<b>2.640</b>	<b>2.010</b>	<b>1.960</b>	<b>1.830</b>	<b>1.510</b>	<b>1.120</b>

Running times for a protein sequence ( $\sigma = 22$ )

$m$	4	8	12	16	20	24	28	32
IR	<b>1.990</b>	2.000	1.990	2.000	1.990	2.000	1.990	1.990
CS	45.190	45.230	45.490	45.650	45.900	46.040	46.400	44.400
BPCS	2.030	2.010	2.020	2.050	2.040	2.030	2.040	2.020
BCS	31.110	22.450	18.620	16.430	15.130	14.090	13.450	12.670
BPBCS	2.130	<b>1.180</b>	<b>0.950</b>	<b>0.590</b>	<b>0.270</b>	<b>0.120</b>	<b>0.070</b>	<b>0.070</b>

Running times for a natural language text buffer ( $\sigma = 93$ )

$m$	4	8	12	16	20	24	28	32
IR	<b>1.850</b>	1.820	1.820	1.850	1.880	1.820	1.860	1.850
CS	36.950	36.680	36.520	36.410	36.230	36.120	36.080	36.210
BPCS	2.050	1.970	1.970	1.990	1.970	1.980	1.990	1.980
BCS	30.410	19.390	15.720	13.640	12.350	11.430	10.820	10.320
BPBCS	2.000	<b>0.990</b>	<b>0.610</b>	<b>0.210</b>	<b>0.050</b>	<b>0.020</b>	<b>0.013</b>	<b>0.010</b>

The above experimental results show that in most cases the BPBCS algorithm obtains the best results and only sporadically is second to the IR algorithm. Moreover, in the case of natural language texts and long patterns, the BPBCS algorithm is about 100 times faster than the IR algorithm.

<sup>1</sup> <http://www.data-compression.info/Corpora/CanterburyCorpus/>



## 7 Conclusions

In this paper we have presented a new efficient algorithm for the Swap Matching problem with short patterns. In particular, we have devised a  $\mathcal{O}(nm^2)$  general algorithm, named BACKWARD-CROSS-SAMPLING, and have provided an efficient implementation of it, based on bit-parallelism.

An extensive experimental comparisons showed that our algorithm is very fast in practice and obtains the best results in most cases, especially with long patterns and large alphabets.

## References

1. Amir, A., Aumann, Y., Landau, G.M., Lewenstein, M., Lewenstein, N.: Pattern matching with swaps. In: IEEE Symposium on Foundations of Computer Science, pp. 144–153 (1997)
2. Amir, A., Aumann, Y., Landau, G.M., Lewenstein, M., Lewenstein, N.: Pattern matching with swaps. *Journal of Algorithms* 37(2), 247–266 (2000)
3. Amir, A., Cole, R., Hariharan, R., Lewenstein, M., Porat, E.: Overlap matching. *Inf. Comput.* 181(1), 57–74 (2003)
4. Amir, A., Landau, G.M., Lewenstein, M., Lewenstein, N.: Efficient special cases of pattern matching with swaps. *Information Processing Letters* 68(3), 125–132 (1998)
5. Antoniou, P., Iliopoulos, C.S., Jayasekera, I., Rahman, M.S.: Implementation of a swap matching algorithm using a graph theoretic model. In: Elloumi, M., et al. (eds.) BIRD 2008. CCIS, vol. 13, pp. 446–455. Springer, Heidelberg (2008)
6. Baeza-Yates, R., Gonnet, G.H.: A new approach to text searching. *Commun. ACM* 35(10), 74–82 (1992)
7. Cantone, D., Faro, S.: Pattern matching with swaps for short patterns in linear time. In: Nielsen, M., et al. (eds.) SOFSEM 2009. LNCS, vol. 5404, pp. 255–266. Springer, Heidelberg (2009)
8. Crochemore, M., Rytter, W.: Text algorithms. Oxford University Press, Oxford (1994)
9. Iliopoulos, C.S., Rahman, M.S.: A new model to solve the swap matching problem and efficient algorithms for short patterns. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 316–327. Springer, Heidelberg (2008)
10. Muthukrishnan, S.: New results and open problems related to non-standard stringology. In: Galil, Z., Ukkonen, E. (eds.) CPM 1995. LNCS, vol. 937, pp. 298–317. Springer, Heidelberg (1995)
11. Navarro, G., Raffinot, M.: A bit-parallel approach to suffix automata: Fast extended string matching. In: Farach-Colton, M. (ed.) CPM 1998. LNCS, vol. 1448, pp. 14–33. Springer, Heidelberg (1998)

# The Height and Range of Watermelons without Wall

## (Extended Abstract)

Thomas Feierl\*

Fakultät für Mathematik  
Universität Wien  
Nordbergstr. 15  
1090 Wien, Austria

**Abstract.** We determine the weak limit of the distribution of the random variables “height” and “range” on the set of  $p$ -watermelons without wall restriction as the number of steps tends to infinity. Additionally, we provide asymptotics for the moments of the random variable “height”.

## 1 Introduction

The model of *vicious walkers* was originally introduced by Fisher [7] as a model for wetting and melting processes. In general, the vicious walkers model is concerned with  $p$  random walkers on a  $d$ -dimensional lattice. In the lock step model, at each time step all of the walkers move one step in one of the allowed directions, such that at no time any two random walkers share the same lattice point.

A configuration that attracted much interest amongst mathematical physicists and combinatorialists is the *watermelon configuration* [1], which is the model underlying this paper (see Figure [1] for an example). This configuration can be studied with or without the presence of an impenetrable wall. By tracing the paths of the vicious walkers through the lattice we can identify the (probabilistic) vicious walkers model with certain sets of non-intersecting lattice paths. It is exactly this equivalent point of view that we adopt in this paper. We proceed with a precise definition. A  $p$ -*watermelon of length  $2n$*  is a set of  $p$  lattice paths in  $\mathbb{Z}^2$  satisfying the following conditions:

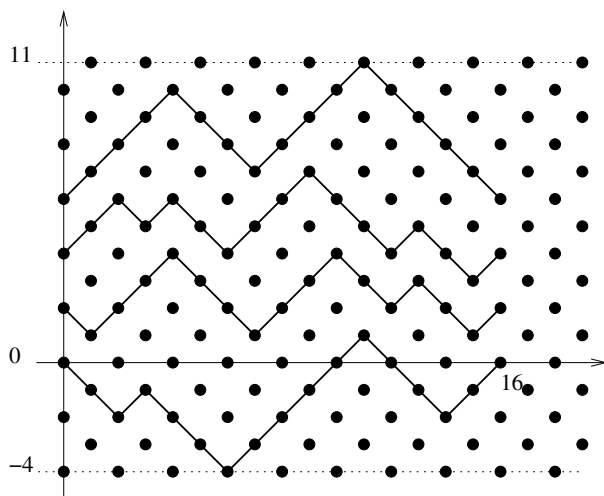
- For  $i = 1, 2, \dots, p$ , the  $i$ -th path starts at position  $(0, 2i - 2)$  and ends at  $(2n, 2i - 2)$ ,
- The paths consist of steps from the set  $\{(1, 1), (1, -1)\}$  only and
- The paths are *non-intersecting*, that is, at no time any two paths share the same lattice point.

An example of a 4-watermelon of length 16 is shown in Figure [1] (for the moment, the dashed lines and the labels should be ignored).

---

\* Research supported by the Austrian Science Foundation FWF, grant S9607-N13.

<sup>1</sup> This term comes from the resemblance of large configurations to the colour patterns of certain watermelons (see [4] Figure 1(b)).



**Fig. 1.** Example of a 4-watermelon of length 16 without wall, height 11, depth 4 and range 15

Since its introduction, the vicious walkers model has been studied in numerous papers. While early results mostly analyse the vicious walkers model in the continuum limit, there are nowadays many results for certain configurations directly based on the lattice path description given above. For example, Guttmann, Owczarek and Viennot [11] related the star and watermelon configurations to the theory of Young tableaux and integer partitions. Later, Krattenthaler, Guttmann and Viennot [16] proved new, exact as well as asymptotic, results for the number of certain configurations of vicious walkers.

The vicious walkers model is also very closely related to random matrix theory, as can be seen from articles by, e.g., Baik [1], Johansson [12] and Nagao and Forrester [18]. More recently, Katori and Tanemura [14] and Gillet [10] studied the diffusion scaling limit of certain configurations of vicious walkers, namely stars and watermelons, respectively.

In 2003, Bonichon and Mosbah [2] presented an algorithm for uniform random generation of watermelons, which relies on the counting results by Krattenthaler, Guttmann and Viennot [16]. Amongst other things, Bonichon and Mosbah studied, by means of numerical experiments, the parameter height on the set of watermelons (with and without wall).

In this paper we rigorously analyse the following two parameters on the set of  $p$ -watermelons:

- The *height* of a watermelon is the maximum ordinate reached by its top most branch.
- The *range* of a watermelon is the difference of the maximum of its top most branch and the minimum of its bottom most branch (the *depth* of the watermelon).

The 4-watermelon depicted in Figure 1 has the height 11 and the range  $11 + 4 = 15$ .

Katori et. al. [13] and Schehr et. al. [19] studied the parameter “height” in the continuous limit, and recovered the leading terms for some of the asymptotics proved in [6, 5]. Additionally, Schehr et. al. gave some arguments concerning the behaviour of the parameter “height” as the number of walkers tends to infinity.

Now, consider the set  $m_n^{(p)}$  of  $p$ -watermelons of length  $2n$ , endowed with the uniform probability measure. We can then speak of the random variables “height”, denoted by  $H_{n,p}$ , and “range”, denoted by  $R_{n,p}$ , on this set. We determine the weak limits of  $H_{n,p}$  and  $R_{n,p}$  as the number  $n$  of steps tends to infinity (see Theorem 1 and Theorem 3, respectively). Additionally, we determine the first two terms in the asymptotic expansion of the moments of  $H_{n,p}$  (see Theorem 2).

Techniques similar to those applied in this paper can also be used to analyse the random variable height on the set of  $p$ -watermelons under the presence of an impenetrable wall. For details we refer to [6].

The paper is organised as follows. The next section contains some well known results that are needed in the subsequent sections. In Section 3 we consider the random variable “height”, and we determine the weak limit as well as asymptotics for all moments. In the last section, we determine the weak limit of the random variable “range”.

This extended abstract contains only sketches of proofs for all results. For a detailed presentation, we refer to the full version of this manuscript [5].

## 2 Preliminaries

In this section we collect several results which will be needed in the two subsequent sections. All these results are either well known in the literature and/or can easily be derived by means of standard techniques. We, therefore, remain very brief, give only a few comments on the proofs and in each case refer to the corresponding literature for details.

We start with an exact enumeration result for the total number of watermelons confined to a horizontal strip. (Recall, that the depth of a watermelon is the minimum ordinate of its bottom most branch.)

**Lemma 1.** *The number  $m_{n,h,k}^{(p)}$  of  $p$ -watermelons without wall, length  $2n$ , height  $< h$  and depth  $> -k$  is given by*

$$m_{n,h,k}^{(p)} = \det_{0 \leq i,j < p} \left( \sum_{\ell \in \mathbb{Z}} \left( \binom{2n}{n + \ell(h+k) + i - j} - \binom{2n}{n + \ell(h+k) + h - i - j} \right) \right).$$

The total number  $m_n^{(p)}$  of  $p$ -watermelons is given by

$$m_n^{(p)} = \det_{0 \leq i,j < p} \left( \binom{2n}{n + i - j} \right).$$

This lemma follows immediately from the well-known Lindström–Gessel–Viennot formula (see [9, Corollary 3] or [17, Lemma 1]), together with an iterated reflection principle.

*Remark 1.* Since any  $p$ -watermelon without wall and length  $2n$  has depth  $> -n-1$ , we see that the number of watermelons with height  $< h$  and no restriction on the depth is given by  $m_{n,h,n+1}^{(p)}$ . For the sake of convenience, this quantity will also be denoted by  $m_{n,h}^{(p)}$ . In this special case, the determinantal expression above simplifies to

$$m_{n,h}^{(p)} = \det_{0 \leq i,j < p} \left( \binom{2n}{n+i-j} - \binom{2n}{n+h-i-j} \right).$$

**Lemma 2.** *We have*

$$m_n^{(p)} = \left(\frac{2}{n}\right)^{\binom{p}{2}} \binom{2n}{n}^p \left(\prod_{i=0}^{p-1} i!\right) (1 + O(n^{-1}))$$

as  $n \rightarrow \infty$ .

The determinant expression for  $m_n^{(p)}$  can be evaluated to a product, from which the asymptotics are easily obtained. For a proof, we refer to [16] (see also [5]).

**Lemma 3.** *For  $|m - z| \leq n^{5/8}$ ,  $z$  bounded, and arbitrary  $N > 1$  we have the asymptotic expansion*

$$\begin{aligned} \frac{\binom{2n}{n+m-z}}{\binom{2n}{n}} &= e^{-m^2/n} \sum_{u=0}^{4N+1} \left(\frac{z}{\sqrt{n}}\right)^u \frac{1}{u!} H_u\left(\frac{m}{\sqrt{n}}\right) \\ &+ e^{-m^2/n} \sum_{u=0}^{4N+1} \left(\frac{z}{\sqrt{n}}\right)^u \sum_{l=1}^{3N+1} \sum_{k=0}^{u-1} \sum_{r=1}^{2l} \frac{F_{r,l}}{k!n^l} \binom{2r}{u-k} H_k\left(\frac{m}{\sqrt{n}}\right) \left(-\frac{m}{\sqrt{n}}\right)^{2r+k-u} \\ &+ O\left(e^{-m^2/n} n^{-1-2N}\right) \end{aligned} \tag{1}$$

as  $n \rightarrow \infty$ . Here, the  $F_{r,l}$  are some constants the explicit form of which is of no importance in the sequel, and  $H_k(z)$  denotes the  $k$ -th Hermite polynomial, that is,

$$\frac{H_k(z)}{k!} = \sum_{m \geq 0} \frac{(-1)^{k-m} (2z)^{2m-k}}{(k-m)! (2m-k)!}, \quad k \geq 0. \tag{2}$$

The lemma above follows from Stirling’s approximation for the factorials. For a detailed proof we refer to [6, Lemma 6].

### 3 Height

In this section we derive asymptotics for the distribution as well as for the moments of the random variable  $H_{n,p}$ . As mentioned before, the number of  $p$ -watermelons with length  $2n$  and height  $< h$  is given by  $m_{n,h}^{(p)} = m_{n,h,n+1}^{(p)}$ . Consequently, we have for the distribution of  $H_{n,p}$

$$\mathbb{P} \{H_{n,p} + 1 \leq h\} = \frac{m_{n,h}^{(p)}}{m_n^{(p)}}.$$

**Theorem 1.** *For each fixed  $t \in (0, \infty)$  we have the asymptotics*

$$\mathbb{P} \left\{ \frac{H_{n,p} + 1}{\sqrt{n}} \leq t \right\} = \frac{2^{-\binom{p}{2}}}{\prod_{j=0}^{p-1} j!} \det_{0 \leq i,j < p} \left( (-1)^i H_{i+j}(0) - H_{i+j}(t) e^{-t^2} \right) + O\left(n^{-1/2} e^{-t^2}\right) \quad (3)$$

as  $n \rightarrow \infty$ , where  $H_a(x)$  denotes the  $a$ -th Hermite polynomial.

*Proof (Sketch).* Set  $\mathbf{x} = (x_0, \dots, x_{p-1})$  and  $\mathbf{y} = (y_0, \dots, y_{p-1})$ , and consider the more general quantity

$$m_{n,h}^{(p)}(\mathbf{x}, \mathbf{y}) = \det_{0 \leq i,j < p} \left( \binom{2n}{n+x_i-y_j} - \binom{2n}{n+h-x_i-y_j} \right).$$

Asymptotics for this quantity are obtained by factoring  $\binom{2n}{n}$  out of each row of the determinant above and applying Lemma 3 to each entry of the determinant.

It is easy to see that the determinant  $m_{n,h}^{(p)}(\mathbf{x}, \mathbf{y})$  is equal to zero whenever  $x_i = x_j$  or  $y_i = y_j$  for some  $i \neq j$  (in that case, two rows/columns are equal). This also holds true for the asymptotics for  $m_{n,h}^{(p)}(\mathbf{x}, \mathbf{y})$ . Moreover, the asymptotic expression obtained by the procedure described above is seen to be a polynomial in the  $x_i$ 's and  $y_j$ 's. Hence,

$$m_{n,h}^{(p)}(\mathbf{x}, \mathbf{y}) = n^{-\binom{p}{2}} \binom{2n}{n} \frac{\prod_{0 \leq i < j < p} (x_j - x_i)(y_j - y_i)}{\prod_{0 \leq j < p} j!^2} \chi(n, h) + O(n^{-1/2} e^{-h^2/n})$$

as  $n \rightarrow \infty$ . Here, the error term is determined by noting that every power of  $x_j$  and  $y_j$  entails a factor of  $n^{-1/2}$  (see Lemma 3). The unknown coefficient  $\chi(n, h)$  can now be determined by comparing coefficients on both sides of the equation above. Comparing the coefficients of  $\prod_{j=0}^{p-1} x_j^j y_j^j$ , we obtain

$$\det_{0 \leq i,j < p} \left( (-1)^i H_{i+j}(0) - H_{i+j} \left( \frac{h}{\sqrt{n}} \right) e^{-h^2/n} \right) = \chi(n, h).$$

If we specialise by setting  $x_j = y_j = j$ , then we see that

$$m_{n,h}^{(p)} = n^{-\binom{p}{2}} \binom{2n}{n}^p \det_{0 \leq i,j < p} \left( (-1)^i H_{i+j}(0) - H_{i+j} \left( \frac{h}{\sqrt{n}} \right) e^{-h^2/n} \right) + O \left( n^{-1/2} e^{-h^2/n} \right).$$

Setting  $h = t\sqrt{n}$  and replacing  $m_{2n}^{(p)}$  with its asymptotic equivalent as given by Lemma 2 we obtain the result.  $\square$

Let us now turn our attention to the moments of the distribution of  $H_{n,p}$ . Clearly, we have for  $s \in \mathbb{N}$ ,

$$\mathbb{E} \left( H_{n,p}^s \right) = \sum_{h \geq 1} h^s \frac{m_{n,h+1}^{(p)} - m_{n,h}^{(p)}}{m_n^{(p)}} = \sum_{h \geq 1} (h^s - (h-1)^s) \frac{m_n^{(p)} - m_{n,h}^{(p)}}{m_n^{(p)}}. \tag{4}$$

The dominant terms of the asymptotics for the moments are going to be expressed by linear combinations of certain infinite exponential sums. Asymptotics for these sums are to be determined now.

**Lemma 4.** For  $\nu \geq 0$  and  $\mu > 0$  define

$$f_{\nu,\mu}(n) = \sum_{h \geq 1} h^\nu e^{-\mu h^2/n}.$$

This sum admits the asymptotic series expansion

$$f_{\nu,\mu}(n) \approx \frac{1}{2} \Gamma \left( \frac{\nu+1}{2} \right) \left( \frac{n}{\mu} \right)^{(\nu+1)/2} + \sum_{m \geq 0} \binom{\mu}{n}^m \frac{(-1)^{\nu+m} B_{2m+\nu+1}}{(2m+\nu+1)!m!},$$

as  $n \rightarrow \infty$ , where  $\Gamma$  denotes the gamma function and  $B_m$  is the  $m$ -th Bernoulli number defined via the equation  $\sum_{j \geq 0} B_j t^j / j! = t / (e^t - 1)$ .

The lemma can be proved with the help of Mellin transform techniques (see, e.g., 8 for an account on these techniques applied to asymptotics).

The rest of this section is devoted to the proof of Theorem 2 below, which gives the final expression for the asymptotics of the moments. In order to present the proof of this theorem in a clear fashion we split it into a series of lemmas. For a more detailed overview of the proof, we refer directly to the proof of Theorem 2.

As a first step, we prove in Lemma 5 a preliminary asymptotic expression for the moments of the height distribution. The presented compact form of the asymptotics makes use of certain linear operators that are going to be defined now.

**Definition 1.** Let  $\Xi_1$  and  $\Xi_0$  denote the linear operators defined by

$$\begin{aligned} \Xi_1 \left( h^\nu e^{-\mu h^2} \right) &= \frac{1}{2} \Gamma \left( \frac{\nu+1}{2} \right) \left( \frac{1}{\mu} \right)^{(\nu+1)/2} \\ \Xi_0 \left( h^\nu e^{-\mu h^2} \right) &= (-1)^\nu \frac{B_{\nu+1}}{(\nu+1)!}, \end{aligned}$$

where  $\nu \geq 0$  and  $\mu > 0$ ,  $\Gamma$  denotes the gamma function and  $B_k$  is the  $k$ -th Bernoulli number.

By Lemma 4 we have

$$f_{\nu,\mu}(n) = \Xi_1 \left( h^\nu e^{-\mu h^2} \right) n^{(\nu+1)/2} + \Xi_0 \left( h^\nu e^{-\mu h^2} \right) + O(n^{-1}), \quad n \rightarrow \infty,$$

so that  $\Xi_1$  and  $\Xi_0$  yield the coefficients of the first two terms in the asymptotic expansion of  $f_{\nu,\mu}(n)$ .

The preliminary expression for the asymptotics of the moments can now be proven in pretty much the same way as in Theorem 3.

**Lemma 5.** *For  $s \in \mathbb{N}$ ,  $s \geq 1$ , the  $s$ -th moment of the random variable “height” satisfies the asymptotics*

$$\begin{aligned} \mathbb{E}H_{n,p}^s &= s\Xi_1 \left( \kappa_p h^{s-1} \right) n^{s/2} - \Xi_1 \left( \binom{s}{2} \kappa_p h^{s-2} + \tau_p h^{s-1} \right) n^{(s-1)/2} \\ &\quad + \Xi_0(\kappa_p) + O\left(n^{s/2-1}\right) \end{aligned} \quad (5)$$

as  $n \rightarrow \infty$ , where

$$\kappa_p = 1 - \frac{2^{-\binom{p}{2}}}{\prod_{0 \leq j < p} j!} \det_{0 \leq i, j < p} \left( (-1)^i H_{i+j}(0) - H_{i+j}(h) e^{-h^2} \right)$$

and

$$\tau_p = (p-1) \frac{2^{-\binom{p}{2}}}{\prod_{0 \leq j < p} j!} \det_{0 \leq i, j < p} \left( \begin{cases} (-1)^i H_{i+j}(0) - H_{i+j}(h) e^{-h^2} & \text{if } i < p-1 \\ (-1)^p H_{p+j}(0) - H_{p+j}(h) e^{-h^2} & \text{if } i = p-1 \end{cases} \right).$$

Here,  $H_k(z)$  denotes the  $k$ -th Hermite polynomial.

*Proof.* The same arguments as in the proof of Theorem 3 give the finer asymptotics

$$\begin{aligned} m_{n,h}^{(p)}(\mathbf{x}, \mathbf{y}) &= n^{-\binom{p}{2}} \binom{2n}{n} \frac{\prod_{0 \leq i < j < p} (x_j - x_i)(y_j - y_i)}{\prod_{0 \leq j < p} j!^2} \\ &\quad \times \left( \chi(n, h) + \sum_{j=0}^{p-1} \left( \xi_j(n, h) \frac{x_j}{\sqrt{n}} + \eta_j(n, h) \frac{y_j}{\sqrt{n}} \right) \right) + O(n^{-1} e^{-h^2/n}) \end{aligned}$$

as  $n \rightarrow \infty$ . By comparing coefficients we have already seen that

$$\chi(n, h) = \det_{0 \leq i, j < p} \left( (-1)^i H_{i+j}(0) - H_{i+j} \left( \frac{h}{\sqrt{n}} \right) e^{-h^2/n} \right).$$



Analogously, we can determine  $\xi_k(n, h)$ . By comparing the coefficients of  $x_k \prod_{j=0}^{p-1} x_j^j y_j^j$  on both sides of the equation above, we obtain the equations

$$0 = \xi_k(n, h) - \xi_{k+1}(n, h), \quad k < p - 1,$$

and

$$\xi_{p-1}(n, h) = \frac{1}{p} \det_{0 \leq i, j < p} \left( \begin{cases} (-1)^i H_{i+j}(0) - H_{i+j} \left( \frac{h}{\sqrt{n}} \right) e^{-h^2/n} & \text{if } i < p - 1 \\ (-1)^p H_{p+j}(0) - H_{p+j} \left( \frac{h}{\sqrt{n}} \right) e^{-h^2/n} & \text{if } i = p - 1 \end{cases} \right).$$

Analogous results hold for the  $\eta_k(n, h)$ ,  $0 \leq k < p$ .

Since  $H_{i+j}(0)$  is non-zero if and only if  $i + j$  is even we deduce  $(-1)^i H_{i+j}(0) = (-1)^j H_{i+j}(0)$ , which implies

$$\xi_{p-1}(n, h) = \eta_{p-1}(n, h).$$

Specialisation to  $x_j = y_j = j$ ,  $0 \leq j < p$ , then leads to

$$\frac{m_{n,h}^{(p)}}{m_n^{(p)}} = \frac{2^{-\binom{p}{2}}}{\prod_{0 \leq j < p} j!} \left( \chi(n, h) + 2 \binom{p}{2} \xi_{p-1}(n, h) n^{-1/2} \right) + O \left( n^{-1} e^{-h^2/n} \right)$$

as  $n \rightarrow \infty$ . The result now follows from the fact that

$$\mathbb{E} \left( H_{n,p}^s \right) = \sum_{h \geq 1} \left( sh^{s-1} - \binom{s}{2} h^{s-2} \right) \left( 1 - \frac{m_{n,h}^{(p)}}{m_n^{(p)}} \right) + O \left( n^{s/2-1} \right), \quad n \rightarrow \infty,$$

which follows from Equation (4) and Lemma 4. □

**Lemma 6.** Let  $H_k(x)$  denote the  $k$ -th Hermite polynomial as defined by Equation (2). We have the determinant evaluation

$$\det_{0 \leq i, j < p} \left( (-1)^{(i+j)/2} H_{i+j}(0) \right) = 2^{\binom{p}{2}} \prod_{j=0}^{p-1} j!. \tag{6}$$

**Lemma 7.** Let  $\mu > 0$  denote a real number. The operator  $\Xi_1$  from Definition 7 satisfies the relation

$$\Xi_1 \left( \frac{d}{dh} \left( h^\nu e^{-\mu h^2} \right) \right) = \begin{cases} -1 & \text{if } \nu = 0 \\ 0 & \text{if } \nu > 0. \end{cases} \tag{7}$$

*Proof.* For  $\nu = 0$  the claim follows immediately from the definition of the operator  $\Xi_1$ . For  $\nu > 0$  we calculate

$$\Xi_1 \left( h^{\nu+1} e^{-\mu h^2} \right) = \frac{\nu}{2\mu} \Xi_1 \left( h^{\nu-1} e^{-\mu h^2} \right),$$

from which the claims follows upon multiplying by  $2\mu$  and rearranging the terms. □

**Lemma 8.** *Let  $\kappa_p$  and  $\tau_p$  denote the determinants defined in Lemma 5. We have the relation*

$$(p - 1) \frac{d}{dh} \kappa_p = \tau_p, \quad p \geq 1. \tag{8}$$

This last lemma is far from being obvious. Nevertheless, the proof has been completely omitted because of its technical nature. For a detailed proof, we refer to the full version [5].

We are now able to state and prove the final expression for the asymptotics of the moments.

**Theorem 2.** *For  $s \in \mathbb{N}$ ,  $s \geq 1$ , the  $s$ -th moment of the random variable “height” satisfies*

$$\begin{aligned} \mathbb{E} (H_{n,p}^s) = s \Xi_1 (\kappa_p h^{s-1}) n^{s/2} + (s - 1) \left( p - 1 - \frac{s}{2} \right) \Xi_1 (\kappa_p h^{s-2}) n^{(s-1)/2} \\ + p - \frac{3}{2} + O \left( n^{s/2-1} \right) \end{aligned} \tag{9}$$

as  $n \rightarrow \infty$ . Here,  $\kappa_p$  is defined as in Lemma 5.

*Proof (Sketch).* As a first step we need to prove that the quantity  $\kappa_p$  is of the form

$$\kappa_p = \sum_{k=0}^K \sum_{m=1}^M \lambda_{k,m} h^{2k} e^{-mh^2}$$

for some numbers  $K$ ,  $M$  and some constants  $\lambda_{k,m}$ . Recall, that the  $k$ -th Hermite polynomial is even (odd) whenever  $k$  is even (odd). This implies that  $\kappa_p$  is an even function of  $h$ . An application of Lemma 6 then shows that the constant term of  $\kappa_p$  is equal to zero, and establishes the desired form for  $\kappa_p$ .

Now, an application of Lemma 7 immediately shows that

$$\Xi_1 \left( \frac{d}{dh} (\kappa_p h^{s-1}) \right) = 0, \quad s > 1,$$

and the product rule for the derivative together with Lemma 8 imply

$$\Xi_1 (\tau_p h^{s-1}) = -(s - 1)(p - 1) \Xi_1 (\kappa_p h^{s-2}), \quad s > 1.$$

The last step in the transition from Equation (5) to (9) is the evaluation of the quantities  $\Xi_1(\tau_p)$  and  $\Xi_0(\kappa_p)$ . For the sake of simplicity, we set  $\frac{1}{C} = 2^{\binom{p}{2}} \prod_{j=0}^{p-1} j!$ , as well as

$$\chi(h) = \det_{0 \leq i, j < p} \left( (-1)^i H_{i+j}(0) - H_{i+j}(0) e^{-h^2} \right).$$

From Lemma 8 and Lemma 7 we deduce that

$$\Xi_1 (\tau_p) = (p - 1) \Xi_1 \left( \frac{d}{dh} \kappa_p \right) = -(p - 1) \Xi_1 \left( C \frac{d}{dh} \chi(h) \right).$$

**Table 1.** This table gives the coefficient of the dominant asymptotic term of  $\mathbb{E}H_{n,p}^s$  as  $n \rightarrow \infty$  for small values of  $s$  and  $p$  (see Theorem 2)

$s\kappa_s^{(p)}$	$s = 1$	$s = 2$	$s = 3$
$p = 1$	$\frac{1}{2}\sqrt{\pi} = 0.88\dots$	$1$	$\frac{3}{4}\sqrt{\pi} = 1.32\dots$
$p = 2$	$\frac{2+\sqrt{2}}{4}\sqrt{\pi} = 1.51\dots$	$\frac{5}{2}$	$\frac{3(12+\sqrt{2})}{16}\sqrt{\pi} = 4.45\dots$
$p = 3$	$\frac{72+45\sqrt{2}-16\sqrt{3}}{96}\sqrt{\pi} = 1.99\dots$	$\frac{25}{6}$	$\frac{1584+315\sqrt{2}-32\sqrt{3}}{385}\sqrt{\pi} = 9.11\dots$

Now, Lemma 7 shows that

$$\Xi_1 \left( \frac{d}{dh} \chi(h) \right) = -1, \quad \text{which implies} \quad \Xi_1(\tau_p) = 1 - p.$$

Recall the definition of the operator  $\Xi_0$ , and note that all odd Bernoulli numbers except for  $B_1$  are equal to zero. This, together with the fact that  $\kappa_p$  is an even function of  $h$ , proves the equality

$$\Xi_0(\kappa_p) = \Xi_0(1 - C\chi(h)).$$

Furthermore, noting that  $\Xi_0(h^\nu e^{-\mu h^2})$  is independent of  $\mu$  we see that

$$\Xi_0(\kappa_p) = B_1 = -\frac{1}{2}.$$

This completes the proof of the theorem. □

## 4 Range

We determine the asymptotics for  $n \rightarrow \infty$  of

$$\mathbb{P}\{R_{n,p} \leq r\} = \frac{1}{m_n^{(p)}} \sum_{h=2p-2}^r \left( m_{n,h+1,r-h+1}^{(p)} - m_{n,h,r-h+1}^{(p)} \right). \quad (10)$$

Note that  $m_{n,h+1,r-h+1}^{(p)} - m_{n,h,r-h+1}^{(p)}$  is the number of watermelons with height exactly  $h$  and range  $\leq r$ .

**Theorem 3.** For each fixed  $t \in (0, \infty)$  we have the asymptotics

$$\mathbb{P}\left\{ \frac{R_{n,p} + 1}{\sqrt{n}} \leq t \right\} \rightarrow \frac{2^{-\binom{p}{2}}}{\prod_{i=0}^{p-1} i!} \int_0^t \left( \frac{d}{dz} T_p(z, w) \Big|_{z=t} \right) dw, \quad n \rightarrow \infty, \quad (11)$$

where

$$T_p(z, w) = \det_{0 \leq i, j < p} \left( (-1)^i \left( \sum_{\ell \in \mathbb{Z}} H_{i+j}(\ell z) e^{-(\ell z)^2} \right) - \left( \sum_{\ell \in \mathbb{Z}} H_{i+j}(\ell z + w) e^{-(\ell z + w)^2} \right) \right).$$

Here,  $H_a$  denotes the  $a$ -th Hermite polynomial.

*Proof (Sketch).* Since  $m_{n,2p-2,k}^{(p)} = 0$  for any  $k$ , Equation (10) can be rewritten as

$$\mathbb{P}\{R_{n,p} \leq r\} = \frac{m_{n,r+1,1}^{(p)}}{m_n^{(p)}} + \frac{1}{m_n^{(p)}} \sum_{h=2p-1}^r \left( m_{n,h,r-h+2}^{(p)} - m_{n,h,r-h+1}^{(p)} \right).$$

The first term on the right-hand side is negligible. To see this, we note that  $m_{n,r+1,1}$  is equal to the number of  $p$ -watermelons with wall and height  $\leq r$ , which is of order  $\binom{2n}{n}^p n^{-p^2}$  as  $n \rightarrow \infty$  (see [6] for details), whereas  $m_n^{(p)}$  is of order  $\binom{2n}{n}^p n^{-\binom{p}{2}}$  (see Lemma 2).

Asymptotics for the sum on the right-hand side can now be established in a fashion analogous to the proof of Theorem 1. A more detailed presentation of these techniques can also be found in [6, Theorem 2]. We find the asymptotics

$$\mathbb{P}\{R_{n,p} \leq r\} \sim \frac{\binom{2n}{n}^p n^{-\binom{p}{2}}}{m_n^{(p)}} \sum_{h=2p-1}^r \left( T_p \left( \frac{r+2}{\sqrt{n}}, \frac{h}{\sqrt{n}} \right) - T_p \left( \frac{r+1}{\sqrt{n}}, \frac{h}{\sqrt{n}} \right) \right)$$

as  $n \rightarrow \infty$ .

Now, Taylor series expansion shows that

$$T_p \left( \frac{r+2}{\sqrt{n}}, \frac{h}{\sqrt{n}} \right) - T_p \left( \frac{r+1}{\sqrt{n}}, \frac{h}{\sqrt{n}} \right) = \frac{1}{\sqrt{n}} T'_p \left( \frac{r+1}{\sqrt{n}}, \frac{h}{\sqrt{n}} \right) + O(n^{-1})$$

as  $n \rightarrow \infty$ , where  $T'$  denotes the derivative of  $T$  with respect to its first argument. Setting  $r+1 = t\sqrt{n}$  we see that

$$\begin{aligned} \sum_{h=2p-1}^r \left( T_p \left( \frac{r+2}{\sqrt{n}}, \frac{h}{\sqrt{n}} \right) - T_p \left( \frac{r+1}{\sqrt{n}}, \frac{h}{\sqrt{n}} \right) \right) \\ \sim \sum_{h=2p-1}^r \frac{1}{\sqrt{n}} T'_p \left( \frac{r+1}{\sqrt{n}}, \frac{h}{\sqrt{n}} \right) \rightarrow \int_0^t T'(t, w) dw \end{aligned}$$

as  $n \rightarrow \infty$ . □

*Remark 2.* For the special case  $p = 1$  we recover a well-known fact originally proven by Chung [3] and Kennedy [15] (see also Vervaat [20]). Namely, the equality of the distributions of the height of Brownian excursions and the range of Brownian bridges. In fact, for  $p = 1$  Theorem 3 reads

$$\mathbb{P}\left\{ \frac{R_{n,1} + 1}{\sqrt{n}} \leq t \right\} \rightarrow \sum_{\ell \in \mathbb{Z}} (1 - 2(\ell t)^2) e^{-(\ell t)^2}, \quad n \rightarrow \infty,$$

which is exactly the limiting distribution of the height of 1-watermelons with wall (see [6]).

## References

1. Baik, J.: Random vicious walks and random matrices. *Comm. Pure Appl. Math.* 53(11), 1385–1410 (2000)
2. Bonichon, N., Mosbah, M.: Nicolas Bonichon and Mohamed Mosbah. Watermelon uniform random generation with applications. *Theoret. Comput. Sci.* 307(2), 241–256 (2003); Random generation of combinatorial objects and bijective combinatorics
3. Chung, K.L.: Excursions in Brownian motion. *Ark. Mat.* 14(2), 155–177 (1976)
4. Essam, J.W., Guttmann, A.J.: Vicious walkers and directed polymer networks in general dimensions. *Phys. Rev. E* (3) 52(6, part A), 5849–5862 (1995)
5. Feierl, T.: The height and range of watermelons without wall. Preprint, arXiv:math.CO/0806.0037
6. Feierl, T.: The height of watermelons with wall. Preprint, arXiv:math.CO/0802.2691
7. Fisher, M.E.: Walks, walls, wetting, and melting. *J. Statist. Phys.* 34(5-6), 667–729 (1984)
8. Flajolet, P., Gourdon, X., Dumas, P.: Mellin transforms and asymptotics: harmonic sums. *Theoret. Comput. Sci.* 144(1-2), 3–58 (1995); Special volume on mathematical analysis of algorithms
9. Gessel, I., Viennot, G.: Determinants, paths, and plane partitions (1989) (preprint), <http://people.brandeis.edu/~gessel/homepage/papers/pp.pdf>
10. Gillet, F.: Asymptotic behaviour of watermelons. Preprint, arXiv:math.PR/0307204 (2003)
11. Guttmann, A.J., Owczarek, A.L., Viennot, X.G.: Vicious walkers and Young tableaux. I. Without walls. *J. Phys. A* 31(40), 8123–8135 (1998)
12. Johansson, K.: Non-intersecting paths, random tilings and random matrices. *Probab. Theory Related Fields* 123(2), 225–280 (2002)
13. Katori, M., Izumi, M., Kobayashi, N.: Maximum distribution of bridges of noncolliding brownian paths. *Phys. Rev. E* 78, 051102 (2008) (electronic)
14. Katori, M., Tanemura, H.: Functional central limit theorems for vicious walkers. *Stoch. Stoch. Rep.* 75(6), 369–390 (2003)
15. Kennedy, D.P.: The distribution of the maximum Brownian excursion. *J. Appl. Probability* 13(2), 371–376 (1976)
16. Krattenthaler, C., Guttmann, A.J., Viennot, X.G.: Vicious walkers, friendly walkers and Young tableaux. II. With a wall. *J. Phys. A* 33(48), 8835–8866 (2000)
17. Lindström, B.: On the vector representations of induced matroids. *Bull. London Math. Soc.* 5, 85–90 (1973)
18. Nagao, T., Forrester, P.J.: Vicious random walkers and a discretization of Gaussian random matrix ensembles. *Nuclear Phys. B* 620(3), 551–565 (2002)
19. Schehr, G., Majumdar, S.M., Comtet, A., Randon-Furling, J.: Exact distribution of the maximal height of watermelons. Preprint, arXiv:0807.0522 (2008)
20. Vervaat, W.: A relation between Brownian bridge and Brownian excursion. *Ann. Probab.* 7, 143–149 (1979)

# Fast Convolutions and Their Applications in Approximate String Matching

Kimmo Fredriksson<sup>1</sup> and Szymon Grabowski<sup>2</sup>

<sup>1</sup> Department of Computer Science, University of Kuopio,  
P.O. Box 1627, 70211 Kuopio, Finland  
fredriks@cs.uku.fi

<sup>2</sup> Computer Engineering Department, Technical University of Łódź,  
Al. Politechniki 11, 90–924 Łódź, Poland  
sgrabow@kis.p.lodz.pl

**Abstract.** We develop a method for performing boolean convolutions efficiently in word RAM model of computation, having a word size of  $w = \Omega(\log n)$  bits, where  $n$  is the input size. The technique is applied to approximate string matching under Hamming distance. The obtained algorithms are the fastest known. In particular, we reduce the complexity of the Amir et al. [1] algorithm for  $k$ -mismatches from  $O(n\sqrt{k \log k})$  to  $O(n + n\sqrt{k/w} \log k)$ .

## 1 Introduction

A word RAM is a random-access machine with unit-cost operations for operands of  $w$  bits, and having instruction set similar to modern computers. Traditional model used e.g. in sorting or string matching algorithms is the comparison model, where the algorithm complexities and lower bounds are measured by the number of pair-wise comparisons required to perform the task. However, the word RAM model is much more natural and realistic given the modern computers. This model has become ever more popular since the appearance of fusion trees [2] (showing that  $n$  integers can be sorted in  $o(n \log n)$  time in word RAM).

The word RAM model has been implicit in most algorithm analyses, but the algorithms generally have not taken the advantage of it. One such algorithm is the famous fast Fourier transform (FFT), an efficient method to compute the discrete Fourier transform. The algorithm assumes word RAM model in which arithmetic operations of  $w = \Theta(\log n)$  bits (where  $n$  is the input size) can be done in constant time. FFT is one of the most powerful tools in various fields of computer science and engineering.

One of the reasons of the wide applicability of FFT is that convolutions and polynomial multiplications can be computed in  $O(n \log n)$  time, and e.g. many approximate string matching problems can be reduced to convolutions. This idea dates back to 1974, when Fischer and Paterson introduced convolution-based techniques [3] for tackling with several string matching problems. Since then,

many new algorithms using the same general approach have been presented, see e.g. [14,5] for a variety of string matching problems and FFT-based solutions.

An interesting result was achieved by Indyk [6]. He gave an  $O(n)$ -time Monte Carlo algorithm for boolean convolutions, which let him obtain a number of other results, for example for wild card matching.

In this paper, we consider approximate string matching permitting mismatches. We are given a text string of length  $n$  and a pattern string of length  $m$  over some integer alphabet of size  $\sigma$ , and we want to either compute the number of mismatches for each possible pattern alignment over the text (Hamming distance) or report every text position where the number of mismatches is at most  $k$ . The classic convolutions-based solution [3] solves the Hamming distance problem in time  $O(\sigma n \log m)$ . This can be improved to  $O(n\sqrt{m \log m})$  time [7,8]. The current state-of-the-art is a hybrid algorithm with  $O(n\sqrt{k \log k})$  time [1]. We improve the classic algorithm (which is used as a component in many other, e.g. in the cited [7,1] algorithms) to run in time  $O(\sigma n \log^2 m/w)$  and consequently the  $O(n\sqrt{k \log k})$  time algorithm to run in time  $O(n + n\sqrt{k/w} \log k)$ .

We note that there are other techniques to obtain sub-quadratic time complexity for Hamming distance or  $k$ -mismatches (cf. Table 1). Landau and Vishkin [9] used a suffix tree augmented with the lowest common ancestor (LCA) data structure, to achieve  $O(nk)$  time. Bit-parallelism can be used to simulate a non-deterministic automaton; the best result along these lines [10] yields  $O(nm/w)$  time. A prime-number encoding scheme was proposed in [11] for  $k$ -mismatches with character classes, to achieve  $O(n\sigma)$  time if, for example, both  $\log^2 m = O(\log n)$  and  $\log m = O(\sigma)$  hold.

In this paper we present two techniques for calculating several convolutions in parallel, using word-level parallelism. One of them samples the text in several locations (or alternatively can be used for searching multiple patterns), the other parallelizes computations across the alphabet. The first technique is more efficient but cannot be used for extremely long patterns (of length close to the length of the text).

We assume word RAM model of computation, with a word length of  $w$  bits. The theoretical model imposes that  $w = \Omega(\log n)$ , where  $n$  is the input size. The

**Table 1.** Our results. We assume that  $w = \Omega(\log n)$ , and  $\sigma_p \leq \min\{\sigma, m\}$  is the number of distinct symbols in the pattern. All results for Hamming distance apply to  $k$ -mismatches as well.

Problem	Previous results	Our results
Hamming distance	$O(\sigma_p n \log m)$ [3] $O(n\sqrt{m \log m})$ [7,8] $O(nm/w)$ [10]	$O(\sigma_p n \log^2 m/w)$ , if $m = O(n \log m/w)$ $O(n + n\sqrt{m/w} \log m)$ $O(\sigma_p n + \sigma_p n \log^2 m/w^{1-\epsilon})$ , for any $m$ $O(n + \sigma_p n \log(m/\sigma_p)(\log m/w + 1/\log n))$
$k$ -mismatches	$O(n\sqrt{k \log k})$ [1] $O((n + \frac{nk^3}{m}) \log k)$ [1] $O(nk)$ [9,12,13]	$O(n + n\sqrt{k/w} \log k)$

practical view is that  $w = 32$  or  $w = 64$  in current typical CPU architectures, and growing<sup>1</sup>. Like other works applying FFT and convolutions, we also assume that multiplication of two  $w$ -bit words can be done in  $O(1)$  time. For large  $w$  this may not be the case. Assuming that multiplication takes  $O(\log w)$  time, our result e.g. for  $k$ -mismatches becomes  $O(n + n\sqrt{k \log w/w} \log k)$ . We note that the wide word assumption occurs more and more often in algorithmics [14,15,16]. If not stated otherwise, logarithms used throughout the paper are in base 2.

## 2 Multiple Convolutions with Vector Packing

Let us have two vectors,  $\tau = \tau_0\tau_1 \dots \tau_{n-1}$  and  $\rho = \rho_0\rho_1 \dots \rho_{m-1}$ , over some integer alphabet. We also assume that  $m < n$  (and often  $m \ll n$ ). We are interested in sums of the form

$$S(i) = \rho \otimes \tau(i) = \sum_{j=0}^{m-1} \rho_j \times \tau_{i+j}$$

for each possible  $i$ . Convolution of two vectors each of length  $n$  takes  $O(n \log n)$  time by using FFT (the convolution theorem), hence computing  $\rho \otimes \tau(i)$  for every  $i$  takes  $O(n \log m)$  total time, if we divide  $\tau$  to  $n/m$  overlapping blocks, each of length  $2m$ . From now on, we assume that this machinery is taken as a black box, and in particular we assume that this division to blocks is implicit. This is the basis of numerous string matching algorithms based on FFT. Basically, various problem instances are solved by encoding the problem to suitable vectors  $\tau$  and  $\rho$ . The first string matching algorithm based on this general idea was by Fischer and Paterson [3]; they gave an algorithm for matching with wild cards.

We now present a general technique of computing several convolutions in parallel. Assume that each value of  $S(i)$  can be represented with  $u$  bits. This means that each word of  $w$  (where  $w = \Omega(\log n)$ ) bits can pack  $b = \lfloor w/u \rfloor$  values. Let us exploit this fact. The vector  $\tau$  is divided to  $b$  blocks (overlapping by  $m - 1$  symbols) each of length  $\ell = n/b + m - 1$ . For simplicity, we assume that  $\ell = \Theta(n/b)$ . (This makes the technique easier to describe [see also Fig. 1], but in practice, we would use the same block division as in the standard algorithm, that is,  $n/m$  blocks of length  $2m$ , and just handle  $b$  of them in parallel.) Then we can define a new vector  $\tau'$  of length  $\ell$ ; call  $\ell' = \lfloor n/b \rfloor$ :

$$\tau'_i = \tau_{i+0\ell'} \times 2^{0u} + \tau_{i+1\ell'} \times 2^{1u} + \tau_{i+2\ell'} \times 2^{2u} + \dots + \tau_{i+(b-1)\ell'} \times 2^{(b-1)u}.$$

Each value  $\tau'_i$  still fits into  $w$  bits, and the whole vector  $\tau'$  can be easily computed in  $O(n)$  time. Note also that the fields cannot interact with each other, i.e. there is no possibility of a carry from one field to the next. Define  $S'(i) = \rho \otimes \tau'(i)$ .

---

<sup>1</sup> E.g. the multimedia extensions, such as the widely available SSE instruction set introduced in 1999 with Intel Pentium III have word size of  $w = 128$ , and the upcoming AVX extensions will at first double this.



**Alg. 1.** Word-packed-convolution( $\tau, n, \rho, m, u$ ).

---

```

1  /* preprocessing */
2   $b \leftarrow \lfloor w/u \rfloor$ 
3   $\ell' \leftarrow \lfloor n/b \rfloor$ 
4  for  $i \leftarrow 0$  to  $\ell' - 1$  do
5     $\tau'_i \leftarrow 0$ 
6    for  $h \leftarrow 0$  to  $b - 1$  do
7       $\tau'_i \leftarrow \tau'_i + \tau_{i+h\ell'} \times 2^{hu}$ 
8  /* FFT computation */
9  Compute  $S'(i) = \rho \otimes \tau'(i)$  for all  $i$  using FFT
10 /* postprocessing */
11 for  $i \leftarrow 0$  to  $n - m + 1$  do
12    $S(i) \leftarrow (S'(i \bmod \ell') \gg [i/\ell']u) \& (2^u - 1)$ 
13 return  $S$ 

```

---

**Observation 1.**  $S'(i) = S(i+0\ell') \times 2^{0u} + S(i+1\ell') \times 2^{1u} + S(i+2\ell') \times 2^{2u} + \dots$ , i.e.  $S'$  is a packed representation of  $S$ .

Now the values  $S'(i)$  for all  $i$  can be computed in  $O(n + \ell \log m)$  time with FFT. Given  $S'$ , we can trivially obtain  $S$  in time  $O(n)$ . Alg. [1](#) shows more details. In Sec. [3](#) we present how this general scheme can be used to obtain efficient string matching algorithms for Hamming distance.

*On computing the convolutions.* We note that “FFT” algorithm in general refers to computing the discrete Fourier transform (DFT), or its inverse, of a (sampled) complex or real function. In other words, DFT is susceptible to the inherent inaccuracy of floating point arithmetic, due to its finite precision (e.g., round-off errors). This is usually not a problem, since  $\Theta(\log m)$  bits of precision is enough in many string matching applications. In our case, however, we need  $\Theta(w)$  bits of precision, where  $w$  may be large as compared to  $\log m$  (or even to  $\log n$ ). However, the *number-theoretic transform* (NTT) operates with modular arithmetic on integers, and thus is accurate. What is important for our case, is that NTT (and its inverse) can be computed with the same algorithms as DFT, and that the convolution theorem holds for NTT as well. Hence we can compute the convolutions efficiently and precisely using fast number-theoretic transforms. Since NTT can be computed with FFT algorithm, we continue to speak about FFT. For more details, see e.g. [17](#).

### 3 Applications

In what follows, we have a pattern  $P = p_0p_1 \dots p_{m-1}$  and text  $T = t_0t_1 \dots t_{n-1}$  over some integer alphabet  $\Sigma = \{1, 2, \dots, \sigma\}$ .  $P$  is said to occur exactly at position  $i$  of  $T$  if  $p_j = t_{i+j}$  for all  $j = 0 \dots m-1$ . Likewise,  $P$  occurs at position  $i$  of  $T$  with a Hamming distance  $k$ , if  $p_j = t_{i+j}$  for  $m-k$  values of  $j$ . The problem is then to report every text position  $i$  that has a pattern occurrence, under some particular matching model.

We now present improved solutions to some basic algorithms to illustrate the technique. Convolutions (computed with FFT) are used as a component in many other algorithms, e.g.,  $\delta$  and  $\gamma$  matching. We do not cover those here. However, our technique has applications in those algorithms as well.

### 3.1 Improved Algorithm for Hamming Distance

The classic FFT-based algorithm for Hamming distance works as follows. For each symbol  $c$  from the alphabet we create a bit-vector  $\rho^c$  of length  $m$ , with bits 1 at positions where  $c$  occurs in  $P$ , and 0s elsewhere, i.e.  $\rho_j^c = 1$  iff  $p_j = c$ . Similarly,  $\tau_i^c = 1$  iff  $t_i = c$ . Having this representation, we calculate  $S^c(i)$  using FFT in  $O(n \log m)$  time. The result for each text position is a multiplication of two binary vectors, i.e., the number of matching pairs of bits 1. This is repeated for all the  $\sigma$  symbols, and the results are summed to obtain  $S(i) = \sum_{c \in \Sigma} S^c(i)$ , which is the total number of matches for a position  $i$ ; in other words, the Hamming distance is  $m - S(i)$ . The algorithm obviously runs in  $O(\sigma n \log m)$  worst case time.

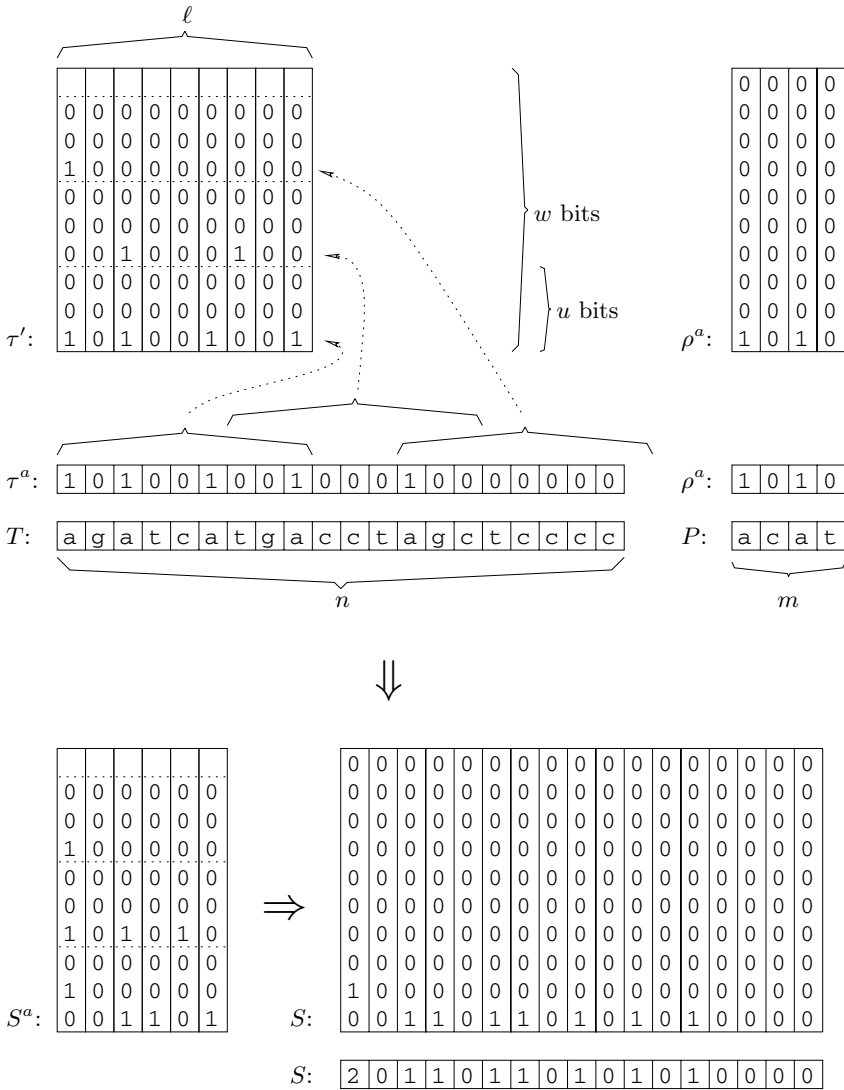
To improve this algorithm, we map  $\tau$  to  $\tau'$  for each symbol (see Fig. [1](#) for an example with unrealistically small  $n$ ). For this, we must fix the value of  $u$ . As the algorithm computes the number of matching characters between  $P$  and a window of  $T$ , the sum is at most  $m$ . Hence the number of bits needed for any value  $S(i)$  is  $u = \lceil \log(m + 1) \rceil$ . We use this value and proceed as above using the new representation. The total time for the convolutions is then  $O(\sigma \ell \log m)$ , which is  $O(\sigma n \log^2 m/w)$ . The preprocessing step takes  $O(\sigma n \log m/w + n)$  time (zeroing the vectors plus filling the actual values). Alternatively, the vectors  $\tau^c$  can also be built incrementally from  $\tau^{c-1}$  with setting and unsetting each affected field separately, which can be done in  $O(n)$  total time; as the preprocessing is never dominating in our algorithms we omit the details of this variant. The values of  $S$  can be computed from  $S'$  trivially in time  $O(\sigma n)$ , but we can add  $b$  values in parallel using directly the packed representation, which gives  $O(\sigma n \log m/w + n)$  total time for the postprocessing. The final complexity is then dominated by the convolutions. Alg. [2](#) gives the pseudo code. We have just obtained:

**Theorem 1.** *In word RAM with word length  $w$  bits, Hamming distance of  $P$  of length  $m = O(n/(w/\log m))$  can be computed in  $O(n + \sigma n \log^2 m/w)$  total worst case time for all  $n$  text positions.*

We note that  $S^c$  needs to be computed only for the cases where  $c$  occurs in  $P$ , as otherwise we know that all  $S^c(i)$  values are 0s. Let  $\sigma_P$  be the number of distinct alphabet symbols occurring in  $P$ , i.e.  $\sigma_P \in \{1 \dots \min(m, \sigma)\}$ . Second, we can derive a better bound for  $u$ , but this time the bound will be different for each symbol  $c$ . That is, assuming that symbol  $c$  occurs  $\sigma_c$  times in  $P$ , we use the value  $u_c = \lceil \log(\sigma_c + 1) \rceil$ . Hence the total time for the convolutions becomes

$$O\left(\sum_c n/w \log \sigma_c \log m\right).$$

Hence the average time depends on the compressibility of  $P$ , or more precisely on the 0-order entropy of  $P$ . The lower the entropy is, the faster the convolutions are computed. Given the convexity of the log function, the worst case is when all  $u_c$  values are equal, and therefore the time is upper bounded by  $O(\sigma_P n/w \log(m/\sigma_P) \log m)$ . Summing up the vectors can be done in  $O(\sigma_P n)$



**Fig. 1.** An (unrealistic) example of mapping  $T$  and  $P$  to  $\tau$  and  $\rho$  for an alphabet symbol **a**. Here  $w = 10$ , and  $u = \lceil \log(m + 1) \rceil = 3$ . For  $\tau'$  and  $\rho$  the least significant bit is at the bottom. The result of the convolutions  $\rho^a \otimes \tau'(i)$  is shown as  $S^a$ , and the corresponding vector  $S$  in both binary and decimal form, that is,  $S(i)$  denotes the number of matching **a**'s of  $P$  in  $T[i \dots i + m - 1]$  (so that the last value denotes the matches for  $T[n - m \dots n - 1]$ ).

time, which may be slower than previously, as we cannot parallelize the process easily anymore. Still, by assuming that  $u_c < \epsilon \log n$ , for some constant  $\epsilon < 1$  (e.g.,  $\epsilon = \frac{1}{2}$ ), for all  $\sigma_c$ , and by noticing that there are at most  $\log m$  different

**Alg. 2.** Word-packed-Hamming-distance( $T, n, P, m$ ).

---

```

1  /* preprocessing */
2   $u \leftarrow \lceil \log_2(m+1) \rceil$ 
3   $b \leftarrow \lfloor w/u \rfloor$ 
4   $\ell' \leftarrow \lfloor n/b \rfloor$ 
5  for  $c \leftarrow 0$  to  $\sigma - 1$  do
6      for  $i \leftarrow 0$  to  $m - 1$  do
7          if  $P[i] = c$  then  $\rho_i^c \leftarrow 1$  else  $\rho_i^c \leftarrow 0$ 
8      for  $i \leftarrow 0$  to  $\ell' - 1$  do
9           $\tau_i^c \leftarrow 0$ 
10         for  $h \leftarrow 0$  to  $b - 1$  do
11             if  $T[i + h\ell'] = c$  then  $\tau_i^c \leftarrow \tau_i^c + 2^{hu}$ 
12     /* FFT computations */
13     for  $c \leftarrow 0$  to  $\sigma - 1$  do
14         Compute  $S^c(i) = \rho^c \otimes \tau^c(i)$  for all  $i$  using FFT
15     /* postprocessing */
16     for  $i \leftarrow 0$  to  $\ell' - 1$  do
17          $S'(i) \leftarrow 0$ 
18         for  $c \leftarrow 0$  to  $\sigma - 1$  do
19              $S'(i) \leftarrow S'(i) + S^c(i)$ 
20     for  $i \leftarrow 0$  to  $n - m + 1$  do
21          $S(i) \leftarrow (S'(i \bmod \ell') \gg \lfloor i/\ell' \rfloor u) \& (2^u - 1)$ 
22     /* Convert match counts to Hamming distance */
23     for  $i \leftarrow 0$  to  $n - 1$  do
24          $S(i) \leftarrow m - S(i)$ 
25     return  $S$ 

```

---

$u_c$  values, one can use look-up tables of size  $O(2^{\varepsilon \log n} \log m) = o(n)$  to convert and add several counter values in parallel, giving

$$O\left(n + n \sum_c \log \sigma_c / \log n\right)$$

time for the postprocessing. Again, assuming that  $\log(m/\sigma_P) < \varepsilon \log n$  and using the convexity of the log function, this is upper bounded by

$$O(n + n\sigma_P \log(m/\sigma_P) / \log n).$$

In summary, we have obtained:

**Theorem 2.** *In word RAM with word length  $w$  bits, Hamming distance of  $P$  of length  $m = O(n/(w/\log \max \sigma_c))$ , where  $\max \sigma_c$  is the number of occurrences of the most frequent alphabet symbol in  $P$ , can be computed in  $O(n + \sigma_P n \log(m/\sigma_P)(\log m/w + 1/\log n))$  total worst case time for all  $n$  text positions.*

Thus computing the convolutions dominates the postprocessing as long as  $w = O(\log n \log m)$ , and the complexity is then  $O(n + \sigma_P n \log(m/\sigma_P) \log m/w)$ .

### 3.2 Matching Permitting $k$ Mismatches

In [1] Amir et al. showed how to find all occurrences of  $P$  from  $T$  permitting at most  $k$  mismatches in time  $O(n\sqrt{k} \log k)$ . This differs from the Hamming distance problem in that the actual distance is not computed for every text position. Their algorithm works as follows. Call a symbol  $c$  frequent, if it occurs

in  $P$  at least  $\alpha = 2\sqrt{k}$  times. The algorithm distinguishes between two cases. Case 1: there are at least  $\beta = \sqrt{k}$  frequent symbols. It is shown how this case can be handled in  $O(kn\alpha/k) = O(n\sqrt{k})$  time. Case 2: there are less than  $\beta$  frequent symbols; (a) the matches of the  $O(\beta)$  frequent symbols are computed using convolutions and FFT, hence this takes time  $O(\beta n \log m) = O(\sqrt{kn} \log m)$ ; (b) those symbols that are not frequent occur less than  $\alpha$  times, and can be handled in  $O(n\alpha)$  total time. The final match count for Case 2 is then the sum of the subcases (a) and (b). The final time complexity is then  $O(\sqrt{kn} \log m)$ , as the Case 2 (a) is dominating. They show how this can be improved to obtain the promised complexity of  $O(n\sqrt{k \log k})$ . See [1] for the details of the Case 1 and Case 2 (b).

Our aim is to improve Case 2 (a), which is based on convolutions and FFT. The improvement basically follows their ideas, but we obtain different optimal values of  $\alpha$  and  $\beta$ . The complexity of the above scheme (the whole algorithm) is  $O(n\alpha + n\beta \log m)$ , with the restriction that  $\alpha\beta \geq 2k$ . We can compute the convolutions more efficiently using our technique (Alg. 2), the only difference here is that the alphabet size is effectively reduced to  $\beta$ . Hence the complexity becomes  $O(n\alpha + n\beta \log^2 m/w)$ . Using  $\alpha$  and  $\beta$  as above, we obtain  $O(n\sqrt{k} + n\sqrt{k} \log^2 m/w)$ . However, the optimum values are  $\alpha = 2\sqrt{k/w} \log m$  and  $\beta = \sqrt{kw}/\log m$ , giving  $O(n\sqrt{k/w} \log m)$  worst case time. However, they also have a linear time algorithm for  $k < k' = O(m^{1/3-\varepsilon})$ , hence the above method is applied only for  $k \geq k'$ , but in this case  $\log m = O(\log k)$ . We have just proved:

**Theorem 3.** *The  $k$ -mismatches problem can be solved in  $O(n + n\sqrt{k/w} \log k)$  time.*

This is faster than the  $O(n\sqrt{k \log k})$  algorithm by a factor of  $\sim \sqrt{w/\log k}$ .

Finally, note that if  $k = \Theta(m)$ , or if one is interested in computing the actual Hamming distance for each text position, then the simpler  $O(n\sqrt{m \log m})$  time base-line algorithm [7,8] can be used instead. Assume again that symbol  $c$  is frequent if it occurs at least  $\alpha$  times. Consider now the infrequent symbols, i.e. those that occur less than  $\alpha$  times. The contribution of these symbols to the match can be computed by a simple linear scan over the text, spending  $O(\alpha)$  time per text position. The number of frequent symbols in  $P$  is then at most  $m/\alpha$ , and their contribution can be handled by using convolutions. Hence the total time is  $O(n\alpha + nm/\alpha \cdot \log m)$ , which is optimized by choosing  $\alpha = \sqrt{m \log m}$ . Similarly as before, if we compute the convolutions with our algorithm, the optimum turns out to be  $\alpha = \sqrt{m/w} \log m$ , and the total time becomes  $O(n\sqrt{m/w} \log m)$ .

### 3.3 Multiple Matching under Hamming Distance

The problem of multiple matching under Hamming distance or  $k$  mismatches model can be stated as follows: Given a text  $T$ , a set  $\mathcal{P}$  of  $r$  patterns,  $\mathcal{P} = \{P^1, P^2, \dots, P^r\}$ , and  $r$  non-negative integers  $k_h$ , for  $h \in \{0 \dots r-1\}$ , report all text positions  $i$  such that there exists an index  $h \in \{0 \dots r-1\}$  for which  $p_j^h = t_{i+j}$  for at least  $m - k_h$  locations  $j$  from the set  $0 \dots m-1$ . A less general problem variant could assume all error thresholds for the  $r$  patterns equal, but for our algorithm handling different  $k_h$  parameters does not pose any difficulty.

A naïve application of the Fischer–Paterson  $k$ -mismatches algorithm for multiple patterns would be to calculate the convolutions for each alphabet symbol and each pattern separately; then the products over the entire alphabet for each pattern are summed up, and whenever the sum for a given pattern reaches its corresponding threshold, a match is reported. In what follows, we keep the  $O(\sigma)$  multiplicative factor but parallelization is used to handle several patterns at the same time.

The technique is similar to what we used for a single pattern. Again the text  $T$  is mapped to  $\tau$ , that is,  $\tau_i^c = 1$  iff  $t_i = c$ , and 0 otherwise. Likewise, each pattern  $P^h$  in the set  $\mathcal{P}$  is mapped to corresponding  $\rho(h)$  vector. However, this time we do not pack  $\tau$  to  $\tau'$ . Instead, we pack several  $\rho$  vectors into a single word of  $w$  bits. Again, the basic method assumes that  $u = \lceil \log(m+1) \rceil$ , and thus each word can pack  $b = \lfloor w/u \rfloor$  values.

Similarly to our previous technique, define a new vector  $\rho'$ :

$$\rho'_i = \rho(1)_i \times 2^{0u} + \rho(2)_i \times 2^{1u} + \rho(3)_i \times 2^{2u} + \dots + \rho(r)_i \times 2^{(r-1)u}.$$

This vector uses  $ru$  bits per element, which may be larger than  $w$ ; in such case we just divide the set into smaller subsets. For simplicity we assume this implicitly. Then we can define and compute  $S'(i) = \rho' \otimes \tau(i)$  for each alphabet symbol, and sum up the results as previously. Each obtained sum, however, is a  $w$ -bit concatenation of  $u$  bit-fields, each of which stores the match count for a single pattern for its group. Testing for a match, i.e., if the match count is at least  $m - k_h$  for  $P^h$ , can be done naïvely, in  $O(r)$  time overall, per text location.

To sum up, the overall time complexity of this algorithm involves  $O(\sigma(n + rm) \log m/w + n + rm)$  preprocessing time,  $O(\sigma n \lceil r/(w/\log m) \rceil \log m)$  time for calculating convolutions with FFT,  $O(\sigma n \log m/w)$  time for summations over the alphabet symbols (this complexity term is absorbed by the previous ones) and  $O(nr)$  time for match counter checks and reporting. Overall, we get  $O(\sigma rm \log m/w + rm + n(\sigma r \log^2 m/w + r))$  worst-case time. Note that this is asymptotically the same as multiplying the complexity of our basic algorithm for single pattern by  $r$ , and that the same result can be obtained by applying the method in different ways.

Using different field widths ( $u$ ) depending on the actual number of occurrences of the symbols in  $P$  works in this setting as well, even if that number is not the same for all the patterns. I.e. each field in the packed representation ( $\rho'_i$ ) can have different width.

## 4 Computing Hamming Distance with Alphabet Packing

So far we discussed a way to represent several subsequences of the vector  $\tau$  (or  $T$ , if we talk about text over some integer alphabet) in a single vector of  $w$ -bit integers before calculating the convolution. Now we demonstrate that the packing idea can also be applied for several alphabet symbols rather than subsequences of  $\tau$  (or  $T$ ). This is interesting for the case of long patterns, in particular the

case of  $m = \Theta(n)$ . In this section we assume integer alphabet and talk about text  $T$  and pattern  $P$ . We consider the Hamming distance problem.

The key component of our construction is a possibly dense subsequence  $\mathcal{A}(N)$  of  $\{0, 1, 2, \dots, N-1\}$ , where  $N = O(w/\log m)$  in our problem, not containing any arithmetic progressions of length  $r = 3$ . Finding properties of such subsequences (also for larger  $r$ ) is a classic research topic in number theory [18]. In particular, Roth [19] proved that such subsequences for  $r = 3$  (a result which was later improved to any  $r > 2$ ) have density zero. In other words,  $|\mathcal{A}(N)|/N$  must tend to zero. We start with a very simple observation that if we multiply all terms of  $\mathcal{A}(N)$  by the same positive integer, the newly-created sequence will not contain any  $r$ -term arithmetic progressions either.

It is easy to generate a sequence  $\mathcal{A}(N)$  such that  $|\mathcal{A}(N)|$  tends to  $N^{\log_3 2} = N^{0.6309\dots}$  with  $N$  growing to infinity. This sequence is a discrete approximation of the famous Cantor ternary set. W.l.o.g. assume that  $N$  is a power of 3. From the interval containing all  $0, 1, \dots, N-1$  integers its middle third is first removed, and the sum of the intervals  $I_1 = 0, 1, \dots, N/3-1$  and  $I_2 = 2N/3, 2N/3+1, \dots, N-1$  is what remains. Note that we do not have any 3-term progression with the first term in  $I_1$  and the last term in  $I_2$ , since the middle term would fall in the deleted range. We continue the middle subinterval removals recursively with  $I_1$  and  $I_2$ , until no more integers can be removed. Here is the resulting set for  $N = 81$ :

$$0, 2, 6, 8, 18, 20, 24, 26, 54, 56, 60, 62, 72, 74, 78, 80.$$

Finding the optimal (possibly densest) subsequence for an arbitrary large  $N$  is still an open problem. The best upper bound result known at the moment [20] states that if  $N/|\mathcal{A}(N)| = \delta$ , where  $\delta \gg (\log \log N)^2/(\log N)^{2/3}$ , and  $N$  is sufficiently large, then  $\mathcal{A}$  must contain progressions of length 3. A lower bound is given in the classic Behrend theorem [21] and states that for an arbitrarily large  $N$  there exists a subsequence  $\mathcal{A}(N)$  such that  $N/|\mathcal{A}(N)| = e^{c\sqrt{\log N}}$ , where  $c$  is some constant. Less precisely, we can say that we can select  $\mathcal{A}(N)$  in such a way to have  $N/|\mathcal{A}(N)| = o(N^\varepsilon)$ , for any  $\varepsilon > 0$ .

We go back to our problem. Let  $\mathcal{A}(N) = \{A_0, A_1, \dots, A_{s-1}\}$ , and we require that its largest term,  $A_{s-1}$ , does not exceed  $w/(2\lceil \log(m+1) \rceil) - 1$ . W.l.o.g. assume that  $s$  divides  $\sigma$ , the alphabet size in the given problem instance. Now we can define vectors  $U^i$ ,  $0 \leq i \leq \sigma/s - 1$ , of length  $n$  each:

$$U_j^i = \chi_{si+0}(t_j) \times 2^{A_0u} + \chi_{si+1}(t_j) \times 2^{A_1u} + \dots + \chi_{si+s-1}(t_j) \times 2^{A_{s-1}u},$$

where  $u = \lceil \log(m+1) \rceil$ , and  $t_j$ ,  $0 \leq j < n$ , are symbols from  $T$ , and the characteristic function  $\chi_x(y)$  returns 1 iff  $x = y$ . Similarly we define vectors  $V_i$ , with respect to the pattern  $P$ . The convolutions are now calculated (with FFT) between vectors  $U_i$  and  $V_i$ , for all  $i$ . From the definition of  $U_i$  and  $V_i$ , and essentially their relation to  $\mathcal{A}(N)$ , we easily notice that the components of a convolution, i.e., the products of  $U_j^i$  and  $V_k^i$ , for given  $i, j$  and  $k$ , all affect different bit-fields (of size  $\lceil \log(m+1) \rceil$  bits) of the machine word that stores the final result, as long as the aligned pairs of symbols are unique. To obtain

the match count requires adding the values in the relevant bit-fields, which we explain briefly on an example. Imagine that  $i = 0$ ,  $s \geq 19$ , and there are three matches for the alphabet symbol 9 for the current alignment of the pattern over the text. The machine word storing the convolution of the corresponding  $U_0$  and  $V_0$  vectors will then have value 3 in its  $A_{18}$ -th bit-field. Mismatches also set some bit-fields but we simply ignore those. Overall, the time complexity of our algorithm is  $O(\sigma n + \sigma n \log m / |\mathcal{A}(w/\log m)|)$ , which is asymptotically  $O(\sigma n + \sigma n \log m (\log m/w)^{1-\varepsilon})$ , i.e., worse than the vector packing idea from Sec. 2. Still, if  $m$  is large, and in particular if  $m = \Theta(n)$ , this idea works better or is even the only one of those two applicable. More formally, we have obtained:

**Theorem 4.** *In word RAM with word length  $w$  bits, Hamming distance of  $P$  of length  $m = O(n)$  can be computed in  $O(\sigma n + \sigma n \log m (\log m/w)^{1-\varepsilon})$  total worst case time for all  $n$  text positions. The value  $\varepsilon$  asymptotically approaches zero with  $w/\log m$  growing to infinity.*

## 5 Conclusions

We presented two techniques to speed-up existing algorithms for Hamming distance calculations and related problems. They are based on the concept of bit-parallelization of FFT-based convolutions. The technique of matching the pattern against several subsequences of the text in parallel can trivially be adopted also for multiple pattern matching under Hamming distance. Note that in the  $k$ -mismatches variation of the problem we can immediately use different values of  $k$  for different patterns. We believe that our technique(s) have applications for many other string matching problems; exploring those possibilities is left for future work. One immediate application of our technique (not mentioned earlier in the paper) is matching with wild cards using the classic algorithm [3]. Other possibilities include especially music information retrieval motivated problems. For example, there exist FFT-based algorithms for  $\delta$ -matching,  $(\delta, \gamma)$ -matching and  $\gamma$ -matching [4,5], and bit-parallelism should be a way to improve their complexities.

Finally we note that other trade-offs for Hamming distance are possible; e.g. the techniques of Theorems 2 and 4 could be combined to avoid using look-up tables in the postprocessing. We leave the details for future work.

## References

1. Amir, A., Lewenstein, M., Porat, E.: Faster algorithms for string matching with  $k$  mismatches. In: Proceedings of the 11th ACM-SIAM Annual Symposium on Discrete Algorithms, San Francisco, CA, pp. 794–803 (2000)
2. Fredman, M.L., Willard, D.E.: Surpassing the information theoretic bound with fusion trees. *J. Comput. System Sci.* 47, 424–436 (1993)
3. Fischer, M.J., Paterson, M.: String matching and other products. In: Karp, R.M. (ed.) Proceedings of the SIAM-AMS Complexity of Computation, Providence, RI, pp. 113–125 (1974)



4. Clifford, P., Clifford, R., Iliopoulos, C.S.: Faster algorithms for  $\delta, \gamma$ -matching and related problems. In: Apostolico, A., Crochemore, M., Park, K. (eds.) CPM 2005. LNCS, vol. 3537, pp. 68–78. Springer, Heidelberg (2005)
5. Amir, A., Lipsky, O., Porat, E., Umanski, J.: Approximate matching in the  $L_1$  metric. In: Apostolico, A., Crochemore, M., Park, K. (eds.) CPM 2005. LNCS, vol. 3537, pp. 91–103. Springer, Heidelberg (2005)
6. Indyk, P.: Faster algorithms for string matching problems: matching the convolution bound. In: Proceedings of the 39th IEEE Annual Symposium on Foundations of Computer Science, pp. 166–173. IEEE Computer Society Press, Los Alamitos (1998)
7. Abrahamson, K.: Generalized string matching. *SIAM Journal on Computing* 16(6), 1039–1051 (1987)
8. Kosaraju, S.R.: Efficient string matching (1987) (manuscript)
9. Landau, G.M., Vishkin, U.: Efficient string matching with  $k$  mismatches. *Theor. Comput. Sci.* 43(2-3), 239–249 (1986)
10. Grabowski, S., Fredriksson, K.: Bit-parallel string matching under Hamming distance in  $O(n \lceil m/w \rceil)$  worst case time. *Information Processing Letters* 105(5), 182–187 (2008)
11. Linhart, C., Shamir, R.: Faster pattern matching with character classes using prime number encoding. *Journal of Computer and System Sciences* 75(3), 155–162 (2009)
12. Galil, Z., Giancarlo, R.: Improved string matching with  $k$  mismatches. *SIGACT News* 17(4), 52–54 (1986)
13. Galil, Z., Giancarlo, R.: Data structures and algorithms for approximate string matching. *J. Complexity* 4(1), 33–72 (1988)
14. Thorup, M.: Combinatorial power in multimedia processors. *SIGARCH Comput. Archit. News* 31(4), 5–11 (2003)
15. Thorup, M.: On  $AC^0$  implementations of fusion trees and atomic heaps. In: Proceedings of the 14th ACM-SIAM Annual Symposium on Discrete Algorithms, Philadelphia, PA, USA, pp. 699–707. Society for Industrial and Applied Mathematics (2003)
16. Knuth, D.: The art of computer programming: Combinatorial algorithms. Prefascicle 1a. Draft of section 7.1.3: Bitwise tricks and techniques (2008), <http://www-cs-faculty.stanford.edu/~knuth/fasc1a.ps.gz>
17. Smith III, J.O.: Mathematics of the Discrete Fourier Transform (DFT), with Audio Applications, 2nd edn. W3K Publishing (2007)
18. Graham, R.L., Rothschild, B.L.: Ramsey theory, 2nd edn. Wiley-Interscience, New York (1990)
19. Roth, K.F.: On certain sets of integers. *J. London Math. Soc.* 28, 104–109 (1953)
20. Bourgain, J.: Roth’s theorem on progressions revisited. *Journal d’Analyse Mathématique* 104(1), 155–192 (2008)
21. Behrend, F.A.: On sets of integers which contain no three in arithmetic progression. *Proc. Nat. Acad. Sci.* 23, 331–332 (1946)

# Better Polynomial Algorithms on Graphs of Bounded Rank-Width

Robert Ganian and Petr Hliněný\*

Faculty of Informatics, Masaryk University  
Botanická 68a, 602 00 Brno, Czech Republic  
ganian@mail.muni.cz, hlineny@fi.muni.cz

**Abstract.** Although there exist many polynomial algorithms for *NP*-hard problems running on a bounded clique-width expression of the input graph, there exists only little comparable work on such algorithms for rank-width. We believe that one reason for this is the somewhat obscure and hard-to-grasp nature of rank-decompositions. Nevertheless, strong arguments for using the rank-width parameter have been given by recent formalisms independently developed by Courcelle and Kanté, by the authors, and by Bui-Xuan et al. This article focuses on designing formally clean and understandable “pseudopolynomial” (XP) algorithms solving “hard” problems (non-FPT) on graphs of bounded rank-width. Those include computing the chromatic number and polynomial or testing the Hamiltonicity of a graph and are extendable to many other problems.

**Keywords:** Rank-width, rank-decomposition, chromatic number, chromatic polynomial, Hamiltonian path.

## 1 Introduction

We postpone all formal definitions till the next section. Rank-width, introduced by Oum and Seymour [20], is a relatively new graph complexity measure which is quickly receiving attention over the past few years. Compared to the (perhaps better known) clique-width measure, rank-width has two major advantages: First, an optimal rank-decomposition can be efficiently constructed if the rank-width is bounded [17]. Second, a rank-decomposition (actually, a suitable modification of it, see Section 3) allows for design of formally cleaner [11], and sometimes much faster parameterized algorithms [2, 12] than previously known ones running on a clique-width expression of the given graph.

The core of the new approach lies in an alternative characterization of rank-decompositions using bilinear graph products [4], or equivalently labeling parse trees [10, 11], or  $R_k$ -joins [2]. These new approaches have led to interesting new

---

\* Both authors have been supported by the the research grants GAČR 201/08/0308 and 201/09/J021, and P. Hliněný also by the research intent MSM0021622419 of the Czech Ministry of Education.

FPT algorithms [2,12] which run much faster with respect to the rank-width parameter than previously known algorithms, as in [5].

The aim of this paper is to extend the ideas of the mentioned algorithms to problems which likely do not have FPT algorithms with respect to rank-width, and hence their presented algorithms are “pseudopolynomial”, a complexity class known as XP. We present some of the basic definitions in Section 2, and describe the parse-tree formalism for handling rank-decompositions of graphs in Section 3. The main new results are then presented and proved in Section 4.

We prove (Theorem 4.1) that the chromatic number of a graph of rank-width  $t$  can be determined in time  $O(n^{h(t)})$  where  $h(t) = O(2^{t(t+1)/2})$ . This algorithm significantly improves over a previous algorithm of Kobler and Rotics [19] which runs in time  $O(n^{4^k})$  on graphs of clique-width  $k$ . When comparing these two algorithms, the readers should keep in mind that our parameter  $t$  is the rank-width of the input graph, and the clique-width  $k$  can reach up to  $2^{t/2-1}$  by [3]. We, moreover, straightforwardly extend our algorithm (Theorem 4.8) to compute the chromatic polynomial, again improving runtime over previous [1]. Finally, we show (Theorem 4.9) how to decide Hamiltonian path in a graph of bounded rank-width.

## 2 Definitions and Basics

We only consider finite undirected simple graphs without loops. We will start by briefly introducing a few needed concepts and then define rank-decompositions and rank-width, while in Section 3 we continue by defining the concepts of  $t$ -labeled graphs and their parse trees. Many of the definitions in the latter section are taken or adapted from our [11].

The reader should be aware of the notion of *fixed-parameter tractable* [6] algorithms (FPT algorithms in short), which are the algorithms running in time  $O(n^c \cdot 2^{f(k)})$  for a constant  $c$ , a parameter  $k$  (rank-width in our case) and any (computable)  $f$ . Some NP-hard problems such as deciding whether a graph is  $q$ -colourable do have FPT algorithms when parameterized by clique-width, see e.g. [5]. On the other hand, [9] have recently proved that various problems, such as the chromatic number or hamiltonicity, likely can not be solved by FPT algorithms parameterized by clique-/rank-width.

In such cases, authors usually look for algorithms which are “pseudopolynomial”—formally in class XP or *uniform XP* [6]—i.e. running in time  $O(n^{f(k)})$  for the parameter  $k$  and a computable function  $f$ . Many examples using the clique-width parameter can be found in [17,13,19,21]. Our goal in this paper is to design and use a mathematically precise and sound formalism for solving problems on graphs of bounded rank-width in XP time. This extends the Myhill–Nerode type formalism which we have introduced in [11] for FPT algorithms on such graphs.

**Branch-width.** A set function  $f : 2^M \rightarrow \mathbb{Z}$  is called *symmetric* if  $f(X) = f(M \setminus X)$  for all  $X \subseteq M$ . A tree is *subcubic* if all its nodes have degree at

most 3. For a symmetric function  $f : 2^M \rightarrow \mathbb{Z}$  on a finite set  $M$ , the branch-width of  $f$  is defined as follows.

A *branch-decomposition* of  $f$  is a pair  $(T, \mu)$  of a subcubic tree  $T$  and a bijective function  $\mu : M \rightarrow \{t : t \text{ is a leaf of } T\}$ . For an edge  $e$  of  $T$ , the connected components of  $T \setminus e$  induce a bipartition  $(X, Y)$  of the set of leaves of  $T$ . The *width* of an edge  $e$  of a branch-decomposition  $(T, \mu)$  is  $f(\mu^{-1}(X))$ . The *width* of  $(T, \mu)$  is the maximum width over all edges of  $T$ . The *branch-width* of  $f$  is the minimum of the width of all branch-decompositions of  $f$ . (If  $|M| \leq 1$ , then we define the branch-width of  $f$  as  $f(\emptyset)$ .)

A natural application of this definition is the branch-width of a graph, introduced by Robertson and Seymour along with better known tree-width. In that case we use  $M = E(G)$ , and  $f$  the connectivity function of  $G$ . There is, however, another interesting application of the aforementioned general notions, in which we consider the vertex set  $V(G) = M$  of a graph  $G$  as the ground set.

**Rank-width** ([20]). For a graph  $G$ , let  $\mathbf{A}_G[U, W]$  be the bipartite adjacency matrix of a bipartition  $(U, W)$  of the vertex set  $V(G)$  defined over the two-element field  $\text{GF}(2)$  as follows: the entry  $a_{u,w}$ ,  $u \in U$  and  $w \in W$ , of  $\mathbf{A}_G[U, W]$  is 1 if and only if  $uw$  is an edge of  $G$ . The *cut-rank* function  $\rho_G(U) = \rho_G(W)$  then equals the rank of  $\mathbf{A}_G[U, W]$  over  $\text{GF}(2)$ . A *rank-decomposition* and *rank-width* of a graph  $G$  is the branch-decomposition and branch-width of the cut-rank function  $\rho_G$  of  $G$  on  $M = V(G)$ , respectively.

**Theorem 2.1** ([17]). *For every fixed  $t$  there is an  $O(n^3)$ -time FPT algorithm that, for a given  $n$ -vertex graph  $G$ , either finds a rank-decomposition of  $G$  of width at most  $t$ , or confirms that the rank-width of  $G$  is more than  $t$ .*

**A Few Rank-width Examples.** Any complete graph of more than one vertex has clearly rank-width 1 since any of its bipartite adjacency matrices consists of all 1s. It is similar with complete bipartite graphs if we split the decomposition along the parts. We illustrate the situation with graph cycles: while  $C_3$  and  $C_4$  have rank-width 1,  $C_5$  and all longer cycles have rank-width equal 2. A rank-decomposition of, say, the cycle  $C_5$  is shown in Fig. 1. Conversely, every subcubic tree with at least 4 leaves has an edge separating at least 2 leaves on each side, and every corresponding bipartition of  $C_5$  gives a matrix of rank 2.

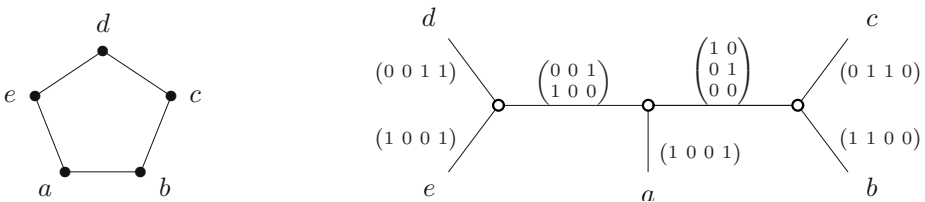


Fig. 1. A rank-decomposition of the graph cycle  $C_5$

Rank-width is closely tied to another width parameter called *clique-width* [7]. A graph has bounded rank-width if and only if it has bounded clique-width. However, there is no equivalent of Theorem 2.1 for clique-width [8], and the value of clique-width can be up to exponentially larger than rank-width [3], both making rank-width a more attractive parameter for designing algorithms. On the other hand, it appears really difficult to design dynamic-programming algorithms running on a given rank-decomposition of a graph.

### 3 Rank-Width Parse Trees and Regularity

In a search for a “more suitable form” of a rank-decomposition, Courcelle and Kanté [4] defined the bilinear products of multiple-coloured graphs, and proposed algebraic expressions over these operators as an equivalent description of a rank-decomposition (cf. Theorem 3.1). Here we introduce (following [10,11]) the same idea in terms of labeling join and parse trees which we propose as more convenient for the results in the next sections. One should note that an analogous idea also underlies the *H*-join decompositions of Bui-Xuan, Telle and Vatshelle [2].

A *t*-labeling of a graph is a mapping  $lab : V(G) \rightarrow 2^{L_t}$  where  $L_t = \{1, 2, \dots, t\}$  is the set of *labels* (this notion is exactly equivalent to multiple-coloured graphs of [4]). Having a graph  $G$  with an (implicitly) associated *t*-labeling  $lab$ , we refer to the pair  $(G, lab)$  as to a *t*-labeled graph and use notation  $\bar{G}$ . Notice that each vertex of a *t*-labeled graph may have zero, one or more labels. We will often view (cf. [4] again) a *t*-labeling of  $G$  equivalently as a mapping  $V(G) \rightarrow \text{GF}(2)^t$  to the *binary vector space* of dimension  $t$ , where  $\text{GF}(2)$  is the two-element finite field.

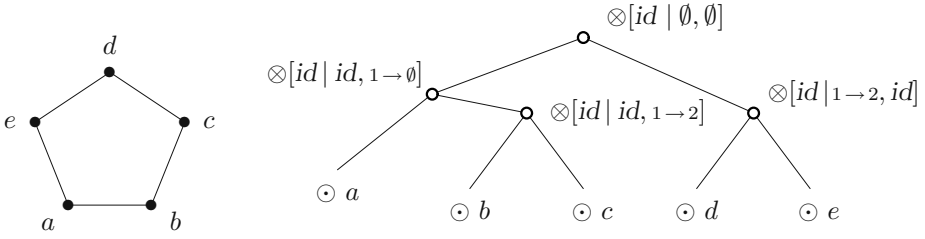
**Labeling Join ([11]).** Considering *t*-labeled graphs  $\bar{G}_1 = (G_1, lab^1)$  and  $\bar{G}_2 = (G_2, lab^2)$ , a *t*-labeling join  $\bar{G}_1 \otimes \bar{G}_2$  is defined on the disjoint union of  $G_1$  and  $G_2$  by adding all edges  $(u, v)$  such that  $|lab^1(u) \cap lab^2(v)|$  is odd, where  $u \in V(G_1), v \in V(G_2)$ . The resulting graph is unlabeled.

A *t*-relabeling is a mapping  $f : L_t \rightarrow 2^{L_t}$ . In linear algebra terms, a *t*-relabeling  $f$  is in a natural one-to-one correspondence with a *linear transformation*  $f : \text{GF}(2)^t \rightarrow \text{GF}(2)^t$ , i.e. a  $t \times t$  binary matrix  $\mathbf{A}_f$ . For a *t*-labeled graph  $\bar{G} = (G, lab)$  we define  $f(\bar{G})$  as the same graph with a vertex *t*-labeling  $lab' = f \circ lab$ . Here  $f \circ lab$  stands for the linear transformation  $f$  applied to the labeling  $lab$ , or equivalently  $lab' = lab \cdot \mathbf{A}_f$  as matrix multiplication. Informally,  $f$  is applied separately to each label in  $lab(v)$  and the outcomes are summed up “modulo 2”; e.g. for  $lab(v) = \{1, 2\}$  and  $f(1) = \{1, 3, 4\}$ ,  $f(2) = \{1, 2, 3\}$ , we get  $f \circ lab(v) = \{2, 4\} = \{1, 3, 4\} \Delta \{1, 2, 3\}$ .

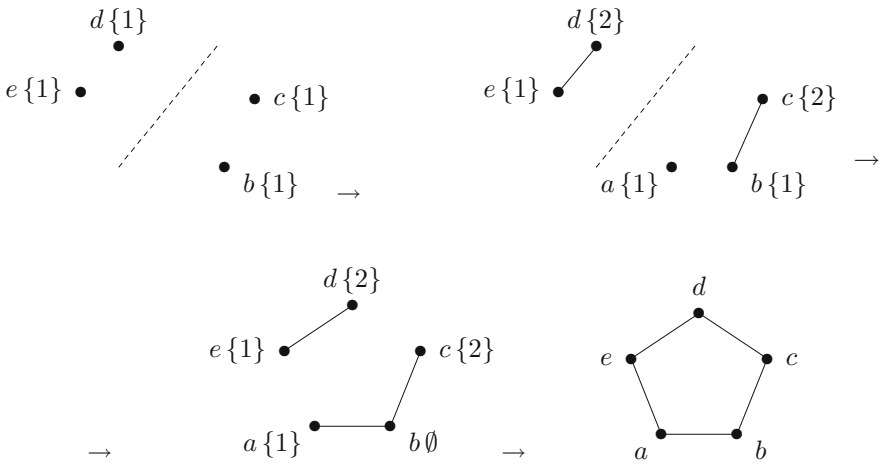
Let  $\odot$  be a nullary operator creating a single new graph vertex of label  $\{1\}$ . For *t*-relabelings  $f_1, f_2, g : L_t \rightarrow 2^{L_t}$ , let  $\otimes[g | f_1, f_2]$  be a binary operator—called *t*-labeling composition (as bilinear product of [4])—over pairs of *t*-labeled graphs  $\bar{G}_1 = (G_1, lab^1)$  and  $\bar{G}_2 = (G_2, lab^2)$  defined as follows:

$$\bar{G}_1 \otimes [g | f_1, f_2] \bar{G}_2 = \bar{H} = (\bar{G}_1 \otimes g(\bar{G}_2), lab)$$

where a new labeling is  $lab(v) = f_i \circ lab^i(v)$  for  $v \in V(G_i)$ ,  $i = 1, 2$ .



**Fig. 2.** An example of a labeling parse tree which generates a 2-labeled cycle  $C_5$ , with symbolic relabelings at the nodes ( $id$  denotes the relabeling preserving all labels, and  $\emptyset$  is the relabeling “forgetting” all labels)



**Fig. 3.** “Bottom-up” generation of  $C_5$  by the parse tree from Fig. 2

A  $t$ -labeling parse tree  $T$ , see also [10, Definition 6.11], is a finite rooted ordered subcubic tree (with the root degree at most 2) such that

- All leaves of  $T$  contain the  $\odot$  symbol, and
- Each internal node of  $T$  contains one of the  $t$ -labeling composition symbols.

A parse tree  $T$  then *generates* (parses) the graph  $G$  which is obtained by successive leaves-to-root applications of the operators in the nodes of  $T$ . See Fig. 2, 3.

Analogously to the work of Courcelle and Kanté we get a crucial statement:

**Theorem 3.1 (Rank-width parsing theorem [4, 11]).** *A graph  $G$  has rank-width at most  $t$  if and only if (some labeling of)  $G$  can be generated by a  $t$ -labeling parse tree. Furthermore, a width- $t$  rank-decomposition of  $G$  can be transformed into a  $t$ -labeling parse tree on  $\Theta(|V(G)|)$  nodes in time  $O(t^2 \cdot |V(G)|^2)$ .*

The tools we use are inspired by the Myhill–Nerode theorem in automata theory. We see this classical theorem as the starting point of formal understanding of

dynamic algorithms: Such an algorithm typically collects “all relevant information” about the studied problem on a local part, and then processes this information “through” the whole input. The task is to determine what the words “all relevant information” mean here.

The basic approach defines a so-called canonical equivalence, e.g. [16,11], which is analogous to the “right congruence” known in theory of regular languages. A strongly enhanced formalism, called a PCE scheme, is given by the authors in [12]. The PCE scheme provides a very fine control over the runtime of FPT algorithms on graphs of bounded rank-width. On the other hand, in this paper we show a formally precise handling of problems for which we do not have FPT algorithms (wrt. rank-width), and which thus require a different, yet similar, formal approach.

## 4 Applications in XP Algorithms

### 4.1 Computing the Chromatic Number

We illustrate our formalism on the graph chromatic number problem, for which we strongly improve the previous algorithm of [19] running on graphs of bounded clique-width. For the purposes of this section, it is useful to think about colouring not as a function from vertices to colours but rather as a vertex-partition of  $G$ . Formally, a *colour partition* of  $G$  is a partition  $\mathcal{N}$  of  $V(G)$  into pairwise disjoint nonempty(!) sets such that each  $X \in \mathcal{N}$  is independent in  $G$ . The *chromatic number* of a graph  $G$  is the minimum  $|\mathcal{N}|$  such that  $\mathcal{N}$  is a colour partition of  $G$ .

**Theorem 4.1.** *Assume that an input graph  $G$  is given in the form of a  $t$ -labeling parse tree  $T$ . Then the chromatic number of  $G$  can be computed in time*

$$O\left(|V(G)|^{h(t)}\right) \text{ where } h(t) = O(2^{t(t+1)/2}).$$

We will also need a few preliminary technical results. Let, for  $X \subseteq V(G)$ ,  $\gamma(\bar{G}, X) = \{lab(u) : u \in X\}$ . Notice that this set of labelings—vectors in  $\text{GF}(2)^t$ —generates a vector subspace  $\langle \gamma(\bar{G}, X) \rangle$ . Considering a  $t$ -labeled graph  $\bar{G} = (G, lab)$  with a proper colour partition  $\{C_1, C_2 \dots C_j\}$ , the core idea is that we only need to record the subspaces  $\langle \gamma(\bar{G}, C_i) \rangle$  for  $i = 1, \dots, j$ .

**Lemma 4.2 (also [2,12]).** *Assume  $t$ -labeled graphs  $\bar{G}$  and  $\bar{H}$ , and arbitrary sets  $X \subseteq V(\bar{G})$ ,  $Y \subseteq V(\bar{H})$ . In the join graph  $\bar{G} \otimes \bar{H}$ , there is no edge between a vertex of  $X$  and a vertex of  $Y$  if and only if the subspace  $\langle \gamma(\bar{G}, X) \rangle$  is orthogonal to the subspace  $\langle \gamma(\bar{H}, Y) \rangle$  in  $\text{GF}(2)^t$ .*

**Proof.** Let  $\bar{G} = (G, lab)$  and  $\bar{H} = (H, lab')$ . We consider arbitrary  $y \in Y$ . Then (as a scalar product in  $\text{GF}(2)$ )  $lab(x) \cdot lab'(y) = 0$  for all  $x \in X$ ; henceforth  $\alpha \cdot lab'(y) = 0$  for all  $\alpha \in \langle \gamma(\bar{G}, X) \rangle$  by means of elementary linear algebra. By a symmetrical argument, we get  $\alpha \cdot \beta = 0$  for all  $\beta \in \langle \gamma(\bar{H}, Y) \rangle$ , which means these subspaces indeed are mutually orthogonal. Conversely, if  $\langle \gamma(\bar{G}, X) \rangle \perp \langle \gamma(\bar{H}, Y) \rangle$ , then there is obviously no edge between  $x \in X$  and  $y \in Y$  by the definition of  $\otimes$  as  $lab(x) \cdot lab'(y) = 0$ . ■

**Corollary 4.3.** *Assume  $t$ -labeled graphs  $\bar{G}_1, \bar{G}_2$  and  $\bar{H}$ , and independent sets  $X_i \subseteq V(\bar{G}_i)$ ,  $i = 1, 2$  and  $Y \subseteq V(\bar{H})$ . If  $\langle \gamma(\bar{G}_1, X_1) \rangle = \langle \gamma(\bar{G}_2, X_2) \rangle$ , then  $X_1 \cup Y$  is independent in  $\bar{G}_1 \otimes \bar{H}$  if and only if  $X_2 \cup Y$  is independent in  $\bar{G}_2 \otimes \bar{H}$ .*

**Lemma 4.4** ([15], cf. [12, Proposition 6.1]). *The number  $S(t)$  of subspaces of the binary vector space  $\text{GF}(2)^t$  satisfies  $S(t) \leq 2^{t(t+1)/4} - 2$  for all  $t \geq 12$ . ■*

Informally, the algorithm for Theorem 4.1 starts by colouring all vertices in the leaves of the labeling parse tree by distinct colours. Notice that if we keep the colours distinct after performing the join operator, the resulting colour partition of the graph will always be proper — each vertex will have a unique colour. However, to compute the chromatic number, we also need to consider two colours merging together during a join operation. The resulting colour partition after such a merge is only proper if there was no edge between the merging colours, and we must make sure that this can be algorithmically determined from the information we store. The dynamic programming algorithm then, at every join operator, considers all possibilities of colour merges and stores some information about these possible colour partitions

**Proof of Theorem 4.1.** Given a graph  $G$ , we will write  $G \models \nu(\mathcal{N})$  to say that a set family  $\mathcal{N} \subseteq 2^{V(G)} \setminus \emptyset$  is a proper colour partition of  $G$ . Also, having two set families  $\mathcal{N}, \mathcal{N}'$ , we denote by  $\mathcal{I}(\mathcal{N}', \mathcal{N})$  the set of all injective mappings  $p : \mathcal{N}' \rightarrow \mathcal{N}$ , and we write  $\mathcal{N} \stackrel{p}{\leftarrow} \mathcal{N}'$  for  $p \in \mathcal{I}(\mathcal{N}', \mathcal{N})$  to denote the family

$$\mathcal{N} \stackrel{p}{\leftarrow} \mathcal{N}' = \{X \cup p(X) : X \in \mathcal{N}'\} \cup (\mathcal{N} \setminus p(\mathcal{N}')) .$$

Informally,  $\mathcal{N} \stackrel{p}{\leftarrow} \mathcal{N}'$  expands the colour partition  $\mathcal{N}$  of  $G$  by merging some of its colour classes with those of  $\mathcal{N}'$  as prescribed by  $p$ .

For any  $t$ -labeled graphs  $\bar{G}_1, \bar{G}_2$  and any colour partitions  $\mathcal{N}_i \subseteq 2^{V(G_i)} \setminus \emptyset$  where  $i = 1, 2$ , we define  $(\bar{G}_1, \mathcal{N}_1) \approx_{\nu, t} (\bar{G}_2, \mathcal{N}_2)$  if and only if  $|\mathcal{N}_1| = |\mathcal{N}_2| = q$  and the following holds true: For all  $t$ -labeled graphs  $\bar{H}$  and all colour partitions  $\mathcal{N} \subseteq 2^{V(H)}$ , and for all  $\mathcal{N}_0 \subseteq \mathcal{N}$  such that  $|\mathcal{N}_0| \leq q$ , it holds

$$(4.5) \quad \begin{aligned} &\exists p_1 \in \mathcal{I}(\mathcal{N}_0, \mathcal{N}_1) : (\bar{G}_1 \otimes \bar{H}) \models \nu((\mathcal{N}_1 \stackrel{p_1}{\leftarrow} \mathcal{N}_0) \cup (\mathcal{N} \setminus \mathcal{N}_0)) \\ \iff &\exists p_2 \in \mathcal{I}(\mathcal{N}_0, \mathcal{N}_2) : (\bar{G}_2 \otimes \bar{H}) \models \nu((\mathcal{N}_2 \stackrel{p_2}{\leftarrow} \mathcal{N}_0) \cup (\mathcal{N} \setminus \mathcal{N}_0)) . \end{aligned}$$

Note that  $(\bar{G}_1, \mathcal{N}_1) \approx_{\nu, t} (\bar{G}_2, \mathcal{N}_2)$  means that there is no real difference between the  $q$ -colour-partitioned  $(\bar{G}_1, \mathcal{N}_1)$  and  $(\bar{G}_2, \mathcal{N}_2)$  with respect to the possibility of merging prescribed colour classes  $\mathcal{N}_0$  of any joined graph  $\bar{H}$  with some existing colour classes in  $\mathcal{N}_1, \mathcal{N}_2$ . Hence,  $\approx_{\nu, t}$  captures all information necessary to decide which subcolourings of  $\bar{G}_i$  extend to colourings of any larger  $\bar{G}_i \otimes \bar{H}$ .

Let  $\Gamma(\bar{G}, \mathcal{N}) = \{\langle \gamma(\bar{G}, X) \rangle : X \in \mathcal{N}\}$  denote a multiset(!) of subspaces of  $\text{GF}(2)^t$ . The crucial finding, inspired by the colouring algorithm in [19], reads:

**(4.6)** For any  $t$ -labeled graphs  $\bar{G}_1, \bar{G}_2$  and any  $\mathcal{N}_i \subseteq 2^{V(G_i)} \setminus \emptyset$ ,  $i = 1, 2$  such that  $\bar{G}_i \models \nu(\mathcal{N}_i)$ , it holds  $(\bar{G}_1, \mathcal{N}_1) \approx_{\nu, t} (\bar{G}_2, \mathcal{N}_2)$  if  $\Gamma(\bar{G}_1, \mathcal{N}_1) = \Gamma(\bar{G}_2, \mathcal{N}_2)$ .



We will use  $\mathcal{N}_i^+$  to denote  $(\mathcal{N}_i \stackrel{p_i}{\leftarrow} \mathcal{N}_0) \cup (\mathcal{N} \setminus \mathcal{N}_0)$ ,  $i = 1, 2$  (cf. [4.5](#)). To prove this claim, we assume  $\Gamma(\bar{G}_1, \mathcal{N}_1) = \Gamma(\bar{G}_2, \mathcal{N}_2)$  and  $(\bar{G}_1 \otimes \bar{H}) \models \nu(\mathcal{N}_1^+)$  for some fixed  $p_1$ . Since  $\mathcal{N}_2^+$  is also a partition of the vertices of  $\bar{G}_2 \otimes \bar{H}$ , for claiming  $\nu(\mathcal{N}_2^+)$  it suffices to verify that all  $C \in \mathcal{N}_2^+$  are independent in the graph  $\bar{G}_2 \otimes \bar{H}$ . That is trivial if  $C \in \mathcal{N}_2$  or  $C \in (\mathcal{N} \setminus \mathcal{N}_0)$ , since both  $\bar{G}_2$  and  $\bar{H}$  were properly coloured and in this case the whole  $C$  is present in one of the original graphs. For the rest, we consider any bijection  $b : \mathcal{N}_1 \rightarrow \mathcal{N}_2$  preserving  $\langle \gamma(\bar{G}_1, X) \rangle = \langle \gamma(\bar{G}_2, b(X)) \rangle$  for all  $X \in \mathcal{N}_1$ , and choose  $p_2 = b \circ p_1$ . Assume  $C = Y \cup p_2(Y) \in \mathcal{N}_2^+$  where  $Y \in \mathcal{N}_0$ . Applying Corollary [4.3](#) with  $X_1 = p_1(Y)$  and  $X_2 = p_2(Y)$ , we conclude that  $C$  is independent in  $\bar{G}_2 \otimes \bar{H}$  if  $p_1(Y) \cup Y$  is independent in  $\bar{G}_1 \otimes \bar{H}$ , and  $p_1(Y) \cup Y \in \mathcal{N}_1^+$ .

Considering the labeling parse tree  $T$  of  $G$ , and a node  $z$  of  $T$ , let  $\bar{G}_z$  denote the  $t$ -labeled graph parsed by the subtree of  $T$  rooted at  $z$ . Thanks to previous Claim [4.6](#), a dynamic algorithm for computing the chromatic number of  $G$  has to remember only the set  $M_T(z)$  of those multisets  $\Gamma(\bar{G}_z, \mathcal{N})$  coming from proper colour partitions  $\mathcal{N}$  of  $V(G_z)$ , at any particular node  $z$  of  $T$ . This information is trivial to construct at each leaf of  $T$ .

We now show how to obtain the set  $M_T(z)$  from the sets  $M_T(x)$  and  $M_T(y)$  of the left son  $x$  and right son  $y$  of our  $z$ . We consider any proper colour partitions  $\mathcal{N}_x$  and  $\mathcal{N}_y$  of  $\bar{G}_x$  and  $\bar{G}_y$ , respectively. Let  $\mathcal{J}(\mathcal{N}_y, \mathcal{N}_x)$  denote the set of all partial injective mappings  $p$  from  $\mathcal{N}_y$  into  $\mathcal{N}_x$  (i.e. of injective mappings from any subset of  $\mathcal{N}_y$  into  $\mathcal{N}_x$ ), and let  $\stackrel{p}{\Rightarrow}$  be a generalization of  $\stackrel{p}{\leftarrow}$  such that

$$\mathcal{N}_x \stackrel{p}{\Rightarrow} \mathcal{N}_y = \{X \cup p(X) : X \in p^{-1}(\mathcal{N}_x)\} \cup (\mathcal{N}_y \setminus p^{-1}(\mathcal{N}_x)) \cup (\mathcal{N}_x \setminus p(\mathcal{N}_y)).$$

It is obvious that any colour partition of  $\bar{G}_z$  induces colour partitions of  $\bar{G}_x$  and  $\bar{G}_y$ , and so every  $\bar{G}_z \models \nu(\mathcal{N}_z)$  is obtained as  $\mathcal{N}_z = (\mathcal{N}_x \stackrel{p}{\Rightarrow} \mathcal{N}_y)$  for some  $\bar{G}_x \models \nu(\mathcal{N}_x)$  and  $\bar{G}_y \models \nu(\mathcal{N}_y)$ , and suitable  $p \in \mathcal{J}(\mathcal{N}_y, \mathcal{N}_x)$ .

For  $p \in \mathcal{J}(\mathcal{N}_y, \mathcal{N}_x)$ , we define the “signature”  $Sig(p)$  as an edge-weighted bipartite graph  $D_p$  on the vertex set  $\mathcal{S} \cup \mathcal{S}$ , where  $\mathcal{S}$  is the family of all subspaces of  $\text{GF}(2)^t$ .  $f = \Psi\Psi' \in \mathcal{S} \times \mathcal{S}$  is an edge of  $D_p$  iff there is  $Y \in \mathcal{N}_y$  in the domain of  $p$  such that  $\Psi' = \langle \gamma(\bar{G}_y, Y) \rangle$  and  $\Psi = \langle \gamma(\bar{G}_x, p(Y)) \rangle$ . The weight of the edge  $f$  is then the number of such witnesses  $Y \in \mathcal{N}_y$ .

Let  $z$  carry the composition operator  $\otimes[g | f_1, f_2]$  in  $T$ , i.e.  $\bar{G}_z = \bar{G}_x \otimes[g | f_1, f_2] \bar{G}_y$ . Notice that  $\Gamma(\bar{G}_z, \mathcal{N}_z)$  where  $\mathcal{N}_z = (\mathcal{N}_x \stackrel{p}{\Rightarrow} \mathcal{N}_y)$  is uniquely determined by  $\Gamma(\bar{G}_x, \mathcal{N}_x)$ ,  $\Gamma(\bar{G}_y, \mathcal{N}_y)$ , by the relabelings  $f_1, f_2$ , and by  $Sig(p)$ . We furthermore define on the same vertex set a special bipartite graph  $D_g^\perp$  (depending on the relabeling  $g$ ) with  $E(D_g^\perp) = \{\Psi\Psi' \in \mathcal{S} \times \mathcal{S} : \Psi \perp g(\Psi')\}$ . The purpose is to explicitly define which colour classes could be merged through  $\otimes[g | f_1, f_2]$  without creating edges inside any class.

From Lemma [4.2](#) we immediately conclude:

**(4.7)**  $\bar{G}_z \models \nu(\mathcal{N}_z)$  where  $\mathcal{N}_z = (\mathcal{N}_x \stackrel{p}{\Rightarrow} \mathcal{N}_y)$  for some  $\bar{G}_x \models \nu(\mathcal{N}_x)$ ,  $\bar{G}_y \models \nu(\mathcal{N}_y)$  and  $p \in \mathcal{J}(\mathcal{N}_y, \mathcal{N}_x)$  if, and only if,  $Sig(p)$  is a subgraph of  $D_g^\perp$ .

With Claim (4.7) at hand it is straightforward how to compute the set  $M_T(z)$  from the sets  $M_T(x)$  and  $M_T(y)$ . We loop through all members  $\Gamma_x \in M_T(x)$  and  $\Gamma_y \in M_T(y)$ , and all admissible signatures  $Sig$  (i.e. nonnegative integer weightings of the above bipartite graph  $D_g^\perp$  by (4.7)), test a simple consistency condition, and then possibly add the resulting  $\Gamma_z$  (easily computable) to  $M_T(z)$ . This consistency condition on  $Sig$  is that, for each its vertex  $\Psi$ , the sum of the weights of the edges of  $Sig$  incident with  $\Psi$  is at most the multiplicity of  $\Psi$  in  $\Gamma_x$  or  $\Gamma_y$ , respectively.

Finally, the chromatic number of  $G$  equals the least cardinality of a member of  $M_T(r)$  where  $r$  is the root of  $T$ . Such a leaves-to-root dynamic algorithm then runs in time  $O(m(G, t)^2 \cdot w(G, t) \cdot S(t)^2 \cdot t^3 \cdot |V(G)|)$ , where  $m(G, t)$  denotes the number of possible distinct  $\Gamma(\bar{G}, \mathcal{N})$ , and  $w(G, t)$  stands for the number of distinct weightings of the graph  $D_g^\perp$ . Each  $\Gamma_z$  is then determined from  $\Gamma_x, \Gamma_y$  and  $Sig$  in  $S(t)^2 t^3$  steps where  $S(t)$  is estimated in Lemma 4.4

For simplicity, we provide only short arguments giving rather weak (but sufficient) bounds on  $m, w$  here:  $m(G, t)$  can be bounded from above by  $|V(G)|^{S(t)}$ —consider that the multiplicity of any subspace in the multiset  $\Gamma(\bar{G}, \mathcal{N})$  is at most the number of nonempty colour classes. With analogous arguments we also get  $w(G, t) \leq |V(G)|^{S(t)^2}$ . These estimates then lead to a runtime bound of order  $|V(G)|^{h(t)}$  where  $h(t) \leq 2S(t) + S(t)^2 + o(t^2) + 1 = O(S(t)^2) = O(2^{t(t+1)/2})$ . ■

### 4.2 Chromatic Polynomial

The chromatic polynomial was first introduced by Birkhoff in the context of the Four Colour problem. Although the concept seems quite technical and obscure in nature, it has since become of independent interest. The *chromatic polynomial* of  $G$  is a polynomial  $P_G(x)$  such that for every nonnegative integer  $x$ ,  $P_G(x)$  equals the number of distinct proper colourings of  $G$  which use  $x$  colours. It is a trivial observation that, given the values of all  $P_G(x)$  for  $x = 1, 2, \dots, n = |V(G)|$ , finding  $P_G(x)$  simply becomes a matter of resolving  $n$  (independent) equations of  $n$  unknowns.

Computing the chromatic polynomial is generally  $\#P$ -complete. It has been noted by [14] that the algorithm of [19] extends towards computing the chromatic polynomial on graphs of bounded clique-width, and the same statement occurs with a proof in [1]. We improve those results with:

**Theorem 4.8.** *Assume that an input graph  $G$  is given in the form of a  $t$ -labeling parse tree  $T$ . Then the chromatic polynomial of  $G$  can be computed in time*

$$O\left(|V(G)|^{h(t)}\right) \text{ where } h(t) = O(2^{t(t+1)/2}).$$

**Proof.** As already explained, we need to modify the algorithm of Theorem 4.1 so that it will compute the number of distinct proper colourings of  $G$  having the prescribed number of colour classes. Fortunately, we do not need to process all possible colour partitions of  $G$ ; thanks to (4.6), we only have to remember the numbers of partitions  $\mathcal{N}$  determining the same value of  $\Gamma(\bar{G}_z, \mathcal{N})$  at a node  $z$ .

Let  $\vec{\alpha} = (\alpha_\Gamma : \Gamma \text{ is a multiset of subspaces of } \text{GF}(2)^t)$  be a vector of free variables. Although  $\vec{\alpha}$  is infinite, we shall actually use a finite part of it. Our algorithm shall compute the linear multivariate (symbolic) polynomial  $R(\vec{G}_z)[\vec{\alpha}] = \sum_\Gamma q_\Gamma \cdot \alpha_\Gamma$  where  $q_\Gamma$  stands for the number of distinct colour partitions  $\mathcal{N}$  of  $G_z$  such that  $\Gamma(\vec{G}_z, \mathcal{N}) = \Gamma$ .

The algorithm generally proceeds as that of Theorem 4.1. Specifically, at a node  $z$  of  $T$  with the sons  $x$  and  $y$ , we compute straightforwardly

$$R(\vec{G}_z)[\vec{\alpha}] = R(\vec{G}_x)[\vec{\alpha}] \cdot R(\vec{G}_y)[\vec{\alpha}]$$

and then apply all the necessary substitutions: For every pair  $\alpha_{\Gamma_1}, \alpha_{\Gamma_2} \in \vec{\alpha}$ , we replace the term  $\alpha_{\Gamma_1} \cdot \alpha_{\Gamma_2}$  with a sum, over all admissible signatures  $Sig$  (4.7), of the terms  $r_{Sig} \cdot \alpha_{\Gamma_{Sig}}$  where  $\Gamma_{Sig}$  is the multiset uniquely determined by  $\Gamma_1, \Gamma_2, Sig$ , and the composition relabelings at  $z$ . The number  $r_{Sig}$  is defined as follows.

Let  $\mathcal{N}_1, \mathcal{N}_2$  be such that  $\Gamma(\vec{G}_x, \mathcal{N}_1) = \Gamma_1$  and  $\Gamma(\vec{G}_y, \mathcal{N}_2) = \Gamma_2$ . Then  $r_{Sig}$  is the number of distinct partial injective mappings  $p \in \mathcal{J}(\mathcal{N}_2, \mathcal{N}_1)$  such that  $Sig(p) = Sig$ . One can check that this quantity does not depend on a particular choice of  $\mathcal{N}_1, \mathcal{N}_2$ , and that it can be straightforwardly computed from  $\Gamma_1, \Gamma_2$  and  $Sig$  along with computing  $\Gamma_{Sig}$ .

Finally, we extract the numbers of colourings of  $G$  from the above computed  $R(\vec{G})[\vec{\alpha}]$ . We note two points: First, a  $c$ -colouring does not have to use all  $c$  colours, and so we have to count with all colour partitions of at most  $c$  classes. Second, a traditional colouring distinguishes between the colours, while our colour classes do not. Hence we obtain the total number of distinct  $c$ -colourings of  $G$  from  $R(\vec{G})[\vec{\alpha}]$  if we substitute  $\alpha_\Gamma = c!/(c - |\Gamma|)!$  and  $\alpha_\Gamma = 0$  if  $|\Gamma| > c$ . ■

### 4.3 Hamiltonian Path

The last algorithm illustrating the strength our approach is based on a Hamiltonian path algorithm for graphs of bounded clique-width [7]. While the goal of the previous two subsections was to demonstrate how one can design new more efficient algorithms on labeling parse trees with tools of linear algebra, here we show how to translate an existing clique-width-based algorithm to a formally better setting within our scheme.

**Theorem 4.9.** *Assume that an input graph  $G$  is given in the form of a  $t$ -labeling parse tree  $T$ . Then one can determine whether  $G$  has a Hamiltonian path in time*

$$O(|V(G)|^{\ell(t)}) \text{ where } \ell(t) = O(4^t).$$

**Proof.** We say that a set of edges  $F \subseteq E(G)$  is *linear* if the subgraph  $G \upharpoonright F = (V(G), F)$  is a collection of disjoint paths. We also write  $G \models \lambda(F)$  if  $F$  is a Hamiltonian path in  $G$ . Having two linear subsets  $F_1 \subseteq E(G_1)$  and  $F_2 \subseteq E(G_2)$ , we define the equivalence relation  $(\vec{G}_1, F_1) \approx_{\lambda, t} (\vec{G}_2, F_2)$  if and only if, for all  $t$ -labeled graphs  $\vec{H}$  and all linear  $F \subseteq E(H)$ , it holds

$$(4.10) \quad \begin{aligned} &\exists F_3 \subseteq E(\vec{G}_1 \otimes \vec{H}) \setminus (E(G_1) \cup E(H)) : (\vec{G}_1 \otimes \vec{H}) \models \lambda(F_1 \cup F_3 \cup F) \\ \iff &\exists F_4 \subseteq E(\vec{G}_2 \otimes \vec{H}) \setminus (E(G_2) \cup E(H)) : (\vec{G}_2 \otimes \vec{H}) \models \lambda(F_2 \cup F_4 \cup F). \end{aligned}$$

Obviously,  $\approx_{\lambda,t}$  captures all information necessary to decide which linear subsets of  $G_1$  extend to Hamiltonian paths in a join graph.

Similarly to [7], for a linear subset  $F \subseteq E(G)$ , we define a multiset of labeling pairs  $\Pi(G, F) = \{(lab(x), lab(y)) : x, y \text{ are the ends of a path in } G \upharpoonright F\}$  (an isolated vertex is a path with the ends  $x = y$ ). Analogously to (4.6) we have:

(4.11) For any  $t$ -labeled graphs  $\bar{G}_1, \bar{G}_2$  and any linear  $F_1 \subseteq E(G_1)$  and  $F_2 \subseteq E(G_2)$ , it holds  $(\bar{G}_1, F_1) \approx_{\lambda,t} (\bar{G}_2, F_2)$  if  $\Pi(\bar{G}_1, F_1) = \Pi(\bar{G}_2, F_2)$ .

Therefore, our algorithm computes, in the leaves-to-root direction on  $T$ , the sets  $N_T(z) = \{\Pi(\bar{G}_z, F) : F \subseteq E(G) \text{ linear}\}$ . Since there are  $2^{2^t} = 4^t$  distinct labeling pairs in  $\text{GF}(2)^t$ , there are at most  $|V(G)|^{4^t}$  distinct multisets  $\Pi(\bar{G}_z, F)$  to be considered in each set  $N_T(z)$ . Obviously,  $G$  has a Hamiltonian path  $F$  if and only if  $N_T(r)$  contains a multiset  $\Pi(\bar{G}, F)$  of cardinality one. The rest proceeds in the same way as the previous algorithms. ■

*Remark 4.12.* One of the advantages of our new proof of Theorem 4.9 is that it immediately extends towards solving directed Hamiltonian path in digraphs of bounded *bi-rank-width* (a directed analogue of rank-width, cf. [18]).

## 5 Concluding Notes

The list of algorithms presented in this article is by no means exhaustive. Other XP algorithms designed for graphs of bounded clique-width (e.g., for the edge-dominating set [19]) can also be straightforwardly translated into our parse tree approach on rank-width, similarly to Theorem 4.9. One can expect that the time complexity of such algorithms will have a “one-level higher” exponent, as we see in Theorem 4.9. As already mentioned, the reason is that rank-width generally has an exponentially smaller value than clique-width (not that the new algorithms would be slower).

Still, the main advantage of designing algorithms on rank-decompositions of graphs is that we can efficiently compute an optimal rank-decomposition by Theorem 2.1. Even better, sometimes it is possible to use our approach to design algorithms which are actually much faster than the best known ones on clique-width; this is the case of computing the chromatic numbers and polynomials in Theorem 4.1. Determining which algorithms designed for clique-width can be radically improved in such a way remains an open question, one that we believe deserves further study.

## References

1. Averbouch, I., Godlin, B., Makowsky, J.A., Rotics, U.: Computing graph polynomials on graphs of bounded clique-width. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 191–204. Springer, Heidelberg (2006)

2. Bui-Xuan, B.-M., Telle, J.A., Vatshelle, M.: *H*-join and algorithms on graphs of bounded rankwidth. Technical Report 378, Dept. of Informatics, University of Bergen, Norway (2008), <http://www.ii.uib.no/publikasjoner/texrap/pdf/2008-378.pdf>
3. Corneil, D.G., Rotics, U.: On the relationship between cliquewidth and treewidth. *SIAM J. Comput.* 34(4), 825–847 (2005)
4. Courcelle, B., Kanté, M.M.: Graph Operations Characterizing Rank-Width and Balanced Graph Expressions. In: Brandstädt, A., Kratsch, D., Müller, H. (eds.) *WG 2007*. LNCS, vol. 4769, pp. 66–75. Springer, Heidelberg (2007)
5. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.* 33(2), 125–150 (2000)
6. Downey, R.G., Fellows, M.R.: *Parameterized complexity*. Monographs in Computer Science. Springer, New York (1999)
7. Espelage, W., Gurski, F., Wanke, E.: How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In: Brandstädt, A., Van Bang Le (eds.) *WG 2001*. LNCS, vol. 2204, pp. 117–128. Springer, Heidelberg (2001)
8. Fellows, M.R., Rosamond, F.A., Rotics, U., Szeider, S.: Clique-width minimization is NP-hard. In: *Proceedings of the 38th annual ACM Symposium on Theory of Computing*, pp. 354–362. ACM Press, New York (2006)
9. Fomin, F., Golovach, P., Lokshtanov, D., Saurabh, S.: Clique-width: On the Price of Generality. In: *Proceedings of the 19th Annual ACM–SIAM Symposium on Discrete Algorithms*, pp. 825–834. ACM Press, New York (2009)
10. Ganian, R.: Automata formalization for graphs of bounded rank-width. Master thesis. Faculty of Informatics of the Masaryk University, Brno, Czech republic (2008)
11. Ganian, R., Hliněný, P.: Automata approach to graphs of bounded rank-width. In: *Proceedings of IWOCOA 2008*, pp. 4–15 (2008)
12. Ganian, R., Hliněný, P.: On Parse Trees and Myhill–Nerode–type Tools for handling Graphs of Bounded Rank-width, p. 28 (2009) (manuscript)
13. Gerber, M.U., Kobler, D.: Algorithms for vertex-partitioning problems on graphs with fixed clique-width. *Theoret. Comput. Sci.* 299(1-3), 719–734 (2003)
14. Gimenez, O., Hliněný, P., Noy, M.: Computing the Tutte Polynomial on Graphs of Bounded Clique-Width. *SIAM J. Discrete Math.* 20, 932–946 (2006)
15. Goldman, J., Rota, G.-C.: The number of subspaces of a vector space. In: Tutte, W.T. (ed.) *Recent Progress in Combinatorics*, pp. 75–83. Academic Press, London (1969)
16. Hliněný, P.: Branch-width, parse trees, and monadic second-order logic for matroids. *J. Combin. Theory Ser. B* 96(3), 325–351 (2006)
17. Hliněný, P., Oum, S.: Finding Branch-decomposition and Rank-decomposition. *SIAM J. Comput.* 38, 1012–1032 (2008)
18. Kanté, M.: The rank-width of directed graphs. arXiv:0709.1433v3 (2008)
19. Kobler, D., Rotics, U.: Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Appl. Math.* 126(2-3), 197–221 (2003)
20. Oum, S., Seymour, P.: Approximating clique-width and branch-width. *J. Combin. Theory Ser. B* 96(4), 514–528 (2006)
21. Rao, M.: MSOL partitioning problems on graphs of bounded treewidth and clique-width. *Theoretical Computer Science* 377, 260–267 (2007)

# Minimax Trees in Linear Time with Applications

Paweł Gawrychowski<sup>1</sup> and Travis Gagie<sup>2,\*</sup>

<sup>1</sup> Institute of Computer Science  
University of Wrocław, Poland  
gawry1@gmail.com

<sup>2</sup> Research Group in Genome Informatics  
Bielefeld University, Germany  
travis.gagie@gmail.com

**Abstract.** A minimax tree is similar to a Huffman tree except that, instead of minimizing the weighted average of the leaves' depths, it minimizes the maximum of any leaf's weight plus its depth. Golumbic (1976) introduced minimax trees and gave a Huffman-like,  $\mathcal{O}(n \log n)$ -time algorithm for building them. Drmota and Szpankowski (2002) gave another  $\mathcal{O}(n \log n)$ -time algorithm, which takes linear time when the weights are already sorted by their fractional parts. In this paper we give the first linear-time algorithm for building minimax trees for unsorted real weights.

## 1 Introduction

In a minimax tree for a multiset  $W = \{w_1, \dots, w_n\}$  of weights, each leaf has a weight  $w_i$ , each internal node has weight equal to the maximum of its children's weights plus 1, and the weight of the root is as small as possible. In other words, if  $\ell_i$  is the depth of the leaf with weight  $w_i$ , then  $\max_i \{w_i + \ell_i\}$  is minimized. Golumbic [21] showed that if we modify Huffman's algorithm [24] to repeatedly replace the two nodes with smallest weights by a node whose weight is equal to their maximum plus 1, instead of their sum, then it builds a minimax tree instead of a Huffman tree. Like Huffman's algorithm, Golumbic's algorithm takes  $\mathcal{O}(n \log n)$  time and can build trees of any degree. Golumbic, Parker [34] and Hoover, Klawe and Pippenger [22] showed how to use Golumbic's algorithm to restrict circuits' fan-in and fan-out without greatly increasing their sizes or depths. While studying prefix codes with minimum maximum pointwise redundancy, Drmota and Szpankowski [10,11] independently introduced minimax trees as code-trees for generalized Shannon codes [36] and gave another  $\mathcal{O}(n \log n)$ -time algorithm for building them, which takes linear time when the weights are already sorted by their fractional parts. To see why the two problems are related, consider that, if  $P = p_1, \dots, p_n$  is a probability distribution and

---

\* This paper was written while the second author was at the University of Eastern Piedmont, Italy, supported by Italy-Israel FIRB Project "Pattern Discovery Algorithms in Discrete Structures, with Applications to Bioinformatics".

each  $w_i = \log_2 p_i$ , then a minimax tree for  $W$  is the code-tree for a prefix code with minimum maximum pointwise redundancy with respect to  $P$ . By analyzing their algorithm, Drmota and Szpankowski proved bounds on the redundancy of arithmetic coding, which Baer [3] recently improved by analyzing Golumbic's algorithm. In this paper we show how Drmota and Szpankowski's algorithm can be made to run in linear time on a word RAM when each index and weight fits in  $\mathcal{O}(1)$  words. Thus, we obtain the first linear-time algorithm for building minimax trees for unsorted real weights.

Between Golumbic's article and Drmota and Szpankowski's, there seems to have been little research on building minimax trees. Several important papers were published, however, on the related problem of building alphabetic minimax trees, in which the leaves' weights must be in a given order from left to right. Hu, Kleitman and Tamaki [23] gave the first  $\mathcal{O}(n \log n)$ -time algorithm for building alphabetic minimax trees for real weights. Kirkpatrick and Klawe [27] and Coppersmith, Klawe and Pippenger [6] gave an algorithm (or, more precisely, two algorithms that are equivalent when the trees are to be binary) that builds alphabetic minimax trees for integer weights in linear time, and showed how to use it to restrict circuits' fan-in and fan-out without greatly increasing their sizes or depths and without changing the numbers of edge crossings (and, thus, preserving planarity). Kirkpatrick and Klawe also showed how to combine their algorithm with binary search in order to build alphabet minimax trees for real weights in  $\mathcal{O}(n \log n)$  time. We note that, if their algorithm for integer weights is viewed as an alphabetic analogue of the Kraft Inequality [31] — as it was by Yeung [38] and Nakatsu [33], who independently rediscovered it — then their algorithm for real weights is an alphabetic analogue of Drmota and Szpankowski's. Kirkpatrick and Przytycka [28] gave an  $\mathcal{O}(\log n)$ -time,  $\mathcal{O}(n/\log n)$ -processor algorithm for integer weights in the CREW PRAM model. Finally, Evans and Kirkpatrick [12] showed how a generalization of Kirkpatrick and Klawe's algorithm can be used to restructure binary search trees.

We became interested in minimax trees while studying adaptive prefix coding. In a previous paper [16] (see also [17,25]) we noted that minimax trees built with Golumbic's algorithm have the same Sibling Property [13,18] as Huffman trees, and turned the Faller-Gallager-Knuth algorithm [30] for adaptive Huffman coding into an algorithm for adaptive Shannon coding. Intriguingly, although static Huffman coding is optimal and static Shannon coding is not, adaptive Shannon coding has a better worst-case bound than adaptive Huffman coding does. In another previous paper [15] we used a data structure due to Kirkpatrick and Przytycka and a technique for generalized selection due to Klawe and Mumey [29], to make Kirkpatrick and Klawe's algorithm for real weights run in  $\mathcal{O}(n \min(\log n, d \log \log n))$  time, where  $d$  is the number of distinct values  $[w_i]$ . In that paper we conjectured that a similar modification could make Drmota and Szpankowski's algorithm run in linear time on unsorted real weights, and in this paper we prove that conjecture.



In Section 2 we consider the preliminary problem of building minimax trees for unsorted integer weights. Notice that, as such weights have no fractional parts, Drmota and Szpankowski's algorithm takes linear time for this problem. However, there are two difficulties when using their algorithm: first, because they considered the weights to be logarithms, they did not address some questions of precision that arise when the weights are large; second, because they were mostly interested in analysis, they were satisfied with computing the depths of minimax trees' leaves in linear time, rather than building the trees themselves. We give two new linear-time algorithms for unsorted integer weights that can handle large weights — i.e., polynomial in  $n$ , so that each fits in a constant number of machine words — and that actually build the minimax trees. In Section 3 we present our main result, a linear-time algorithm for building minimax trees for unsorted real weights. Our algorithm is based on Drmota and Szpankowski's but, whereas theirs uses sorting and binary search, ours uses generalized selection, as well as a new data structure to test the Kraft Inequality. In Section 4 we discuss how our algorithms from Sections 2 and 3 can be applied to problems in, e.g., data compression, group testing and circuit design. In Section 5 we briefly discuss two possible directions for future work. Our results generalize to higher degrees and larger code alphabets but, for the sake of simplicity, in the current version of this paper we consider only binary trees and alphabets; by log we always mean  $\log_2$ .

## 2 Minimax Trees for Integer Weights

In this section we give two  $\mathcal{O}(n)$ -time algorithms for building a minimax tree for a multiset of integer weights, both based on the following lemma (which we note applies to any weights, not only integers) and corollary. We write  $M(W)$  to denote the weight of the root of a minimax tree for  $W$ .

**Lemma 1.** *If  $W = \{w_1, \dots, w_n\}$  is a multiset of weights and*

$$W' = \left\{ \max(w_1, \max_i\{w_i\} - n + 1), \dots, \max(w_n, \max_i\{w_i\} - n + 1) \right\},$$

*then  $M(W') = M(W)$ . Moreover, any minimax tree for  $W'$  becomes a minimax tree for  $W$  when we replace the leaves' weights equal to  $\max_i\{w_i\} - n + 1$  by the weights in  $W$  less than or equal to  $\max_i\{w_i\} - n + 1$ , in any order.*

*Proof.* Consider a minimax tree  $T$  for  $W$ . Without loss of generality, we can assume  $T$  is strictly binary — i.e., that every internal node has exactly two children — and, therefore, that it has height at most  $n - 1$ . If  $n = 1$ , then  $W = w_1 = \max_i\{w_i\} - n + 1$ . Otherwise, all the leaves have depth at least 1, so  $M(W) \geq \max_i\{w_i\} + 1$ . Consider any leaf (if one exists) with weight less than  $\max_i\{w_i\} - n + 1$  and depth  $\ell$ . Since  $\max_i\{w_i\} - n + 1 + \ell \leq \max_i\{w_i\} < M(W)$ , increasing that leaf's weight to  $\max_i\{w_i\} - n + 1$  and updating its ancestors' weights, does not change the weight  $M(W)$  of the root. It follows that  $M(W') = M(W)$ .



Now consider a minimax tree  $T'$  for  $W'$ . If we replace the leaves' weights equal to  $\max_i\{w_i\} - n + 1$  by the weights in  $W$  less than or equal to  $\max_i\{w_i\} - n + 1$  and update all the nodes' weights, then the weight  $M(W')$  of the root cannot increase nor, by definition, decrease to less than  $M(W)$ . Since  $M(W') = M(W)$ , it follows that the re-weighted tree is a minimax tree for  $W$ .  $\square$

**Corollary 1.** *When all the weights in  $W$  are integers, we can sort  $W'$  in  $\mathcal{O}(n)$  time.*

*Proof.* When all the weights in  $W$  at least  $\max_i\{w_i\} - n + 1$  are integers, all the weights in  $W'$  are integers in the interval  $[\max_i\{w_i\} - n + 1, \max_i\{w_i\}]$ . Since this interval has length  $n - 1$ , we can sort  $W'$  in  $\mathcal{O}(n)$  time using either direct addressing, which takes  $\mathcal{O}(n)$  extra space, or radix sort, which takes no extra space [14].  $\square$

For our first algorithm, we build and sort  $W'$ ; build a minimax tree for  $W'$  using an implementation of Golumbic's algorithm that takes  $\mathcal{O}(n)$  time when the weights are already sorted; and replace the leaves' weights equal to  $\max_i\{w_i\} - n + 1$  by the weights in  $W$  less than or equal to  $\max_i\{w_i\} - n + 1$ . We note that Van Leeuwen [37] showed how to implement Huffman's algorithm to take  $\mathcal{O}(n)$  time when the weights are already sorted. We could implement Golumbic's algorithm analogously, but we think the implementation below is simpler.

**Lemma 2.** *Golumbic's algorithm can be implemented to take  $\mathcal{O}(n)$  time when the weights are already sorted.*

*Proof.* We start with the weights stored in a linked list in nondecreasing order, and set a pointer to the head of the list. We then repeat the following procedure until there is only one node left in the list, which is the root of a minimax tree for the given weights: we move the pointer along the list to the last weight less than or equal to the maximum of the first two weights plus 1; remove the first two nodes from the list; make those nodes the children of a new node with weight equal to the maximum of their weights plus one; and insert the new node immediately to the right of the pointer. Notice we remove two nodes for each one we insert, so the total number of nodes is  $2n - 1$ . Therefore, since the pointer passes over each node once, this implementation takes  $\mathcal{O}(n)$  time.  $\square$

Building and sorting  $W'$  takes  $\mathcal{O}(n)$  time, by Corollary 1; building a minimax tree for  $W'$  takes  $\mathcal{O}(n)$  time, by Lemma 2; replacing the leaves' weights equal to  $\max_i\{w_i\} - n + 1$  by the weights in  $W$  less than or equal to  $\max_i\{w_i\} - n + 1$  takes  $\mathcal{O}(n)$  time, because it can be done in any order. By Lemma 1, the resulting tree is a minimax tree for  $W$ .

**Theorem 1.** *Given a multiset  $W$  of  $n$  integer weights, we can build a minimax tree for  $W$  in  $\mathcal{O}(n)$  time.*

Our second algorithm differs in its second step: instead of using Golumbic's algorithm to build a minimax tree for  $W'$ , we use Kirkpatrick and Klawe's

$\mathcal{O}(n)$ -time algorithm for integer weights to build an alphabetic minimax tree for the sequence  $V$  consisting of the weights in  $W'$  in non-increasing order. The algorithm's correctness follows from the Kraft Inequality.

**Theorem 2 (Kraft, 1949).** *If there exists a binary tree whose leaves have depths  $\ell_1, \dots, \ell_n$ , then  $\sum_i 1/2^{\ell_i} \leq 1$ . Conversely, if  $\sum_i 1/2^{\ell_i} \leq 1$  and  $\ell_1 \leq \dots \leq \ell_n$ , then there exists an ordered binary tree whose leaves, from left to right, have depths  $\ell_1, \dots, \ell_n$ .*

By the latter part of Theorem 2 and a standard exchange argument — i.e., if a minimax tree contains two leaves such that the deeper one has a higher weight than the shallower one, then we can swap their weights — there exists a minimax tree for  $W'$  in which the leaves' weights are non-increasing from left to right. Therefore, by definition, any alphabetic minimax tree for  $V$  is a minimax tree for  $W'$ .

### 3 Minimax Trees for Real Weights

In this section we give the first  $\mathcal{O}(n)$ -time algorithm for building minimax trees for unsorted real weights. As we noted in the introduction, our algorithm is based on Drmota and Szpankowski's algorithm but avoids sorting, which is the step that determines their algorithm's  $\mathcal{O}(n \log n)$  complexity. In addition to yielding an optimal algorithm for an interesting problem with applications in, e.g., data compression, group testing and circuit design — described in Section 4 — we believe the techniques we use in this section may be of independent interest.

To build a prefix code with minimum maximum pointwise redundancy with respect to a given probability distribution  $P = p_1, \dots, p_n$ , Drmota and Szpankowski start with a Shannon code for  $P$ , in which the codeword for the  $i$ th character has length  $\lceil \log(1/p_i) \rceil$ , for each  $i$ ; they sort the logarithms by their fractional parts, i.e.,  $\log(1/p_1) - \lfloor \log(1/p_1) \rfloor, \dots, \log(1/p_n) - \lfloor \log(1/p_n) \rfloor$ ; and they find the largest value  $x$  such that  $\lceil \log(1/p_1) - x \rceil, \dots, \lceil \log(1/p_n) - x \rceil$  obey the Kraft Inequality. A binary tree with leaves at these depths is the code-tree for a prefix code with minimum maximum pointwise redundancy with respect to  $P$ , and a minimax tree for  $\{\log p_1, \dots, \log p_n\}$ .

We can use Drmota and Szpankowski's algorithm to build a minimax tree given any multiset of weights because, for any value  $c$ , if  $W = \{w_1, \dots, w_n\}$  and  $W' = \{w_1 + c, \dots, w_n + c\}$  then, by definition,  $M(W') = M(W) + c$  and any minimax tree for  $W'$  becomes a minimax tree for  $W$  when we subtract  $c$  from each leaf's weight. In particular, if  $c = -\log(\sum_i 2^{w_i})$  then  $\sum_i 2^{w_i+c} = 2^c \sum_i 2^{w_i} = 1$ ; therefore,  $W' = \{\log p_1, \dots, \log p_n\}$  for some probability distribution  $P = p_1, \dots, p_n$  and we can use Drmota and Szpankowski's algorithm directly to build minimax trees for  $W'$  and, thus, for  $W$ . Without loss of generality, we henceforth assume the given multiset  $W$  of weights is equal to  $\{\log p_1, \dots, \log p_n\}$  for some probability distribution  $P$  (so, in particular, each  $w_i \leq 0$ ). We can restate the theorem Drmota and Szpankowski proved to establish the correctness of their algorithm — and which also establishes the correctness of our own — in terms of minimax trees instead of prefix codes, as follows:

**Theorem 3 (Drmota and Szpankowski, 2002).** *If  $W = \{w_1, \dots, w_n\}$  is a multiset of weights,  $X = \{x_1, \dots, x_n\} = \{|w_1| - \lfloor |w_1| \rfloor, \dots, |w_n| - \lfloor |w_n| \rfloor\}$  and  $x_i$  is the largest element in  $X \cup \{0\}$  such that*

$$\sum_{x_j \leq x_i} 1/2^{\lfloor |w_j| \rfloor} + \sum_{x_j > x_i} 1/2^{\lceil |w_j| \rceil} \leq 1,$$

*then any minimax tree for  $\{-\lfloor |w_j| \rfloor : x_j \leq x_i\} \cup \{-\lceil |w_j| \rceil : x_j > x_i\}$  becomes a minimax tree for  $W$  when we replace each leaf's weight  $-\lfloor |w_j| \rfloor$  or  $-\lceil |w_j| \rceil$  by  $w_j$ .*

If  $x_1 \leq \dots \leq x_n$  and  $x_i > 0$  then, by Theorem 3,  $i$  is the largest index such that  $\{\lfloor |w_j| \rfloor : x_j \leq x_i\} \cup \{\lceil |w_j| \rceil : x_j > x_i\}$  satisfies the Kraft Inequality. To build a minimax tree for  $W$  with Drmota and Szpankowski's algorithm, we compute and sort  $X$ ; use binary search to find  $i$ , in each round testing whether the Kraft Inequality holds; build a minimax tree for  $\{-\lfloor |w_1| \rfloor, \dots, -\lfloor |w_i| \rfloor, -\lceil |w_{i+1}| \rceil, \dots, -\lceil |w_n| \rceil\}$ ; and replace each leaf's weight  $-\lfloor |w_j| \rfloor$  or  $-\lceil |w_j| \rceil$  by  $w_j$ . Our version differs in three ways: we use generalized selection instead of sorting and binary search; we use a new data structure to test the Kraft Inequality; and we use either of our algorithms from Section 2 to build the minimax tree for  $\{-\lfloor |w_1| \rfloor, \dots, -\lfloor |w_i| \rfloor, -\lceil |w_{i+1}| \rceil, \dots, -\lceil |w_n| \rceil\}$ . In the remainder of this section we first show how to use generalized selection to find  $i$  in  $\mathcal{O}(n)$  time, excluding the time needed to test the Kraft Inequality; we then show how to perform all the necessary tests in a total of  $\mathcal{O}(n)$  time using our new data structure. Since each of our algorithms from Section 2 takes  $\mathcal{O}(n)$  time, it follows that we can build a minimax tree for  $W$  in  $\mathcal{O}(n)$  time.

To find  $x_i$  in  $\mathcal{O}(n)$  time with general selection, we start with the multiset  $X_1 = X \cup \{0\}$  and repeat the following procedure until we reach the empty set: in the  $r$ th round, we use the linear-time selection algorithm due to Blum *et al.* [4] to find the current multiset  $X_r$ 's median  $x_m$ , then test whether

$$\sum_{x_j \leq x_m} 1/2^{\lfloor |w_j| \rfloor} + \sum_{x_j > x_m} 1/2^{\lceil |w_j| \rceil} \leq 1;$$

if so, we remove those elements of  $X_r$  that are less than or equal to  $x_m$  and recurse on the resulting multiset; if not, we remove those elements of  $X_r$  that are greater than or equal to  $x_m$  and recurse. The element  $x_i$  is the largest median we consider for which the test is positive. Since the size of the multisets decreases by a factor of at least 2 in each round, we use  $\mathcal{O}(\log n)$  rounds and we find all the medians in a total of  $\mathcal{O}(n)$  time.

By the same arguments we used to prove Lemma 1, we can assume, without loss of generality, that  $\lceil |w_j| \rceil \leq n - 1$  for each  $j$ . To test the Kraft Inequality, we use a data structure consisting of two  $n$ -bit binary fractions,  $S_1$  and  $S_2$ , each broken into  $(\log n)$ -bit blocks and initially set to 0. For  $1 \leq k \leq n - 1$ , adding  $1/2^k$  to either fraction takes  $\mathcal{O}(1)$  amortized time, for the same reason that incrementing a binary counter takes  $\mathcal{O}(1)$  amortized time (see, e.g., [7, Section 17.3]). Nondestructively testing whether  $S_1 + S_2 \leq 1$  takes  $\mathcal{O}(n/\log n)$

time, because adding each corresponding pair of blocks takes  $\mathcal{O}(1)$  time and, by induction, the number carried from each pair to the next is at most 1; resetting either fraction to 0 takes  $\mathcal{O}(1)$  time for each block, i.e.,  $\mathcal{O}(n/\log n)$  time in total.

Before starting to search for  $x_i$ , we set  $S_1 = \sum_j 1/2^{\lceil |w_j| \rceil}$  in  $\mathcal{O}(n)$  time. Throughout our generalized selection, we maintain the invariant that, at the beginning of the  $r$ th round,

$$S_1 = \sum_j 1/2^{\lceil |w_j| \rceil} + \sum_{0 < x_j < \min(X_r)} 1/2^{\lceil |w_j| \rceil}$$

and  $S_2 = 0$ . In the  $r$ th round, we set

$$S_2 = \sum_{\min(X_r) \leq x_j \leq x_m} 1/2^{\lceil |w_j| \rceil}$$

in  $\mathcal{O}(|X_r|)$  time. Since

$$\begin{aligned} S_1 + S_2 &= \sum_j 1/2^{\lceil |w_j| \rceil} + \sum_{0 < x_j < \min(X_r)} 1/2^{\lceil |w_j| \rceil} + \sum_{\min(X_r) \leq x_j \leq x_m} 1/2^{\lceil |w_j| \rceil} \\ &= \sum_{x_j \leq x_m} 1/2^{\lceil |w_j| \rceil} + \sum_{x_j > x_m} 1/2^{\lceil |w_j| \rceil}, \end{aligned}$$

we can test the Kraft Inequality in  $\mathcal{O}(n/\log n)$  time by checking whether  $S_1 + S_2 \leq 1$ . If the test is positive, then we add  $S_2$  to  $S_1$  in  $\mathcal{O}(n/\log n)$  time; if the test is negative, then we do not change  $S_1$ . In either case, straightforward calculation shows that, afterwards,

$$S_1 = \sum_j 1/2^{\lceil |w_j| \rceil} + \sum_{0 < x_j < \min(X_{r+1})} 1/2^{\lceil |w_j| \rceil}$$

so the first part of our invariant is maintained. Finally, we reset  $S_2 = 0$  in  $\mathcal{O}(n/\log n)$  time, so the second part of our invariant is maintained. Since  $|X_r| = \mathcal{O}(n/2^r)$ , the  $r$ th round takes a total of  $\mathcal{O}(n/2^r + n/\log n)$  time. Since  $\sum_{r \geq 1} n/2^r = n$  and we use  $\mathcal{O}(\log n)$  rounds, it follows that our whole generalized selection takes  $\mathcal{O}(n)$  time. This completes the proof of our main result:

**Theorem 4.** *Given a multiset  $W$  of  $n$  real weights, we can build a minimax tree for  $W$  in  $\mathcal{O}(n)$  time.*

## 4 Applications

Our results from Sections 2 and 3 allow us to solve the following problems in linear time:

- I. Build a prefix code with minimum maximum pointwise redundancy with respect to a given distribution;

- II. Given a good estimate of the distribution over an alphabet, build a good prefix code;
- III. Given a good estimate of the distribution over a set, design a good group test to find the unique target;
- IV. Build a minimax tree for a given multiset of real weights;
- V. Build a Shannon code for a given distribution;
- VI. Build a tree whose leaves have at most given depths;
- VII. Restrict a given circuit to have bounded fan-in or fan-out;
- VIII. Build a minimax tree for a given multiset of integer weights.

The authors cited in the introduction have already shown, however, that Problem [I](#) takes  $\mathcal{O}(n)$  more time than [IV](#) [\[10,11,13\]](#), [V](#) than [VI](#) [\[10,11,16\]](#), and [VI](#) and [VII](#) than [VIII](#) [\[21,34,42\]](#). Therefore, in the current version of this paper, we discuss only Problems [II](#), [III](#), [IV](#) and [VIII](#). We showed in Sections [2](#) and [3](#), respectively, that Problem [VIII](#) and [IV](#) take  $\mathcal{O}(n)$  time. In the remainder of this section we define what we mean by “good” in Problems [II](#) and [III](#), and show they take  $\mathcal{O}(n)$  more time than [IV](#). (Problems [II](#) and [III](#) are, in fact, equivalent to each other and to [I](#), and analogous to a problem we considered in our paper [\[15\]](#) on building alphabetic minimax trees.) It follows that all the problems listed above take  $\mathcal{O}(n)$  time.

Suppose we want to build a good prefix code with which to compress a file, but are given only a sample of its characters. Let  $P = p_1, \dots, p_n$  be the normalized distribution of characters in the file, let  $Q = q_1, \dots, q_n$  be the normalized distribution of characters in the sample and suppose our codewords are  $C = c_1, \dots, c_n$ . An ideal code for  $Q$  assigns the  $i$ th character a codeword of length  $\log(1/q_i)$  (which may not be an integer), and the average codeword’s length using such a code is  $H(P) + D(P||Q)$ , where  $H(P) = \sum_i p_i \log(1/p_i)$  is the entropy of  $P$  and  $D(P||Q) = \sum_i p_i \log(p_i/q_i)$  is the relative entropy between  $P$  and  $Q$ . The entropy measures our expected surprise at a character drawn uniformly at random from the file, given  $P$ ; the relative entropy (also known as the informational divergence or Kullback-Leibler pseudo-distance) measures the increase in our expected surprise when we estimate  $P$  by  $Q$ , and is often used to quantify how well  $Q$  approximates  $P$  (see, e.g., [\[8\]](#)).

Consider the best worst-case bound we can achieve, given only  $Q$ , on how much the average codeword’s length exceeds  $H(P) + D(P||Q)$ . A result by Katona and Nemetz [\[26\]](#) implies we do not generally achieve a constant bound on the difference when  $C$  is a Huffman code for  $Q$ . (Given  $P$ , of course, the best bound we could achieve on how much the average codeword’s length exceeds  $H(P)$ , would be the redundancy of a Huffman code for  $P$ .) For example, if  $q_1, \dots, q_n$  are proportional to  $F_n, \dots, F_1$ , where  $F_i$  denotes the  $i$ th Fibonacci number (i.e.,  $F_1 = F_2 = 1$  and  $F_i = F_{i-1} + F_{i-2}$  for  $i \geq 3$ ), then the codewords’ lengths are  $1, \dots, n-2, n-1, n-1$  in any Huffman code for  $Q$ . If  $p_n$  is sufficiently close to 1, then

$$H(P) + D(P||Q) \approx \log(1/q_n) = \log \sum_{i=1}^n F_i = n \log \phi + \mathcal{O}(1)$$

but the average codeword’s length  $\sum_i p_i |c_i| \approx n - 1$ , so for large  $n$  the difference is about  $(1/\log \phi - 1)n \approx 0.44n$ , where  $\phi \approx 1.62$  is the golden ratio.

As long as  $q_i > 0$  whenever  $p_i > 0$ , the average codeword’s length

$$\begin{aligned} \sum_i p_i |c_i| &= \sum_i p_i \left( \log(1/p_i) + \log(p_i/q_i) + \log q_i + |c_i| \right) \\ &= H(P) + D(P\|Q) + \sum_i p_i (\log q_i + |c_i|) \end{aligned}$$

(if  $q_i = 0$  but  $p_i > 0$  for some  $i$ , then  $D(P\|Q)$  is infinite). Notice each  $|c_i|$  is the length of a branch in the code-tree for  $C$ . Therefore, the best bound we can achieve is

$$\begin{aligned} \min_C \max_P \left\{ \sum_i p_i (\log q_i + |c_i|) \right\} \\ &= \min_C \max_i \{ \log q_i + |c_i| \} \\ &= M(\log q_1, \dots, \log q_n), \end{aligned}$$

which is less than 1 by inspection of Drmota and Szpankowski’s algorithm (see also [8, Theorem 5.4.3] and [11,35,3]). Moreover, we achieve this bound when the code-tree for  $C$  has the same shape as a minimax tree for  $\{\log q_1, \dots, \log q_n\}$ . In other words, Problem III takes  $\mathcal{O}(n)$  more time than IV.

Now suppose we want to design a good group test (see, e.g., [12]) to find the unique target in a set, given only an estimate  $Q$  — presumably gained from past experience or experimentation — of the probability distribution  $P$  according to which the target is chosen. A group test allows us to choose, repeatedly, a subset of the elements and check whether the target is among them. We can represent a group test as a decision tree in which each leaf is labelled with an element and each internal node is labelled with the concatenation of its children’s labels. Because such a decision tree can be viewed as the code-tree for a prefix code, and vice versa, the expected number of checks we make exceeds  $H(P) + D(P\|Q)$  by as little as possible when the decision tree for our group test has the same shape as a minimax tree for  $\{\log q_1, \dots, \log q_n\}$ . In other words, Problem III is equivalent to III and, therefore, also takes  $\mathcal{O}(n)$  more time than IV.

## 5 Future Work

We are currently studying whether either Drmota and Szpankowski’s solution to Problem I or our solution to III can give us an intuitive explanation of why adaptive Shannon coding has a better worst-case bound than does adaptive Huffman coding. On the one hand, worst-case bounds (especially for online algorithms; see, e.g., [5]) are often proven by considering a game between the algorithm and an omniscient adversary, and minimizing the maximum pointwise redundancy at each step seems somehow related (more than just by name) to the minimax strategy for the algorithm. On the other hand, adaptive prefix coding can be

viewed as a procedure in which we repeatedly build a prefix code based on a sample — i.e., the characters already encoded.

We are still studying alphabetic minimax trees and have started studying minimax trees with unequal edge costs. We believe this latter variant will prove particularly interesting, for three reasons: first, it is not known how to build efficiently a Huffman tree with unequal edge costs (see, e.g., [19,20]); second, the earliest efficient approximation algorithm (of which we are aware) for building a Huffman tree with unequal edge costs, was Krause's generalization [32] of Shannon coding; third, there is an analogue of the Kraft Inequality for binary trees with unequal edge costs [9].

## Acknowledgments

Many thanks to Michael Baer, Giovanni Manzini and the referees for their advice.

## References

1. Ahlswede, R., Wegener, I.: Search Problems. Wiley, Chichester (1987)
2. Aigner, M.: Combinatorial Search. Wiley, Chichester (1988)
3. Baer, M.B.: Tight bounds on minimum maximum pointwise redundancy. In: Proceedings of the International Symposium on Information Theory, pp. 1944–1948 (2008)
4. Blum, M., Floyd, R.W., Pratt, V.R., Rivest, R.L., Tarjan, R.E.: Time bounds for selection. *Journal of Computer and System Sciences* 7(4), 448–461 (1973)
5. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
6. Coppersmith, D., Klawe, M.M., Pippenger, N.: Alphabetic minimax trees of degree at most  $t$ . *SIAM Journal on Computing* 15(1), 189–192 (1986)
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press and McGraw-Hill (2001)
8. Cover, T.M., Thomas, J.A.: Elements of Information Theory, 2nd edn. Wiley, Chichester (2006)
9. De Prisco, R., Persiano, G.: Characteristic inequalities for binary trees. *Information Processing Letters* 53(4), 201–207 (1995)
10. Drmota, M., Szpankowski, W.: Generalized Shannon code minimizes the maximal redundancy. In: Proceedings of the 5th Latin American Symposium on Theoretical Informatics, pp. 306–318 (2002)
11. Drmota, M., Szpankowski, W.: Precise minimax redundancy and regret. *IEEE Transactions on Information Theory* 50(11), 2686–2707 (2004)
12. Evans, W.S., Kirkpatrick, D.G.: Restructuring ordered binary trees. *Journal of Algorithms* 50(2), 168–193 (2004)
13. Faller, N.: An adaptive system for data compression. In: Record of the 7th Asilomar Conference on Circuits, Systems and Computers, pp. 593–597 (1973)
14. Franceschini, G., Muthukrishnan, S., Pătrașcu, M.: Radix sorting with no extra space. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 194–205. Springer, Heidelberg (2007)
15. Gagie, T.: A new algorithm for building alphabetic minimax trees. *Fundamenta Informaticae* (to appear)

16. Gagie, T.: Dynamic Shannon coding. In: Albers, S., Radzik, T. (eds.) *ESA 2004*. LNCS, vol. 3221, pp. 359–370. Springer, Heidelberg (2004)
17. Gagie, T.: Dynamic Shannon coding. *Information Processing Letters* 102(2-3), 113–117 (2007)
18. Gallager, R.G.: Variations on a theme by Huffman. *IEEE Transactions on Information Theory* 24(6), 668–674 (1978)
19. Golin, M.J., Kenyon, C., Young, N.E.: Huffman coding with unequal letter costs. In: *Proceedings of the 34th Symposium on Theory of Computing*, pp. 785–791 (2002)
20. Golin, M.J., Li, J.: More efficient algorithms and analyses for unequal letter cost prefix-free coding. *IEEE Transactions on Information Theory* 52(8), 3412–3424 (2008)
21. Golubic, M.C.: Combinatorial merging. *IEEE Transactions on Computers* 25(11), 1164–1167 (1976)
22. Hoover, H.J., Klawe, M.M., Pippenger, N.: Bounding fan-out in logical networks. *Journal of the ACM* 31(1), 13–18 (1984)
23. Hu, T.C., Kleitman, D.J., Tamaki, J.: Binary trees optimum under various criteria. *SIAM Journal on Applied Mathematics* 37(2), 246–256 (1979)
24. Huffman, D.A.: A method for the construction of minimum-redundancy codes. *Proceedings of the IRE* 40, 1089–1101 (1952)
25. Karpinski, M., Nekrich, Y.: A fast algorithm for adaptive prefix coding. *Algorithmica* 55(1), 29–41 (2009)
26. Katona, G.O.H., Nemetz, T.O.H.: Huffman codes and self-information. *IEEE Transactions on Information Theory* 22(3), 337–340 (1976)
27. Kirkpatrick, D.G., Klawe, M.M.: Alphabetic minimax trees. *SIAM Journal on Computing* 14(3), 514–526 (1985)
28. Kirkpatrick, D.G., Przytycka, T.M.: An optimal parallel minimax tree algorithm. In: *Proceedings of the 2nd Symposium on Parallel and Distributed Processing*, pp. 293–300 (1990)
29. Klawe, M.M., Mumey, B.: Upper and lower bounds on constructing alphabetic binary trees. *SIAM Journal on Discrete Mathematics* 8(4), 638–651 (1995)
30. Knuth, D.E.: Dynamic Huffman coding. *Journal of Algorithms* 6(2), 163–180 (1985)
31. Kraft, L.G.: A device for quantizing, grouping, and coding amplitude-modulated pulses. MSc thesis, Massachusetts Institute of Technology (1949)
32. Krause, R.M.: Channels which transmit letters of unequal duration. *Information and Control* 5(1), 13–24 (1962)
33. Nakatsu, N.: Bounds on the redundancy of binary alphabetical codes. *IEEE Transactions on Information Theory* 37(4), 1225–1229 (1991)
34. Parker Jr., D.S.: Combinatorial merging and Huffman’s algorithm. *IEEE Transactions on Computers* 28(5), 365–367 (1979)
35. Rezaei, F., Charalambous, C.D.: Robust coding for uncertain sources: a minimax approach. In: *Proceedings of the International Symposium on Information Theory*, pp. 1539–1543 (2005)
36. Shannon, C.E.: A mathematical theory of communication. *Bell System Technical Journal* 27, 379–423, 623–645 (1948)
37. van Leeuwen, J.: On the construction of Huffman trees. In: *Proceedings of the 3rd International Colloquium on Automata, Languages and Programming*, pp. 382–410 (1976)
38. Yeung, R.W.: Alphabetic codes revisited. *IEEE Transactions on Information Theory* 37(3), 564–572 (1991)



# Planar Biconnectivity Augmentation with Fixed Embedding

Carsten Gutwenger, Petra Mutzel, and Bernd Zey

Department of Computer Science, Technische Universität Dortmund, Germany  
{carsten.gutwenger,petra.mutzel,bernd.zey}@tu-dortmund.de

**Abstract.** A *combinatorial embedding*  $\Pi$  of a planar graph  $G = (V, E)$  is defined by the cyclic order of incident edges around each vertex in a planar drawing of  $G$ . The *planar biconnectivity augmentation problem with fixed embedding (PBA-Fix)* asks for a minimum edge set  $E' \subseteq V \times V$  that augments  $\Pi$  to a combinatorial embedding  $\Pi'$  of  $G + E'$  such that  $G + E'$  is biconnected and  $\Pi$  is preserved, i.e.,  $\Pi'$  restricted to  $G$  yields again  $\Pi$ .

In this paper, we show that PBA-Fix is NP-hard in general, i.e., for not necessarily connected graphs, by giving a reduction from 3-PARTITION. For connected graphs, we present an  $\mathcal{O}(|V|(1 + \alpha(|V|)))$  time algorithm solving PBA-Fix optimally. Moreover, we show that—considering each face of  $\Pi$  separately—this algorithm meets the lower bound for the general biconnectivity augmentation problem proven by Eswaran and Tarjan [4].

## 1 Introduction

The problem of augmenting a graph to reach a certain connectivity is a fundamental graph theoretic problem. For a fixed  $k \in \mathbb{N}$ , the *general augmentation problem* asks for a minimum number of edges to add to a graph such that the graph becomes  $k$ -connected. In this paper, we study the case  $k = 2$ , i.e., we want to add a minimum number of edges to obtain a biconnected graph. Though this problem can be solved in linear time [6], it becomes  $\mathcal{NP}$ -hard if we consider planar graphs and demand that the augmented graph is still planar; see [7]. We call this problem the *planar biconnectivity augmentation problem (PBA)*. It has applications in graph drawing, where certain drawing algorithms require a planar and biconnected graph, e.g., [4]; if we want to apply such an algorithm to an arbitrary planar graph, we first need to augment the graph by adding edges. It is desirable to add only few edges, since we do not want to change the structure of the graph too much.

Several approximation algorithms for PBA have been proposed. Kant and Bodlaender [7] presented a rather simple 2-approximation algorithm for PBA running in  $\mathcal{O}(n \log n)$  time for a graph with  $n$  vertices, and an optimal algorithm for the restricted case, where all cutvertices belong to the same triconnected component, with  $\mathcal{O}(n^{2.5})$  runtime. Better approximation algorithms for the general case have been proposed but are all incorrect. Kant and Bodlaender [7] gave a

3/2-approximation algorithm, but Fialko and Mutzel [2] gave a counter-example showing that their algorithm has an approximation factor not better than 2. In the same paper, they proposed a new algorithm with approximation factor 5/3. However, we could show that even this algorithm cannot have an approximation factor better than 2; see [5, 12].

Since PBA is hard in general, it is worth studying a variant of this problem, where we are not only given a planar graph  $G$  but also a combinatorial embedding of  $G$  that has to be preserved. A *combinatorial embedding*  $\Pi$  of a planar graph  $G$  is given by the cyclic order of the incident edges around each vertex in a planar drawing of  $G$ . If  $G$  is connected,  $\Pi$  also defines the set of *faces*, i.e., the edges bounding the regions into which the plane is subdivided by a drawing realizing  $\Pi$ . However, if  $G$  is disconnected, the faces are only defined for each connected component, leaving the freedom to place a drawing of a connected component into any face of another connected component.

The problem we study in this paper is formally defined as follows:

**Definition 1 (PBA-Fix).** Given a planar graph  $G$  and a combinatorial embedding  $\Pi$  of  $G$ , the *planar biconnectivity augmentation problem with fixed embedding (PBA-Fix)* asks for a smallest edge set  $E'$  such that  $G + E'$  is planar and biconnected, and there exists a combinatorial embedding of  $G + E'$  preserving  $\Pi$ .

In this paper, we show that PBA-Fix can be solved efficiently for connected graphs by presenting an  $\mathcal{O}(n(1 + \alpha(n)))$  time algorithm, but becomes  $\mathcal{NP}$ -hard if disconnected graphs are allowed. The crucial part for showing  $\mathcal{NP}$ -hardness stems from the problem of assigning connected components of the graph to the correct faces, which is not specified by the combinatorial embedding.

## 1.1 Preliminaries

We first introduce some basic definitions and results.

Let  $G = (V, E)$  be a graph. A path in  $G$  is a *chain* if all its inner vertices have degree two in  $G$ . A maximal biconnected subgraph of  $G$  is a *biconnected component* or *block* of  $G$ , and a vertex of  $G$  is a *cutvertex* if its removal increases the number of connected components. The *BC-forest*  $\mathcal{B} = (V_c \cup V_b, E_b)$  of  $G$  is a bipartite graph representing the relationships between the blocks and cutvertices of  $G$ . It contains two types of nodes: the B-nodes  $V_b$  are the blocks of  $G$  and the C-nodes  $V_c$  are the cutvertices of  $G$ ; there is an edge between a B-node  $b$  and a C-node  $c$  iff vertex  $c$  is contained in block  $b$ . The *leaves* in a BC-forest, i.e., the blocks containing exactly one cutvertex, are also called *pendants*. If  $G$  is connected,  $\mathcal{B}$  is also connected and thus called *BC-tree*.

**Theorem 1 ([1]).** Let  $G = (V, E)$  be a graph with  $|V| > 2$ , and let  $h$  be the number of connected components,  $p$  the number of pendants,  $d$  the maximum degree of a C-node, and  $q$  the number of isolated B-nodes in the BC-forest of  $G$ . Then

$$\max \{d + h - 2, \lceil p/2 \rceil + q\}$$

edges are necessary and sufficient to biconnect  $G$ , if  $p + q > 1$ .

Consider the BC-forest  $\mathcal{B}$  of a graph  $G$ , and let  $p$  be the number of its leaves. A C-node  $c^*$  whose degree is at least  $\deg(c^*) \geq \lceil p/2 \rceil + 2$  is called *massive*. If  $\mathcal{B}$  does not contain a massive C-node, we say that  $\mathcal{B}$  is *balanced*. We remark that the maximum in Theorem [1](#) becomes  $\lceil p/2 \rceil + q$  if  $\mathcal{B}$  is balanced, and  $d + h - 2$  otherwise.

We denote with  $P$  the set of all pendants in  $\mathcal{B}$ . If  $p_0$  is a pendant, a *simple vertex* in  $p_0$  refers to any vertex in the block  $p_0$  that is not the unique cutvertex of  $G$  contained in  $p_0$ . *Connecting two pendants*  $p_1$  and  $p_2$  means that we insert an edge  $(v, w)$  in  $G$ , such that  $v$  and  $w$  are any two simple vertices in  $p_1$  and  $p_2$ , respectively.

We say two pendants  $p_1$  and  $p_2$  satisfy the *leaf-connection condition* iff  $p_1$  and  $p_2$  lie in the same tree of  $\mathcal{B}$  and the path from  $p_1$  to  $p_2$  contains either two nodes with degree at least three or one B-node with degree at least four. The following lemma shows, that such pendants allow us to decrease the number of pendants by two by only inserting a single edge.

**Lemma 1** ([\[6, 2\]](#)). *If two pendants  $p_1$  and  $p_2$  satisfy the leaf-connection condition, a new edge connecting both pendants decreases the number of pendants by two. In case the graph is connected and the corresponding BC-tree contains at least three pendants, the previous relation is an equivalence relation.*

Therefore, we also say that two pendants satisfying the leaf-connection condition can be *profitably* connected. For a connected graph with at least three pendants, a set  $B \subseteq P$  is a *bundle* if

- (i) any two pendants in  $B$  can be connected without losing planarity;
- (ii) connecting any two pendants in  $B$  creates a new pendant; and
- (iii)  $B$  is maximal with respect to condition (i) and (ii).

Condition (ii) in the definition of bundles and Lemma [1](#) guarantee that the path between two pendants of the same bundle  $B$  contains exactly one node with degree at least three. We call this node the *parent* of  $B$ . If  $B$  contains just one pendant, say  $p_1$ , its parent is defined as the C-node  $c$  satisfying the following conditions:

- (i) The path between  $p_1$  and  $c$  is a chain;
- (ii) adding the edge  $(c_p, c)$  in  $G$  preserves planarity, where  $c_p$  is the only cutvertex adjacent to  $p_1$ ; and
- (iii) the path from  $p_1$  to  $c$  is the longest among all C-nodes satisfying (i) and (ii).

We call a bundle  $B$  together with its parent a *label*. Depending on the type of the parent, we also say C-label if the parent is a C-node, and B-label otherwise. The *size* of a label is the number of pendants in the bundle  $B$ . The definition of labels implies that two pendants belonging to different labels always satisfy the leaf-connection condition.

### 1.2 Organization

This paper is organized as follows. In Sect. 2, we show that PBA-Fix is NP-hard in general. Then, Sect. 3 presents an algorithm for planar biconnectivity augmentation with fixed embedding for connected graphs. We prove the optimality of this algorithm (Sect. 3.1) by showing that it meets the lower bound for biconnectivity augmentation (Theorem 1) within each face and analyze its runtime and space requirements (Sect. 3.2). Finally, Sect. 4 concludes the paper with one open problems.

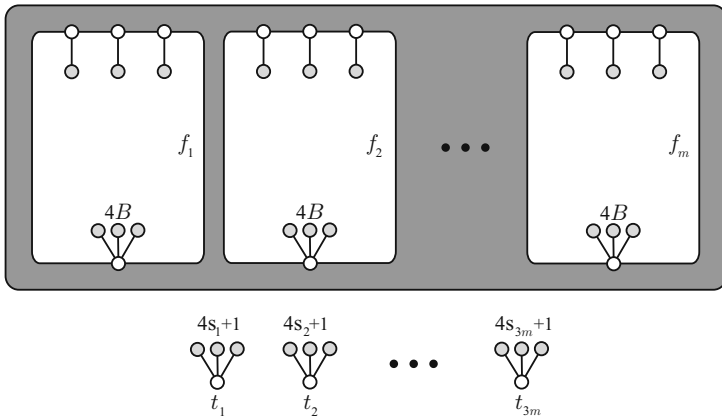
## 2 $\mathcal{NP}$ -Hardness for Non-connected Graphs

In this section we show that PBA-Fix is  $\mathcal{NP}$ -hard in general. We prove  $\mathcal{NP}$ -hardness of PBA-Fix by constructing a polynomial-time reduction from the strongly  $\mathcal{NP}$ -complete decision problem 3-PARTITION, which is defined as follows: Given  $3m$  positive integers  $s_1, \dots, s_{3m}$  and a positive integer bound  $B$  such that  $\sum_{i=1}^{3m} s_i = mB$  with  $B/4 < s_i < B/2$  for  $i = 1, \dots, 3m$ , can  $s_1, \dots, s_{3m}$  be partitioned into  $m$  triplets  $S_1, \dots, S_m$  such that  $\sum_{s \in S_j} s = B$  for  $j = 1, \dots, m$ ?

**Theorem 2.** *PBA-Fix is  $\mathcal{NP}$ -hard for disconnected graphs.*

*Proof.* Consider an instance  $\mathcal{I} = \langle s_1, \dots, s_{3m}; B \rangle$  of 3-PARTITION. We construct a planar, disconnected graph  $G_{\mathcal{I}} = (V, E)$  and an embedding  $\Pi_{\mathcal{I}}$  of  $G_{\mathcal{I}}$  with the property that  $\mathcal{I}$  has a valid partition if and only if  $G_{\mathcal{I}}$  can be made biconnected by adding  $4mB + 3m$  edges while preserving  $\Pi_{\mathcal{I}}$ ; compare Fig. 1.

$G_{\mathcal{I}}$  consists of  $3m + 1$  connected components:  $3m$  simple trees representing the integers  $s_1, \dots, s_{3m}$  and one dense graph (the shaded part in Fig. 1) with  $m$  relevant faces, i.e., one face for each triplet. More precisely, the  $j$ -th tree  $t_j$  has  $4s_j + 1$  leaves and each face  $f_i$ ,  $i = 1, \dots, m$ , contains  $4B + 3$  pendants. Hence,



**Fig. 1.** The constructed graph  $G_{\mathcal{I}}$  for the reduction from 3-PARTITION to PBA-Fix

$G_{\mathcal{T}}$  has  $8mB + 6m$  pendants in total. The embedding  $\Pi_{\mathcal{T}}$  of  $G_{\mathcal{T}}$  is fixed as shown by Fig. 11

If  $\mathcal{T}$  has a valid partition assigning the  $i$ -th integer to the  $\varphi(i)$ -th triplet, the graph can obviously be made biconnected by inserting  $4mB + 3m$  edges: We embed each tree  $t_i$  into face  $f_{\varphi(i)}$ , connect one of its pendants with one of the size-one labels in  $f_{\varphi(i)}$ , and the other pendants with pendants from the label of size  $4B$  in  $f_{\varphi(i)}$ . Since  $\sum_{s \in S_i} s = B$  for  $i = 1, \dots, m$ , each pendant can be connected profitably leading to  $4mB + 3m$  edges in total.

Now assume that an optimal solution of PBA-Fix for  $G_{\mathcal{T}}$  contains  $4mB + 3m$  edges. We show in two steps that  $\mathcal{T}$  has a valid partition by determining properties of the augmented graph. Firstly, we observe that each face  $f_i$  contains exactly three  $t_j$ -trees. If this would not be the case an augmentation of a face with at most two trees would induce at least one expensive pendant, since  $s_j < B/2$  holds for each  $j$  and therefore  $3 + 4s_j + 4s_k \leq 3 + 4(B - 1) < 4B$  for all  $j, k \in \{1, \dots, 3m\}$ .

Secondly, we show that the assignment of the  $3m$  trees to the  $m$  faces induces a valid partition. Therefore,  $4s_j + 4s_k + 4s_l = 4B$  must hold for each face  $f_i$  containing the trees  $t_j, t_k, t_l$ . Assume, there exists a face  $f_i$  with the property that  $4s_j + 4s_k + 4s_l > 4B$ . Then, the number of pendants in  $f_i$  is  $4(B + s_j + s_k + s_l) + 6 \geq 8B + 7$  and it follows directly from Theorem 11 that at least  $\lceil p/2 \rceil = 4B + 4$  edges are necessary for making  $f_i$  biconnected. Theorem 11 also implies that each face requires at least  $d + h - 2 = 4B + 3$  edges for augmentation, since we have  $h = 4$  connected components (the three trees and the subgraph induced by the face) and the maximum degree in the corresponding BC-forest is  $d = 4B + 1$ . Therefore, the whole augmenting set would include at least  $4B + 4 + (m - 1)(4B + 3) = 4mB + 3m + 1$  edges and  $s_j + s_k + s_l = B$  holds for each face, thus inducing a valid partitioning of the integers.

The size of  $G_{\mathcal{T}}$  is bounded by  $\mathcal{O}(mB)$  and  $G_{\mathcal{T}}$  can also be constructed in time  $\mathcal{O}(mB)$ . Since PBA-Fix obviously belongs to  $\mathcal{NP}$  and 3-PARTITION is  $\mathcal{NP}$ -complete in the strong sense [3], implying that there is no optimal pseudo-polynomial algorithm for 3-PARTITION unless  $\mathcal{P} = \mathcal{NP}$ , the theorem follows.  $\square$

### 3 The Algorithm for Connected Graphs

Though PBA-Fix is  $\mathcal{NP}$ -hard in general, we show in this section that the problem can be solved efficiently for connected graphs. We first present the algorithm and then prove its optimality (Sect. 3.1) and runtime and space requirements (Sect. 3.2). Further details can be found in [12].

Procedure PLANARAUGMENTATIONFIX depicted in Alg. 1 sketches the algorithm. Its subprocedures HANDLEPENDANT, FINDMATCHING, and UPDATE are described in the following paragraphs.

We are given a connected, planar graph  $G = (V, E)$  and a combinatorial embedding  $\Pi$  of  $G$ , and the procedure inserts a minimum number of edges into  $\Pi$  such that  $G$  becomes biconnected. The approach is based on iteratively

---

**Algorithm 1.** PLANARAUGMENTATIONFIX

---

**Input:** a planar, connected graph  $G = (V, E)$  and a combinatorial embedding  $\Pi$  of  $G$ 

```

1: for all faces  $f \in \Pi$  do
2:   construct the BC-tree  $\mathcal{B}_f$  of the subgraph induced by face  $f$ 
3:   if number of pendants  $\leq 3$  then
4:     connect the  $p$  pendants with  $p - 1$  edges
5:   else
6:     for all pendants  $p_i$  of  $\mathcal{B}_f$  do
7:       HANDLEPENDANT( $p_i$ )
8:     end for
9:   end if
10:  while number of labels  $> 1$  do
11:    let  $\ell$  be a label of maximum size
12:     $(p_1, p_2) \leftarrow \text{FINDMATCHING}(\ell)$ 
13:    insert a new edge into  $f$  connecting simple vertices of  $p_1$  and  $p_2$ 
14:    UPDATE()
15:  end while
16:  if number of pendants  $> 0$  then
17:    connect the remaining  $p$  pendants with  $p - 1$  edges
18:  end if
19: end for

```

---

connecting pendants of different labels, i.e., each such inserted edge connects simple vertices of the corresponding blocks.

Since the embedding  $\Pi$  of the input graph is fixed, it is sufficient to consider and augment each face  $f$  separately. Therefore, we compute the BC-tree  $\mathcal{B}_f$  of the subgraph induced by  $f$ , i.e., the graph induced by the edges on the boundary of  $f$ . We have to make sure that  $\mathcal{B}_f$  reflects the embedding of the graph, since the idea of the actual augmentation is to select the largest label and connect its leftmost pendant with the rightmost pendant of the neighboring label to its left. This is accomplished by procedure FINDMATCHING, which traverses the BC-tree in clockwise direction from the first pendant of a label to the next pendant; here, a doubly linked list as representation is sufficient to perform this task efficiently. If both pendants belong to the same label, their ordering in the list is adjusted to reflect the embedding (this is easy since these pendants have to come after each other) and the traversal continues from the latter pendant; adjusting the ordering in the lists assures that we do not have to traverse paths in the BC-tree between two pendants of the same label twice. Otherwise, both pendants are neighbors with respect to the embedding and fulfill the leaf-connection condition, since they belong to different labels.

After inserting an edge, the BC-tree needs to be updated, since the *insertion path*, i.e., the unique path between the two pendants we connect, and the inserted edge induce a new cycle in  $G$ . The following observation details the changes in the BC-tree.

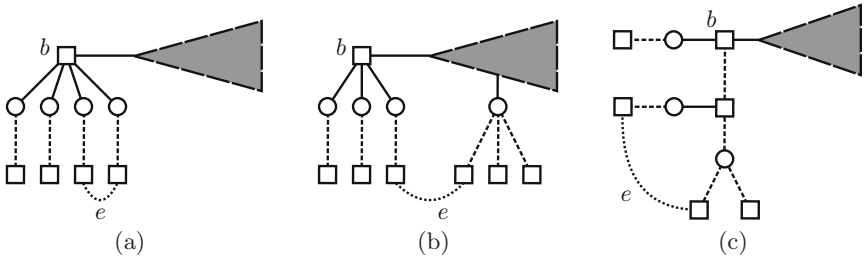
**Observation 1** ([6, 9]). *Let  $G$  be a connected graph,  $\mathcal{B}_G$  its BC-tree, and  $p_1$  and  $p_2$  two pendants of  $\mathcal{B}_G$ . Let  $G'$  and  $\mathcal{B}'_G$  be the graph and BC-tree after inserting an edge between two simple vertices of  $p_1$  and  $p_2$ , and denote with  $P$  the path in  $\mathcal{B}_G$  from  $p_1$  to  $p_2$ . Then,  $\mathcal{B}'_G$  is obtained from  $\mathcal{B}_G$  as follows:*

1. *All B-nodes on  $P$  are merged into one B-node  $b^*$ .*
2. *Any C-node on  $P$  with degree two is eliminated.*
3. *Every C-node on  $P$  with degree at least three remains in  $\mathcal{B}'_G$  adjacent to nodes not on  $P$ , and is now also adjacent to  $b^*$ .*

For maintaining the BC-tree, we use the dynamic data structure proposed by Westbrook and Tarjan [11]. Here, nodes of the BC-tree are not contracted after inserting a new edge. Instead, the initial BC-tree remains unchanged and the current tree structure is realized by an additional union-find data structure on the BC-nodes. Moreover, we root the BC-tree at a B-node with degree at least two, which allows us to find the path between two pendants in time linear in the length of the path, applying a straightforward bottom up search in the tree.

A crucial part of our approach is the correct computation of the labels. For the initial assignment of pendants to labels, procedure HANDLEPENDANT starts a bottom-up traversal from each pendant until a node with degree at least three is reached. If this node is a C-node it always becomes the parent of a label containing at least the pendant from where we started the bottom-up traversal; further pendants might be added to this label as well. A B-node becomes the parent of a label if its degree is three and the subtrees rooted at its two children do not contain a node with degree greater than two. Since this property can only be decided after considering all pendants by HANDLEPENDANT, we introduce a second structure for bundles called pseudo-labels. A *pseudo-label* has a B-node  $b$  as parent and contains the size-one C-labels whose parents are adjacent to  $b$ . If such a pseudo-label has size two, it becomes a real label and furthermore, the adjacent size-one C-labels are deleted. Otherwise, the adjacent C-nodes remain parents.

Furthermore, we apply pseudo-labels for maintaining labels as described in the following. After inserting an edge the BC-tree and the labels have to be updated. The size of each of the two labels to which the pendants we connect belong decreases by one. A pendant whose label reaches size one needs to be reconsidered by HANDLEPENDANT, since it probably has to be reassigned to another label. Furthermore, labels whose parents are not part of the insertion path might require an update. In particular, pseudo-labels might become real labels if the number of adjacent size-one C-labels decreases to two. Fig. 2 illustrates three possible cases how the contained size-one C-labels of a pseudo-label may become obsolete after inserting an edge  $e$ . Like mentioned before, the parents of the size-one C-labels do not lie on the insertion path. However, they are affected by the new edge and the related pseudo-label is transformed into a B-label. In Fig. 2(a) and (b) the parent of the pseudo-label is part of the insertion path, whereas in (c) this property does not hold and the pseudo-label only changes due to the new adjacent chain. Therefore, we first test each pseudo-label whose parent lies on the insertion path for its size and transform it into a B-label, if



**Fig. 2.** Three cases where a pseudo-label with parent  $b$  becomes a real B-label after inserting an edge  $e$ . The shaded parts represent subtrees, the dashed lines depict chains, C-nodes are represented by circles, and B-nodes by rectangles.

necessary. Secondly, further affected pseudo-labels, like the one in Fig. 2(c), are updated implicitly, since their parents are found during the traversal in HANDLEPENDANT which is called for the remaining pendant of a new size-one label.

In the special case, when the number of pendants decreases to at most three or there is only one label left, the  $p$  remaining pendants are connected with  $p - 1$  edges by adding an edge between simple vertices of consecutive pendants. The consideration of the involved labels and pseudo-labels, as well as the updates of the BC-tree, are accomplished by procedure UPDATE.

### 3.1 Optimality

We first examine the relationship between maximum labels and massive or critical C-nodes. We call a C-node *critical* if its degree is  $\lceil \frac{p}{2} \rceil + 1$ .

**Lemma 2.** *If a BC-tree  $\mathcal{B}$  is not balanced, its massive C-node  $c^*$  must be the parent of a label, say  $\ell_0$ . Furthermore,  $\ell_0$  is the unique maximum label in  $\mathcal{B}$ .*

*Proof.* Let  $p$  denote the number of pendants in  $\mathcal{B}$ . We prove both statements by contradiction.

Assume that  $c^*$  is not the parent of a label, let  $d$  be its degree and denote with  $t_1, \dots, t_d$  the  $d$  subtrees of  $\mathcal{B}$  we would get after removing  $c^*$ . Then, there cannot exist any chain from a pendant to  $c^*$ , and therefore each  $t_i$ ,  $i = 1, \dots, d$ , would contain more than one pendant of  $\mathcal{B}$ . Since  $c^*$  is massive,  $d$  is at least  $\lceil \frac{p}{2} \rceil + 2$ . Thus, there would be at least  $2(\lceil \frac{p}{2} \rceil + 2)$  pendants; however, the BC-tree has only  $p$  pendants.

Now, let  $j$  be the number of pendants contained in label  $\ell_0$ , i.e., the pendants from  $j$  subtrees out of  $t_1, \dots, t_d$  belong to  $\ell_0$ . Then, the other  $d - j$  subtrees each contain at least two pendants. Now assume there exists another C-node  $c_2$  that is the parent of another label with at least the same size as  $\ell_0$ . Altogether the BC-tree would then have at least  $2 \cdot j + 2(d - j - 1) \geq p + 2$  pendants, which is again a contradiction.  $\square$

Analogously, critical C-nodes are also always parents of labels. In general, the second statement in Lemma 2 is not true for critical nodes. However, we can



show—again by contradiction—that no label is greater than a label whose parent is a critical C-node. We state this in the following lemma and skip the proof because it is similar to the previous one.

**Lemma 3.** *Let  $c$  be a critical C-node in a BC-tree  $\mathcal{B}$ . Then,  $c$  is the parent of a label, say  $\ell_c$ , and there is no label in  $\mathcal{B}$  whose size is greater than the size of  $\ell_c$ .*

To show the correctness and optimality of our approach, we remark some further properties of the algorithm. Since edges are inserted between neighboring pendants, i.e., all remaining pendants lie on the same side of the new edge, planarity is ensured and the embedding is preserved. Furthermore, the augmented edges obviously assure that the graph becomes biconnected.

The correct computation of labels is essential for guaranteeing optimality of the solution. The initial assignment of pendants to their labels is obviously correct. After inserting an edge, some labels might require an update. Since the two involved labels—the labels whose pendants we connect—are considered directly, HANDLEPENDANT is called for the remaining pendants if they reach size one, and pseudo-labels on the path are also taken into account, thus all labels are updated correctly.

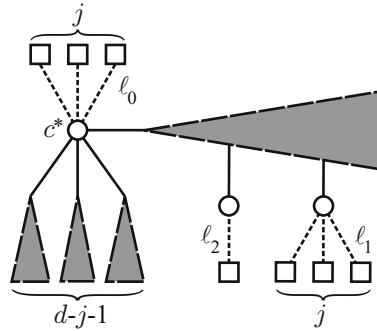
**Theorem 3.** *The solution computed by PLANARAUGMENTATIONFIX is an optimal solution for PBA-Fix.*

*Proof.* To complete the proof we now show that each face-induced subgraph is made biconnected by the minimum number of edges, namely  $\max\{d - 1, \lceil \frac{p}{2} \rceil\}$  as given by Theorem 1, where  $d$  is the maximum degree of a C-node and  $p$  the number of pendants in the BC-tree  $\mathcal{B}$ . To prove this, we consider two cases separately, which are based on the existence of a massive c-node.

Firstly, we consider the case, where  $\mathcal{B}$  contains a massive C-node  $c^*$ . In this case, the algorithm must not exceed the lower bound of adding  $\deg(c^*) - 1$  new edges to biconnect the graph. By Lemma 2 the parent of the unique maximum label  $\ell_0$  in  $\mathcal{B}$  is  $c^*$ . Therefore, the algorithm automatically selects  $\ell_0$  to find an appropriate matching. The following induction on  $\deg(c^*)$  implies that the bound  $\deg(c^*) - 1$  is achieved:

*Base Case:* Since a node with degree  $\leq 3$  cannot be massive, the base case is  $\deg(c^*) = 4$ . By definition,  $\deg(c^*) \geq \lceil \frac{p}{2} \rceil + 2$  and thus  $p = 4$  (we cannot have less pendants than  $\deg(c^*)$ ). Hence,  $\mathcal{B}$  consists of one label  $\ell_0$  with 4 pendants, and  $3 = \deg(c^*) - 1$  edges are inserted to biconnect the graph.

*Inductive step:* We have  $\deg(c^*) \geq 5$ . If there is only one label,  $\mathcal{B}$  is a star and  $\deg(c^*) - 1$  edges are inserted. Otherwise, a new edge between a pendant of  $\ell_0$  and another label is inserted. Therefore, the degree of  $c^*$  decreases by one and the BC-tree remains unbalanced since  $\lceil \frac{p}{2} \rceil$  decreases by one as well. It follows by the induction hypothesis that  $((\deg(c^*) - 1) - 1) + 1 = \deg(c^*) - 1$  edges are inserted.



**Fig. 3.** Sketch of the situation in the BC-tree with a critical C-node  $c^*$ , its pertinent label  $\ell_0$ , and the two other involved labels  $\ell_1$  and  $\ell_2$

Secondly, we consider the case, where  $\mathcal{B}$  is balanced and show that the algorithm achieves the lower bound of  $\lceil \frac{p}{2} \rceil$ . The proof is again inductive, this time on the number of pendants  $p$ .

*Base Cases:* The base cases are  $p \in \{2, 3\}$ . In both cases  $\lceil \frac{p}{2} \rceil$  edges are inserted.

*Inductive step:* We have  $p \geq 4$ . First, notice that  $\mathcal{B}$  contains at least two labels, and therefore the new edge satisfies the leaf-connection condition. Hence, the number of pendants decreases by two and, if the resulting BC-tree remains balanced, we can conclude that only  $\lceil \frac{p-2}{2} \rceil + 1 = \lceil \frac{p}{2} \rceil$  edges are inserted.

Therefore, we show by contradiction that no massive C-node arises during the algorithm. Thus, assume that  $c^*$  is the new massive C-node after inserting a new edge. Let  $p$  and  $p'$  denote the number of pendants, and  $\text{deg}^{\text{before}}$  and  $\text{deg}^{\text{after}}$  the degree of a node before and after the insertion, respectively. Therefore,  $p' = p - 2$  holds and from the definition of a massive c-node  $\text{deg}^{\text{after}}(c^*) \geq \lceil p'/2 \rceil + 2 = \lceil \frac{p-2}{2} \rceil + 2$ . By Observation [III](#) the degree of a C-node lying on the insertion path between the two connected pendants either reduces by one or it is contracted to the new B-node. Therefore,  $c^*$  cannot lie on the insertion path, since otherwise

$$\text{deg}^{\text{before}}(c^*) = \text{deg}^{\text{after}}(c^*) + 1 \geq \left\lceil \frac{p'}{2} \right\rceil + 3 = \left\lceil \frac{p}{2} \right\rceil + 2$$

and  $c^*$  would already be a massive c-node.

Hence,  $d := \text{deg}^{\text{before}}(c^*) = \text{deg}^{\text{after}}(c^*) = \lceil p'/2 \rceil + 2 = \lceil p/2 \rceil + 1$  holds and therefore,  $c^*$  must have been a critical C-node before the edge-insertion. Let  $\ell_0$  denote the label whose parent is  $c^*$  and let  $j$  be its size; thus, there are  $d - j$  subtrees adjacent to  $c^*$  and each of them contains at least two pendants. There have to exist two other labels  $\ell_1$  and  $\ell_2$  in the BC-tree since  $\ell_0$  is not involved in the new edge; compare Fig. [3](#). One of these labels also needs to contain at least  $j$  pendants because this label was selected by the algorithm; w.l.o.g. let this label be  $\ell_1$ . Both of the labels  $\ell_1$  and  $\ell_2$  are part of the same subtree adjacent to  $c^*$ , since otherwise  $c^*$  would be a part of the insertion path. Altogether, the number of pendants of the BC-tree before inserting the edge is

$$\begin{aligned}
&\geq \underbrace{j}_{\ell_0} + \underbrace{j}_{\ell_1} + \underbrace{2(d-j-1)}_{\text{other subtrees}} + \underbrace{1}_{\ell_2} \\
&= 2d - 1 \\
&\geq 2\left(\left\lceil \frac{p}{2} \right\rceil + 1\right) - 1 \\
&\geq p + 1
\end{aligned}$$

This is the desired contradiction, and thus we have shown that the algorithm always achieves the lower bound of  $\max\{d-1, \lceil \frac{p}{2} \rceil\}$  edges for each face-induced subgraph and the theorem follows.  $\square$

### 3.2 Runtime and Space

We prove now the main result of this section.

**Theorem 4.** *Let  $G = (V, E)$  be a planar, connected graph and  $\Pi$  a combinatorial embedding of  $G$ . Then, PLANARAUGMENTATIONFIX computes an optimal solution of PBA-Fix for  $G$  and  $\Pi$  in  $\mathcal{O}(|V|(1 + \alpha(|V|)))$  time while requiring only linear space, where  $\alpha$  denotes the inverse of the Ackermann function.*

*Proof.* We already proved optimality of the solution in Theorem 3, so it remains to analyze the runtime and space requirements of PLANARAUGMENTATIONFIX. We first remark that the number of edges in a planar graph is always linear in the number of nodes. The algorithm considers each face of  $\Pi$  separately and constructs the corresponding face-induced subgraph. Since each edge of  $G$  belongs to at most two faces, the total number of edges in all face-induced subgraphs is in  $\mathcal{O}(|V|)$ . Therefore, we can derive the total runtime and space requirements from the runtimes and space requirements for handling each face.

Let  $G_f = (V_f, E_f)$  be the subgraph induced by face  $f$ . The BC-tree of  $G_f$  can be computed in time  $\mathcal{O}(|V_f| + |E_f|)$  by finding the biconnected components [10] and requires  $\mathcal{O}(|V_f|)$  space. Using dynamic BC-trees as proposed by Westbrook and Tarjan [21], which utilize a union-find data structure to enable efficient update operations, all updates on the BC-tree require a runtime of  $\mathcal{O}(|V_f|(1 + \alpha(|V_f|)))$  in total. In each iteration, the largest label can be selected by using buckets for labels with equal size. Since a newly inserted edge affects only a constant number of labels and the size of a label changes by at most two, each largest label can be found in constant time. Furthermore, after inserting an edge all B-nodes on the insertion path are merged into one B-node; see Observation 1. Hence, each path in the BC-tree is considered a constant number of times during FINDMATCHING and similar arguments hold for procedure HANDLEPENDANT.

Therefore, the runtime for augmenting a face-induced subgraph is dominated by the BC-tree updates, thus resulting in a total runtime of  $\mathcal{O}(|V|(1 + \alpha(|V|)))$  for the whole algorithm.  $\square$

## 4 Conclusion

We have shown that the complexity status of PBA-Fix depends on the connectivity of the input graph  $G$ . For the general case, i.e., if disconnected graphs

are allowed, PBA-Fix is  $\mathcal{NP}$ -hard; however, if  $G$  is connected it is efficiently solvable and we have presented an  $\mathcal{O}(n(1 + \alpha(n)))$  time algorithm, where  $n$  is the number of vertices in  $G$ . An implementation of this algorithm in our graph drawing library OGDF [8] is currently under development.

In case of a non-connected graph  $G$ , where an assignment of the faces of the connected components to each other is given,  $G$  can be made connected in linear time using the same procedure as described in [9] without losing the optimality for PBA-Fix.

We close with an open problem: *can the runtime of the algorithm for connected graphs be improved to linear runtime?* At the moment, we rely on general dynamic BC-trees, giving us union-find complexity for our algorithm.

## References

- [1] Eswaran, K., Tarjan, R.: Augmentation problems. *SIAM Journal on Computing* 5(4), 653–665 (1976)
- [2] Fialko, S., Mutzel, P.: A new approximation algorithm for the planar augmentation problem. In: *Proc. SODA 1998*, pp. 260–269. SIAM, Philadelphia (1998)
- [3] Garey, M.R., Johnson, D.S.: Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing* 4(4), 397–411 (1975)
- [4] Gutwenger, C., Mutzel, P.: Planar polyline drawings with good angular resolution. In: Whitesides, S.H. (ed.) *GD 1998*. LNCS, vol. 1547, pp. 167–182. Springer, Heidelberg (1999)
- [5] Gutwenger, C., Mutzel, P., Zey, B.: On the hardness and approximability of planar biconnectivity augmentation. In: Ngo, H.Q. (ed.) *COCOON 2009*. LNCS, vol. 5609, pp. 249–257. Springer, Heidelberg (2009)
- [6] Hsu, T.-S., Ramachandran, V.: On finding a smallest augmentation to biconnect a graph. *SIAM Journal on Computing* 22(5), 889–912 (1993)
- [7] Kant, G., Bodlaender, H.L.: Planar graph augmentation problems. In: Dehne, F., Sack, J.-R., Santoro, N. (eds.) *WADS 1991*. LNCS, vol. 519, pp. 286–298. Springer, Heidelberg (1991)
- [8] OGDF: An Open Graph Drawing Framework (2009), <http://www.ogdf.net>
- [9] Rosenthal, A., Goldner, A.: Smallest augmentations to biconnect a graph. *SIAM Journal on Computing* 6(1), 55–66 (1977)
- [10] Tarjan, R.: Depth-first search and linear graph algorithms. *SIAM Journal on Computing* 1(2), 146–160 (1972)
- [11] Westbrook, J., Tarjan, R.: Maintaining bridge-connected and biconnected components on-line. *Algorithmica* 7(5&6), 433–464 (1992)
- [12] Zey, B.: Algorithms for planar graph augmentation. Master’s thesis, Dortmund University of Technology (2008), [http://ls11-www.cs.uni-dortmund.de/people/gutweng/diploma\\_thesis\\_zey.pdf](http://ls11-www.cs.uni-dortmund.de/people/gutweng/diploma_thesis_zey.pdf)

# Trivially-Perfect Width

Ling-Ju Hung<sup>1</sup>, Ton Kloks\*, and Chuan Min Lee<sup>2,\*\*</sup>

<sup>1</sup> Department of Computer Science and Information Engineering  
National Chung Cheng University  
Min-Hsiung, Chia-Yi 621, Taiwan  
hunglc@cs.ccu.edu.tw

<sup>2</sup> Department of Computer and Communication Engineering  
Ming Chuan University  
5 De Ming Rd., Guishan District, Taoyuan County 333, Taiwan  
joneslee@mail.mcu.edu.tw

**Abstract.** The  $\mathcal{G}$ -width of a class of graphs  $\mathcal{G}$  is defined as follows. A graph  $G$  has  $\mathcal{G}$ -width  $k$  if there are  $k$  independent sets  $\mathbb{N}_1, \dots, \mathbb{N}_k$  in  $G$  such that  $G$  can be embedded into a graph  $H \in \mathcal{G}$  with the property that for every edge  $e$  in  $H$  which is not an edge in  $G$ , there exists an  $i$  such that both endpoints of  $e$  are in  $\mathbb{N}_i$ . For the class  $\mathfrak{TP}$  of trivially-perfect graphs we show that  $\mathfrak{TP}$ -width is NP-complete and we present fixed-parameter algorithms.

## 1 Introduction

The recognition problem of probe interval graphs was introduced by Zhang *et al.* [8,14]. This problem stems from the physical mapping of chromosomal DNA of humans and other species. Since then probe graphs of many other graph classes have been investigated by various authors. We generalize the concept to the graph-class-width parameters.

**Definition 1.** Let  $\mathcal{G}$  be a class of graphs which contains all cliques. The  $\mathcal{G}$ -width of a graph  $G$  is the minimum number  $k$  of independent sets  $\mathbb{N}_1, \dots, \mathbb{N}_k$  in  $G$  such that there exists an embedding  $H \in \mathcal{G}$  of  $G$  with the property that for every edge  $e = (x, y)$  in  $H$  which is not an edge of  $G$ , there exists an  $i$  with  $x, y \in \mathbb{N}_i$ .

In this paper we investigate the width-parameter for the class  $\mathfrak{TP}$  of trivially-perfect graphs, henceforth called the *trivially-perfect width*, or  $\mathfrak{TP}$ -width. If a graph  $G$  has  $\mathfrak{TP}$ -width  $k$  then we call  $G$  also a *k-probe trivially-perfect graph*. This paper deals with the recognition problem of  $k$ -probe trivially-perfect graphs. We refer to the *partitioned case* of the problem when a collection of, possibly overlapping, independent sets  $\mathbb{N}_i$ ,  $i = 1, \dots, k$  is a part of the input. We call such a collection of independent sets a *witness*.

---

\* This author is supported by the National Science Council of Taiwan under grant NSC 97-2811-E-194-001.

\*\* The author is partially supported by the National Science Council of Taiwan under grant NSC 97-2218-E-130-002-MY2.

Trivially-perfect graphs were first studied by Wolk [13]. However, Golumbic gave the class its name [4].

**Definition 2** ([4]). *A graph is trivially perfect if for each induced subgraph the independence number is equal to the number of maximal cliques.*

We use Wolk's characterization of the class.

**Theorem 1** ([13]). *A graph is trivially perfect if and only if every connected induced subgraph has a universal vertex*<sup>1</sup>.

We denote the class of trivially-perfect graphs by  $\mathfrak{TP}$ . The class can also be characterized by excluding the  $C_4$  and  $P_4$  as induced subgraphs; thus the class of trivially-perfect graphs is exactly the class of *chordal cographs*. In this paper we study the  $\mathfrak{TP}$ -width of graphs.

It follows from Theorem 1 that a connected graph  $G = (V, E)$  is trivially perfect if and only if there exists a rooted tree  $T$ , with node-set  $V$  such that two vertices  $x$  and  $y$  are adjacent in  $G$  if and only if one lies on the path from the root to the other. Thus the set of vertices of any path from the root to a leaf in  $T$  induces a maximal clique in  $G$ , and these are all the maximal cliques in  $G$ .

It can be seen that the class of partitioned  $k$ -probe trivially-perfect graphs can be characterized by a finite set of forbidden induced, partitioned subgraphs, see, e.g., [6, 11, 12]. We think that a similar statement holds also for the unpartitioned case, but we have no proof of this yet.

In the following, we write some of our notational customs. For two sets  $A$  and  $B$  we write  $A + B$  and  $A - B$  instead of  $A \cup B$  and  $A \setminus B$ . We write  $A \subseteq B$  if  $A$  is a subset of  $B$  with possible equality and we write  $A \subset B$  if  $A$  is a subset of  $B$  and  $A \neq B$ . For a set  $A$  and an element  $x$  we write  $A + x$  instead of  $A + \{x\}$  and  $A - x$  instead of  $A - \{x\}$ . It will be clear from the context when  $x$  is an element instead of a set.

A graph  $G$  is a pair  $G = (V, E)$  where  $V$  is a *finite* set, of which the elements are called the vertices of  $G$ , and where  $E$  is a set of two-element subsets of  $V$ , of which the elements are called the edges of  $G$ . We denote edges of a graph as  $(x, y)$  and we call  $x$  and  $y$  the endvertices of the edge. For a vertex  $x$  we write  $N(x)$  for its set of neighbors and we write  $N[x] = N(x) + x$  for the closed neighborhood of  $x$ . For a subset  $W \subseteq V$  we write  $N(W) = \bigcup_{x \in W} N(x) - W$  for its neighborhood and we write  $N[W] = N(W) + W$  for its closed neighborhood. Usually we use  $n = |V|$  to denote the number of vertices of  $G$  and we use  $m = |E|$  to denote the number of edges of  $G$ .

For a graph  $G = (V, E)$  and a subset  $S \subseteq V$  of vertices we write  $G[S]$  for the subgraph *induced* by  $S$ , that is, the graph with  $S$  as its set of vertices and with those edges of  $E$  that have both endvertices in  $S$ . For a subset  $W \subseteq V$  we write  $G - W$  for the graph  $G[V - W]$ . For a vertex  $x$  we write  $G - x$  rather than  $G - \{x\}$ . We usually denote graph classes by calligraphic capitals.

---

<sup>1</sup> A vertex is *universal* if it is adjacent to all other vertices.

The paper is organized as follows: in Section 2, we show that the rankwidth of  $k$ -probe trivially-perfect graphs is bounded and for constant  $k$ ,  $k$ -probe trivially-perfect graphs can be recognized in  $O(n^3)$  by formulating the problem in  $C_2MS$ -logic. In Section 3, we give a linear time algorithm for the recognition of partitioned  $k$ -probe trivially-perfect graphs. In Section 4, we show that  $\mathfrak{B}$ -width is NP-complete. In Section 5, we give a fixed-parameter algorithm to check if a given graph is a  $k$ -probe trivially-perfect graph.

Since the results are comparable to those of rankwidth, it seems worthwhile to examine  $\mathcal{G}$ -width for specific classes of graphs. We started with one of the ‘easiest’ in this paper. Although for this class rankwidth gives an alternative solution, a closer examination of the method seems not a bad idea anyway. Next in line are the distance-hereditary graphs; for those no monadic-order formulation is available at the moment. We hope that, with this paper, we obtained a sufficient understanding of the methods to solve this, and other classes in future research.

## 2 Trivially-Perfect Width Is Fixed-Parameter Tractable

In this section we show that for constant  $k$ ,  $k$ -probe trivially-perfect graphs can be recognized in  $O(n^3)$  time.

The following Theorem 2 is a characterization which can be formulated in  $C_2MS$ -logic 3. We will prove that  $k$ -probe trivially-perfect graphs have *rankwidth* at most  $2^k$  shortly. It is known that problems which can be formulated in  $C_2MS$ -logic can be solved in  $O(n^3)$  time on graphs with bounded rankwidth 3.

**Definition 3.** Let  $(G, \mathcal{N})$  be a partitioned graph with a witness

$$\mathcal{N} = \{\mathbb{N}_i \mid i = 1, \dots, k\}$$

of  $k$ , possibly overlapping independent sets in  $G$ . A vertex  $\omega$  is probe universal if for every vertex  $x \neq \omega$  either

- (i)  $(x, \omega) \in E$ , or
- (ii) there exists  $i \in \{1, \dots, k\}$  with  $\{x, \omega\} \subseteq \mathbb{N}_i$ .

**Theorem 2.** A graph  $G = (V, E)$  is a  $k$ -probe trivially-perfect graph if and only if there exist independent sets  $\mathbb{N}_i$ ,  $i = 1, \dots, k$ , such that every connected induced subgraph has a probe universal vertex.

*Proof.* Assume that  $G$  is a  $k$ -probe trivially-perfect graph. Let  $\mathcal{N}$  be a witness of  $k$  independent sets  $\mathbb{N}_i$  and let  $H$  be an embedding, obtained by adding edges in these independent sets. Let  $C \subseteq V$  be a subset of vertices such that  $G[C]$  is connected. Then  $H[C]$  is also connected. By Theorem 1  $H[C]$  has a universal vertex  $\omega$ . Let  $x$  be another vertex of  $G[C]$ . Since  $H$  is an embedding of  $(G, \mathcal{N})$ , either  $x$  and  $\omega$  are adjacent in  $G$ , or there exists an  $i$  such that  $x, \omega \in \mathbb{N}_i$ .

Assume that there exists a witness  $\mathcal{N}$  of  $k$  independent sets  $\mathbb{N}_i$  such that every connected induced subgraph of  $G$  has a probe universal vertex. Let  $C$  be

a component of  $G$ . We show that  $G[C]$  can be embedded into a trivially perfect graph. Let  $\omega$  be a probe universal vertex in  $G[C]$ . By induction we may assume that  $G[C] - \omega$  with the induced witness has a trivially-perfect embedding  $H'$ . We obtain an embedding  $H$  of  $G[C]$  by adding  $\omega$  as a universal vertex to  $H'$ .  $\square$

**Definition 4** ([10]). A rank-decomposition of a graph  $G = (V, E)$  is a pair  $(T, \tau)$  where  $T$  is a ternary tree and  $\tau$  a bijection from the leaves of  $T$  to the vertices of  $G$ . Let  $e$  be an edge in  $T$  and consider the two sets  $A$  and  $B$  of leaves of the two subtrees of  $T - e$ . Let  $M_e$  be the submatrix of the adjacency matrix of  $G$  with rows indexed by the vertices of  $A$  and columns indexed by the vertices of  $B$ . The width of  $e$  is the rank over  $GF(2)$  of  $M_e$ . The width of  $(T, \tau)$  is the maximum width over all edges  $e$  in  $T$  and the rankwidth of  $G$  is the minimum width over all rank-decompositions of  $G$ .

The class of graphs with rankwidth at most 1 is exactly the class of distance-hereditary graphs [2,5,10]. Note that every trivially-perfect graph is distance hereditary [1], since every induced path has length 1 or 2 by Theorem [1].

**Theorem 3.**  $k$ -Probe trivially-perfect graphs have rankwidth at most  $2^k$ .

*Proof.* Consider a rank-decomposition  $(T, \tau)$  with width 1 for an embedding  $H$  of a  $k$ -probe trivially-perfect graph  $G$ . Consider an edge  $e$  in  $T$  and assume that  $M_e$  is an all 1s-matrix. Each independent set  $N_i$  creates a 0-submatrix in  $M_e$ . If  $k = 1$  this proves that the rankwidth of  $G$  is at most 2. In general, for  $k \geq 0$ , note that there are at most  $2^k$  different neighborhoods from one leaf-set of  $T - e$  to the other. It follows that the rank of  $M_e$  is at most  $2^k$ .  $\square$

**Theorem 4.** For each  $k \geq 0$  there exists an  $O(n^3)$  algorithm which checks whether a graph  $G$  with  $n$  vertices is a  $k$ -probe trivially-perfect graph; that is,  $\mathfrak{TP}$ -width is in FPT.

*Proof.* By Theorem [3]  $k$ -probe trivially-perfect graphs have bounded rankwidth. It is well-known that  $C_2MS$ -problems can be solved in  $O(n^3)$  time for graphs of bounded rankwidth (see [3], and follow pointers from there). By Theorem [2], the recognition of  $k$ -probe trivially-perfect graphs is such a problem.  $\square$

### 3 Partitioned $k$ -Probe Trivially-Perfect Graphs

Obviously, the result of the previous section holds as well when the collection of independent sets  $N_1, \dots, N_k$  is a part of the input. Thus for each  $k$  there is an  $O(n^3)$  algorithm that checks whether a graph  $G$  with a witness of  $k$  independent sets  $N_i$ , can be embedded into a trivially-perfect graph. However, there are a few drawbacks to this solution. First of all, Theorem [4] only shows the *existence* of an  $O(n^3)$  recognition algorithm. In any case, *a priori*, it is unclear how to obtain the algorithm explicitly. Furthermore, the constants involved in the algorithm make the solution impractical. Already there is an exponential blow-up when one moves from  $\mathfrak{TP}$ -width to rankwidth.



In this section we show that there exists an easy algorithm for the recognition of partitioned  $k$ -probe trivially-perfect graphs by recursively eliminating a probe universal vertex.

**Proposition 1.** *Let  $(G, \mathcal{N})$  be a partitioned graph with a witness*

$$\mathcal{N} = \{\mathbb{N}_i \mid i = 1, \dots, k\}$$

*of  $k$ , possibly overlapping independent sets. Then  $(G, \mathcal{N})$  is a partitioned  $k$ -probe trivially-perfect graph if and only if every component of  $G$ , with the induced witness, is a partitioned  $k$ -probe trivially-perfect graph.*

**Theorem 5.** *There exists a linear-time algorithm to check whether a partitioned graph  $(G, \mathcal{N})$  with a witness  $\mathcal{N}$  of  $k$  independent sets, is a partitioned  $k$ -probe trivially-perfect graph.*

*Proof.* If  $G$  is disconnected then, by Proposition 1, we can check each component individually. Assume  $G$  is connected. It is easy to see that we can compute an elimination ordering by probe universal vertices in linear time, by keeping a list of vertices ordered by degree. □

*Remark 1.* Note that the algorithm described in Theorem 5 is fully polynomial. The algorithm can be used to compute an embedding in  $O(n^2)$  time.

## 4 $\mathfrak{P}$ -Width Is NP-Complete

Let  $\mathfrak{T}$  be the class of complete graphs (cliques). We first show that  $\mathfrak{T}$ -width is NP-complete.

**Theorem 6.**  *$\mathfrak{T}$ -Width is NP-complete.*

*Proof.* Let  $(G, \mathcal{N})$  with witness  $\mathcal{N} = \{\mathbb{N}_i \mid i = 1, \dots, k\}$  be a partitioned  $k$ -probe complete graph. Thus every nonedge of  $G$  has its endvertices in one of the independent sets  $\mathbb{N}_i$ . That is,  $\mathcal{N}$  forms a clique-cover of the edges of  $\bar{G}$ . This proves that a graph  $G$  has  $\mathfrak{T}$ -width at most  $k$  if and only if the edges of  $\bar{G}$  can be covered with  $k$  cliques. The problem to cover the edges of a graph by a minimum number of cliques is NP-complete [9]. □

**Theorem 7.**  *$\mathfrak{P}$ -Width is NP-complete.*

*Proof.* Let  $G = (V, E)$  be a graph with  $n$  vertices and  $m$  edges. Add a vertex  $\omega$  and make  $\omega$  adjacent to all vertices of  $G$ . Additionally, add a clique  $C$  of  $n^2$  vertices and make every vertex of  $C$  adjacent to every vertex of  $G$ . Let  $G'$  be the graph constructed in this way. Note that, when we add edges between nonadjacent vertices of  $V$  we obtain an embedding of  $G'$  into a trivially-perfect graph. We show that this is the only feasible embedding.

For each nonedge  $\{x, y\}$  in  $G$  we now have a collection of  $C_4$ 's using  $x, y$ , the vertices of the clique  $C$  and  $\omega$ . Assume that there is an embedding of  $G'$

into a trivially-perfect graph with  $x$  and  $y$  not adjacent. Then each vertex of  $C$  is adjacent to  $\omega$ . Thus each vertex of  $C$  must be in one of the independent sets  $\mathbb{N}_s$ ,  $s = 1, \dots, k$ , and no two are in the same independent set since  $C$  is a clique. Then  $k \geq n^2$  which is a contradiction, since making a clique of  $G$  creates a trivially-perfect embedding, and this needs at most  $\binom{n}{2} - m$  independent sets. Thus the only feasible embedding makes a clique of  $G$ . That is, the  $\mathfrak{P}$ -width of  $G'$  is the same as the  $\mathfrak{T}$ -width of  $G$ , and by Theorem 6 this is hard to compute.  $\square$

## 5 A Fixed-Parameter Algorithm to Compute $\mathfrak{P}$ -Width

In this section we show that there exists for each  $k$  an  $O(n^3)$  algorithm which checks if a graph  $G$  is a  $k$ -probe trivially-perfect graph.

Let  $(G, \mathcal{N})$  be a partitioned graph with a witness

$$\mathcal{N} = \{\mathbb{N}_i \mid i = 1, \dots, k\}$$

of  $k$  independent sets. We call the vertices of the independent sets *nonprobes* and we call the vertices which are not in any independent set *probes*. The  $k$ -label  $\alpha(x)$  of a vertex  $x$  is the 0/1-vector of length  $k$  with the  $i^{\text{th}}$  entry  $\alpha_i(x)$  equal to 1 if and only if  $x \in \mathbb{N}_i$ . We write  $\alpha(x) \leq \alpha(y)$  if  $\alpha_i(x) \leq \alpha_i(y)$  for all  $i = 1, \dots, k$ . We write  $\alpha(x) \perp \alpha(y)$  if there is no  $i$  with  $\alpha_i(x) = \alpha_i(y) = 1$ .

We use  $(G, \alpha)$  to denote a labeled graph. If  $X$  is a subset of vertices then we write  $\alpha(X)$  for the restriction of the labeling  $\alpha$  to the vertices of  $X$ . For a labeled subset  $X$  we write  $(X, \alpha)$ , instead of  $(G[X], \alpha)$  and instead of  $(G[X], \alpha(X))$ .

Consider the equivalence relation  $\equiv$  defined by  $x \equiv y$  if  $N(x) = N(y)$ . Denote the equivalence class of a vertex  $x$  by  $[x]$ . Define the partial order  $\preceq$  by:

$$[x] \preceq [y] \quad \text{if} \quad N(x) \subseteq N(y)$$

Likewise, we consider the equivalence relation  $\equiv'$  defined by  $x \equiv' y$  if  $N[x] = N[y]$ . The equivalence class of a vertex  $x$  under this relation is denoted by  $[x]$ . We consider the partial order defined by:

$$[x] \preceq [y] \quad \text{if} \quad N[x] \subseteq N[y]$$

**Lemma 1.** *A graph  $G$  is trivially perfect if and only if for every pair of adjacent vertices  $x$  and  $y$ , either  $[x] \preceq [y]$  or  $[y] \preceq [x]$ .*

*Proof.* Note that a graph  $G$  has two adjacent vertices with incomparable closed neighborhoods, if and only if  $G$  contains an induced  $P_4$  or  $C_4$ .  $\square$

**Definition 5 ([7]).** *A module  $M$  in a graph  $G = (V, E)$  is a set of vertices such that for every vertex  $y \notin M$  either*

1.  $N(y) \cap M = \emptyset$ , or
2.  $M \subseteq N(y)$ .

A module  $M$  is trivial if  $|M| \leq 1$  or if  $M = V$ .

*Remark 2.* Assume  $G = (V, E)$  is connected and trivially perfect. Let  $T$  be a rooted tree with node-set  $V$  such that two vertices are adjacent in  $G$  if and only if in  $T$ , one lies on the path from the root to the other. We refer to  $T$  as the *tree-model* of  $G$ . Note that for each node  $x$  in  $T$  the vertices in the subtree rooted at this node form a module  $X$  in  $G$ . The neighborhood  $N_G(X)$  of this module is the path in  $T$  from the root to the parent of  $x$ .

**Definition 6.** A probe module is a labeled set  $(X, \alpha)$  which induces a partitioned  $k$ -probe trivially-perfect graph with the additional property that for each vertex  $y \notin X$  there exists a  $y$ -extension, which is a label  $\alpha(y)$  such that either

- (a)  $N(y) \cap X = \emptyset$ , or
- (b) For each  $x \in X$   $[x \in N(y) \Leftrightarrow \alpha(y) \perp \alpha(x)]$ .

**Definition 7.** Two disjoint probe modules  $(X, \alpha)$  and  $(Y, \beta)$  are twins if  $(X + Y, \gamma)$  is a probe module with the inherited labeling  $\gamma(X) = \alpha$  and  $\gamma(Y) = \beta$ , such that either

- (i) no vertex of  $X$  is adjacent to any vertex of  $Y$ , or
- (ii) one of  $X$  and  $Y$  is a probe clique, and for every pair of vertices  $x \in X$  and  $y \in Y$ ,  $x$  and  $y$  are adjacent if and only if  $\gamma(x) \perp \gamma(y)$ .

**Definition 8.** Let  $(X, \alpha)$  be a probe module in a connected graph  $G$ . Then  $(X, \alpha)$  embeds if the labeling  $\alpha$  extends such that  $(G, \alpha)$  has an embedding  $H$  which has  $H[X]$  as a module. The graph  $H$  is an embedding of  $X$ .

**Definition 9.** Let  $(X, \alpha)$  be a probe module. A label-set of  $(X, \alpha)$  is a maximal subset of vertices of  $X$  with the same label. The characteristic  $\chi(X)$  is the set of labels for which the label-set is nonempty.

*Remark 3.* For ease of description we describe a trivially-perfect graph  $G$  also by its *cotree*. This representation is a binary tree where the leaves are labeled by the vertices of  $G$  and the internal nodes labeled by the join-operator  $\otimes$  or the union-operator  $\oplus$ . In case of a join-operator, the set of leaves in at least one of the two subtrees must induce a clique, since  $G$  is chordal.

**Lemma 2 (The Telescope Lemma).** Let  $(X, \alpha)$  and  $(Y, \alpha)$  be twin probe-modules. Assume that  $\chi(X) \supseteq \chi(Y)$ . Then  $(X + Y, \alpha)$  embeds if and only if  $(X, \alpha)$  embeds.

*Proof.* Let  $H$  be an embedding of  $(X + Y, \alpha)$ . Consider the cotree of  $H$ . Since  $H[X + Y]$  is a module in  $H$ , we may assume that  $(X + Y)$  forms a subtree. Let  $H'$  be the trivially-perfect graph obtained from  $H$  by replacing  $H[X + Y]$  by the union or join of  $H[X]$  and  $H[Y]$ , whichever is appropriate. Then  $H'$  is an embedding of  $(X, \alpha)$ ; we obtain a cotree by making  $X$  and  $Y$  twin-branches.

Now let  $H$  be an embedding of  $(X, \alpha)$ . Assume that a vertex  $z \notin (X + Y)$  is adjacent in  $G$  to a vertex  $y \in Y$ . Then  $\alpha(z) \perp \alpha(y)$  in any  $z$ -extension of

$(X + Y, \alpha)$ . There exists a vertex  $x \in X$  such that  $\alpha(x) = \alpha(y)$ , which implies that  $z$  is also adjacent to  $x$  in  $G$ .

Consider a cotree of  $H - Y$  such that  $X$  forms a subtree. Add  $H[Y]$  to  $H - Y$  as a twin-branch of  $H[X]$  and let  $H'$  be the graph that results. We prove that we can add edges between  $Y$  and  $N_H(X) - Y$ , such that  $X + Y$  becomes a module in  $H'$ .

Let  $z \notin (X + Y)$ . If  $z$  is not adjacent to any vertex of  $X$  in  $H$  then  $z$  is also not adjacent to any vertex in  $Y$ . Assume  $z$  is adjacent to  $X$  in  $H$ . Let  $y \in Y$ , and assume that  $z$  is not adjacent to  $y$  in  $G$ . We prove that  $\alpha_i(z) = \alpha_i(y) = 1$  for some entry  $i$ . There exists a vertex  $x \in X$  such that  $\alpha(x) = \alpha(y)$ . If  $z$  is not adjacent to  $x$  in  $G$ ,  $\alpha_i(y) = \alpha_i(x) = \alpha_i(z) = 1$  for some entry  $i$ . Assume  $z$  is adjacent to  $x$  in  $G$ . Then  $\alpha(z) \perp \alpha(x)$  in any  $z$ -extension of  $(X + Y, \alpha)$ . Since  $(X + Y, \alpha)$  is a probe module and  $\alpha(x) = \alpha(y) \perp \alpha(z)$ ,  $z$  is adjacent to  $y$ , which is a contradiction.  $\square$

**Definition 10.** A true – or false twinset is a set of vertices such that every pair is a true – or false twin, respectively.<sup>2</sup> A  $k$ -twinset is either a false twinset with at least 3 vertices or a true twinset with at least  $k + 2$  vertices.

**Lemma 3 (The Twinset Lemma).** Let  $S$  be  $k$ -twinset. Then  $G$  has  $\mathfrak{TP}$ -width at most  $k$  if and only if  $G - x$  has  $\mathfrak{TP}$ -width at most  $k$  for any  $x \in S$ .

*Proof.* Assume that  $G$  has a false twinset  $\{x, y, z\}$ . Assume that  $G - x$  has an embedding  $H$ . If one of  $y$  and  $z$  is a nonprobe in  $H$ , then we can make a copy for  $x$  as a true twin. Note that creating a true twin does not introduce a  $P_4$  or  $C_4$  so the new embedding is also trivially perfect. Now assume that both  $y$  and  $z$  are probes. Then their neighborhood in  $H$  must induce a clique. We may now add  $x$  as a false twin of  $y$  and  $z$  in  $H$ . Note that also in this case no  $P_4$  or  $C_4$  is introduced.

Assume that  $G$  has a true twinset  $S$  with  $k + 2$  vertices. Let  $x \in S$  and let  $H$  be an embedding of  $G - x$ . Since  $S - x$  is a clique there exists an ordering of the vertices of  $S - x$  such that for every pair  $a, b \in S - x$ ,  $N_H[a] \subseteq N_H[b]$  or  $N_H[b] \subseteq N_H[a]$ . Let  $y$  be the smallest vertex in this ordering. If  $y$  has a neighbor in  $H$  which is not a neighbor in  $G$ , then this is a new neighbor of all the vertices in  $S - x$ . This is a contradiction, since  $S - x$  is a clique with  $k + 1$  vertices, and creating a common neighbor for  $S - x$  would require  $k + 1$  independent sets.  $\square$

**Definition 11.** A  $k$ -witness  $\mathcal{N}$  is well-linked if for every  $\mathbb{N}_i \in \mathcal{N}$ , every vertex  $x \notin \mathbb{N}_i$  has a neighbor in  $\mathbb{N}_i$ .

**Lemma 4.** Every  $k$ -probe trivially-perfect graph has a witness with  $k$  independent sets which is well-linked.

*Proof.* Starting with any witness, repeatedly add a vertex  $x$  to an independent set  $\mathbb{N}_i$  if it has no neighbor in that set.  $\square$

<sup>2</sup> A true twin is a pair of vertices  $x$  and  $y$  with  $N[x] = N[y]$ . A false twin is a pair of vertices  $x$  and  $y$  with  $N(x) = N(y)$ .

**Lemma 5 (The Well-Linkedness Lemma).** *Let  $(G, \mathcal{N})$  be a  $k$ -probe trivially-perfect graph with a well-linked witness  $\mathcal{N}$  and corresponding labeling  $\alpha$ . Let  $H$  be an embedding. For every nonadjacent pair  $x$  and  $y$  in  $G$  with  $N_H(x) \subseteq N_H[y]$ ,*

$$(x) \preceq (y) \iff \alpha(x) \geq \alpha(y)$$

*Proof.* Assume  $\alpha(x) \geq \alpha(y)$ . Let  $z \in N_G(x)$ . Then  $z \in N_H[y]$ . Since  $x$  and  $y$  are not adjacent,  $z \neq y$ . Thus  $z \in N_H(y)$ . If  $z \notin N_G(y)$ , then there exists an  $i$  with  $\{z, y\} \subseteq \mathbb{N}_i$ . Now  $\alpha(x) \geq \alpha(y)$  implies that also  $x \in \mathbb{N}_i$ , which contradicts that  $z$  is adjacent to  $x$ . Hence  $(x) \preceq (y)$ .

Assume  $(x) \not\preceq (y)$ , that is,  $N_G(x) \not\subseteq N_G(y)$ . *A fortiori*,  $x$  and  $y$  are not adjacent. Assume  $\neg(\alpha(x) \geq \alpha(y))$ . Then there exists an  $i$  with  $y \in \mathbb{N}_i$  and  $x \notin \mathbb{N}_i$ . Since  $\mathcal{N}$  is well-linked, there exists a vertex  $z \in N_G(x) \cap \mathbb{N}_i$ . Since  $(x) \not\preceq (y)$ ,  $z \in N_G(y)$ , contradicting that  $z$  and  $y$  are both in  $\mathbb{N}_i$ .  $\square$

**Definition 12.** *Let  $(X, \alpha)$  be a probe module. A vertex  $\gamma \in N(X)$  is  $X$ -minimal if there exists no  $y \in N(X)$  with  $(y) \neq (\gamma)$  and  $(y) \preceq (\gamma)$  and also no  $z \in N(X)$  with  $[z] \neq [\gamma]$  and  $[z] \preceq [\gamma]$ .*

*Remark 4.* Notice that  $X$ -minimality of a vertex is independent of the actual labeling of the probe module  $(X, \alpha)$ .

**Lemma 6.** *Assume  $G$  has no  $k$ -twinset. Assume that  $(X, \alpha)$  embeds as a branch in the tree-model of a well-linked embedding. Let  $\mathcal{Y}$  be the set of  $X$ -minimal vertices. Then  $|\mathcal{Y}| \leq 2^{k+1} + k - 1$ .*

*Proof.* Consider a well-linked embedding  $H$ . Let  $T$  be a tree-model of  $H$ . Consider the path  $M$  from the root to the ancestor of  $X$  in  $T$  and let  $M_0, M_1, \dots$  be a partition of  $M$  into modules. By minimality of the embedding we may assume that each vertex of  $M_i$  has a neighbor in every subtree of  $M_i$ . Assume they are ordered such that  $N_H[x_i] \subset N_H[x_{i+1}]$  for each  $x_i \in M_i$  and  $x_{i+1} \in M_{i+1}$ , for  $i = 0, 1, \dots$

Notice that each label-set of each  $M_s$  is a module in  $G$ . Since there is no  $k$ -twinset, each label-set of nonprobes has at most 2 vertices and each label-set of probes has at most  $k + 1$  vertices. Thus

$$|M_s| \leq 2(2^k - 1) + (k + 1) = 2^{k+1} + k - 1$$

By the Well-Linkedness Lemma, a vertex  $x \in M_s$  is minimal if it has a label  $\alpha(x)$  such that all other label-sets  $\alpha' \geq \alpha(x)$  in  $M_0, \dots, M_s$  are empty. It follows that there are at most  $\sum_{i=0}^k \binom{k}{i} = 2^k$  label-sets of minimal vertices, at most  $2^k - 1$  of minimal nonprobes, each containing at most 2 elements, and at most one label-set of minimal probes, containing at most  $k + 1$  elements. Thus the number of minimal elements is bounded by  $2^{k+1} + k - 1$ .  $\square$

**Lemma 7.** *Assume  $G$  is a connected  $k$ -probe trivially-perfect graph without  $k$ -twinset. Let  $(X, \alpha)$  be a probe module that embeds as a branch into a well-linked embedding  $H$ . Let  $T$  be a tree-model of  $H$  and let  $M_0$  be the lowest set of ancestors of  $X$  in  $T$  that forms a module in  $H$ . There exists a set  $\Omega$ , of size  $|\Omega| \leq 2^{2(k+1)}$  such that  $M_0 \subseteq \Omega$ . This set  $\Omega$  can be computed in linear time.*

*Proof.* Start with  $\Omega = \emptyset$ . Repeatedly compute the set of  $X$ -minimal vertices in  $G$ , add them to  $\Omega$ , and delete them from the graph. After at most  $2^k$  repetitions, each label-set of  $M_0$  is contained in  $\Omega$ . Since each set of maximal elements has at most  $2^{k+1} + k - 1$  vertices,

$$|\Omega| \leq 2^k(2^{k+1} + k - 1) \leq 2^{2k+1} + 2^{2k} \leq 2^{2(k+1)} \quad \square$$

**Definition 13.** A pattern is a cotree of a  $k$ -labeled trivially perfect graph such that for every internal node the characteristics of the two subtrees are incomparable.

*Remark 5.* Consider a cotree of an embedding of a labeled graph  $(G, \alpha)$ . By Lemma 2, we may repeatedly prune branches for which the characteristic is contained in the characteristic of the other branch. The result is a pattern.

**Lemma 8.** There are  $O(2^{(k+3)2^{2^k}})$  non-isomorphic patterns.

*Proof.* The characteristic of every internal node is the union of the characteristics of its children. This union is larger than the two constituent sets since those are incomparable. A binary tree with depth at most  $2^k$  has at most  $2^{2^k} - 1$  internal nodes. The number of binary trees with  $t$  internal nodes can be bounded by the Catalan number  $C_t = \binom{2t}{t} \frac{1}{t+1}$ . Thus the number of cotrees with  $t + 1$  leaves is bounded by  $2^t C_t \sim \frac{2^{3t}}{t^{3/2}\sqrt{\pi}}$ . There are at most  $2^{k(t+1)}$  labelings for the leaves. Thus the number of patterns is bounded by  $c2^{kc} \frac{2^{3c}}{c^{3/2}\sqrt{\pi}}$ , where  $c = 2^{2^k}$ .  $\square$

*Remark 6.* Similar to Lemma 6 it can be shown that the number of feasible, incomparable induced patterns of branches incident with an internal node of the cotree is bounded by a constant. Actually, this proves a well-quasi-ordering, which implies a finite set of forbidden induced subgraphs for  $\mathfrak{FB}$ -width  $\leq k$ . We elaborate on this in the full version of this paper.

**Theorem 8.** For each  $k$ , there exists an  $O(n^3)$ -time algorithm for the recognition of  $k$ -probe trivially-perfect graphs.

*Proof.* Consider a partition  $(M_0, M_1, \dots, M_s)$  of the vertices into probe modules. Initially, each module consists of a single vertex. For each probe module we keep the possible embeddings, either as a clique-module or as a branch, as a list of patterns. The algorithm tries to merge modules into new modules. By Lemma 7 there are only a constant number of possible extensions for each module. Assume that a probe module  $(X, \alpha)$  unions with some other probe modules. If there is a module  $(Y, \beta)$  with  $\chi(Y) \subseteq \chi(X)$  then  $(Y, \beta)$  merges together with  $(X, \alpha)$  for those labelings. There are at most  $2^k$  module extensions in which the characteristic enlarges. A suitable merge of two probe modules can be found in  $O(n^2)$  time.  $\square$

## 6 Conclusion

So far, we have limited our research to classes of graphs that have bounded rankwidth. For classes such as threshold graphs and cographs we were able to show that the width parameter is fixed-parameter tractable. One of the classes for which this is still open is the class of distance-hereditary graphs.

## Acknowledgement

Ton Kloks is currently a guest of the Department of Computer Science and Information Engineering of National Chung Cheng University. He gratefully acknowledges the funding for this research by the National Science Council of Taiwan and the hospitality of the department. This research is supported by the National Science Council of Taiwan under grant NSC 97-2221-E-194-055.

## References

1. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph classes: A survey. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia (1999)
2. Chang, M.-S., Hsieh, S.-Y., Chen, G.-H.: Dynamic programming on distance-hereditary graphs. In: Leong, H.-V., Jain, S., Imai, H. (eds.) ISAAC 1997. LNCS, vol. 1350, pp. 344–353. Springer, Heidelberg (1997)
3. Courcelle, B., Oum, S.: Vertex minors, monadic second-order logic, and a conjecture by Seese. *Journal of Combinatorial Theory, Series B* 97, 91–126 (2007)
4. Golombic, M.C.: Trivially perfect graphs. *Discrete Mathematics* 24, 105–107 (1978)
5. Howorka, E.: A characterization of distance-hereditary graphs. *Quarterly Journal of Mathematics* 28, 417–420 (1977)
6. Kruskal, J.: Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture. *Transactions of the American Mathematical Society* 95, 210–225 (1960)
7. Möhring, R., Radermacher, F.: Substitution decomposition for discrete structures and connections with combinatorial optimization. In: Burkard, R., Cuninghame-Green, R., Zimmermann, U. (eds.) *Algebraic and Combinatorial Methods in Operations Research*. *Annals of Discrete Mathematics*, vol. 19. Mathematics studies, vol. 95, pp. 257–355. Elsevier/North-Holland, Amsterdam (1984)
8. McMorris, F.R., Wang, C., Zhang, P.: On probe interval graphs. *Discrete Applied Mathematics* 88, 315–324 (1998)
9. Orlin, J.: Contentment in graph theory: covering graphs with cliques. *Indagationes Mathematicae* 39, 406–424 (1977)
10. Oum, S.: Graphs of bounded rank-width, PhD Thesis, Princeton University (2005)
11. Pouzet, M.: Applications of well-quasi-ordering and better-quasi-ordering. In: Rivai, I. (ed.) *Graphs and Order*, pp. 503–519. D. Reidel Publishing Company (1985)
12. Thomassé, T.: On better-quasi-ordering countable series-parallel orders. *Transactions of the American Mathematical Society* 352, 2491–2505 (2000)
13. Wolk, E.S.: A note on The comparability graph of a tree. *Proceedings American Mathematical Society* 16, 17–20 (1965)
14. Zhang, P., Schon, E.A., Fischer, S.G., Cayanis, E., Weiss, J., Kistler, S., Bourne, P.E.: An algorithm based on graph theory for the assembly of contigs in physical mapping of DNA. In: *CABIOS*, vol. 10, pp. 309–317 (1994)

# Lightweight Parameterized Suffix Array Construction

Tomohiro I<sup>1</sup>, Satoshi Deguchi<sup>1</sup>, Hideo Bannai<sup>1</sup>, Shunsuke Inenaga<sup>2</sup>,  
and Masayuki Takeda<sup>1</sup>

<sup>1</sup> Department of Informatics, Kyushu University

<sup>2</sup> Faculty of Information Science and Electrical Engineering, Kyushu University  
744 Motoooka, Nishiku, Fukuoka, 819-0395 Japan  
{tomohiro.i,satoshi.deguchi,bannai,takeda}@i.kyushu-u.ac.jp,  
inenaga@c.csce.kyushu-u.ac.jp

**Abstract.** We present a first algorithm for direct construction of parameterized suffix arrays and parameterized longest common prefix arrays for non-binary strings. Experimental results show that our algorithm is much faster than naïve methods.

## 1 Introduction

Parameterized pattern matching is a form of pattern matching first introduced by Baker [1], that allows for interchange in the alphabet. More formally, let  $\Pi$  be the set of *parameter symbols* and  $\Sigma$  be the set of *constant symbols*. Strings over  $\Pi \cup \Sigma$  are called *parameterized strings* (*p-strings*). Two p-strings of the same length are said to parameterized match (p-match) if one string can be transformed into the other by using a bijection on  $\Sigma \cup \Pi$ , with the restriction that the bijection must be the identity on the constant symbols of  $\Sigma$ . In other words, the bijection maps any  $a \in \Sigma$  to  $a$  itself, while symbols of  $\Pi$  can be interchanged. Examples of applications of parameterized pattern matching are software maintenance [12], plagiarism detection [3], and RNA structural matching [4].

For the standard pattern matching problem, there exist several data structures that can be obtained by preprocessing the text string so that pattern matching can be performed efficiently. The most famous are the suffix tree [5] and suffix array [6]. These data structures can both be constructed directly in linear time [5,7,8,9,10,11,12], independent of the alphabet size. Most operations on a suffix tree can be efficiently simulated with the suffix array and several other auxiliary arrays, including an array containing the lengths of longest common prefixes of the suffixes (LCP array), composing an *enhanced* suffix array [13,14]. The array representation has become more preferable as it requires less memory, and is generally faster to construct and work on, due to memory access locality.

For p-string pattern matching, Baker [2] introduced the *parameterized suffix tree* (*p-suffix tree*), which is similar in concept to the suffix tree. Baker gave an  $O(n(\pi + \log(\pi + \sigma)))$  time algorithm to construct the p-suffix tree for a given text string, where  $n$  is the text length,  $\pi = |\Pi|$  and  $\sigma = |\Sigma|$ . Kosaraju [15]



proposed an algorithm to construct p-suffix trees in  $O(n(\log \pi + \log \sigma))$  time. Both algorithms are based on McCreight's construction algorithm for standard suffix trees [7]. Shibuya [4] gave an on-line construction algorithm working in  $O(n(\log \pi + \log \sigma))$  time, which is based on Ukkonen's construction algorithm for standard suffix trees [8]. Given a pattern  $p$  of length  $m$ , we can compute the set  $Pocc$  of all positions of  $t$  where the corresponding substring of  $t$  p-matches pattern  $p$  in  $O(m \log(\pi + \sigma) + |Pocc|)$  time, using the p-suffix tree of a text  $t$ .

Concerning the array representation, parameterized suffix arrays were considered by Deguchi *et al.* [16]. The parameterized pattern matching problem can be solved in  $O(m \log n + |Pocc|)$  time with a simple binary search, or  $O(m + \log n + |Pocc|)$  with a binary search utilizing PLCP information, or  $O(m \log(\pi + \sigma) + |Pocc|)$  time if we consider *enhanced* p-suffix arrays. As with the case of standard suffix trees and arrays, the array representation is superior in memory usage and memory access locality. Deguchi *et al.* presented a linear time algorithm for direct construction of the parameterized suffix array and parameterized LCP (PLCP) array for binary strings. To the best of our knowledge, no efficient algorithm for direct construction of a parameterized suffix array and PLCP array for non-binary strings exist, and the best theoretical worst-case time bound is  $O(n^2)$ , using a standard radix sort on strings.

This paper presents a new algorithm for efficient construction of p-suffix arrays and PLCP arrays for non-binary strings. For p-suffix array construction, our algorithm combines any string sorting algorithm with linear time pre- and post processing. Though we are unable to reduce the theoretical worst case time bound, our algorithm considerably reduces the number of suffixes to be sorted using the string sorting algorithm, hence greatly reducing the running time. For the PLCP array, we modify the linear time LCP array construction algorithm of [17], so that it can be used for p-strings. However, due to properties of p-strings, it is still open if the theoretical time bound of our algorithm is linear. Computational experiments show both our algorithms are generally much faster than naïve approaches for various texts.

## 2 Preliminaries

Let  $\Sigma$  and  $\Pi$  be two disjoint finite sets of *constant symbols* and *parameter symbols*, respectively. An element of  $(\Sigma \cup \Pi)^*$  is called a *p-string*. The length of any p-string  $s$  is the total number of constant and parameter symbols in  $s$  and is denoted by  $|s|$ . For any p-string  $s$  of length  $n$ , the  $i$ -th symbol is denoted by  $s[i]$  for each  $1 \leq i \leq n$ , and the *substring* starting at position  $i$  and ending at position  $j$  is denoted by  $s[i : j]$  for  $1 \leq i \leq j \leq n$ . In particular,  $s[1 : j]$  and  $s[i : n]$  denote the *prefix* of length  $j$  and the *suffix* of length  $n - i + 1$  of  $s$ , respectively. For any two strings  $s$  and  $t$ ,  $lcp(s, t)$  denotes the length of the *longest common prefix* of  $s$  and  $t$ .

**Definition 1 (Parameterized Matching).** *Any two p-strings  $s$  and  $t$  of the same length  $m$  are said to parameterized match (p-match) iff one of the following conditions hold for every  $1 \leq i \leq m$ :*

1.  $s[i] = t[i] \in \Sigma$ ,
2.  $s[i], t[i] \in \Pi$ ,  $s[i] \neq s[j]$  and  $t[i] \neq t[j]$  for any  $1 \leq j < i$ ,
3.  $s[i], t[i] \in \Pi$ ,  $s[i] = s[i - k]$  for any  $1 \leq k < i$  iff  $t[i] = t[i - k]$ .

We write  $s \simeq t$  when  $s$  and  $t$  p-match.

For example, let  $\Pi = \{a, b, c\}$ ,  $\Sigma = \{X, Y\}$ ,  $s = \text{abaXabY}$  and  $t = \text{bcbXbcY}$ . Observe that  $s \simeq t$ .

Let  $\mathcal{N}$  be the set of non-negative integers. For any non-negative integers  $i \leq j \in \mathcal{N}$ , let  $[i, j] = \{i, i + 1, \dots, j\} \subset \mathcal{N}$ .

**Definition 2.** We define  $pv : (\Sigma \cup \Pi)^* \rightarrow (\Sigma \cup \mathcal{N})^*$  to be the function such that for any p-string  $s$  of length  $n$ ,  $pv(s) = u$  where, for  $1 \leq i \leq n$ ,

$$u[i] = \begin{cases} s[i] & \text{if } s[i] \in \Sigma, \\ 0 & \text{if } s[i] \in \Pi \text{ and } s[i] \neq s[j] \text{ for any } 1 \leq j < i, \\ i - k & \text{if } s[i] \in \Pi \text{ and } k = \max\{j \mid s[i] = s[j], 1 \leq j < i\}. \end{cases}$$

In the running example,  $pv(s) = 002X24Y$  with  $s = \text{abaXabY}$ .

The following proposition is clear from Definition 2

**Proposition 1.** For any p-string  $s$  of length  $n$ , it holds for any  $1 \leq i \leq j \leq n$  that  $pv(s[i : j]) = v[1 : j - i + 1]$ , where  $v = pv(s[1 : n])$ .

**Proposition 2** ([2]). For any two p-strings  $s$  and  $t$  of the same length,  $s \simeq t$  iff  $pv(s) = pv(t)$ .

In the running example, we then have  $s \simeq t$  and  $pv(s) = pv(t) = 002X24Y$ .

We also define the dual of the  $pv$  function, as follows:

**Definition 3.** We define  $fw : (\Sigma \cup \Pi)^* \rightarrow (\Sigma \cup \mathcal{N} \cup \{\infty\})^*$  to be the function such that for any p-string  $s$  of length  $n$ ,  $fw(s) = w$  where, for  $1 \leq i \leq n$ ,

$$w[i] = \begin{cases} s[i] & \text{if } s[i] \in \Sigma, \\ \infty & \text{if } s[i] \in \Pi \text{ and } s[i] \neq s[j] \text{ for any } i < j \leq n, \\ k - i & \text{if } s[i] \in \Pi \text{ and } k = \min\{j \mid s[i] = s[j], i < j \leq n\}. \end{cases}$$

Here,  $\infty$  denotes a value for which  $i < \infty$  for any  $i \in \mathcal{N}$

In the running example,  $fw(s) = 242X\infty\infty Y$  with  $s = \text{abaXabY}$ .

**Proposition 3.** For any p-string  $s$  of length  $n$ , it holds for any  $1 \leq i \leq n$  that  $fw(s[i : n]) = w[i : n]$ , where  $w = fw(s)$ .

For any p-string  $s$  of length  $n$ ,  $pv(s)$  and  $fw(s)$  can be computed in  $O(n)$  time with extra  $O(\pi)$  space, using a table of size  $\pi$  recording the last position of each parameter symbol in the left-to-right (resp. right-to-left) scanning of  $s$  [2].

---

<sup>1</sup> In practice,  $n$  can be used in place of  $\infty$  as long as we are considering a single p-string of length  $n$ , and its substrings.

*Problem 1 (P-matching problem).* Given any two p-strings  $t$  and  $p$  of length  $n$  and  $m$  respectively,  $n \geq m$ , compute  $Pocc(t, p) = \{i \mid t[i : i + m - 1] \simeq p\}$ .

Proposition 2 implies that  $Pocc(t, p) = \{i \mid pv(p) = pv(t[i : i + m - 1])\}$ .

**Lemma 1 ([18]).** *Problem 1 on alphabet  $\Sigma \cup \Pi$  is reducible in linear time to Problem 1 on alphabet  $\Pi$ .*

Due to the above lemma, in the remainder of the paper, we consider only p-strings in  $\Pi^*$ . Then, note that for any p-string  $s$  of length  $n$ ,  $pv(s) \in \{[0, n - 1]\}^n$  and  $fw(s) \in \{[1, n - 1] \cup \{\infty\}\}^n$ . We also see that if  $pv(s)[i] > 0$  then  $fw(s)[i - pv(s)[i]] = pv(s)[i]$ . Similarly, if  $fw(s)[i] < n$  then  $pv(s)[i + fw(s)[i]] = fw(s)[i]$ .

Let  $\preceq$  denote the standard lexicographic ordering on strings of an integer alphabet. To simplify discussions on the end of strings, we assume that for any p-string  $s$ ,  $pv(s)[i] = -1$  for any  $i > |s|$ .

In this paper, we will consider construction of the following data structures.

**Definition 4 (P-suffix Array).** *For any p-string  $s \in \Pi^n$  of length  $n$ , its p-suffix array  $PSA_s$  is an array of length  $n$  such that  $PSA_s[i] = j$ , where  $pv(s[j : n])$  is the lexicographically  $i$ -th element of  $\{pv(s[k : n]) \mid 1 \leq k \leq n\}$ .*

**Definition 5 (PLCP Array).** *For any p-string  $s \in \Pi^n$  of length  $n$ , its PLCP array  $PLCP_s$  is an array of length  $n$  such that*

$$PLCP_s[i] = \begin{cases} -1 & \text{if } i = 1, \\ lcp(pv(s[PSA[i - 1] : n]), pv(s[PSA[i] : n])) & \text{if } 2 \leq i \leq n. \end{cases}$$

We abbreviate  $PLCP_s$  as  $PLCP$  when clear from the context. The following is a useful auxiliary array that we will use for the construction of  $PLCP$ .

**Definition 6 (Rank Array).** *For any p-string  $s \in \Pi^n$  of length  $n$ , its rank array  $rank_s$  is an array of length  $n$  such that  $rank_s[PSA_s[i]] = i$ , for any  $1 \leq i \leq n$ .*

We abbreviate  $rank_s$  as  $rank$  when clear from the context. Note that  $rank_s[i]$  can be computed in linear time from  $PSA_s$  for all  $i$  where  $1 \leq i \leq n$ .

Table 1 shows an example of a p-suffix array, PLCP array and rank array for the string  $s = \text{babbcacaabcb}$ .

The  $PSA$ ,  $PLCP$ , and  $rank$  arrays can naturally be used in similar ways as the suffix, LCP, and rank arrays for standard strings.  $PSA$  and  $PLCP$  arrays can be constructed by a linear time traversal on the p-suffix tree. However, unlike standard suffix arrays and lcp arrays, direct linear time algorithms that do not construct the tree as an intermediate data structure are not known, except for the case of binary alphabets [16].

The main difficulty in developing efficient algorithms for constructing  $PSA$  and  $PLCP$  is that for any p-string  $s$ , a suffix  $pv(s)[i : n]$  of  $pv(s)$  is not necessarily equal to  $pv(s[i : n])$  of the suffix  $s[i : n]$ . As an important consequence, for any p-strings  $s, t$  with  $lcp(pv(s), pv(t)) > 0$ ,  $pv(s) \preceq pv(t)$  does not necessarily imply  $pv(s[2 : |s|]) \preceq pv(t[2 : |t|])$ , which is a property essential for efficient construction algorithms in the standard case.

**Table 1.**  $PSA_t$  and  $PLCP_t$  for p-string  $s = \text{babbcacaabcb}$

$i$	$PSA[i]$	$PLCP[i]$	$\text{rank}[i]$	1 2 3 4 5 6 7 8 9 10 11 12	$\text{zlen}[PSA[i]]$	type
1	12	-1	9	0	1	A
2	11	1	6	0 0	2	A
3	9	2	12	0 0 0 2	3	B
4	4	4	4	0 0 0 2 2 1 6 4 2	3	B
5	7	2	10	0 0 1 0 4 2	2	B
6	2	6	8	0 0 1 0 4 2 2 1 6 4 2	2	B
7	10	2	5	0 0 2	2	C
8	6	3	11	0 0 2 1 0 4 2	2	C
9	1	7	3	0 0 2 1 0 4 2 2 1 6 4 2	2	C
10	5	3	7	0 0 2 2 1 0 4 2	2	C
11	8	1	2	0 1 0 0 2	1	C
12	3	5	1	0 1 0 0 2 2 1 6 4 2	1	C

### 3 Algorithm

For our lightweight algorithm, we will use the number of contiguous zeroes in the prefix of each suffix  $pv(s[i : n])$ , to sort them coarsely.

**Definition 7.** For any p-string  $s$  of length  $n$ , we define  $\text{zlen}_s[i]$  as the length of contiguous zeroes in the prefix of  $pv(s[i : n])$  for any  $1 \leq i \leq n$ . That is,

$$\text{zlen}_s[i] = \max\{j \mid pv(s[i : i + j - 1]) = 0^j, 1 \leq j \leq n - i + 1\}.$$

We abbreviate  $\text{zlen}_s$  as  $\text{zlen}$  when clear from the context. Note that  $\text{zlen}_s[i]$  can be computed in amortized linear time for all  $i$  where  $1 \leq i \leq n$ .

#### 3.1 Constructing P-Suffix Array

$\text{zlen}$  divides the set of suffixes  $s[i : n] (1 \leq i \leq n)$  of p-string  $s$  into 3 types.

- Type A: those consisting of zeroes only, that is  $\text{zlen}_s[i] = |s| - i + 1$ .
- Type B: those with  $\text{zlen}_s[i] < |s| - i + 1$  and  $\text{zlen}_s[i] > pv(s[i : n])[\text{zlen}_s[i] + 1]$
- Type C: those with  $\text{zlen}_s[i] < |s| - i + 1$  and  $\text{zlen}_s[i] = pv(s[i : n])[\text{zlen}_s[i] + 1]$ .

Our algorithm consists of the following steps.

1. Calculate  $\text{zlen}_s[i]$  for all  $i$  in linear time and determine its type.
2. Determine the positions of type A suffixes using  $\text{zlen}_s[i]$ .
3. Coarsely sort all type B and C suffixes in linear time by radix sort using  $\text{zlen}_s[i]$  and the first non-zero value  $pv(s[i : n])[\text{zlen}_s[i] + 1]$  in  $pv(s[i : n])$ .
4. Determine the positions of the type B suffixes using any sorting algorithm.
5. Determine the positions of the remaining type C suffixes in linear time.

Each step can be calculated in linear time except for Step 4 which depends on the underlying sort algorithm. We will describe the details of each step below.

First, we determine the positions of the type A suffixes (Step 2). Let  $A_s$  denote the number of type A suffixes of  $s$ . For the type A suffixes, it is obvious that  $PSA_s[1 : A_s] = n, n - 1, \dots, n - A_s + 1$ .

For the type B and C suffixes (Step 3), they are divided into *blocks* of suffixes that have the same  $zlen$  value and first non-zero value. We can determine the order of these blocks in linear time and space by radix sort, that is, bucket sort first in ascending order of the first non-zero value and then in descending order of the  $zlen$  value. For any suffix  $i$ , the first non-zero value is not greater than  $zlen[i]$ . Therefore, we can also do this operation by a single bucket sort in descending order of  $\sum_{t=1}^{zlen[i]} t - pv(s[i : n])[zlen[i] + 1] = \frac{zlen[i](zlen[i]+1)}{2} - pv(s[i : n])[zlen[i] + 1]$ . This alternative method works in linear time and space, as long as  $\frac{z(z+1)}{2} = O(n)$ , where  $z = \max\{zlen_s[i] \mid 1 \leq i \leq n - A_s\}$ .

Step 4 is then conducted using any string sorting algorithm within each type B block. Note that within a block, it suffices to see the order of  $pv(s[i : n])[zlen_s[i] + 2 : n]$  since they will have a common prefix of length  $zlen_s[i] + 1$ .

Let  $\overline{PSA}_s$  denote the intermediate array obtained just after processing Step 4, that is, only the type C suffixes are not in position yet. The next lemmas describe the key properties for sorting these remaining suffixes in linear time (Step 5).

**Lemma 2.** *Let  $s$  be a  $p$ -string of length  $n$ . For any  $i, j$  ( $1 \leq i, j \leq n - 1$ ), if  $fw(s)[i] \geq fw(s)[j]$  and  $pv(s[i + 1 : n]) \prec pv(s[j + 1 : n])$ , then  $pv(s[i : n]) \prec pv(s[j : n])$ .*

*Proof.* Assume on the contrary that  $pv(s[i : n]) \succ pv(s[j : n])$ . Let  $l$  be  $lcp(pv(s[i : n]), pv(s[j : n]))$ . Since  $pv(s[i : n]) \succ pv(s[j : n])$ , then

$$pv(s[i : n])[1 : l] = pv(s[j : n])[1 : l], \quad pv(s[i : n])[l + 1] > pv(s[j : n])[l + 1].$$

(i)  $fw(s)[j] > l$ . Since  $fw(s)[i] \geq fw(s)[j]$ , then  $pv(s[i + 1 : n])[1 : l - 1] = pv(s[j + 1 : n])[1 : l - 1]$ ,  $pv(s[i + 1 : n])[l] > pv(s[j + 1 : n])[l]$ . We get  $pv(s[i + 1 : n]) \succ pv(s[j + 1 : n])$ , a contradiction.

(ii)  $fw(s)[j] = l$ . By assumption,  $pv(s[i : n])[l + 1] > pv(s[j : n])[l + 1] = fw(s)[j] = l$ . However, by Definition 2,  $pv(s[i : n])[l + 1] \leq l$ , a contradiction.

(iii)  $fw(s)[j] < l$ . Since  $fw(s)[i] = fw(s)[j]$ , then  $pv(s[i + 1 : n])[1 : l - 1] = pv(s[j + 1 : n])[1 : l - 1]$ ,  $pv(s[i + 1 : n])[l] > pv(s[j + 1 : n])[l]$ . We get  $pv(s[i + 1 : n]) \succ pv(s[j + 1 : n])$ , a contradiction.  $\square$

**Lemma 3.** *Let  $s$  be a  $p$ -string of length  $n$ . For any  $i$  ( $1 \leq i \leq n - 1$ ), if  $pv(s[i : n])$  is a type C suffix, then  $pv(s[i + 1 : n]) \prec pv(s[i : n])$ .*

*Proof.* Assume on the contrary that  $pv(s[i + 1 : n]) \succ pv(s[i : n])$ . Since  $pv(s[i : n])$  is a type C suffix,  $zlen[i + 1] \geq zlen[i]$ . Then for some  $k$  ( $zlen[i] + 2 \leq k \leq n - i + 1$ ),  $pv(s[i + 1 : n])[1 : k - 1] = pv(s[i : n])[1 : k - 1]$  and  $pv(s[i + 1 : n])[k] > pv(s[i : n])[k]$ . In addition, it follows from  $pv(s[i : n])[zlen[i] + 1] = zlen[i]$  that  $pv(s[i + 1 : n])[zlen[i] + 1 : n] = pv(s[i : n])[zlen[i] + 2 : n - 1]$ . Here,

$$\begin{aligned} pv(s[i : n])[k] &= pv(s[i + 1 : n])[k - 1] = pv(s[i : n])[k - 1] \\ &= pv(s[i + 1 : n])[k - 2] = pv(s[i : n])[k - 2] \\ &= \dots = pv(s[i + 1 : n])[zlen[i] + 1] = pv(s[i : n])[zlen[i] + 1] = zlen[i]. \end{aligned}$$

```

// Initialize:
// head[i] = k: min position in type C block with k = zlen[PSA[j]]
// block[i] = if(PSA[i]-1 is type C) then zlen[PSA[i]-1] else 0
for (i = 1; i < n; i++) {
    j = block[psa[i]];
    if (j != 0) {
        // s[psa[i]-1:n] is type C
        psa[head[j]] = psa[i] - 1; // determine its position in block j
        head[j]++; // increment head position of block
    }
}
}

```

**Fig. 1.** Algorithm for sorting type C suffixes in linear time. Note that the initialization of head and block arrays can be done in linear time.

Hence  $pv(s[i+1:n])[1:k-1] = 0^{zlen[i]}zlen[i]^{k-1-zlen[i]}$ . This implies that  $pv(s[i+1:n])[k] \leq zlen[i] = pv(s[i:n])[k]$ , a contradiction.  $\square$

**Theorem 1.** *Let  $s$  be a  $p$ -string of length  $n$ . If the positions of type A and B suffixes are determined, the positions of the remaining type C suffixes can be determined in linear time.*

*Proof.* From Lemma 2, if  $pv(s[i+1:n]) \prec pv(s[j+1:n])$  and  $pv(s[i:n]), pv(s[j:n])$  are in the same type C block, we have  $pv(s[i:n]) \prec pv(s[j:n])$  since  $fw$  values are equal. Therefore, we scan  $\overline{PSA}_s$  in increasing order and determine the correct position of type C suffix  $pv(s[i:n])$  if the position of suffix  $pv(s[i+1:n])$  is already determined. This is guaranteed by Lemma 3, since the position of suffix  $pv(s[i+1:n])$  precedes that of  $pv(s[i:n])$ . Hence, we can determine the positions of the type C suffixes by running through  $\overline{PSA}_s$  once.  $\square$

Fig. 1 shows our linear time sorting algorithm for type C suffixes.

### 3.2 Constructing PLCP Array

This section considers the construction of PLCP arrays, given the  $p$ -suffix array. For standard LCP arrays, Kasai *et al.* [17] showed a linear time algorithm for its construction, given the suffix array. However, the same algorithm cannot be applied directly to PLCP arrays because of the difficulties mentioned at the end of Section 2. In the following, we show some characteristics of PLCP arrays and propose P-Kasai, a modified version of the algorithm of [17].

For the type A suffixes, it is obvious that  $PLCP_s[1:A_s] = -1, 1, 2, \dots, A_s - 1$ . Hence we mainly consider computing  $PLCP_s[i]$  where  $A_s < i \leq n$ .

**Lemma 4.** *Let  $s$  be a  $p$ -string of length  $n$  and  $l_i = PLCP_s[rank[i]]$ . For any  $i$  ( $1 \leq i < n$ ), if  $l_i > 0$  then*

$$\begin{cases} l_{i+1} \geq l_i - 1 & \text{if } pv(s[j+1:n]) \prec pv(s[i+1:n]), \\ l_{j+1} \geq l_i - 1 & \text{if } pv(s[j+1:n]) \succ pv(s[i+1:n]), \end{cases}$$

where  $j = PSA[rank[i] - 1]$ .

*Proof.* Since  $pv(s[j : j+l_i-1]) = pv(s[i : i+l_i-1])$ , then  $pv(s[j+1 : j+l_i-1]) = pv(s[i+1 : i+l_i-1])$ . If  $pv(s[j+1 : n]) \prec pv(s[i+1 : n])$ , then for any  $k$  ( $rank[j+1] \leq k \leq rank[i+1]$ ),

$$pv(s[j+1 : j+l_i-1]) = pv(s[PSA[k] : PSA[k]+l_i-2]) = pv(s[i+1 : i+l_i-1]).$$

Hence,  $l_{i+1} \geq l_i - 1$ . Similarly, if  $pv(s[j+1 : n]) \succ pv(s[i+1 : n])$ , we can get  $l_{j+1} \geq l_i - 1$ .  $\square$

Note that the case of  $l_i > 0$  and  $pv(s[j+1 : n]) \succ pv(s[i+1 : n])$  does not occur for *LCP* arrays, which is the key property for amortized linear time construction algorithm [17]. In the case for *PLCP* arrays, for example, we can see in Table [1] that  $pv(s[2 : 12]) \prec pv(s[10 : 12])$  but  $pv(s[3 : 12]) \succ (s[11 : 12])$ .

A necessary condition for  $pv(s[j+1 : n]) \succ pv(s[i+1 : n])$  is given below:

**Lemma 5.** *Let  $s$  be a  $p$ -string of length  $n$  and  $l_i = PLCP_s[rank[i]]$ . For any  $i$  ( $1 \leq i < n$ ), if  $pv(s[j+1 : n]) \succ pv(s[i+1 : n])$  then  $pv(s[i : n])[l_i+1] = l_i$ , where  $j = PSA[rank[i]-1]$ .*

*Proof.* First,  $pv(s[j+1 : n])[1 : l_i-1] = pv(s[i+1 : n])[1 : l_i-1]$ , and by Definition [2]  $pv(s[i : n])[l_i+1] \leq l_i$ . If we assume  $pv(s[i : n])[l_i+1] < l_i$ , then

$$pv(s[j+1 : n])[l_i] \leq pv(s[j : n])[l_i+1] < pv(s[i : n])[l_i+1] = pv(s[i+1 : n])[l_i].$$

This implies that  $pv(s[j+1 : n]) \prec pv(s[i+1 : n])$ , a contradiction.  $\square$

Focusing on the case of  $pv(s[i+1 : n]) \prec pv(s[i : n])$ , we have:

**Lemma 6.** *Let  $s$  be a  $p$ -string of length  $n$  and  $l_i = PLCP_s[rank[i]]$ . For any  $i$  ( $1 \leq i < n$ ), if  $pv(s[i : n])[l_i+1] = l_i$  and  $pv(s[i+1 : n]) \prec pv(s[i : n])$  then  $zlen[i+1] \geq l_i$ .*

*Proof.* Assume contrary that  $zlen[i+1] < l_i$ . Since  $pv(s[i+1 : n])[l_i] = 0$ , there exists  $m = \min\{k \mid 2 \leq k < l_i, pv(s[i+1 : n])[k] \neq 0\}$ , that is,  $zlen[i+1] = m-1$ , and then  $zlen[i] = m$ . This implies  $pv(s[i : n]) \prec pv(s[i+1 : n])$ , a contradiction.  $\square$

Lemmas [5] and [6] lead to the next lemma.

**Lemma 7.** *Let  $s$  be a  $p$ -string of length  $n$  and  $l_i = PLCP_s[rank[i]]$ . For any  $i$  ( $1 \leq i < A_s - 1$ ), if  $pv(s[j+1 : n]) \succ pv(s[i+1 : n])$  and  $pv(s[i+1 : n]) \prec pv(s[i : n])$  then*

$$\begin{cases} l_{i+1} = A_s & \text{if } rank[i+1] = A_s + 1 \text{ and } l_i > A_s, \\ l_{i+1} \geq l_i & \text{otherwise,} \end{cases}$$

where  $j = PSA[rank[i]-1]$ .

*Proof.* It follows from Lemma [5] and [6] that  $zlen[i+1] \geq l_i$ . If  $rank[i+1] = A_s + 1$ , it is obvious that  $zlen[PSA[rank[i+1]-1]] = zlen[PSA[A_s]] = A_s$ . In addition if  $l_i > A_s$ , then  $pv(s[1 : A_s]) = 0^{A_s}$  is a prefix of  $pv(s[i+1 : n])$ . Hence  $l_{i+1} = A_s$ .

On the other hand, in the case of  $rank[i+1] > A_s + 1$ , we have  $zlen[PSA[rank[i+1]-1]] \geq l_i$ , since type B and C suffixes are sorted in descending order of  $zlen$ . If  $rank[i+1] = A_s + 1$  but  $l_i \leq A_s$ , then  $zlen[PSA[rank[i+1]-1]] = A_s \geq l_i$ . Hence in either case,  $zlen[PSA[rank[i+1]-1]] \geq l_i$ , and then  $l_{i+1} \geq l_i$ .  $\square$

```

plcp[1] = -1;
for (b = 2; psa[b] == n - b; b++) {
    plcp[b] = b - 1; // PLCP for type A suffixes
}
// b = As + 1
k = 1;
for (i = 1; i <= n - b + 1; i++) {
    j = psa[rank[i]-1];
    if (plcp[rank[i]] > k)
        k = plcp[rank[i]];
    while (pv(s[i:n])[k+1] == pv(s[j:n])[k+1])
        k++;
    plcp[rank[i]] = k;
    if (rank[j+1] < rank[i+1])
        k--; // Kasai's algorithm up to here
    else { // below is modification
        if (plcp[rank[j+1]] < k - 1)
            plcp[rank[j+1]] = k - 1;
        if (rank[i+1] < rank[i]) {
            if (rank[i+1] == b && k > b - 1)
                k = b - 1;
        } else
            k = 1;
    }
}
}

```

Fig. 2. Algorithm for constructing PLCP array (P-Kasai)

Lemma 7 helps to compute  $l_{i+1}$  in the case of  $pv(s[j+1:n]) \succ pv(s[i+1:n])$ . Fig. 2 shows our algorithm for constructing PLCP array.

## 4 Computational Experiments

We compare our algorithms and naive algorithms on randomly generated text, and some files taken from the The Canterbury Corpus<sup>2</sup>, and those used in [19]<sup>3</sup> (Lightweight). All experiments were conducted on an Apple Mac Pro (Early 2008) with 3.2GHZ dual core Xeons and 18GB of memory, running MacOSX 10.5 Leopard. Programs were written in the C language and compiled with the gcc compiler and -O3 option.

Table 2 shows results on random data for various text lengths, and a fixed alphabet size of 255. Table 3 shows results on random data for various alphabet sizes and a fixed text length of 1,000,000. Table 4 shows results on various texts from several corpora. Radix and Bucket denote the two alternatives for the coarse sorting in Step 3. We use a standard quick sort on strings for the sorting algorithm of Step 4. Qsort denotes a naive algorithm using a standard quick sort

<sup>2</sup> <http://corpus.canterbury.ac.nz/>

<sup>3</sup> <http://web.unipmn.it/~manzini/lightweight/corpus/>



**Table 2.** Running times for random strings of length  $n$  and alphabet size = 255

$n$	p-suffix array			PLCP array		avg $lcp$	avg $zlen$	%C
	Radix	Bucket	Qsort	P-Kasai	Naive			
1024	<u>0.000099</u>	0.000170	0.000710	<u>0.000023</u>	0.000207	23.6	19.4	7.58
2048	<u>0.000241</u>	0.000319	0.001672	<u>0.000050</u>	0.000374	25.5	19.6	7.57
4096	<u>0.000689</u>	0.000778	0.004032	<u>0.000147</u>	0.000595	27.1	19.7	7.66
8192	<u>0.001759</u>	0.001840	0.009391	<u>0.000304</u>	0.001116	28.1	19.6	7.73
16384	<u>0.003840</u>	0.003908	0.018881	<u>0.000629</u>	0.002363	29.0	19.7	7.71
32768	0.009518	<u>0.009440</u>	0.042851	<u>0.001291</u>	0.004697	29.6	19.6	7.73
65536	0.022920	<u>0.022261</u>	0.096403	<u>0.002856</u>	0.010423	30.3	19.7	7.72
131072	0.050810	<u>0.048990</u>	0.201429	<u>0.006450</u>	0.020761	31.0	19.7	7.73
262144	0.124608	<u>0.120097</u>	0.458788	<u>0.015064</u>	0.039021	31.9	19.7	7.72
524288	0.287857	<u>0.276315</u>	0.998540	<u>0.062928</u>	0.083262	33.0	19.7	7.72

**Table 3.** Running times for random strings of length 1,000,000 and alphabet size  $\pi$

$\pi$	p-suffix array			PLCP array		avg $lcp$	avg $zlen$	%C
	Radix	Bucket	Qsort	P-Kasai	Naive			
2	0.351	<u>0.327</u>	1.144	0.203	<u>0.124</u>	19.8	1.5	75.0
4	0.387	<u>0.363</u>	0.851	0.228	<u>0.109</u>	11.5	2.2	55.5
8	0.430	<u>0.404</u>	0.839	0.242	<u>0.112</u>	10.7	3.2	40.6
16	0.493	<u>0.459</u>	0.975	0.239	<u>0.121</u>	12.2	4.7	29.4
32	0.522	<u>0.495</u>	1.114	0.225	<u>0.131</u>	15.1	6.8	21.2
64	0.543	<u>0.528</u>	1.346	0.204	<u>0.145</u>	19.4	9.7	15.2
128	0.584	<u>0.572</u>	1.645	0.187	<u>0.161</u>	25.7	13.9	10.8
256	0.633	<u>0.613</u>	2.111	<u>0.170</u>	0.186	34.2	19.7	7.70
512	0.697	<u>0.688</u>	2.708	<u>0.159</u>	0.224	45.1	28.0	5.48
1024	<u>0.754</u>	<u>0.754</u>	3.570	<u>0.147</u>	0.264	61.1	39.8	3.89
2048	0.851	<u>0.825</u>	4.688	<u>0.141</u>	0.360	84.8	56.4	2.75
4096	0.940	<u>0.921</u>	6.309	<u>0.132</u>	0.454	118.6	79.8	1.95
8192	<u>0.984</u>	0.990	8.373	<u>0.127</u>	0.628	165.8	113.1	1.38
16384	1.039	<u>1.068</u>	11.570	<u>0.124</u>	0.864	230.3	160.0	0.978
32768	1.289	<u>1.074</u>	15.510	<u>0.119</u>	1.266	317.0	226.2	0.691
65536	<u>0.967</u>	1.070	21.972	<u>0.115</u>	1.896	430.8	320.0	0.489
131072	<u>0.834</u>	0.992	30.362	<u>0.112</u>	3.058	578.2	453.5	0.345
262144	<u>0.667</u>	0.946	42.425	<u>0.111</u>	5.780	766.4	640.4	0.245
524288	<u>0.509</u>	0.941	58.855	<u>0.105</u>	9.795	1019.7	907.2	0.173

on all suffixes. For the PLCP array, we compare the P-Kasai algorithm and a naive algorithm. %C denotes the percentage of type C suffixes. The running times were measured by user time, averaged over 100 and 10 iterations for random strings and files, respectively. In the tables, they are presented in second.

Our algorithms are clearly much faster than naive methods except for the PLCP computation when the average  $lcp$  is small.

**Table 4.** Running times for files from several corpora

File	p-suffix array			PLCP array		length	avg <i>lcp</i>	avg <i>zlen</i>	%C
	Radix	Bucket	Qsort	P-Kasai	Naive				
Artificial									
aaa.txt	0.0060	<u>0.0044</u>	175.3258	<u>0.0023</u>	17.0399	100000	49999.5	1.0	99.9
alphabet.txt	0.0060	<u>0.0040</u>	168.2780	<u>0.0024</u>	16.9446	100000	49999.5	26.0	99.9
random.txt	0.0350	<u>0.0321</u>	0.0947	<u>0.0049</u>	0.0102	100000	17.7	9.6	15.2
Canterbury									
alice29.txt	0.0492	<u>0.0477</u>	0.1174	<u>0.0088</u>	0.0163	152089	13.6	5.5	31.0
asyoulik.txt	0.0413	<u>0.0379</u>	0.0896	<u>0.0070</u>	0.0125	125179	13.4	5.9	27.5
cp.html	0.0078	<u>0.0073</u>	0.0175	<u>0.0010</u>	0.0036	24603	18.6	6.2	25.1
fields.c	0.0027	<u>0.0025</u>	0.0062	<u>0.0004</u>	0.0016	11150	18.6	5.2	30.0
grammar.lsp	0.0006	<u>0.0006</u>	0.0016	<u>0.0001</u>	0.0004	3721	13.5	5.0	30.2
lcet10.txt	0.1645	<u>0.1544</u>	0.4040	0.0534	<u>0.0510</u>	426754	15.8	5.6	30.5
plravn12.txt	0.1862	<u>0.1849</u>	0.4506	0.0728	<u>0.0537</u>	481861	13.6	6.1	29.0
xargs.l	0.0008	<u>0.0007</u>	0.0021	<u>0.0001</u>	0.0004	4227	11.6	6.2	28.6
Large									
E.coli	<u>2.9639</u>	2.9823	7.0833	1.4862	<u>1.0672</u>	4638690	19.3	2.2	55.9
bible.txt	3.1321	<u>3.0253</u>	6.5019	1.1278	<u>0.9816</u>	4047392	18.0	5.5	31.6
world192.txt	2.0905	<u>2.0194</u>	3.9638	<u>0.6075</u>	0.7444	2473400	31.3	5.9	28.2
Misc.									
pi.txt	0.4567	<u>0.4529</u>	0.8871	0.2449	<u>0.1174</u>	1000000	11.0	3.6	36.6
Lightweight									
chr22.dna	31.8458	<u>29.8698</u>	4119.5402	<u>12.6971</u>	249.1254	34553758	1980.9	2.0	59.5
etext99	8.4958	<u>7.8785</u>	14.1768	<u>1.9445</u>	3.3963	6291456	94.6	5.6	29.7
howto	80.2950	<u>77.1156</u>	151.0006	<u>13.6105</u>	49.7969	39422105	273.8	4.8	34.9

## 5 Conclusion and Future Work

Using several characteristics of parameterized suffixes, we introduced techniques to speed up the direct construction of parameterized suffix arrays and PLCP arrays. The worst case time complexity of sorting the suffixes is  $O(n^3)$  when using a standard Quicksort on strings. For example, considering  $pv(\text{abbaabb} \dots) = 0013131 \dots$ , gives one type B block of size  $n/2$  requiring  $O(n^3)$  time. However, since the size of the blocks to be sorted is reduced considerably compared to  $n$ , the total time required for our algorithm is much faster than a naïve use of Quicksort.

From a theoretical viewpoint, a naïve radix sort would give an  $O(n^2)$  time algorithm. It is an open problem if there exists better worst-case time algorithms for p-suffix array construction. Similarly, for PLCP arrays, the P-Kasai algorithm runs in  $O(n^2)$  time. However, we do not know if this bound is tight, or if there exist linear time algorithms for PLCP array construction.

## References

1. Baker, B.S.: A program for identifying duplicated code. *Computing Science and Statistics* 24, 49–57 (1992)
2. Baker, B.S.: Parameterized pattern matching: Algorithms and applications. *Journal of Computer and System Sciences* 52(1), 28–42 (1996)
3. Fredriksson, K., Mozgovoy, M.: Efficient parameterized string matching. *Information Processing Letters* 100(3), 91–96 (2006)
4. Shibuya, T.: Generalization of a suffix tree for RNA structural pattern matching. *Algorithmica* 39(1), 1–19 (2004)
5. Weiner, P.: Linear pattern-matching algorithms. In: *Proc. of 14th IEEE Ann. Symp. on Switching and Automata Theory*, pp. 1–11 (1973)
6. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. *SIAM J. Computing* 22(5), 935–948 (1993)
7. McCreight, E.M.: A space-economical suffix tree construction algorithm. *Journal of the ACM* 23(2), 262–272 (1976)
8. Ukkonen, E.: On-line construction of suffix trees. *Algorithmica* 14(3), 249–260 (1995)
9. Farach, M.: Optimal suffix tree construction with large alphabets. In: *Proc. 38th Annual Symposium on Foundations of Computer Science*, pp. 137–143 (1997)
10. Kärkkäinen, J., Sanders, P.: Simple linear work suffix array construction. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) *ICALP 2003*. LNCS, vol. 2719, pp. 943–955. Springer, Heidelberg (2003)
11. Kim, D.K., Sim, J.S., Park, H., Park, K.: Linear-time construction of suffix arrays. In: Baeza-Yates, R., Chávez, E., Crochemore, M. (eds.) *CPM 2003*. LNCS, vol. 2676, pp. 186–199. Springer, Heidelberg (2003)
12. Ko, P., Aluru, S.: Space efficient linear time construction of suffix arrays. In: Baeza-Yates, R., Chávez, E., Crochemore, M. (eds.) *CPM 2003*. LNCS, vol. 2676, pp. 200–210. Springer, Heidelberg (2003)
13. Abouelhoda, M.I., Kurtz, S., Ohlebusch, E.: Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms* 2(1), 53–86 (2004)
14. Kim, D.K., Jeon, J.E., Park, H.: An efficient index data structure with the capabilities of suffix trees and suffix arrays for alphabets of non-negligible size. In: Apostolico, A., Melucci, M. (eds.) *SPIRE 2004*. LNCS, vol. 3246, pp. 138–149. Springer, Heidelberg (2004)
15. Kosaraju, S.: Faster algorithms for the construction of parameterized suffix trees. In: *Proc. FOCS 1995*, pp. 631–637 (1995)
16. Deguchi, S., Higashijima, F., Bannai, H., Inenaga, S., Takeda, M.: Parameterized suffix arrays for binary strings. In: *Proc. PSC 2008*, pp. 84–94 (2008)
17. Kasai, T., Lee, G., Arimura, H., Arikawa, S., Park, K.: Linear-time longest-common-prefix computation in suffix arrays and its applications. In: Amir, A., Landau, G.M. (eds.) *CPM 2001*. LNCS, vol. 2089, pp. 181–192. Springer, Heidelberg (2001)
18. Amir, A., Farach, M., Muthukrishnan, S.: Alphabet dependence in parameterized matching. *Information Processing Letters* 49(3), 111–115 (1994)
19. Manzini, G., Ferragina, P.: Engineering a lightweight suffix array construction algorithm. *Algorithmica* 40, 33–50 (2004)

# On the Crossing Numbers of Cartesian Products of Stars and Graphs on Five Vertices<sup>\*</sup>, <sup>\*\*</sup>

Marián Klešč

Department of Mathematics, Faculty of Electrical Engineering and Informatics,  
Technical University, 042 00 Košice, Slovak Republic  
marian.klesc@tuke.sk

**Abstract.** There are known crossing numbers of Cartesian products of stars with all graphs of order at most four. In this paper we are dealing with the Cartesian products of stars with graphs on five vertices. We give the exact values of crossing numbers for some of these graphs and we summarise all known results concerning crossing numbers of these graphs. In addition, we give the crossing number of the join product of star and the cycle  $C_5$  with one additional edge.

**Keywords:** graph, drawing, crossing number, star, Cartesian product.

## 1 Introduction

Let  $G$  be a simple and undirected graph with vertex set  $V(G)$  and edge set  $E(G)$ . The *crossing number*  $cr(G)$  of the graph  $G$  is the minimum number of pairwise intersections of edges in all drawings of  $G$  in the plane. It is easy to see that a drawing with minimum number of crossings (an *optimal* drawing) is always a *good* drawing, meaning that no edge crosses itself, no two edges cross more than once, and no two edges incident with the same vertex cross. The investigation on crossing numbers of graphs is a classical and however very difficult problem. Garey and Johnson [4] have proved that this problem is NP-complete. According to their special structure, Cartesian products of special graphs are one of few graph classes for which the exact values of crossing numbers were obtained. (For a definition of Cartesian product, see [1].)

Let  $D$  ( $D(G)$ ) be a good drawing of the graph  $G$ . We denote the number of crossings in the drawing  $D$  by  $cr_D(G)$ . Let  $G_i$  and  $G_j$  be edge-disjoint subgraphs of the graph  $G$ . We denote by  $cr_D(G_i, G_j)$  the number of crossings between edges of  $G_i$  and edges of  $G_j$ , and by  $cr_D(G_i)$  the number of crossings among edges of  $G_i$  in  $D$ . It is easy to see that for any three edge-disjoint subgraphs  $G_i$ ,  $G_j$ , and  $G_k$  of the graph  $G$  the following equations hold:

$$cr_D(G_i \cup G_j) = cr_D(G_i) + cr_D(G_j) + cr_D(G_i, G_j),$$

---

<sup>\*</sup> The research was supported by the Slovak VEGA grant No. 1/0636/08.

<sup>\*\*</sup> This work was supported by the Slovak Research and Development Agency under the contract No. APVV-0073-07.

$$cr_D(G_i \cup G_j, G_k) = cr_D(G_i, G_k) + cr_D(G_j, G_k). \tag{1}$$

Let  $P_n$  and  $C_n$  be the *path* and the *cycle* of length  $n$ , respectively, and the *star*  $S_n$  be the complete bipartite graph  $K_{1,n}$ . In [1] Beineke and Ringel asked on the crossing numbers of Cartesian products of small graphs with paths, cycles and stars. For the path, the crossing numbers of  $G \times P_n$  are known for all graphs  $G$  of order at most five, see [8,10,11,14], and for the cycle, the crossing numbers of  $G \times C_n$  are given for all graphs  $G$  with at most four vertices [1,10,11,16]. In [15], the known crossing numbers of  $G \times C_n$  are summarised for graphs on five vertices. The crossing numbers of stars and all graphs of order three or four are given in [1,8,10,11]. For some graphs of order five, the crossing numbers of Cartesian products with stars are given in [13]. We extend these results and we establish the crossing numbers for Cartesian products of stars with several graphs of order five.

In this paper, some proofs are based on Kleitman’s result on crossing numbers of complete bipartite graphs. More precisely, in [9] he proved that






















$$cr(K_{m,n}) = \left\lfloor \frac{m}{2} \right\rfloor \left\lfloor \frac{m-1}{2} \right\rfloor \left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{n-1}{2} \right\rfloor, \quad \text{if } m \leq 6. \tag{2}$$

For convenience, the number  $\lfloor \frac{m}{2} \rfloor \lfloor \frac{m-1}{2} \rfloor \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor$  is often denoted by  $Z(m, n)$  in our paper. In the proofs of this paper, we will often use the term “region” also in nonplanar drawings. In this case, crossings are considered to be vertices of the “map”.

## 2 Cartesian Products of Stars and Graphs of Order Five

In the Table 1, one can find all 21 connected graphs on five vertices. In [2] Bokal proved the conjecture given by Jendrol’ and Ščerbová [8] that  $cr(K_{1,n} \times P_m) = (m - 1) \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor$  for the path  $P_m$  of length  $m$ . Hence,  $cr(G_1 \times S_n) = cr(P_4 \times S_n) = 3 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor$ . The same author in [3] proved that for every tree  $T$  with  $n_2$  vertices of degree two and  $n_3$  vertices of degree three  $cr(T \times S_n) = \lfloor \frac{n}{2} \rfloor ((n_2 + 2n_3) \lfloor \frac{n-1}{2} \rfloor + 1)$ . Thus,  $cr(G_3 \times S_n) = 3 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + \lfloor \frac{n}{2} \rfloor$ . As there is a drawing of the graph  $G_4 \times S_n$  with  $3 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + \lfloor \frac{n}{2} \rfloor$  crossings and the graph  $G_4 \times S_n$  contains  $G_3 \times S_n$  as a subgraph,  $cr(G_4 \times S_n) = 3 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + \lfloor \frac{n}{2} \rfloor$ . In [13], the crossing numbers of the graphs  $G_{11} \times S_n$  and  $G_{14} \times S_n$  are given. Huang and Zhao proved in [7] that the crossing number of the complete tripartite graph  $K_{1,4,n}$  is  $n(n - 1)$ . As the graph  $S_4 \times S_n$  is a subdivision of the graph  $K_{1,4,n}$ ,  $cr(S_4 \times S_n) = cr(G_2 \times S_n) = n(n - 1)$ . Both graphs  $G_6$  and  $G_9$  contain the graph  $G_2$  as a subgraph and therefore,  $cr(G_6 \times S_n) \geq cr(G_2 \times S_n)$  and  $cr(G_9 \times S_n) \geq cr(G_2 \times S_n)$ . It is not difficult to find drawings of both graphs  $G_6 \times S_n$  and  $G_9 \times S_n$  with exactly  $n(n - 1)$  crossings. This implies that  $cr(G_6 \times S_n) = cr(G_9 \times S_n) = n(n - 1)$ . The crossing number of the graph  $G_{10} \times S_n$  is  $4 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + 2n$ , see [12]. The graph  $G_{17} \times S_n$  contains  $G_{10} \times S_n$  as a subgraph and hence,  $cr(G_{17} \times S_n) \geq cr(G_{10} \times S_n)$ . As there is a drawing of the graph  $G_{17} \times S_n$  with  $4 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + 2n$  crossings, the crossing number of

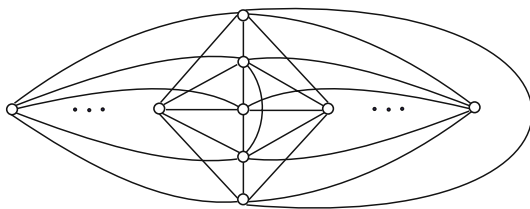
**Table 1.** The known crossing numbers for Cartesian products of stars and graphs of order five

$G_i$	$cr(G_i \times S_n)$	$G_i$	$cr(G_i \times S_n)$
$G_1$ 	$3 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor$ [2]	$G_{12}$ 	
$G_2$ 	$n(n-1)$ [7]	$G_{13}$ 	$4 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + \lfloor \frac{n}{2} \rfloor$
$G_3$ 	$3 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + \lfloor \frac{n}{2} \rfloor$ [3]	$G_{14}$ 	$n(n-1)$ [13]
$G_4$ 	$3 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + \lfloor \frac{n}{2} \rfloor$	$G_{15}$ 	$4 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + 2n + \lfloor \frac{n}{2} \rfloor$
$G_5$ 		$G_{16}$ 	
$G_6$ 	$n(n-1)$	$G_{17}$ 	$4 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + 2n$
$G_7$ 		$G_{18}$ 	$4 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + 2n + \lfloor \frac{n}{2} \rfloor$
$G_8$ 		$G_{19}$ 	$4 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + 2n + \lfloor \frac{n}{2} \rfloor$ [5]
$G_9$ 	$n(n-1)$	$G_{20}$ 	
$G_{10}$ 	$4 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + 2n$ [12]	$G_{21}$ 	
$G_{11}$ 	$n(n-1)$ [13]		

the graph  $G_{17} \times S_n$  is  $4 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + 2n$  also. He and Huang [5] proved that the crossing number of the Cartesian product  $G_{19} \times S_n$  is  $4 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + 2n + \lfloor \frac{n}{2} \rfloor$ . In the next sections we give the exact values of crossing numbers of the graphs  $G_i \times S_n$  for  $i = 13, 15,$  and  $18$ .

### 3 The Crossing Number of $G_{13} + nK_1$

Our aim is to establish the crossing number of the graph  $G_{13} \times S_n$ . To prove that  $cr(G_{13} \times S_n) = 4 \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + \lfloor \frac{n}{2} \rfloor$ , we need to know the crossing number of the join of the graph  $G_{13}$  with  $n$  isolated vertices. The join product of two



**Fig. 1.** The graph  $G_{13} + nK_1$

graphs  $G_i$  and  $G_j$ , denoted by  $G_i + G_j$ , is obtained from vertex-disjoint copies of  $G_i$  and  $G_j$  by adding all edges between  $V(G_i)$  and  $V(G_j)$ . For  $|V(G_i)| = m$  and  $|V(G_j)| = n$ , the edge set of  $G_i + G_j$  is the union of disjoint edge sets of the graphs  $G_i, G_j$ , and the complete bipartite graph  $K_{m,n}$ . In this section, we denote the graph  $G_{13}$  by  $H$ . The graph  $H$  consists of one 5-cycle, denoted by  $C_5(H)$  in the paper, and of one additional edge. The graph  $H + nK_1$  consists of one copy of the graph  $H$  and  $n$  vertices  $t_1, t_2, \dots, t_n$ , where every vertex  $t_i, i = 1, 2, \dots, n$ , is adjacent to five vertices of  $H$ . Let for  $i = 1, 2, \dots, n, T^i$  denote the subgraph induced by five edges incident with the vertex  $t_i$  and let  $F^i = H \cup T^i$ . For the simpler labelling, let  $H_n$  denote the graph  $H + nK_1$ , in this paper. In Figure 1 one can easily see that

$$H + nK_1 = H_n = H \cup K_{5,n} = H \cup \left( \bigcup_{i=1}^n T^i \right). \tag{3}$$

**Theorem 1.**  $cr(H + nK_1) = 4\lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + \lfloor \frac{n}{2} \rfloor$  for  $n \geq 1$ .

*Proof.* The drawing in Figure 1 shows that  $cr(H + nK_1) \leq Z(5, n) + \lfloor \frac{n}{2} \rfloor$  and that the theorem is true if equality holds. We prove the reverse inequality by induction on  $n$ . As the graph  $H + K_1$  is planar, the case  $n = 1$  is trivial. The graph  $H + 2K_1$  contains a subgraph homeomorphic to  $K_5$  and hence,  $cr(H + 2K_1) \geq 1$ . A suitable drawing of the graph  $H + 2K_1$  with one crossing shows that  $cr(H + 2K_1) \leq 1$ , and the case  $n = 2$  is also true. Suppose now that for  $n \geq 3$

$$cr(H_{n-2}) \geq Z(5, n - 2) + \left\lfloor \frac{n - 2}{2} \right\rfloor \tag{4}$$

and consider such a drawing  $D$  of  $H_n$  that

$$cr_D(H_n) < Z(5, n) + \left\lfloor \frac{n}{2} \right\rfloor. \tag{5}$$

Assume that there are two different subgraphs  $T^i$  and  $T^j$  that do not cross each other in  $D$ . Without loss of generality, let  $cr_D(T^{n-1}, T^n) = 0$ . The subdrawing of  $T^{n-1} \cup T^n$  induced from  $D$  divides the plane in such a way that no three vertices of degree two (the vertices of  $H$ ) are placed on the boundary of some region. If the edges of  $T^{n-1} \cup T^n$  cross in  $D$  the cycle  $C_5(H)$ , then  $cr_D(H, T^{n-1} \cup T^n) \geq 1$ .

Otherwise the vertices  $t_{n-1}$  and  $t_n$  are placed in  $D$  in different regions in the view of the subdrawing of  $C_5(H)$ . In this case the edge of  $H$  not belonging to  $C_5(H)$  crosses some edge of  $T^{n-1} \cup T^n$  and  $cr_D(H, T^{n-1} \cup T^n) \geq 1$  again. Moreover, as  $cr(K_{5,3}) = 4$ , in  $D$  every subgraph  $T^i$ ,  $i = 1, 2, \dots, n - 2$ , crosses  $T^{n-1} \cup T^n$  at least four times. Since  $H_n = H + nK_1 = H_{n-2} \cup (T^{n-1} \cup T^n)$  and  $H_{n-2} = K_{5,n-2} \cup H$ , using (1) we have

$$\begin{aligned} cr_D(H_n) &= cr_D(H_{n-2}) + cr_D(T^{n-1} \cup T^n) + cr_D(K_{5,n-2}, T^{n-1} \cup T^n) \\ &+ cr_D(H, T^{n-1} \cup T^n) \geq Z(5, n - 2) + \left\lfloor \frac{n - 2}{2} \right\rfloor + 4(n - 2) + 1 \\ &\geq Z(5, n) + \left\lfloor \frac{n}{2} \right\rfloor. \end{aligned}$$

This contradicts (5).

Hence,  $cr_D(T^i, T^j) \neq 0$  for all  $i, j = 1, 2, \dots, n$ ,  $i \neq j$ . Moreover, using (1) and (3) together with  $cr(K_{5,n}) = Z(5, n)$  we have

$$\begin{aligned} cr_D(H_n) &= cr_D(K_{5,n}) + cr_D(H) + cr_D(K_{5,n}, H) \\ &\geq Z(5, n) + cr_D(H) + cr_D(K_{5,n}, H). \end{aligned}$$

This, together with the assumption (5), implies that

$$cr_D(H) + cr_D(K_{5,n}, H) < \left\lfloor \frac{n}{2} \right\rfloor \tag{6}$$

and hence, in  $D$  there is at least one subgraph  $T^i$  which does not cross  $H$ .

Without loss of generality, let  $cr_D(H, T^n) = 0$  and let  $F^n$  be the subgraph  $H \cup T^n$  of the graph  $H_n$ . In the drawing  $D$  there is at least one subgraph  $T^i$ ,  $i \in \{1, 2, \dots, n - 1\}$ , for which  $cr_D(F^n, T^i) \leq 2$ , otherwise, as  $H_n = K_{5,n-1} \cup F^n$ , we have

$$\begin{aligned} cr_D(H_n) &= cr_D(K_{5,n-1}) + cr_D(F^n) + cr_D(K_{5,n-1}, F^n) \\ &\geq Z(5, n - 1) + 3(n - 1) \geq Z(5, n) + \left\lfloor \frac{n}{2} \right\rfloor. \end{aligned}$$

This contradicts (5). Consider now the subdrawing  $D^*$  of  $F^n$  induced by  $D$ . Our next analysis depends on whether or not the 5-cycle  $C_5(H)$  has an internal crossing in  $D^*$ .

Assume first, that the edges of  $C_5(H)$  do not cross each other. Since the edges of  $T^n$  do not cross the edges of  $H$ , all edges of  $T^n$  are placed in  $D^*$  in one of two regions, say outside, in the view of the subdrawing of  $C_5(H)$  and the edge of  $H$  not belonging to  $C_5(H)$  is placed inside the cycle  $C_5(H)$ . The unique such drawing  $D^*$  is shown in Figure 2. It is easy to see that if, in  $D$ , some vertex  $t_i$ ,  $i \in \{1, 2, \dots, n - 1\}$ , is placed in the region  $\beta_1$ , then  $H$  is crossed by at least two edges joining  $t_i$  with the vertices of  $H$ . Moreover, as  $T^i$  crosses  $T^n$ ,  $cr_D(F^n, T^i) \geq 3$ . Consider now the region  $\beta_2$ . One vertex of  $H$  does not appear



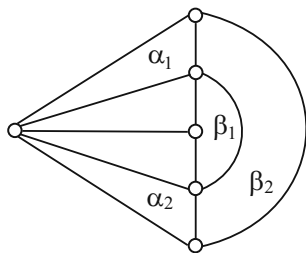


Fig. 2. The subdrawing of  $F^n = H \cup T^n$

on the boundary of this region. So, if  $t_i$  is placed in  $D$  in the region  $\beta_2$ , then  $cr_D(H, T^i) \geq 1$  and  $cr_D(T^n, T^i) \geq 1$ . On the boundary of the region  $\alpha_1$  there are only two vertices of  $H$ , and therefore, if the vertex  $t_i$  is placed in  $D$  in the region  $\alpha_1$ ,  $cr_D(F^n, T^i) \geq 3$ . For  $cr_D(F^n, T^i) = 3$ , the necessary condition is that one edge of  $T^i$  joining  $t_i$  with the vertex of  $H$  on the boundaries of the regions  $\alpha_2$  and  $\beta_1$  crosses the edge of  $H$  on the boundary of  $\alpha_1$ . If no edge of  $T^i$  crosses this edge of  $H$ , then  $cr_D(F^n, T^i) \geq 4$ . Regarding to the symmetry of  $D^*$ , the same holds for the region  $\alpha_2$ . For the remaining three regions of  $D^*$ , three vertices of  $H$  do not appear on its boundary and one of them does not appear on the boundaries of the neighbouring regions. So, if  $t_i$  is placed in  $D$  in some of the mentioned three regions, then the edges of  $T^i$  cross the edges of  $F^n$  at least four times.

Let  $r$  be the number of vertices  $t_i$ ,  $i \in \{1, 2, \dots, n-1\}$ , which are placed in  $D$  in the region  $\beta_2$ . In the drawing  $D$ , every such subgraph  $T^i$  crosses  $F^n$  at least two times and at least one of these crossings appears on the edges of  $H$ . Let  $s$  be the number of vertices  $t_i$  placed in  $D$  in the region  $\beta_1$  and such vertices  $t_i$  placed in the regions  $\alpha_1$  and  $\alpha_2$  for which  $T^i$  crosses  $F$  only three times. Every such subgraph  $T^i$  crosses also  $H$ . Since  $cr_D(F^n, T^j) \leq 2$  for some  $j \in \{1, 2, \dots, n-1\}$ , we have  $r \geq 1$ , and it follows from (6) that  $r + s < \lfloor \frac{n}{2} \rfloor$ . Now

$$\begin{aligned} cr_D(H_n) &= cr_D(K_{5,n-1}) + cr_D(F^n) + cr_D(K_{5,n-1}, F^n) \\ &\geq Z(5, n-1) + 2r + 3s + 4(n-r-s-1) \\ &= Z(5, n) - 4 \left\lfloor \frac{n-1}{2} \right\rfloor + 4n - 4 - 2r - s. \end{aligned}$$

This, together with the assumption (5), gives

$$2r + s > 4n - 4 - 4 \left\lfloor \frac{n-1}{2} \right\rfloor - \left\lfloor \frac{n}{2} \right\rfloor = 3 \left\lfloor \frac{n}{2} \right\rfloor.$$

On the other hand,  $r + s < \lfloor \frac{n}{2} \rfloor$  and the inequality

$$2r + s > 3 \left\lfloor \frac{n}{2} \right\rfloor > 3(r + s)$$

implies that

$$r + 2s < 0.$$

This contradiction with  $r > 0$  and  $s \geq 0$  confirms that there is no drawing of the graph  $H_n$  with fewer than  $Z(5, n) + \lfloor \frac{n}{2} \rfloor$  crossings in which the edges of the 5-cycle  $C_5(H)$  do not cross each other.

Assume now that the edges of  $C_5(H)$  cross each other in  $D$ . We remark that in  $D$  there is at least one subgraph  $T^j$ ,  $j \in \{1, 2, \dots, n - 1\}$ , for which  $cr_D(F^n, T^j) \leq 2$ . As  $cr_D(T^n, T^i) \neq 0$  for all  $i = 1, 2, \dots, n - 1$ , the inequality  $cr_D(F^n, T^j) \leq 2$  implies that  $cr_D(C_5(H), T^j) \leq 1$ . Since  $cr_D(C_5(H), T^n) = 0$ , the vertex  $t_n$  of  $T^n$  lies in  $D^*$  in the region with all five vertices of  $C_5(H)$  on its boundary, and the condition  $cr_D(C_5(H), T^j) \leq 1$  enforces that in the subdrawing of  $C_5(H) \cup T^n$  there is a region with at least four vertices of  $C_5(H)$  on its boundary. This is possible only in the case when two edges incident with a common vertex of  $C_5(H)$  cross, a contradiction with the requirement that the drawing  $D$  is good. This completes the proof.  $\square$

### 4 The Crossing Number of $G_i \times S_n$ for $i = 13, 15$ , and 18

Let  $K$  be a connected graph on five vertices. Consider a graph  $G_K$  obtained by joining all vertices of  $K$  to five vertices of a connected graph  $G$  such that every vertex of  $K$  be adjacent to exactly one vertex of  $G$ . Let  $G_K^*$  be the graph obtained from  $G_K$  by contracting the edges of  $K$ .

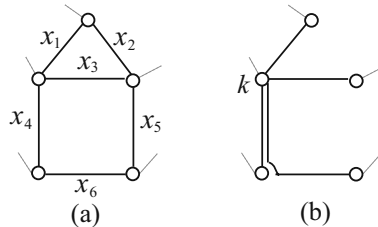
**Lemma 1.** *If  $G$  is a connected graph and  $K = G_{13}$ , then  $cr(G_K^*) \leq cr(G_K)$ .*

*Proof.* Assume an optimal drawing of  $G_K$ . Let  $x_1, x_2, \dots, x_6$  denote the numbers of crossings on the edges of the graph  $K$  in this drawing as shown in Figure 3(a). The drawing in Figure 3(b) shows that if  $x_4 \leq x_2 + x_5$ , then  $K$  can be contracted such that the resulting drawing does not have more crossings than the original. Due to symmetry of the graph  $G_{13}$ , the same holds if  $x_5 \leq x_1 + x_4$ .

Assume that the statement of Lemma 1 is not true. Then there is a good drawing of the graph  $G_K$  in which  $x_4 > x_2 + x_5$  and  $x_5 > x_1 + x_4$ . Combining these inequalities we have the inequality

$$x_4 > x_2 + x_5 > x_2 + x_1 + x_4,$$

which implies that



**Fig. 3.** The contraction of  $G_{13}$

$$x_1 + x_2 < 0.$$

This contradiction with  $x_i \geq 0$  for  $i = 1, 2, \dots, 5$  completes the proof.  $\square$

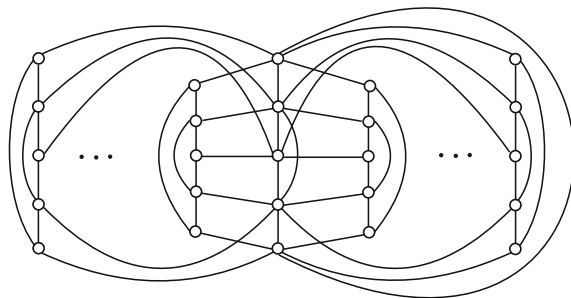
**Lemma 2.** *If  $G$  is a connected graph and  $K = G_{15}$ , then  $cr(G_K^*) \leq cr(G_K) - 1$ .*

*Proof.* As the graph  $K = G_{15}$  contains  $K_{2,3}$  as a subgraph and the graph  $K_{2,3}$  is not outer-planar, in every good drawing of the graph  $G_K$  there is at least one crossing on the edges of the  $K_{2,3}$ -subgraph of  $K$ . In the graph  $G_K$ , the subgraph  $K$  contains two vertices of degree five and three vertices of degree three. Let us denote by  $v_1$  and  $v_2$  the vertices of degree five and let  $e_1, e_2, e_3$  and  $f_1, f_2, f_3$  be the edges of the  $K_{2,3}$ -subgraph of  $K$  incident with the vertices  $v_1$  and  $v_2$ , respectively. Assume an optimal drawing of the graph  $G_K$ . If some of the edges  $e_1, e_2$ , and  $e_3$  is crossed, then the subdrawing obtained by deleting the edges  $e_1, e_2$ , and  $e_3$  has at least one crossing less than the original. If none of the edges  $e_1, e_2$ , and  $e_3$  is crossed, then there is at least one crossing on the edges  $f_1, f_2$ , and  $f_3$ . In this case, the subdrawing of the subgraph  $G_K - \{f_1, f_2, f_3\}$  has at least one crossing less than the original. Both subgraphs  $G_K - \{e_1, e_2, e_3\}$  and  $G_K - \{f_1, f_2, f_3\}$  are homeomorphic to the graph  $G_K^*$ , and therefore  $cr(G_K^*) \leq cr(G_K) - 1$ .  $\square$

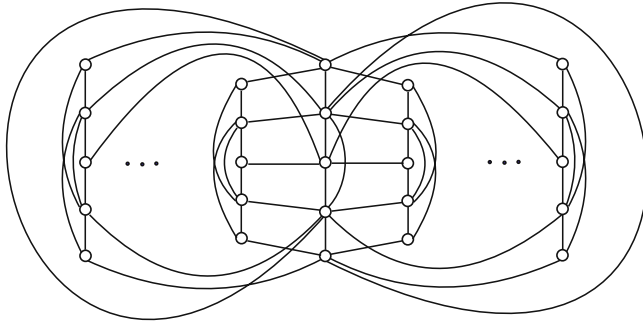
**Theorem 2.**  $cr(G_{13} \times S_n) = 4\lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + \lfloor \frac{n}{2} \rfloor$  for  $n \geq 1$ .

*Proof.* In Figure 4 there is the drawing of the graph  $G_{13} \times S_n$  with  $4\lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + \lfloor \frac{n}{2} \rfloor$  crossings. Hence,  $cr(G_{13} \times S_n) \leq 4\lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + \lfloor \frac{n}{2} \rfloor$ . To prove the reverse inequality we assume that there is a drawing of the graph  $G_{13} \times S_n$  with fewer than  $4\lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + \lfloor \frac{n}{2} \rfloor$  crossings. As contracting the edges of every non-central copy of  $G_{13}$  in the graph  $G_{13} \times S_n$  results in a graph isomorphic to the graph  $G_{13} + nK_1$ , in accordance with Lemma 1 we have  $cr(G_{13} + nK_1) < 4\lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + \lfloor \frac{n}{2} \rfloor$ . This contradiction with Theorem 1 completes the proof.  $\square$

**Theorem 3.**  $cr(G_{15} \times S_n) = cr(G_{18} \times S_n) = 4\lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + 2n + \lfloor \frac{n}{2} \rfloor$  for  $n \geq 1$ .



**Fig. 4.** The graph  $G_{13} \times S_n$



**Fig. 5.** The graph  $G_{15} \times S_n$

*Proof.* In Figure 5 there is the drawing of the graph  $G_{15} \times S_n$  with  $4\lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + 2n + \lfloor \frac{n}{2} \rfloor$  crossings. Hence,  $cr(G_{15} \times S_n) \leq 4\lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + 2n + \lfloor \frac{n}{2} \rfloor$ . To prove the reverse inequality we assume that there is a drawing of the graph  $G_{15} \times S_n$  with fewer than  $4\lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + 2n + \lfloor \frac{n}{2} \rfloor$  crossings. As contracting the edges of every non-central copy of  $G_{15}$  in the graph  $G_{15} \times S_n$  results in a graph isomorphic to the graph  $K_{1,1,3,n}$ , in accordance with Lemma 2 we have  $cr(K_{1,1,3,n}) < 4\lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + n + \lfloor \frac{n}{2} \rfloor$ . This is in contradiction with the result of Ho Pak Tunk [6] which states that  $cr(K_{1,1,3,n}) = 4\lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + n + \lfloor \frac{n}{2} \rfloor$  for all  $n \geq 1$ . Hence,  $cr(G_{15} \times S_n) = 4\lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + 2n + \lfloor \frac{n}{2} \rfloor$ .

In Figure 5 it is possible to draw  $n + 1$  edges (one edge in every copy of  $G_{15}$ ) without increasing the number of crossings and obtain the drawing of the graph  $G_{18} \times S_n$  with  $4\lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + 2n + \lfloor \frac{n}{2} \rfloor$  crossings. Thus, we have the upper bound. On the other hand, the graph  $G_{18} \times S_n$  contains  $G_{15} \times S_n$  as a subgraph and this implies that  $cr(G_{18} \times S_n) = 4\lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor + 2n + \lfloor \frac{n}{2} \rfloor$ . This completes the proof. □

## Acknowledgement

The author is indebted to the referees for their useful comments.

## References

1. Beineke, L.W., Ringel, R.D.: On the crossing numbers of products of cycles and graphs of order four. *J. Graph Theory* 4, 145–155 (1980)
2. Bokal, D.: On the crossing numbers of Cartesian products with paths. *J. Combin. Theory B* 97, 381–384 (2007)
3. Bokal, D.: On the crossing numbers of Cartesian products with trees. *J. Graph Theory* 56, 287–300 (2007)
4. Garey, M.R., Johnson, D.S.: Crossing number is NP-complete. *SIAM J. Algebraic and Discrete Methods* 4, 312–316 (1983)
5. He, P., Huang, Y.: The crossing number of  $W_4 \times S_n$ . *Journal of Zhengzhou University (Natural Science Edition)* 39, 14–18 (2007)

6. Ho, P.T.: The crossing number of  $K_{1,1,3,n}$ . *Ars Combinatoria* (to appear)
7. Huang, Y., Zhao, T.: The crossing number of  $K_{1,4,n}$ . *Discrete Math.* 308, 1634–1638 (2008)
8. Jendrol, S., Ščerbová, M.: On the crossing numbers of  $S_m \times P_n$  and  $S_m \times C_n$ . *Časopis pro pěstování matematiky* 107, 225–230 (1982)
9. Kleitman, D.J.: The crossing number of  $K_{5,n}$ . *J. Combin. Theory* 9, 315–323 (1970)
10. Klešč, M.: On the crossing numbers of Cartesian products of stars and paths or cycles. *Mathematica Slovaca* 41, 113–120 (1991)
11. Klešč, M.: The crossing numbers of products of paths and stars with 4-vertex graphs. *J. Graph Theory* 18, 605–614 (1994)
12. Klešč, M.: The crossing number of  $K_{2,3} \times P_n$  and  $K_{2,3} \times S_n$ . *Tatra Mountains Math. Publ.* 9, 51–56 (1996)
13. Klešč, M.: On the crossing numbers of products of stars and graphs of order five. *Graphs and Combinatorics* 17, 289–294 (2001)
14. Klešč, M.: The crossing numbers of Cartesian products of paths with 5-vertex graphs. *Discrete Math.* 233, 353–359 (2001)
15. Klešč, M.: Some crossing numbers of products of cycles. *Discuss. Math. Graph Theory* 25, 197–210 (2005)
16. Ringelsen, R.D., Beineke, L.W.: The crossing number of  $C_3 \times C_n$ . *J. Combin. Theory* 24, 134–136 (1978)

# Factorizations of Complete Graphs into Spanning Trees with All Possible Maximum Degrees<sup>\*</sup>

Petr Kovář and Michael Kubesa

VŠB – Technical University of Ostrava, Department of Applied Mathematics,  
17. listopadu 15, 708 33 Ostrava, Czech Republic  
petr.kovar@vsb.cz, michael.kubesa@vsb.cz

**Abstract.** Fronček and Kovářová provided in [2] and [3] spanning trees of order  $2n$  that factorize  $K_{2n}$  for every  $n \geq 2$  and for every feasible diameter  $d$ ,  $3 \leq d \leq 2n - 1$ . We extend their work and give a spanning tree on  $2n$  vertices with a maximum degree  $\Delta$  that factorize  $K_{2n}$  for every  $n \geq 2$  and for every feasible  $2 \leq \Delta \leq n$ . We give a construction for both symmetric and non-symmetric spanning trees.

## 1 Introduction

Factorization of complete graphs into isomorphic subgraphs is a well established topic of graph theory and design theory. During the years various classes of graphs were examined whether they do factorize the corresponding complete graph, provided certain necessary conditions are satisfied. For a surprisingly long time hardly anything was known about factorization of complete graphs  $K_{2n}$  into isomorphic spanning trees. Note that a spanning tree factorization of a complete graph on an odd number of vertices is not possible since the number of edges of the spanning tree does not divide the number of edges of the complete graph. Till 1997 only factorizations into hamiltonian paths and doublestars were known (a doublestar arises by joining the centers of two stars  $K_{1,n-1}$  by an edge). In 1997 comes the first major step by Eldergill. In his M.Sc. thesis [1] he gave a necessary and sufficient condition for the existence of a cyclic factorization of  $K_{2n}$  into symmetric trees and characterized completely which trees of order 10 do and which do not factorize  $K_{10}$ . About non-symmetric trees of order higher than 10 nothing was known, since the cyclic factorization cannot be used. Another breakthrough was the paper [2] by Fronček who introduces new techniques of 2-cyclic and bi-cyclic spanning tree factorization of  $K_{2n}$ . These methods are based on the  $\rho$ -labeling (see below), thus they can be used for factorization of  $K_{2n}$  only for odd  $n$ . In 2004 Kovářová introduced in [4]  $k$ -cyclic factorization of  $K_{2kn}$  which extends the decomposition for any  $k \geq 2$  and  $n$  odd. Finally, also in 2004, in [3] Kovářová introduced the swapping labeling which can be used for

---

<sup>\*</sup> Research for this article was partially supported by the institutional project MSM6198910027.

decompositions of  $K_{2n}$  for  $n$  even and lifts the requirement of  $n$  being a multiple of an odd number (see below).

From what was said it is obvious that finding a symmetric spanning tree factorization is easier than for non-symmetric spanning trees. We solve the problem announced in the abstract first for symmetric trees and in Sec. 4 we look also for a non-symmetric spanning tree factorization of  $K_{2n}$  for every feasible maximum degree.

## 2 Definitions and Notation

We consider only graphs with no loops or multiple edges. We begin by stating well known definitions.

A *symmetric tree* is a tree  $T$  that contains an edge  $e = xy$  (called a *bridge*) such that there exists an automorphism  $\phi$  of  $T$ , in which  $\phi(x) = y$  and  $\phi(y) = x$ . In other words we take two copies (called *banks*) of an arbitrary tree  $T'$  and a couple of corresponding vertices  $x, y$ , one in each copy, and join them by an edge  $e = xy$ . The resulting graph is a symmetric tree and all symmetric trees can be obtained in this way. Notice, that a symmetric tree is always of even order. A *broom*  $B_n(k)$  is a tree, which arises from a path  $P$  of length  $n - k - 1$  by attaching one endvertex of  $P$  to the center of a star  $K_{1,k}$ , where  $1 \leq k \leq n - 1$ . For  $k = 1$   $B_n(k)$  is the path  $P_n$  and for  $k = n - 1$   $B_n(k)$  is isomorphic to the star  $K_{1,n-1}$ . A *caterpillar*  $R$  is a tree such that by deleting all leaves we obtain a path  $P$  or an isolated vertex. This nontrivial path  $P$  (if it exists) is called the *spine* of  $R$ .

**Definition 1.** Let  $H$  be a graph on  $n$  vertices. A decomposition of the graph  $H$  is a set of pairwise edge disjoint subgraphs  $\mathcal{G} = \{G_1, G_2, \dots, G_t\}$  of  $H$  such that every edge of  $H$  belongs to precisely one of the subgraphs  $G_i$ ,  $1 \leq i \leq t$ . If each subgraph  $G_i$  is isomorphic to a graph  $G$  we speak about a  $G$ -decomposition of  $H$ . If  $G$  is a connected factor of  $H$ , then the  $G$ -decomposition is often called a  $G$ -factorization.

There are two obvious necessary conditions for the existence of a factorization of the complete graph  $K_m$  into spanning trees isomorphic to a given tree  $T$ . First,  $m$  must be even, since the number of edges of  $T$ ,  $m - 1$ , must divide  $m(m - 1)/2$ , the number of edges of  $K_m$ . Therefore, we consider only factorizations of the complete graph  $K_{2n}$ . There will always be  $n = m/2$  factors isomorphic to  $T$ . Second, the number of factors is  $n$  and each vertex in  $K_{2n}$  must be in every factor of degree at least one, thus the maximum degree of  $T$  cannot exceed  $n$ .

**Definition 2.** Let  $\lambda$  be an injection from the vertex set of a graph  $G$  with  $h$  edges into the set  $L = \{0, 1, \dots, 2h\}$  and let the length of every edge  $e = xy$  in  $G$  be determined by  $\ell(xy) = \min\{|\lambda(x) - \lambda(y)|, 2h + 1 - |\lambda(x) - \lambda(y)|\}$ . Then we say the graph  $G$  has a  $\rho$ -labeling if  $G$  contains edges of all lengths from 1 to  $h$ .

The  $\rho$ -labeling was introduced by Rosa [5]. In the text below we usually identify vertices and their labels. This will be convenient and it shall lead to no misunderstanding since the labelings are injective. In particular, we will refer to the  $xy$  edges with end-vertices labeled by  $\lambda(x), \lambda(y)$  as  $(\lambda(x), \lambda(y))$  edges.

For factorizations of regular complete bipartite graphs  $K_{n,n}$  Fronček introduced in [2] the *bipartite  $\rho$ -labeling*.

**Definition 3.** Let  $G$  be a bipartite graph with  $n$  edges and the vertex set  $V(G) = V_0 \cup V_1$ . Let  $\lambda$  be an injection  $\lambda : V_i \rightarrow S_i$ , where  $S_i$  is a subset of the set  $\{0_i, 1_i, \dots, (n - 1)_i\}$ ,  $i = 0, 1$ . The length of an edge  $x_0y_1$  for  $x_0 \in V_0$  and  $y_1 \in V_1$  with  $\lambda(x_0) = a_0$  and  $\lambda(y_1) = b_1$  is defined as

$$\ell_{01}(x_0y_1) = \begin{cases} b - a & \text{for } b \geq a, \\ n + (b - a) & \text{for } b < a. \end{cases}$$

If the set of all lengths of  $n$  edges is equal to  $\{0, 1, 2, \dots, n - 1\}$ , then  $\lambda$  is a bipartite  $\rho$ -labeling.

The following theorem was shown in [2].

**Theorem 1.** Let a bipartite graph  $G$  with  $n$  edges have a bipartite  $\rho$ -labeling. Then there exists a  $G$ -decomposition of  $K_{n,n}$  into  $n$  copies of  $G$ .

Notice that if  $G$  is a spanning graph we obtain a  $G$ -factorization of  $K_{n,n}$ .

The following concept was introduced by Eldergill in [1]. Recall that  $\phi$  is the automorphism of a symmetric tree that takes one bank onto the other.

**Definition 4.** A labeling of a symmetric tree  $G$  with  $2n - 1$  edges and banks  $T, T'$  is  $\rho$ -symmetric graceful if  $T$  has a  $\rho$ -labeling and  $\phi(i) = i + n$  for each vertex  $i$  in  $T$ , where  $\phi(i)$  is the corresponding vertex in  $T'$  and the addition is performed modulo  $2n$ .

**Theorem 2.** Let  $G$  be a symmetric graph with  $2n - 1$  edges. Then there exists a cyclic  $G$ -decomposition of  $K_{2n}$  if and only if  $G$  has a  $\rho$ -symmetric graceful labeling.

Before we describe the main tool used in our constructions, we have to define the concept of edge lengths that are used in labeling suitable for factorizations.

**Definition 5.** Let  $G$  be a graph with  $V(G) = V_0 \cup V_1, V_0 \cap V_1 = \emptyset$ , and  $|V_0| = |V_1| = r$ . Let  $\lambda$  be an injection,  $\lambda : V_i \rightarrow \{0_i, 1_i, \dots, (r - 1)_i\}$ ,  $i = 0, 1$ .

The pure length of an edge  $x_iy_i$  with  $x_i, y_i \in V_i$ , where  $i \in \{0, 1\}$ , for  $\lambda(x_i) = p_i$  and  $\lambda(y_i) = q_i$  is defined as

$$\ell_{ii}(x_iy_i) = \min\{|p - q|, r - |p - q|\}.$$

The mixed length of an edge  $x_0y_1$  with  $x_0 \in V_0, y_1 \in V_1$ , for  $\lambda(x_0) = p_0$  and  $\lambda(y_1) = q_1$ , is defined as

$$\ell_{01}(x_0y_1) = \begin{cases} q - p & \text{for } q \geq p, \\ q - p + r & \text{for } q < p, \end{cases}$$



where  $p, q \in \{0, 1, \dots, r - 1\}$  are the vertex labels without subscripts. The edges  $x_i y_i$  for  $i = 0, 1$  with the pure length  $\ell_{ii}$  are called pure  $ii$ -edges and the edges  $x_0 y_1$  with the mixed length  $\ell_{01}$  are called mixed edges.

**Definition 6.** Let  $G$  be a graph with  $4n + 1$  edges,  $V(G) = V_0 \cup V_1$ ,  $V_0 \cap V_1 = \emptyset$ , and  $|V_0| = |V_1| = 2n + 1$ . Let  $\lambda$  be an injection,  $\lambda : V_i \rightarrow \{0_i, 1_i, \dots, (2n)_i\}$ ,  $i = 0, 1$  and let the lengths be defined in Definition 5. We say  $G$  has a blended labeling if

- (1)  $\{\ell_{ii}(x_i y_i) : x_i y_i \in E(G)\} = \{1, 2, \dots, n\}$  for  $i = 0, 1$  and
- (2)  $\{\ell_{01}(x_0 y_1) : x_0 y_1 \in E(G)\} = \{0, 1, \dots, 2n\}$ .

Blended labeling was introduced by Fronček 2 as a generalization of the  $\rho$ -symmetric graceful labeling. Notice that in comparison to Definition 4 the requirement for  $T$  to be symmetric is lifted. However, the order of  $G$  has to be 2 modulo 4. For a proof of the following theorem see 2.

**Theorem 3.** Let  $G$  be a graph on  $4n + 2$  vertices with  $4n + 1$  edges. If  $G$  allows a blended labeling, then there exists a  $G$ -decomposition of  $K_{4n+2}$  into  $2n + 1$  isomorphic copies of  $G$ .

If  $G$  is a spanning tree in  $K_{4n+2}$  with a blended labeling then by Theorem 3 it factorizes  $K_{4n+2}$ .

The swapping labeling was introduced by Kovářová in 4. Unlike the blended labeling this labeling guarantees the decomposition of complete graphs of order 0 modulo 4.

**Definition 7.** A graph  $G$  with  $4n - 1$  edges has a swapping labeling if the following is satisfied. The vertex set  $V(G) = V_0 \cup V_1$ ,  $V_0 \cap V_1 = \emptyset$ , and  $|V_0| = |V_1| = 2n$ . Let  $\lambda$  be an injection,  $\lambda : V_i \rightarrow \{0_i, 1_i, \dots, (2n - 1)_i\}$  for  $i = 0, 1$ . Suppose the lengths are defined as in Definition 5, then

- (1)  $\{\ell_{ii}(x_i y_i) : x_i y_i \in E(G)\} = \{1, 2, \dots, n\}$ , for  $i = 0, 1$ ,
- (2) there exists an isomorphism  $\varphi$  s.t.  $G$  is isomorphic to  $G'$ , where  $V(G') = V(G)$  and  $E(G') = E(G) \setminus \{(k_0, (k+n)_0), (l_1, (l+n)_1)\} \cup \{(k_0, (l+n)_1), ((k+n)_0, l_1)\}$ ,
- (3)  $\{\ell_{01}(x_0 y_1) : x_0 y_1 \in E(G)\} = \{0, 1, \dots, 2n - 1\} \setminus \{\ell_{01}(k_0, (l+n)_1)\}$ .

**Theorem 4.** Let  $G$  be a graph on  $4n$  vertices with  $4n - 1$  edges. If  $G$  allows a swapping labeling, then there exists a  $G$ -decomposition of  $K_{4n}$  into  $2n$  isomorphic copies of  $G$ .

A proof is in 4. If  $G$  is a spanning tree in  $K_{4n}$  with a swapping labeling then by Theorem 4 it factorizes  $K_{4n}$ .

### 3 Symmetric Spanning Trees for Every $2 \leq \Delta \leq n$

**Lemma 5.** There exists a symmetric spanning tree  $T$  on  $2n$  vertices with maximum degree  $\Delta$  that allows a  $\rho$ -symmetric graceful labeling for every  $n \geq 2$  and  $2 \leq \Delta \leq n$ .

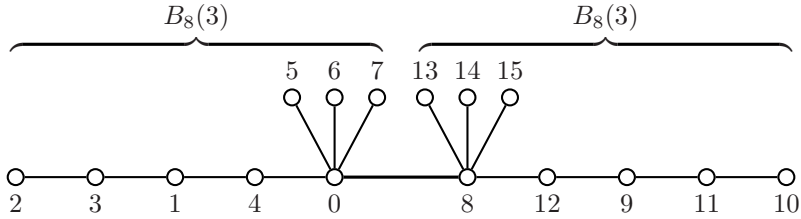


Fig. 1. A symmetric tree of order  $2n = 16$  and  $\Delta = 5$

*Proof.* The proof is constructive. Let  $3 \leq \Delta \leq n$ . We take a broom  $B_n(\Delta - 2)$  (see Fig. 1). It is well-known ([5]) that every caterpillar allows a graceful labeling. Since  $B_n(\Delta - 2)$  is a caterpillar (or a path for  $\Delta - 2 = 1$ ), we find for  $B_n(\Delta - 2)$  a graceful labeling (Fig. 1). Then we take a copy of  $B_n(\Delta - 2)$  and we label its vertices so that if a vertex in  $B_n(\Delta - 2)$  is labeled by  $i$ ,  $i \in \{0, 1, \dots, n - 1\}$ , the corresponding vertex in the copy of  $B_n(\Delta - 2)$  is labeled by  $i + n$ . Finally, we add the edge  $(j, j + n)$  between the two disjoint copies  $B_n(\Delta - 2)$ , where  $j$  is the label of the vertex of degree  $\Delta - 1$  in  $B_n(\Delta - 2)$ ,  $j \in \{0, 1, \dots, n - 1\}$ . Hence, we obtain a symmetric tree  $T$  of order  $2n$  with two vertices of maximum degree  $\Delta$  and with a  $\rho$ -symmetric graceful labeling for every  $2 \leq \Delta \leq n$  (Fig. 1).  $\square$

By Lemma 5 and Theorem 2 we have the following corollary.

**Theorem 6.** *There exists a symmetric spanning tree  $T$  on  $2n$  vertices with maximum degree  $\Delta$  that factorizes  $K_{2n}$  for every  $n \geq 2$  and  $2 \leq \Delta \leq n$ .*

### 4 Non-symmetric Spanning Trees for Every $3 \leq \Delta \leq n$

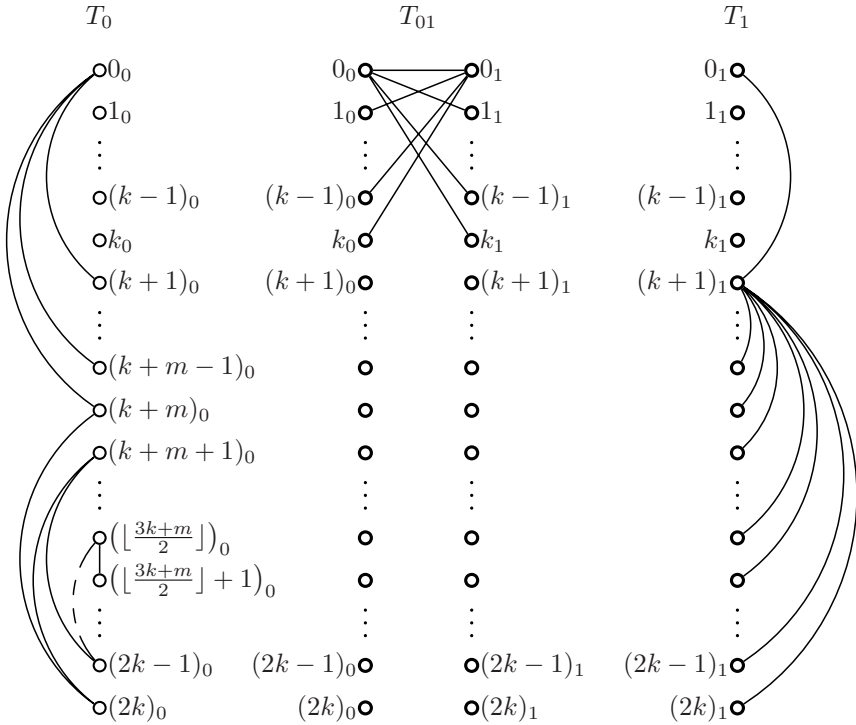
Since  $K_4$  has only a  $P_4$ -factorization,  $P_4$  being the hamiltonian path, and every hamiltonian path of order  $2n$  is a symmetric spanning tree, we consider only  $n \geq 3$  and  $\Delta \geq 3$ .

**Lemma 7.** *There exists a non-symmetric spanning tree  $T$  on  $2n$  vertices with maximum degree  $\Delta$  that allows a blended labeling for every odd  $n \geq 3$  and every  $3 \leq \Delta \leq n$ .*

*Proof.* The proof is constructive. Let  $n = 2k + 1$ . We distinguish two cases and show that trees  $T$  for  $k + 2 \leq \Delta \leq 2k + 1$  and  $T'$  for  $3 \leq \Delta \leq k + 1$  allow a blended labeling (Fig. 2 and 3). To describe the blended labeling, we split  $T$  and  $T'$ , respectively, into three subtrees  $T_0, T_{01}, T_1$  and  $T'_0, T'_{01}, T'_1$ , respectively.  $T_0$  contains only pure 00-edges,  $T_{01}$  only mixed edges and  $T_1$  only pure 11-edges of the tree  $T$  with a blended labeling (Fig. 2). Similarly,  $T'$  with a blended labeling can be split into  $T'_0, T'_{01}, T'_1$  (Fig. 3).

*Case 1.* Let  $k + 2 \leq \Delta \leq 2k + 1$ . Suppose  $\Delta = k + m + 1$ , where  $1 \leq m \leq k$ .

The subtree  $T_{01}$  will be a double star with mixed edges  $(0_0, 0_1), (0_0, 1_1), (0_0, 2_1), \dots, (0_0, k_1)$  of lengths  $0, 1, \dots, k$  (in this order) and  $(0_1, 1_0), (0_1, 2_0),$



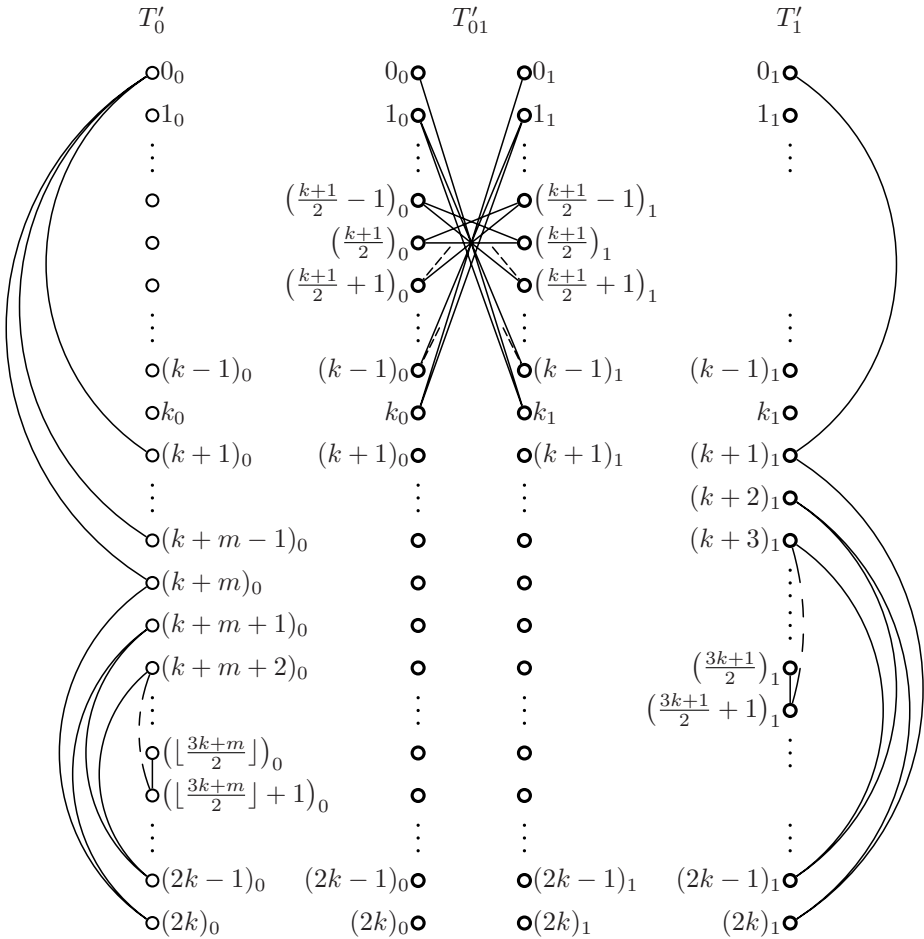
**Fig. 2.** A non-symmetric tree  $T$  with a blended labeling for  $k - m$  odd and  $\Delta = k + m + 1$

$\dots, (0_1, k_0)$  of lengths  $2k, 2k - 1, \dots, k + 1$  in this order. There is a bipartite  $\rho$ -labeling of  $T_{01}$  in Fig. 2.

The subtree  $T_0$  contains pure 00-edges  $(0_0, (k + 1)_0), (0_0, (k + 2)_0), \dots, (0_0, (k + m)_0)$  of lengths  $k, k - 1, \dots, k - m + 1$  in this order, and pure 00-edges of the path  $(k + m)_0, (2k)_0, (k + m + 1)_0, (2k - 1)_0, (k + m + 2)_0, (2k - 2)_0, \dots, (\lfloor \frac{3k+m}{2} \rfloor)_0, (\lfloor \frac{3k+m}{2} \rfloor + 1)_0$  for odd  $k - m$  or  $(k + m)_0, (2k)_0, (k + m + 1)_0, (2k - 1)_0, (k + m + 2)_0, (2k - 2)_0, \dots, (\lfloor \frac{3k+m}{2} \rfloor + 1)_0, (\lfloor \frac{3k+m}{2} \rfloor)_0$  for even  $k - m$  (Fig. 2). The path is of length  $k - m$ . It is not difficult to check that the path contains pure 00-edges of lengths  $k - m, k - m - 1, k - m - 2, \dots, 1$  in this order. Notice that for  $m = k$  the path is of length 0. Since vertices in  $T_0$  are labeled by  $0_0, 1_0, \dots, (2k)_0$  and the pure 00-edges achieve all lengths from 1 to  $k$ , the subtree  $T_0$  has a  $\rho$ -labeling.

The subtree  $T_1$  contains pure 11-edges  $(0_1, (k + 1)_1), ((k + 1)_1, (k + 2)_1), ((k + 1)_1, (k + 3)_1), \dots, ((k + 1)_1, (2k)_1)$  of lengths  $k, 1, 2, \dots, k - 1$  (in this order). Thus  $T_1$  allows a  $\rho$ -labeling.

It is obvious from Fig. 2 that the subtrees  $T_0$  and  $T_{01}$  share only the vertex  $0_0$  and similarly  $T_{01}$  and  $T_1$  share only the vertex  $0_1$ . Hence we have a non-symmetric spanning tree  $T$  of order  $4k + 2$  with a blended labeling, which has a single vertex (namely  $0_0$ ) of degree  $k + m + 1 = \Delta$  for every  $m, 1 \leq m \leq k$ , and the remaining vertices are only of degrees  $k + 2, k, 2$  or  $1$ .



**Fig. 3.** A non-symmetric tree  $T'$  with a blended labeling for  $k$  odd,  $k - m$  even and  $\Delta = m + 1$

*Case 2.* Let  $3 \leq \Delta \leq k + 1$ . Suppose  $\Delta = m + 1$ , where  $2 \leq m \leq k$ .

If  $k$  is even then the subtree  $T'_{01}$  is the path  $0_0, k_1, 1_0, (k - 1)_1, \dots, (\frac{k}{2} - 1)_0, (\frac{k}{2} + 1)_1, (\frac{k}{2})_0, (\frac{k}{2})_1, (\frac{k}{2} + 1)_0, (\frac{k}{2} - 1)_1, \dots, (k - 1)_0, 1_1, k_0, 0_1$  with mixed edges of lengths  $k, k - 1, k - 2, \dots, 2, 1, 0, 2k, 2k - 2, \dots, k + 3, k + 2, k + 1$  in this order (Fig. 3). If  $k$  is odd then  $T'_{01}$  is the path  $0_0, k_1, 1_0, (k - 1)_1, \dots, (\frac{k+1}{2} + 1)_1, (\frac{k+1}{2} - 1)_0, (\frac{k+1}{2})_1, (\frac{k+1}{2})_0, (\frac{k+1}{2} - 1)_1, (\frac{k+1}{2} + 1)_0, \dots, (k - 1)_0, 1_1, k_0, 0_1$  with mixed edges of lengths  $k, k - 1, k - 2, \dots, 2, 1, 0, 2k, 2k - 1, \dots, k + 3, k + 2, k + 1$  in this order (Fig. 3). In both cases the path  $T'_{01}$  has a bipartite  $\rho$ -labeling, which is easily verified.

The subtree  $T'_0$  is the same as the subtree  $T_0$  in *Case 1*. Finally, the subtree  $T'_1$  contains pure 11-edges of the path  $0_1, (k + 1)_1, (2k)_1, (k + 2)_1,$

$(2k-1)_1, \dots, (\frac{3k+2}{2}-1)_1, (\frac{3k+2}{2})_1$  for  $k$  even or of the path  $0_1, (k+1)_1, (2k)_1, (k+2)_1, (2k-1)_1, \dots, (\frac{3k+1}{2}+1)_1, (\frac{3k+1}{2})_1$  for  $k$  odd. In both cases the subtree  $T'_1$  contains pure 11-edges of all lengths from 1 to  $k$ . Therefore,  $T'_1$  allows a  $\rho$ -labeling.

Again the subtrees  $T'_0$  and  $T'_{01}$  share only a single vertex, namely  $0_0$ , and similarly  $T'_{01}$  and  $T'_1$  share only the vertex  $0_1$ . Hence, we have a non-symmetric spanning tree  $T'$  of order  $4k+2$  with a blended labeling, which has exactly one vertex, namely  $0_0$ , of degree  $m+1 = \Delta$  for every  $m$ , where  $2 \leq m \leq k$ , and the remaining vertices are of degree 2 or 1. □

By a similar argument we can show the following lemma.

**Lemma 8.** *There exists a non-symmetric spanning tree  $T$  on  $2n$  vertices with a maximum degree  $\Delta$  that allows a swapping labeling for every even  $n \geq 4$  and  $3 \leq \Delta \leq n$ .*

*Proof.* The proof is constructive. Let  $n = 2k$ . We distinguish three cases and show that trees  $T$  for  $k+2 \leq \Delta \leq 2k$  and  $T'$  for  $4 \leq \Delta \leq k+1$  and a broom  $B(2, 4k-3)$  for  $\Delta = 3$  allows a swapping labeling. We can split  $T$  and  $T'$ , respectively, to three subtrees  $T_0, T_{01}, T_1$  and  $T'_0, T'_{01}, T'_1$ , respectively (Figs. 4 and 5). Then  $T_0$  will contain only pure 00-edges,  $T_{01}$  only mixed edges and  $T_1$  only pure 11-edges of the tree  $T$  with a swapping labeling. Similarly  $T'$  with the swapping labeling is split to  $T'_0, T'_{01}, T'_1$  (Fig. 5).

*Case 1.* Let  $k+2 \leq \Delta \leq 2k$ . We take  $\Delta = k+m+2$ , where  $0 \leq m \leq k-2$ .

The subtree  $T_{01}$  will be the double star with mixed edges  $(0_0, 0_1), (0_0, 1_1), (0_0, 2_1), \dots, (0_0, (k-1)_1)$  of lengths from 0 to  $k-1$  and  $(0_1, 1_0), (0_1, 2_0), \dots, (0_1, (k-1)_0)$  of lengths  $2k-1, 2k-2, \dots, k+1$  (Fig. 4).

The subtree  $T_0$  contains pure 00-edges  $(0_0, k_0), (0_0, (k+1)_0), (0_0, (k+2)_0), \dots, (0_0, (k+m)_0)$  of lengths  $k, k-1, \dots, k-m$ , and pure 00-edges of the path  $0_0, (k+m+1)_0, (2k-1)_0, (k+m+2)_0, (2k-2)_0, (k+m+3)_0, (2k-3)_0, \dots, (\lfloor \frac{3k+m}{2} \rfloor)_0, (\lfloor \frac{3k+m}{2} \rfloor + 1)_0$  for odd  $k-m$  or  $0_0, (k+m+1)_0, (2k-1)_0, (k+m+2)_0, (2k-2)_0, (k+m+3)_0, (2k-3)_0, \dots, (\lfloor \frac{3k+m}{2} \rfloor + 1)_0, (\lfloor \frac{3k+m}{2} \rfloor)_0$  for even  $k-m$  (Fig. 4). The length of the path is  $k-m-1$ .

It is not difficult to check that the path contains pure 00-edges, of lengths  $k-m-1, k-m-2, \dots, 1$ .

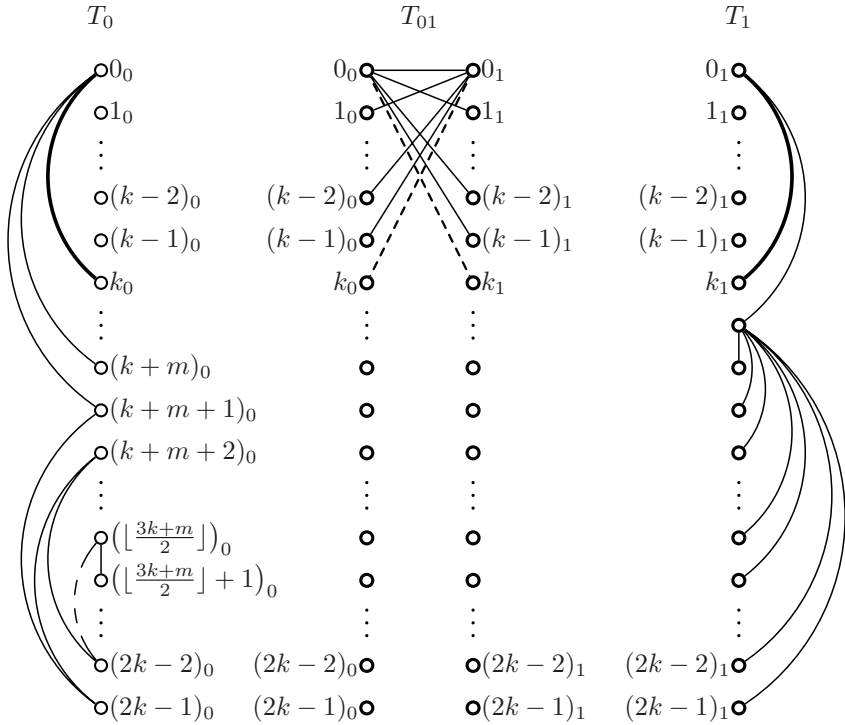
The subtree  $T_1$  contains pure 11-edges  $(0_1, k_1), (0_1, (k+1)_1)$  and  $((k+1)_1, (k+2)_1), ((k+1)_1, (k+3)_1), \dots, ((k+1)_1, (2k-1)_1)$  of lengths  $k, k-1$  and  $1, 2, \dots, k-2$ .

We see that the subtrees  $T_0$  and  $T_{01}$  share only single vertex  $0_0$  and similarly,  $T_{01}$  and  $T_1$  share only  $0_1$  in  $T$ . Further, if we replace the pure edge  $(0_0, k_0)$  by the mixed edge  $(0_0, k_1)$  and the pure edge  $(0_1, k_1)$  by the mixed edge  $(0_1, k_0)$  (both of length  $k$ ) in  $T$ , then we obtain the spanning tree isomorphic to  $T$  (dashed in Fig. 4).

Hence, we have a non-symmetric spanning tree  $T$  of order  $4k$  with a swapping labeling, which has a single vertex  $0_0$  of degree  $k+m+2 = \Delta$  for every  $m$ ,  $0 \leq m \leq k-2$ , and the remaining vertices are only of degrees  $k+2, k-1, 2$ , or 1.

*Case 2.* Let  $4 \leq \Delta \leq k+1$ . We take  $\Delta = m+3$ , where  $1 \leq m \leq k-2$ .

If  $k$  is even then  $T'_{01}$  is the path  $0_0, (k-1)_1, 1_0, (k-2)_1, \dots, (\frac{k}{2}-1)_0, (\frac{k}{2})_1, (\frac{k}{2})_0, (\frac{k}{2}-1)_1, \dots, (k-2)_0, 1_1, (k-1)_0, 0_1$  with mixed edges of lengths  $k-1, k-2,$



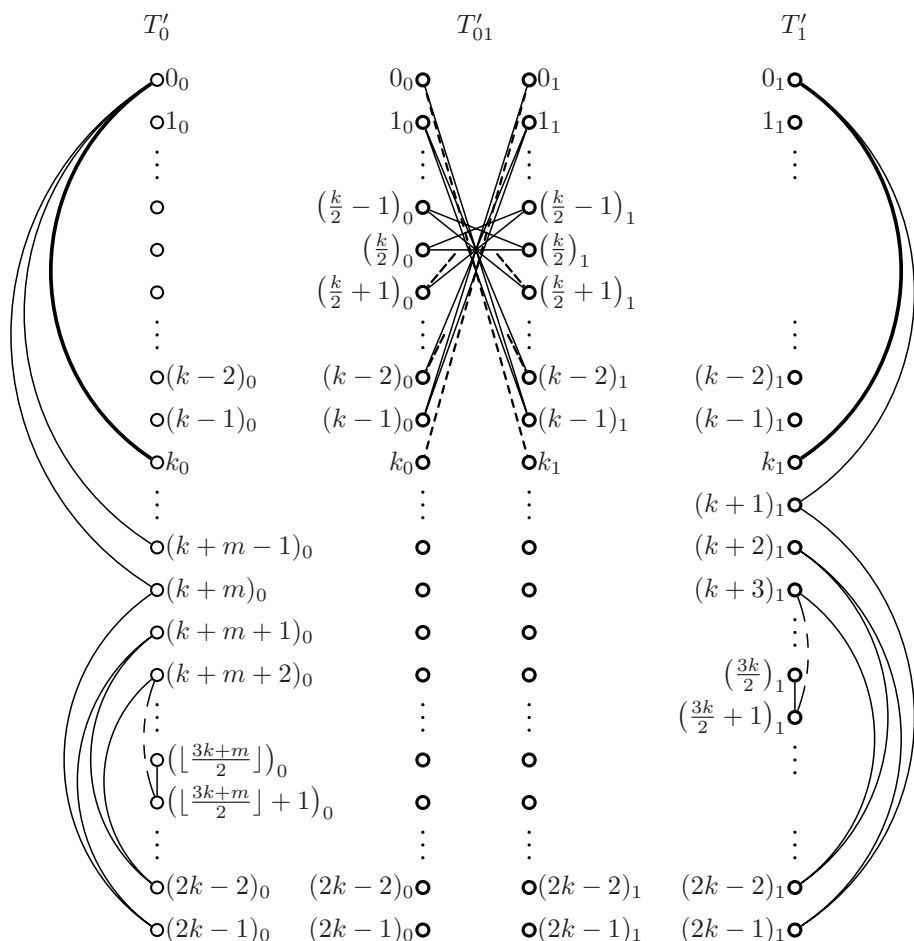
**Fig. 4.** A non-symmetric tree  $T$  with a swapping labeling for  $k - m$  odd and  $\Delta = k + m + 1$

$k - 3, \dots, 1, 0, 2k - 1, \dots, k + 3, k + 2, k + 1$  (Fig. 5). If  $k$  is odd then  $T'_{01}$  is the path  $0_0, (k - 1)_1, 1_0, (k - 2)_1, \dots, (\frac{k-1}{2})_1, (\frac{k-1}{2})_0, (\frac{k-1}{2})_1, (\frac{k-1}{2} + 1)_0, \dots, (k - 1)_0, 1_1, k_0, 0_1$  with mixed edges of lengths  $k - 1, k - 2, k - 3, \dots, 1, 0, 2k - 1, \dots, k + 3, k + 2, k + 1$ .

The subtree  $T'_0$  contains pure 00-edges  $(0_0, k_0), (0_0, (k + 1)_0), \dots, (0_0, (k + m - 1)_0)$  of lengths  $k, k - 1, \dots, k - m + 1$  and pure 00-edges of the path  $0_0, (k + m)_0, (2k - 1)_0, (k + m + 1)_0, (2k - 2)_0, (k + m + 2)_0, (2k - 3)_0, \dots, (\lfloor \frac{3k+m}{2} \rfloor)_0, (\lfloor \frac{3k+m}{2} \rfloor + 1)_0$  for odd  $k - m$  or  $0_0, (k + m)_0, (2k)_0, (k + m + 1)_0, (2k - 1)_0, (k + m + 2)_0, (2k - 2)_0, \dots, (\lfloor \frac{3k+m}{2} \rfloor + 1)_0, (\lfloor \frac{3k+m}{2} \rfloor)_0$  for even  $k - m$ . We see that the edges of the path have lengths  $k - m, k - m - 1, k - m - 2, \dots, 1$  (Fig. 5).

Finally, subtree  $T'_1$  contains pure 11-edge  $(0_1, k_1)$  of length  $k$  and pure 11-edges of the path  $0_1, (k + 1)_1, (2k - 1)_1, (k + 2)_1, (2k - 2)_1, \dots, (\frac{3k}{2} + 1)_1, (\frac{3k}{2})_1$  for  $k$  even or of the path  $0_1, (k + 1)_1, (2k)_1, (k + 2)_1, (2k - 1)_1, \dots, (\frac{3k+1}{2} - 1)_1, (\frac{3k+1}{2})_1$  for  $k$  odd. In both cases the subtree  $T'_1$  contains pure 11-edges of all lengths from 1 to  $k$ .

Again, we see that the subtrees  $T'_0$  and  $T'_{01}$  share only single vertex  $0_0$  and similarly,  $T'_{01}$  and  $T'_1$  share only vertex  $0_1$  in  $T'$ . If we replace the pure edge  $(0_0, k_0)$  by the mixed edge  $(0_0, k_1)$  and the pure edge  $(0_1, k_1)$  by the mixed edge



**Fig. 5.** A non-symmetric tree  $T'$  with a swapping labeling for  $k$  even,  $k - m$  even and  $\Delta = m + 3$

$(0_1, k_0)$  (both of length  $k$ ) in  $T'$ , then we obtain the spanning tree isomorphic to  $T'$ . Hence, we have the non-symmetric spanning tree  $T'$  of order  $4k$  with the swapping labeling, which has exactly one vertex  $0_0$  of degree  $m + 3 = \Delta$  for every  $m$ ,  $1 \leq m \leq k - 2$ , and the remaining vertices are of degrees 3, 2, or 1.

*Case 3.* Let  $\Delta = 3$ . We take a broom  $B(2, 4k - 3)$  and show that  $B(2, 4k - 3)$  admits a swapping labeling for  $k \geq 3$ . Again we can split  $B(2, 4k - 3)$  to three subtrees  $B_0, B_{01}, B_1$ . Then  $B_0$  will contain only pure 00-edges,  $B_{01}$  only mixed edges and  $B_1$  only pure 11-edges of the broom  $B(2, 4k - 3)$  with a swapping labeling.

The subtree  $B_{01}$  will be the path  $k_0, (2k - 1)_1, (k + 1)_0, (2k - 2)_1, \dots, (\frac{3k}{2} - 1)_0, (\frac{3k}{2})_1, (\frac{3k}{2})_0, (\frac{3k}{2} - 1)_1, \dots, (2k - 2)_0, (k + 1)_1, (2k - 1)_0, k_1$  for even  $k$  or

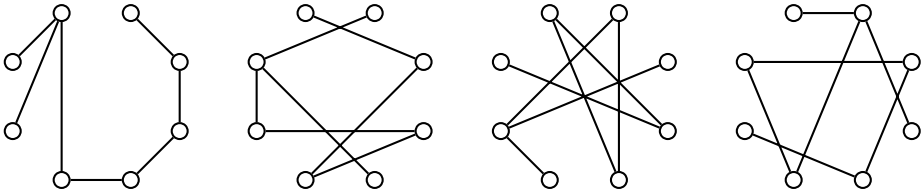


Fig. 6. A  $B(2, 5)$ -factorization of  $K_8$

$k_0, (2k - 1)_1, (k + 1)_0, (2k - 2)_1, \dots, (\frac{3k-1}{2} + 1)_1, (\frac{3k-1}{2})_0, (\frac{3k-1}{2})_1, (\frac{3k-1}{2} + 1)_1, \dots, (2k - 2)_0, (k + 1)_1, (2k - 1)_0, k_1$  for odd  $k$  with mixed edges of all lengths from 0 to  $2k - 1$  with the exception of length  $k$ .

The subtree  $B_0$  contains pure 00-edges of the path  $k_0, 0_0, (k - 1)_0, 1_0, \dots, (\frac{k}{2} - 2)_0, (\frac{k}{2} + 1)_0$  and the edges  $((\frac{k}{2} + 1)_0, (\frac{k}{2})_0), ((\frac{k}{2} + 1)_0, (\frac{k}{2} - 1)_0)$  for even  $k$  or edges of the path  $k_0, 0_0, (k - 1)_0, 1_0, \dots, (\frac{k-1}{2} + 2)_0, (\frac{k-1}{2} - 1)_0$  and the edges  $((\frac{k-1}{2} - 1)_0, (\frac{k}{2})_0), ((\frac{k-1}{2} - 1)_0, (\frac{k-1}{2} + 1)_0)$  for odd  $k$ . We see that  $B_0$  contains pure 00-edges of all lengths from 1 to  $k$ .

The subtree  $B_1$  is the path  $k_1, 0_1, (k - 1)_1, 1_1, \dots, (\frac{k}{2} - 1)_1, (\frac{k}{2})_1$  for even  $k$  or the path  $k_1, 0_1, (k - 1)_1, 1_1, \dots, (\frac{k-1}{2} + 1)_1, (\frac{k-1}{2})_1$  for odd  $k$ . Thus  $B_1$  contains pure 11-edges of all lengths from 1 to  $k$ .

We see that the subtrees  $B_0$  and  $B_{01}$  share only single vertex  $k_0$  and similarly,  $B_{01}$  and  $B_1$  share only vertex  $k_1$  in  $B(2, 4k - 3)$ . If we replace the edge  $(0_0, k_0)$  by the edge  $(0_0, k_1)$  and the edge  $(0_1, k_1)$  by the edge  $(0_1, k_0)$  (both of mixed length  $k$ ) in  $B(2, 4k - 3)$ , then we obtain the spanning tree isomorphic to  $B(2, 4k - 3)$ .

Hence,  $B(2, 4k - 3)$  allows the swapping labeling for every  $k \geq 3$ . For  $k = 2$  a  $B(2, 5)$ -factorization of  $K_8$  is in Fig. 6. □

Finally, from Lemmas 7 and 8, and Theorems 3 and 4 follows our main result.

**Theorem 9.** *Let  $n, \Delta$  be integers, where  $n \geq 3$  and  $3 \leq \Delta \leq n$ . There exists a non-symmetric spanning tree  $T$  on  $2n$  vertices with a maximum degree  $\Delta$  that factorizes  $K_{2n}$  for every  $n \geq 3$  and  $3 \leq \Delta \leq n$ .*

## References

1. Eldergill, P.: Decompositions of the complete graph with an even number of vertices. M.Sc. Thesis, McMaster University, Hamilton (1997)
2. Fronček, D.: Bi-cyclic Decompositions of Complete Graphs Into Spanning Trees. *Discrete Math.* 307, 1317–1322 (2007)
3. Kovářová, T.: Decompositions of Complete Graphs Into Isomorphic Spanning Trees With Given Diameters. *JCMCC* 54, 67–81 (2005)
4. Kovářová, T.: Spanning Tree Factorizations of Complete Graphs, Ph.D. Thesis. VŠB – Technical University of Ostrava (2004)
5. Rosa, A.: On Certain Valuations of the Vertices of a Graph, *Theory Graphs*. In: *Int. Symp. Rome 1966*, pp. 349–355. Gordon and Breach, N.Y. and Dunod, Paris (1967)



# On the Maximal Number of Cubic Subwords in a String\*

Marcin Kubica<sup>1</sup>, Jakub Radoszewski<sup>1</sup>, Wojciech Rytter<sup>1,2</sup>, and Tomasz Walen<sup>1</sup>

<sup>1</sup> Department of Mathematics, Computer Science and Mechanics,  
University of Warsaw, Warsaw, Poland

{kubica,jrad,rytter,walen}@mimuw.edu.pl

<sup>2</sup> Faculty of Mathematics and Informatics,  
Copernicus University, Toruń, Poland

**Abstract.** We investigate the problem of the maximum number of cubic subwords (of the form  $www$ ) in a given word. We also consider square subwords (of the form  $ww$ ). The problem of the maximum number of squares in a word is not well understood. Several new results related to this problem are produced in the paper. We consider two simple problems related to the maximum number of subwords which are squares or which are highly repetitive; then we provide a nontrivial estimation for the number of cubes. We show that the maximum number of squares  $xx$  such that  $x$  is not a primitive word (nonprimitive squares) in a word of length  $n$  is exactly  $\lfloor \frac{n}{2} \rfloor - 1$ , and the maximum number of subwords of the form  $x^k$ , for  $k \geq 3$ , is exactly  $n - 2$ . In particular, the maximum number of cubes in a word is not greater than  $n - 2$  either. Using very technical properties of occurrences of cubes, we improve this bound significantly. We show that the maximum number of cubes in a word of length  $n$  is between  $\frac{45}{100}n$  and  $\frac{4}{5}n$ .

## 1 Introduction

A repetition is a word composed (as a concatenation) of several copies of another word. The exponent is the number of copies. We are interested in natural exponents higher than 2. In [4] the authors considered also exponents which are not integer.

In this paper we investigate the bounds for the maximum number of highly repetitive subwords in a word of length  $n$ . A word is highly repetitive iff it is of the form  $x^k$  for some integer  $k$  greater than 2. In particular, cubes  $w^3$  and squares  $x^2$  with nonprimitive  $x$  are highly repetitive.

The subject of computing maximum number of squares and repetitions in words is one of the fundamental topics in combinatorics on words [16,19] initiated by A. Thue [25], as well as it is important in other areas: lossless compression, word representation, computational biology etc.

---

\* Supported by grant N206 004 32/0806 of the Polish Ministry of Science and Higher Education.

The behaviour of the function  $squares(n)$  of maximum number of squares is not well understood, though the subject of squares was studied by many authors, see [7,8,15]. The best known results related to the value of  $squares(n)$  are, see [11,13,14]:

$$n - o(n) \leq squares(n) \leq 2n - O(\log n)$$

In this paper we concentrate on larger powers of words and show that in this case we can have much better estimations. Let  $cubes(n)$  denote the maximum number of cubes in a word of length  $n$ . We show that:

$$\frac{45}{100} n \leq cubes(n) \leq \frac{4}{5} n$$

There are known efficient algorithms for the computation of integer powers in words, see [13,9,20,21].

The powers in words are related to maximal repetitions, also called *runs*. It is surprising that the bounds for the number of runs are much tighter than for squares, this is due to the work of many people [2,5,6,12,17,18,22,23,24].

Our main result is a new estimation of the number of cubic subwords. We use a new interesting technique in the analysis: the proof of the upper bound is reduced to the proof of an invariant of some abstract algorithm (in our invariant lemma). There is still some gap between upper and lower bound but it is much smaller than the corresponding gap for the number of squares.

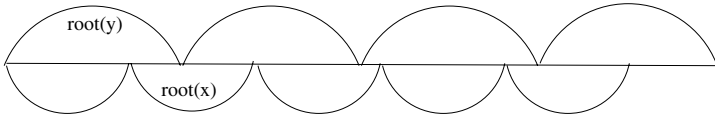
## 2 Basic Properties of Highly Repetitive Subwords

We consider *words* over a finite alphabet  $A$ ,  $u \in A^*$ ; by  $\varepsilon$  we denote an empty word; the positions in a word  $u$  are numbered from 1 to  $|u|$ . For  $u = u_1 \dots u_k$ , by  $u[i..j]$  we denote a *subword* of  $u$  equal to  $u_i \dots u_j$ . We say that a positive integer  $p$  is a *period* of a word  $u = u_1 \dots u_k$  if  $u_i = u_{i+p}$  holds for  $1 \leq i \leq k - p$ . If  $w^k = u$  ( $k$  is a non-negative integer) then we say that  $u$  is the  $k^{th}$  power of the word  $w$ .

The *primitive root* of a word  $u$ , denoted  $root(u)$ , is the shortest word  $w$ , such that  $w^k = u$  for some positive  $k$ . We call a word  $u$  *primitive* if  $root(u) = u$ , otherwise it is called *nonprimitive*. It can be proved that the primitive root of a word  $u$  is the only primitive word  $w$ , such that  $w^k = u$  for some positive  $k$ . A *square* is the  $2^{nd}$  power of some word, and an *np-square* (a nonprimitive square) is a square of a word, that is **not** primitive. A *cube* is a  $3^{rd}$  power of some word.

In this paper we focus on the last occurrences of subwords. Hence, whenever we say that word  $u$  *occurs at position*  $i$  of the word  $v$  we mean its **last** occurrence, that is  $v[i..i + |u| - 1] = u$  and  $v[j..j + |u| - 1] \neq u$  for  $j > i$ . The following lemma is used extensively throughout the article.

**Lemma 1 (Periodicity Lemma [10,19]).** *If a word of length  $n$  has two such periods  $p$  and  $q$ , that  $p + q \leq n + \gcd(p, q)$ , then  $\gcd(p, q)$  is also a period of the word.*



**Fig. 1.** The situation when one hr-word is a (long) prefix of another hr-word implies that  $root(x) = root(y)$ , consequently  $x$  is a suffix of  $y$

We often use, so called, weak version of this lemma, where we only assume that  $p + q \leq n$ .

A word is said to be *highly repetitive* (hr-word) if it is a  $k^{th}$  power of a nonempty word, for  $k \geq 3$ .

**Lemma 2.** *If a hr-word  $x$  is a prefix of a hr-word  $y$  and  $|x| \geq |y| - |root(y)|$ , then  $x$  is also a suffix of  $y$ .*

*Proof.* Due to the periodicity lemma, both words have the same smallest period and it is a common divisor of the lengths of their primitive roots, see Figure 1. Consequently, we have  $root(x) = root(y)$  and  $x$  is a suffix of  $y$ . □

**Lemma 3.** *Assume that  $x$  and  $y$  are two hr-words, where  $y = z^3$  and  $x$  is a subword of  $y$  starting at position  $j \leq \lceil \frac{|root(z)|}{2} \rceil + 1$  and ending at position  $k > |z^2|$ . Then,  $|root(x)| = |root(y)|$ .*

*Proof.* Let  $x = w^k$ , for some  $k \geq 3$ . First, let us note that if the hypothesis of the lemma holds, then  $|x| > \frac{3}{2}|z|$  — this can be verified by careful examination of simple cases: for even and odd values of  $|z|$ . Let us also observe, that  $|root(x)|$  and  $|root(y)|$  are both periods of  $x$ . Moreover:

$$|x| = |w^k| = |w| + \frac{k-1}{k}|x| \geq |w| + \frac{2}{3}|x| > |w| + |z| \geq |root(x)| + |root(y)|$$

From this, by the periodicity lemma, we obtain that  $g = \gcd(|root(x)|, |root(y)|)$  is also a period of  $x$ . However,  $root(x)$  and  $root(y)$  are subwords of  $x$ , so  $g = |root(x)| = |root(y)|$ . □

### 3 Some Simple Bounds

In this section we give some simple estimations of the number of square subwords with nonprimitive roots and cubic subwords.

**Lemma 4.** *Let  $u$  be a word. Let us consider highly repetitive subwords of  $u$  of the form  $v^k$ , for  $k \geq 3$  and  $v$  primitive. For each such subword we consider its (last) occurrence in  $u$ . For each position  $i$  in  $u$ , at most one such subword can have its (last) occurrence at position  $i$ .*

*Proof.* Let us assume, that we have two different hr-words  $x$  and  $y$  with their last occurrences starting at position  $i$ , and let us assume that  $x$  is shorter. Then, we have:  $|x| \geq |y| - |\text{root}(y)|$ , otherwise the considered occurrence of  $x$  would not be the last one.

Now we can apply Lemma 2 —  $x$  is not only a prefix of  $y$ , but also its suffix. Hence,  $x$  appears later in the text and the last occurrence of  $x$  in  $u$  does not start at position  $i$ . This contradiction proves, that the assumption that the last occurrences of  $x$  and  $y$  start at position  $i$  is false.  $\square$

The following fact is a straightforward consequence of Lemma 4.

**Theorem 1.** *The maximum number of highly repetitive subwords of a word of length  $n \geq 2$  is exactly  $n - 2$ .*

*Proof.* From Lemma 4 we know, that at each position there can be at most one last occurrence of a nonempty hr-word. Moreover, the minimum possible length of such a word is 3. So, it cannot occur at positions  $n$  and  $n - 1$ . On the other hand, this upper bound is reached by the word  $a^n$ .  $\square$

As a corollary, we obtain a simple upper bound for the number of cubes, since cubes are hr-words.

**Corollary 1.** *Let us consider a word  $u$  of length  $n$ . The number of nonempty cubes appearing in  $u$  is not greater than  $n - 2$ .*

We improve this upper bound substantially in the next section. However, it requires a lot of technicalities. Another implication of Theorem 1 is a tight bound for the number of np-squares.

**Theorem 2.** *Let  $u$  be a word of length  $n$ . The maximum number of nonempty np-squares appearing in  $u$  is exactly  $\lfloor \frac{n}{2} \rfloor - 1$ .*

*Proof.* Each nonempty np-square can be viewed as  $v^{2i}$  for some nonempty primitive  $v$  and  $i \geq 2$ . However, each such np-square contains a subword  $v^{2i-1}$ , which is not an np-square, but still a hr-word. Hence, the number of nonempty subwords of the form  $v^{2i-1}$  (for primitive  $v$  and  $i \geq 2$ ), appearing in the given word, is not smaller than the number of nonempty np-squares.

Please observe, that Theorem 1 limits the total number of both subwords of the form  $v^{2i}$  and  $v^{2i-1}$ , by  $n - 2$ . Hence, the total number of nonempty np-squares appearing in the given word is not greater than  $\frac{n}{2} - 1$ , and since it is integer, it is not greater than  $\lfloor \frac{n}{2} \rfloor - 1$ . On the other hand, this upper bound is reached by the word  $a^n$ .  $\square$

## 4 The Number of Cubic Subwords

In this section we show, that the upper bound on the number of different cubes in a word of length  $n$  is  $\frac{4}{5}n$ . We also show example words containing  $0.45n$  different cubes. The following lemma states the main idea of the proof of the upper bound.

**Lemma 5.** *Let  $v^3$  and  $w^3$  be two nonempty cubes occurring in a word  $u$  at positions  $i$  and  $j$  respectively, such that:*

$$i < j \leq i + \left\lceil \frac{|\text{root}(v)|}{2} \right\rceil$$

*Then either  $|\text{root}(w)| = |\text{root}(v)|$  or  $|\text{root}(w)| \geq 2 \cdot |\text{root}(v)| - (j - i - 1)$ .*

*Proof.* Let us denote  $p = |\text{root}(v)|$ ,  $q = |\text{root}(w)|$ , and let  $k$  be the position of the last letter of  $w^3$ .

Let us first consider the case, when the (last) occurrence of  $w^3$  is totally inside  $v^3$ . Please note, that  $k$  must then be within the last of the three  $v$ 's, since otherwise  $w^3$  would occur in  $u$  at position  $j + p$  or further (see also Fig. 2). Hence, due to Lemma 3, we obtain  $q = p$ .

In the opposite case, let  $x$  be the maximal prefix of  $w^3$  that lays inside  $v^3$ . If  $p \neq q$  then, by the periodicity lemma,  $p + q$  must be greater than  $|x|$  (please note that if  $p + q \leq |x|$  then obviously both  $\text{root}(v)$  and  $\text{root}(w)$  are subwords of  $x$ ). Therefore:

$$p + q > |x| > |v^3| - (j - i) \geq 3p - (j - i)$$

and hence  $q \geq 2p - (j - i) + 1$ . □

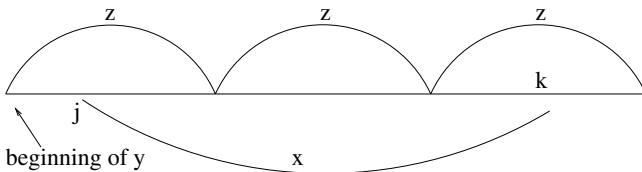
Let us introduce a notion of *p-occurrence*.

**Definition 1.** *A p-occurrence is the (last) occurrence of a cube with primitive root of length p.*

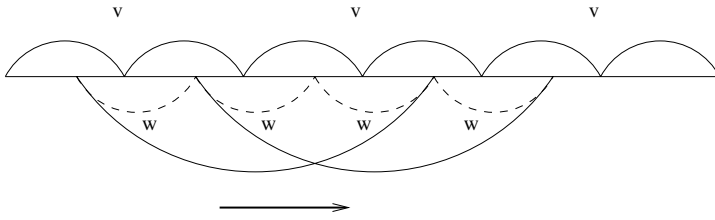
It turns out, that the primitive roots of cubes appearing close to each other cannot be arbitrary. It is formally expressed by the following lemma.

**Lemma 6.** *Let  $a_1, a_2, \dots, a_{p+1}$  be an increasing sequence of positions in a word  $u$ , such that  $a_{j+1} \leq a_j + p$  for  $j = 1, 2, \dots, p$ . It is not possible that there are p-occurrences at all these positions.*

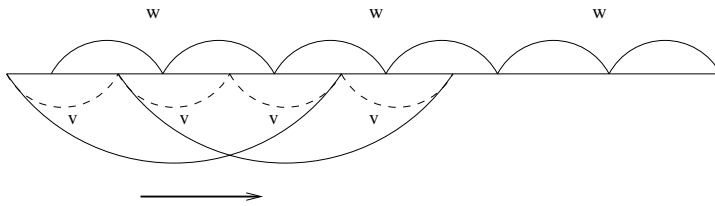
*Proof.* Let us assume, to the contrary, that at each of the positions  $a_1, a_2, \dots, a_{p+1}$  there is a  $p$ -occurrence. Please note, that the inequalities from the hypothesis of the lemma imply that the primitive roots of cubes occurring at these positions are all cyclic rotations of each other. There are only  $p$  different



**Fig. 2.** The situation from Lemma 3



**Fig. 3.** Positions of cubes  $v^3$  and  $w^3$  for the case  $l < k$ :  $a_{j+1}$  is not the last occurrence of  $w^3$



**Fig. 4.** Positions of cubes  $v^3$  and  $w^3$  for the case  $k < l$ :  $a_j$  is not the last occurrence of  $v^3$

rotations of such primitive roots, so due to the pigeonhole principle, some two of them must be equal.

It suffices to show, that all these cubes have the same length, because then two of them must be equal, and one of them is not the last occurrence of the cube.

So, let us assume to the contrary, that some of the considered cubes have different lengths. Let  $a_j$  and  $a_{j+1}$  be such two considered positions, that cubes ( $v^3$  and  $w^3$  respectively) occurring at these positions have different lengths ( $3kp$  and  $3lp$  respectively, for  $k \neq l$ ). Let us consider two cases. If  $l < k$ , then  $3kp - 3lp \geq 3p$ , and  $w^3$  occurs in  $u$  at position  $a_{j+1} + p$  or further.

On the other hand, if  $k < l$ , then  $3lp - 3kp \geq 3p$  and  $v^3$  appears in  $u$  at position  $a_j + p$  or further. So, in both cases we obtain a contradiction. Hence it is not possible, that the lengths of the cubes differ.  $\square$

Let us introduce a notion of independent prefixes.

**Definition 2.** We say that  $v$  is the independent prefix of  $u$  if it is the shortest prefix of  $u$ , that is:

1. a 1-letter word, if there is no occurrence of a cube at the first position of  $u$ , or otherwise
2. such a word  $v$ , for which the last occurrence of a cube in  $u$ , that starts within  $v$  is a  $q$ -occurrence (for some  $q \geq 1$ ), and after this occurrence there are exactly  $\lceil \frac{q}{2} \rceil$  positions (within  $v$ ) without any occurrences of cubes (in  $u$ ).

It is not obvious, that the above definition is valid. Therefore, we prove the following lemma:

**Lemma 7.** *For every word  $u$ , there exists an independent prefix  $v$  of  $u$ .*

*Proof.* If there is no occurrence of a cube at the first position of  $u$ , then obviously  $v = u[1..1]$ . In the opposite case, let us assume that the independent prefix does not exist. Let  $q$  be the maximum such value, that some  $q$ -occurrence exists in  $u$ , and let  $i$  be the rightmost position in  $u$ , that contains a  $q$ -occurrence. From Lemma 5,  $\lceil \frac{q}{2} \rceil$  positions following  $i$  do not contain any occurrences of cubes. So, the prefix  $u[1..i + \lceil \frac{q}{2} \rceil]$  satisfies the properties of an independent prefix — a contradiction.  $\square$

**Lemma 8.** *Let  $v$  be the independent prefix of  $u$ . The number of different nonempty cubes that occur in  $u$  and start within  $v$  is not greater than  $\frac{4}{5}|v|$ .*

*Proof.* Please note, that if  $v$  satisfies the first condition of Definition 2, then the conclusion trivially holds. Therefore, from now on we assume that  $|v| > 1$ .

Let  $c_i$  be a sequence describing the occurrences starting within  $v$ :  $c_i = 0$  iff there are no occurrences in position  $v[i]$ , and  $c_i = q$  iff there is a  $q$ -occurrence in position  $v[i]$ . Please note that:

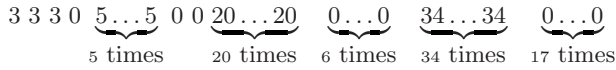
- a) Let  $i < j$  be such indices, that  $c_i, c_j > 0$  and  $c_{i+1} = \dots = c_{j-1} = 0$ . If  $j - i > \lceil \frac{c_i}{2} \rceil$ , then the prefix of  $u$  of length  $i + \lceil \frac{c_i}{2} \rceil$  or shorter is an independent prefix of  $u$  — a contradiction. So, for any such  $i$  and  $j$  we have  $j - i \leq \lceil \frac{c_i}{2} \rceil$ .
- b) From Lemma 5 we obtain that  $c_j \geq 2c_i - (j - i - 1)$ .
- c) From Lemma 6 and due to a) we have that no  $q + 1$  consecutive positive elements of  $c$  can be equal  $q$ .

From now on, we abstract from the actual word  $u$ , and focus only on the properties of sequence  $c$ . We show, that the ratio  $R$  of non-zero elements of  $c$  to the length of  $c$  does not exceed  $\frac{4}{5}$ .

Let us observe that if  $c$  contains such a pair of equal elements  $c_i = c_j > 0$ , that all the elements between them are equal zero, then all the elements between  $c_i$  and  $c_j$  can be removed from  $c$  without decreasing  $R$ . Also, if  $c$  contains a subsequence of consecutive elements equal to  $q$  ( $q > 0$ ) of length less than  $q$  then this subsequence can be extended to length  $q$  without decreasing  $R$ . Let  $c'$  be a sequence obtained from  $c$  by performing the described modification steps (as many times as possible). Please note that none of these steps violates properties b) or c). We will show, that even for  $c'$  the ratio of non-zero elements does not exceed  $\frac{4}{5}$ .

Every possible sequence  $c'$  can be generated by the (nondeterministic) pseudocode shown below. The following variables are used in the pseudocode:

- $p$  — the value of the last positive element of  $c'$
- $len$  — the length of the sequence  $c'$  without  $\lceil p/2 \rceil$  trailing zeros
- $occ$  — the number of positive elements in  $c'$
- $l$  — the gap between consecutive different positive elements of  $c'$
- $\alpha$  — the difference between the actual value of a positive element of  $c'$  and the lower bound from Lemma 5



**Fig. 5.** An example of sequence  $c'$ . The length of the sequence is 88 and it contains 62 positive elements. The ratio is  $62/88 \approx 0.70 < 4/5$ .

Each step of the **repeat-until** loop corresponds to extending sequence  $c'$ , i.e. adding  $l$  zeros and  $p$  elements of value  $p$ .

Note that the algorithm specified by the pseudocode is nondeterministic in a few different aspects — the initial value of  $p$ , the number of steps of the **repeat-until** loop and values of  $l$  and  $\alpha$ .

```


$p :=$  some positive integer;  

 $occ := p; \quad len := p;$   

output:  $\underbrace{p\dots p}_{p\ \text{times}}$   

repeat  

    Invariant  $I(p, occ, len) : \frac{occ}{len + \frac{p}{2}} \leq \frac{4}{5}$ .  

     $l :=$  some integer from interval  $[0, \lceil \frac{p}{2} \rceil]$ ;  

     $\alpha :=$  some non-negative integer;  

     $p := 2p - l + \alpha;$   

     $occ := occ + p;$   

     $len := len + l + p;$   

    output:  $\underbrace{0\dots 0}_{l\ \text{times}} \underbrace{p\dots p}_{p\ \text{times}}$   

until done


```

In order to prove the  $\frac{4}{5}$  bound, we need to show that inequality

$$\frac{occ}{len + \lceil \frac{p}{2} \rceil} \leq \frac{4}{5}$$

holds for every possible execution of the above pseudocode. But this inequality is a consequence of the fact that  $I(p, occ, len)$  is an invariant of the **repeat-until** loop (Lemma 9). □

**Lemma 9 (Invariant Lemma).** *Inequality  $I(p, occ, len)$ :*

$$\frac{occ}{len + \frac{p}{2}} \leq \frac{4}{5}$$

*is an invariant of the **repeat-until** loop from the above pseudocode.*

*Proof.* It is easy to check that before the first execution of the **repeat-until** loop inequality  $I(p, occ, len)$  holds. Therefore, we only need to prove that if



$n$	word	#cubes	ratio
20	01110101011011011000	7	0.35
30	000000110110110101101011010101	11	0.36
40	1101101101110111011100010001000100100100	16	0.40
50	111111111100100100101001010010100101010010101000	20	0.40
60	10100101001010010101001010010100101001010010101001010 1001010100	25	0.41
70	000000110110110101101011010110101101011010110101101101 01011010101101010111	30	0.42
80	11011011010110110101101101011010110101101011010110101 011010110101011010101101010111	34	0.42
90	11101101101110110110111011011011101101110110110110111 0110111011011011101101110110111011101110	40	0.44
100	1000101010010101001010100101001010100101001010010101001 0100101001010100101001010010101001010010100101010111	44	0.44
200	0000100010001000010001000100001000100010000100001000100 00100010001000010001000100001000100010001000100010000 1000100010000100010000100010001000010001000010000100010 0010000100010000100010000100001000010000110111011101110	91	0.45

Fig. 6. Examples of words with high number of distinct cubic subwords

$I(p, occ, len)$  holds then  $I(p', occ', len')$  also holds, where  $p', occ'$  and  $len'$  are the values obtained as a result of a single step of the **repeat-until** loop, i.e.:

$$p' = 2p - l + \alpha, \quad occ' = occ + 2p - l + \alpha, \quad len' = len + 2p + \alpha$$

Let us restate  $I(p', occ', len')$  equivalently in the following way:

$$5 \cdot occ + 10p - 5l + 5\alpha \leq 4 \cdot len + 8p + 4\alpha + 4 \cdot \frac{2p - l + \alpha}{2} \tag{1}$$

Since  $I(p, occ, len)$  can be expressed as  $5 \cdot occ \leq 4 \cdot len + 4 \cdot \frac{p}{2}$ , in order to show (1), it is sufficient to prove that:

$$10p - 5l + 5\alpha \leq 8p + 4\alpha + 2 \cdot (2p - l + \alpha) - 2p \tag{2}$$

As a result of some rearrangement, (2) can be expressed as  $0 \leq 3l + \alpha$  and this inequality trivially holds. □

**Theorem 3.** *The number of different nonempty cubes that occur in a word of length  $n$  is not greater than  $\frac{4}{5}n$ .*

*Proof.* This theorem is a consequence of Lemmas 7 and 8 — it can be proved by simple induction on  $n$ , where the inductive step consists of removing the independent prefix. □

**Theorem 4.** *For infinitely many positive integers  $n$  there exist words  $x$  of length  $n$  with at least  $0.45 \cdot n$  different nonempty cubic subwords.*

*Proof.* A trivial lower bound on the number of different cubic subwords is the word  $a^n$  with  $\lfloor \frac{n}{3} \rfloor$  cubic occurrences. The table presented in Figure 6 contains examples of some words with higher number of cubic subwords. These words have been computed using extensive computer experiments. For example, we have found a word  $w$  of length 200 over binary alphabet containing 91 different nonempty cubic subwords. For any positive integer  $k$  we can construct a word  $x = w_1 w_2 \dots w_k$  (over the alphabet  $\{0, 1, \dots, 2k-1\}$ ), where  $w_i$  is a word created from  $w$  by replacing all occurrences of letter 0 by  $2(i-1)$ , and 1 by  $2i-1$ . Such word  $x$  has length  $200k$  and contains at least  $91k$  cubic subwords (91 in each subword  $w_i$ ). This gives a ratio  $91/200 \cdot n > 0.45 \cdot n$ .  $\square$

## 5 Conclusions

In this paper we prove a tight bound for the number of nonprimitive squares in a word of length  $n$ . Unfortunately, this does not improve the overall bound of the number of squares — the main open problem is improving the bound for primitive squares.

We also give some estimations of the number of cubes in a string of length  $n$ . Although they are much better than the best known estimations for squares in general, they can still be subject to improvement — both the lower and the upper bound do not seem to be tight.

## References

1. Apostolico, A., Preparata, F.P.: Optimal off-line detection of repetitions in a string. *Theor. Comput. Sci.* 22, 297–315 (1983)
2. Baturo, P., Piątkowski, M., Rytter, W.: The number of runs in sturmian words. In: Ibarra, O.H., Ravikumar, B. (eds.) CIAA 2008. LNCS, vol. 5148, pp. 252–261. Springer, Heidelberg (2008)
3. Crochemore, M.: An optimal algorithm for computing the repetitions in a word. *Inf. Process. Lett.* 12(5), 244–250 (1981)
4. Crochemore, M., Fazekas, S.Z., Iliopoulos, C.S., Jayasekera, I.: Bounds on powers in strings. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 206–215. Springer, Heidelberg (2008)
5. Crochemore, M., Ilie, L.: Maximal repetitions in strings. *J. Comput. Syst. Sci.* 74(5), 796–807 (2008)
6. Crochemore, M., Ilie, L., Tinta, L.: Towards a solution to the “Runs” conjecture. In: Ferragina, P., Landau, G.M. (eds.) CPM 2008. LNCS, vol. 5029, pp. 290–302. Springer, Heidelberg (2008)
7. Crochemore, M., Rytter, W.: Squares, cubes, and time-space efficient string searching. *Algorithmica* 13(5), 405–425 (1995)
8. Crochemore, M., Rytter, W.: *Jewels of Stringology*. World Scientific, Singapore (2003)
9. Damanik, D., Lenz, D.: Powers in sturmian sequences. *Eur. J. Comb.* 24(4), 377–390 (2003)
10. Fine, N.J., Wilf, H.S.: Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society* 16, 109–114 (1965)

11. Freankel, A.S., Simpson, J.: How many squares can a string contain? *J. of Combinatorial Theory Series A* 82, 112–120 (1998)
12. Giraud, M.: Not so many runs in strings. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) *LATA 2008*. LNCS, vol. 5196, pp. 232–239. Springer, Heidelberg (2008)
13. Ilie, L.: A simple proof that a word of length  $n$  has at most  $2n$  distinct squares. *J. of Combinatorial Theory Series A* 112, 163–164 (2005)
14. Ilie, L.: A note on the number of squares in a word. *Theoretical Computer Science* 380, 373–376 (2007)
15. Iliopoulos, C.S., Moore, D., Smyth, W.F.: A characterization of the squares in a fibonacci string. *Theor. Comput. Sci.* 172(1-2), 281–291 (1997)
16. Karhumaki, J.: Combinatorics on words. Notes in pdf
17. Kolpakov, R.M., Kucherov, G.: Finding maximal repetitions in a word in linear time. In: *FOCS*, pp. 596–604 (1999)
18. Kolpakov, R.M., Kucherov, G.: On maximal repetitions in words. In: Ciobanu, G., Păun, G. (eds.) *FCT 1999*. LNCS, vol. 1684, pp. 374–385. Springer, Heidelberg (1999)
19. Lothaire, M.: *Applied Combinatorics on Words*. Cambridge University Press, Cambridge (2005)
20. Main, M.G.: Detecting leftmost maximal periodicities. *Discrete Applied Mathematics* 25(1-2), 145–153 (1989)
21. Main, M.G., Lorentz, R.J.: An  $o(n \log n)$  algorithm for finding all repetitions in a string. *J. Algorithms* 5(3), 422–432 (1984)
22. Puglisi, S.J., Simpson, J., Smyth, W.F.: How many runs can a string contain? *Theor. Comput. Sci.* 401(1-3), 165–171 (2008)
23. Rytter, W.: The number of runs in a string: Improved analysis of the linear upper bound. In: Durand, B., Thomas, W. (eds.) *STACS 2006*. LNCS, vol. 3884, pp. 184–195. Springer, Heidelberg (2006)
24. Rytter, W.: The number of runs in a string. *Inf. Comput.* 205(9), 1459–1469 (2007)
25. Thue, A.: Uber unendliche zeichenreihen. *Norske Vid. Selsk. Skr. I Math-Nat.* 7, 1–22 (1906)

# Solution of Peter Winkler's Pizza Problem\*

Josef Cibulka<sup>1</sup>, Jan Kynčl<sup>2</sup>, Viola Mészáros<sup>2,3</sup>, Rudolf Stolař<sup>1</sup>, and Pavel Valtr<sup>2</sup>

<sup>1</sup> Department of Applied Mathematics,  
Charles University, Faculty of Mathematics and Physics,  
Malostranské nám. 25, 118 00 Praha 1, Czech Republic  
cibulka@kam.mff.cuni.cz, ruda@kam.mff.cuni.cz

<sup>2</sup> Department of Applied Mathematics and  
Institute for Theoretical Computer Science,  
Charles University, Faculty of Mathematics and Physics,  
Malostranské nám. 25, 118 00 Praha 1, Czech Republic  
kync1@kam.mff.cuni.cz

<sup>3</sup> Bolyai Institute, University of Szeged,  
Aradi vértanúk tere 1, 6720 Szeged, Hungary  
viola@math.u-szeged.hu

**Abstract.** Bob cuts a pizza into slices of not necessarily equal size and shares it with Alice by alternately taking turns. One slice is taken in each turn. The first turn is Alice's. She may choose any of the slices. In all other turns only those slices can be chosen that have a neighbor slice already eaten. We prove a conjecture of Peter Winkler by showing that Alice has a strategy for obtaining  $4/9$  of the pizza. This is best possible, that is, there is a cutting and a strategy for Bob to get  $5/9$  of the pizza. We also give a characterization of Alice's best possible gain depending on the number of slices. For a given cutting of the pizza, we describe a linear time algorithm that computes Alice's strategy gaining at least  $4/9$  of the pizza and another algorithm that computes the optimal strategy for both players in any possible position of the game in quadratic time. We distinguish two types of turns, shifts and jumps. We prove that Alice can gain  $4/9$ ,  $7/16$  and  $1/3$  of the pizza if she is allowed to make at most two jumps, at most one jump and no jump, respectively, and the three constants are the best possible.

## 1 Introduction

Peter Winkler posed the following problem at the conference Building Bridges, honouring the 60th birthday of László Lovász, in Budapest in 2008. Bob and Alice are sharing a pizza. Bob cuts the pizza into slices of not necessarily equal size. Afterwards they take turns alternately to divide it among themselves. One

---

\* Work on this paper was supported by the project 1M0545 of the Ministry of Education of the Czech Republic. Viola Mészáros was also partially supported by OTKA Grant K76099 and by the grant no. MSM0021620838 of the Ministry of Education of the Czech Republic. Josef Cibulka and Rudolf Stolař were also supported by the Czech Science Foundation under the contract no. 201/09/H057.

slice is taken in each turn. In the first turn Alice takes any slice. In the forthcoming turns one may take a slice if it is adjacent to some previously taken slice. This is called the Polite Pizza Protocol. How much of the pizza can Alice gain?

The original puzzle, that is determining whether Bob can get more than half of the pizza, was devised by Dan Brown in 1996. Bob can easily ensure for himself one half of the pizza. For example, he may cut the pizza into an even number of slices of equal size. Then Bob always obtains exactly one half. Peter Winkler found out that Bob can actually get  $5/9$  of the pizza if he cuts the pizza properly—see Theorem 4 for such cuttings. He conjectured that Alice can obtain  $4/9$  of the pizza for any cutting. The main aim of this paper is to show a strategy of Alice proving this conjecture.

The pizza after Bob’s cutting may be represented by a circular sequence  $P = p_0p_1 \dots p_{n-1}$  and by the sizes  $|p_i| \geq 0$  (for  $i = 0, 1, \dots, n - 1$ ); for simplicity of notation, throughout the paper we do not separate the elements of (circular) sequences by commas. The size of  $P$  is defined by  $|P| := \sum_{i=0}^{n-1} |p_i|$ . Throughout the paper the indices are counted modulo  $n$ .

For  $1 < j \leq n$ , if one of the players chooses a slice  $p_i$  in the  $(j - 1)$ -st turn and the other player chooses  $p_{i-1}$  or  $p_{i+1}$  in the  $j$ -th turn, then the  $j$ -th turn is called a *shift*, otherwise it is called a *jump*.

If some strategy of a player allows the player to make at most  $j$  jumps, then we call it a  *$j$ -jump strategy*. We remark that given a circular sequence  $P$  of length  $n$ , Alice has exactly  $n$  zero-jump strategies on  $P$ , determined by Alice’s first choice. Except for her first and her last turn, Alice has two choices in each turn and exactly one of them is a shift and the other one is a jump. The last turn is always a shift in which she takes the last slice. Let  $\Sigma$  be a particular strategy of one of the players. We say that  $\Sigma$  is a strategy with *gain*  $g$  if it guarantees the player a subset of slices with sum of sizes at least  $g$ .

If the number of slices is even, Alice has the following zero-jump strategy with gain  $|P|/2$ . She partitions the slices of the pizza into two classes, even and odd, according to their parity in  $P$ . In the first turn Alice takes a slice from the class with the sum of slice sizes at least  $|P|/2$ . In all her forthcoming turns she makes shifts, thus forcing Bob to eat from the other class in each of his turns.

Here is our main result.

**Theorem 1.** *For any  $P$ , Alice has a two-jump strategy with gain  $4|P|/9$ .*

For  $n \geq 1$ , let  $g(n)$  be the maximum  $g \in [0, 1]$  such that for any cutting of the pizza into  $n$  slices, Alice has a strategy with gain  $g|P|$ .

**Theorem 2.** *Let  $n \geq 1$ . Then*

$$g(n) = \begin{cases} 1 & \text{if } n = 1, \\ 4/9 & \text{if } n \in \{15, 17, 19, 21, \dots\}, \\ 1/2 & \text{otherwise.} \end{cases}$$

*Moreover, Alice has a zero-jump strategy with gain  $g(n)|P|$  when  $n$  is even or  $n \leq 7$ , she has a one-jump strategy with gain  $g(n)|P|$  for  $n \in \{9, 11, 13\}$ , and she has a two-jump strategy with gain  $g(n)|P|$  for  $n \in \{15, 17, 19, 21, \dots\}$ .*

If we make a restriction on the number of Alice's jumps we get the following results.

**Theorem 3.** (a) *Alice has a zero-jump strategy with gain  $|P|/3$  and the constant  $1/3$  is the best possible.*

(b) *Alice has a one-jump strategy with gain  $7|P|/16$  and the constant  $7/16$  is the best possible.*

Theorem 4, together with Theorem 2, describes all minimal cuttings for which Bob has a strategy with gain  $5|P|/9$ .

**Theorem 4.** *For any  $\omega \in [0, 1]$ , Bob has a one-jump strategy with gain  $5|P|/9$  if he cuts the pizza into 15 slices as follows:  $P_\omega = 0010100(1 + \omega)0(2 - \omega)00202$ . These cuttings describe, up to scaling, rotating and flipping the pizza upside-down, all the pizza cuttings into 15 slices for which Bob has a strategy with gain  $5|P|/9$ .*

*If Bob can make slices of only two different sizes, then he can gain  $5|P|/9$  by cutting the pizza into 21 slices of sizes 0 and 1 as follows: 001010010101001010101. This cutting describes, up to scaling, rotating and flipping the pizza upside-down, all the pizza cuttings into at most 21 slices of at most two different sizes for which Bob has a strategy with gain  $5|P|/9$ .*

We omit the proof of Theorem 2, Theorem 3 and Theorem 4 from this extended abstract. These theorems are proved in the full version of the paper [1], where we also describe a linear-time algorithm for finding Alice's two-jump strategy with gain  $g(n)|P|$ :

**Theorem 5.** *There is an algorithm that, given a cutting of the pizza with  $n$  slices, performs a precomputation in time  $O(n)$ . Then, during the game, the algorithm decides each of Alice's turns in time  $O(1)$  in such a way that Alice makes at most two jumps and her gain is at least  $g(n)|P|$ .*

We note that there is a straightforward quadratic-time dynamic algorithm finding optimal strategies for each of the two players.

We remark that, unlike in Theorem 1, the number of Alice's jumps in her optimal strategy cannot be bounded by a constant. In fact, we can force both players to take the slices in a prescribed order by cutting the pizza into slices of sizes  $1, 1/2, 1/4, \dots$

The following question is still open.

*Problem 6.* Is there an algorithm that uses  $o(n^2)$  time for some precomputations and then computes each optimal turn in constant time?

We remark that we even don't know if Alice's optimal first turn can be computed in time  $o(n^2)$ .

Independently of us and approximately at the same time, K. Knauer, P. Micek and T. Ueckerdt [4] also proved Theorem 1 and some related results.

In a follow-up paper [2] we discuss generalizations of the Pizza Problem.

## 2 The Lower Bound

When the number of slices is even, Alice can always gain at least  $|P|/2$ . Here we prove the lower bound on her gain when  $n \geq 3$  is odd.

### 2.1 Preliminaries

Note that if both players use zero-jump strategy then the set of slices taken by either of the players forms a sequence of every other slice of the pizza. That motivates the following rearrangement. If the number of slices is odd, instead of the circular sequence  $P = p_0 p_1 \dots p_{n-1}$  we will be working with the related circular sequence  $V = v_0 v_1 \dots v_{n-1} = p_0 p_2 \dots p_{n-1} p_1 p_3 \dots p_{n-2}$  that we call *the characteristic cycle* (see Fig. 1). The size of the characteristic cycle is denoted by  $|V|$ . Clearly  $|V| = |P|$ .

An *arc* is a sequence of at most  $n - 1$  consecutive elements of  $V$ . If we talk about the first or the last element of an arc, we always consider it with respect to the linear order on the arc inherited from the characteristic cycle  $V$ . For an arc  $X = v_i v_{i+1} \dots v_{i+l-1}$ , its *length* is  $\ell(X) := l$  and its *size* is  $|X| := \sum_{j=i}^{i+l-1} |v_j|$ . An arc of length  $(n + 1)/2$  is called a *half-circle*.

At any time during a game, a player may decide to make only shifts further on. The player will take one or two arcs of the characteristic cycle afterwards. An example of such a game when Alice decided to make no more jumps is depicted on Fig. 2. Slices taken before the decision point are labeled with \*. The slices she took after the decision point are forming two arcs that are separated in between by some arc of previously taken slices.

**Observation 7.** *Consider a position after Alice’s turn  $A_j, j \neq 1, n$ . We have  $V = T_1 R_1 T_2 R_2$ , where  $\ell(T_1) = \ell(T_2) + 1 = (j + 1)/2$ ,  $\ell(R_1) = \ell(R_2)$ ,  $T_1$  and  $T_2$  are two arcs of already taken slices, and  $R_1$  and  $R_2$  are two arcs containing the remaining slices. Suppose that all the remaining turns of Alice ( $A_{j+2}, A_{j+4}, \dots, A_n$ ) are shifts. Then, regardless of Bob’s remaining turns  $B_{j+1}, \dots, B_{n-1}$ , the slices taken by Alice in the turns  $A_{j+2}, A_{j+4}, \dots, A_n$  necessarily form two arcs  $X_1$  and  $X_2$  such that  $X_1 T_1 X_2$  is a half-circle of  $V$ .*

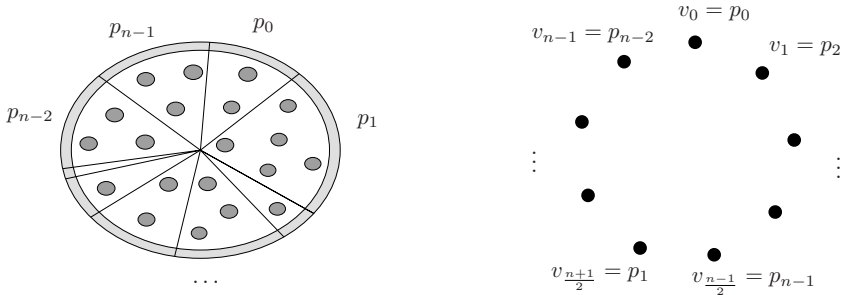
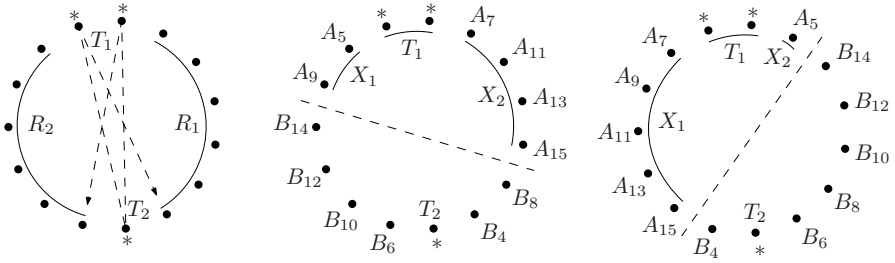
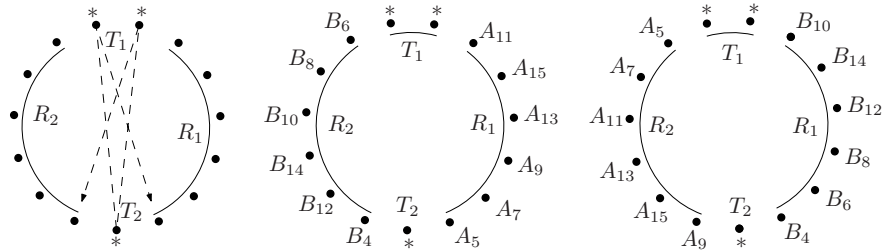


Fig. 1. A cutting of a pizza and the corresponding characteristic cycle



**Fig. 2.** Situation before Bobs turn with the two possible options marked by arrows (left) and two of the possible ends of the game where Alice made no more jumps (middle and right)



**Fig. 3.** Two possible choices of Bob’s next turn (left) and the two possible ends of the game where Bob made no more jumps (middle and right)

*In addition, for any half-circle  $Y_1T_1Y_2$ , Bob can choose his turns  $B_{j+2}, \dots, B_{n-1}$  so that  $X_1 = Y_1$  and  $X_2 = Y_2$ .*

If Bob decides to make only shifts for the rest of the game, he takes one arc afterwards. Namely, if there are two arcs of already taken slices in  $V$  at his decision point, then the arc that will be taken by Bob is neighboring these two arcs at both of its ends (see Fig. 3).

**Observation 8.** *Consider a position after Alice’s turn  $A_j, j \neq 1, n$ . We have  $V = T_1R_1T_2R_2$ , where  $\ell(T_1) = \ell(T_2) + 1 = (j + 1)/2$ ,  $\ell(R_1) = \ell(R_2)$ ,  $T_1$  and  $T_2$  are two arcs of already taken slices, and  $R_1$  and  $R_2$  are two arcs containing the remaining slices. Bob’s turn  $B_{j+1}$  may be on the last slice of  $R_1$  or on the first slice of  $R_2$ . If  $B_{j+1}$  is on the last slice of  $R_1$  and all the remaining turns of Bob are shifts then, regardless of Alice’s remaining turns, Bob will take  $R_1$  and Alice will take  $R_2$  in this phase of the game. Similarly, if  $B_{j+1}$  is on the first slice of  $R_2$  and all the remaining turns of Bob are shifts then, regardless of Alice’s remaining turns, Bob will take  $R_2$  and Alice will take  $R_1$  in this phase of the game.*



## 2.2 Minimal Triples

For each  $v$  in  $V$  the *potential* of  $v$  is the minimum of the sizes of half-circles covering  $v$ . The maximum of the potentials in  $V$  is the *potential* of  $V$ , which we further denote by  $p(V)$ . It is an immediate conclusion that Alice has a strategy with gain  $p(V)$  because by choosing an element with potential equal to  $p(V)$  and making only shifts afterwards Alice obtains at least  $p(V)$ . Therefore we may assume that  $p(V) < |V|/2$ .

A *covering triple* of half-circles is a triple of half-circles such that each element of  $V$  appears in at least one of the three half-circles. We allow two half-circles of the covering triple to be equal.

A covering triple is *minimal* if:

1. It contains a half-circle of minimum size (among all  $n$  half-circles),
2. All half-circles forming the triple have size at most  $p(V)$ , and
3. None of the half-circles may be replaced in the triple by a half-circle of strictly smaller size.

**Claim 9.** *Each half-circle of minimum size lies in at least one minimal triple.*

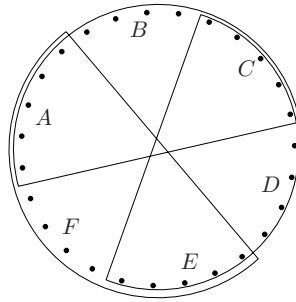
*Proof.* Take a half-circle  $H_1$  of minimum size. Consider  $v_k$  and  $v_{k+(n-3)/2}$  the two uncovered elements neighboring  $H_1$  on  $V$ . Let  $H_2$  be the half-circle of size at most  $p(V)$  that covers  $v_k$  and as many elements of  $V$  not covered by  $H_1$  as possible. We define  $H_3$  in the same way for  $v_{k+(n-3)/2}$ . The above triple of half-circles covers  $V$ . If it is not the case, then take an uncovered element  $v$ . Consider a half-circle  $H$  that has minimal size among half-circles covering  $v$ . At least one of  $v_k$  and  $v_{k+(n-3)/2}$  is covered by  $H$ . This contradicts the choice of  $H_2$  or  $H_3$ . So we get that the given triple of half-circles forms a covering triple. Now while any of the half-circles can be replaced in the triple by a half-circle of strictly smaller size, we replace it. Obviously  $H_1$  won't be replaced as it is a half-circle of minimum size. Consequently the triple we get is a minimal triple.  $\square$

**Observation 10.** *If the size of a half-circle in a minimal triple is  $z$  then Alice has a zero-jump strategy with gain  $z$ .*

*Proof.* As in a minimal triple all half-circles are of size at most  $p(V)$  and Alice has a zero-jump strategy with gain  $p(V)$ , the statement of the observation follows.  $\square$

**Claim 11.** *Let  $p(V) < |V|/2$ . Then any minimal triple contains three pairwise different half-circles, and thus there is a partition of  $V$  into six arcs  $A, B, C, D, E, F$  such that the half-circles in the minimal triple are  $ABC, CDE$  and  $EFA$  (see Fig. 4). The lengths of the arcs satisfy  $l(A) = l(D) + 1 \geq 2, l(C) = l(F) + 1 \geq 2$  and  $l(E) = l(B) + 1 \geq 2$ . Also the sizes of the arcs  $B, D$  and  $F$  are positive.*

*Proof.* If two of the three half-circles in a minimal triple are equal then  $V$  can be covered by two half-circles of the triple. Since each half-circle in the triple has size at most  $p(V)$ , the total size of the pizza is at most  $2p(V) < |V|$ , a



**Fig. 4.** The partitioning of the characteristic cycle given by the covering half-circles

contradiction. If at least one of  $B, D, F$  has length or size 0, we argue exactly in the same way.

We have  $l(ABC) + l(EFA) = n + 1 = l(A) + \dots + l(F) + 1$ , therefore  $l(A) = l(D) + 1 \geq 2$ . The other two equalities are analogous.  $\square$

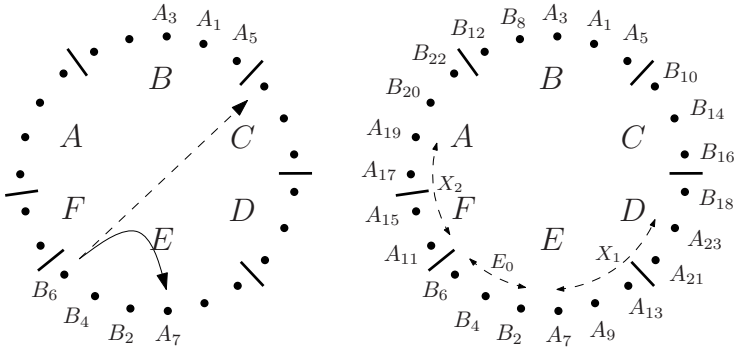
### 2.3 An Auxiliary One-Jump Strategy

Throughout this section we assume that  $p(V) < |V|/2$ . We fix any minimal triple of half-circles. By Claim 11, it yields a partition of  $V$  into six arcs  $A, B, C, D, E, F$  such that the half-circles in the triple are  $ABC, CDE, EFA$  (see Fig. 4). We further use the notation  $a := |A|, b := |B|$ , and so on.

We define a *median slice* of an arc  $X = v_i v_{i+1} \dots v_{i+l}$  to be a slice  $v_k \in X$  such that  $\sum_{j=i}^{k-1} |v_j| \leq |X|/2$  and  $\sum_{j=k+1}^{i+l} |v_j| \leq |X|/2$ . Observe that any arc of positive length has at least one median slice.

**Claim 12.** *Alice has a one-jump strategy for  $V$  with gain  $b/2 + \min\{c+d, f+a\}$  if  $p(V) < |V|/2$ .*

*Proof.* By Claim 11 we have that  $l(B) > 0$ . In the first turn Alice takes a median slice  $v_k$  of  $B$ . Consequently Bob is forced to start in  $E$ . He may take the element  $v_{k+(n-1)/2}$  or  $v_{k+(n+1)/2}$ . Alice makes only shifts while the shift implies taking an element of  $B$ . In the meantime Bob necessarily takes elements from  $E$ . In the turn, when Alice’s shift would imply taking an element outside of  $B$ , Alice makes a jump instead. In that moment some initial arc  $E_0$  of  $E$  starting from the boundary of  $E$  is already taken. Let  $E_1$  be the remaining part (subarc) of  $E$ . Alice takes the available element of  $E_1$ . Note that such an element exists since in the sequence  $P$ , all the neighbors of the slices of  $B$  are in  $E$ . She makes only shifts afterwards. By Observation 7, all the elements taken by her after the jump form two arcs  $X_1$  and  $X_2$ , each of them neighboring  $E_0$  on  $V$  (see Fig. 5). The half-circle  $X_1 E_0 X_2$  can replace either  $CDE$  or  $EFA$  in the fixed minimal triple. Thus due to the minimality of the triple, the size of  $X_1 X_2$  is always at least the



**Fig. 5.** One-jump strategy: Alice chooses a jump rather than a shift (left) and makes no more jumps afterwards (right)

size of either  $CD$  or  $FA$ . As Alice obtained at least the half of  $B$  before the jump, in the end she gains at least  $b/2 + \min\{c + d, f + a\}$ .  $\square$

**Corollary 13.** *Alice has a one-jump strategy for  $V$  with gain  $(a + b + c)/4 + (d + e + f)/2$  if  $p(V) < |V|/2$ .*

*Proof.* By Claim [12](#) Alice has a strategy with gain  $b/2 + \min\{c + d, f + a\}$ . Without loss of generality we may assume this sum is  $g_1 := b/2 + c + d$ . Alice also has a strategy with gain  $g_2 := e + f + a$  by Observation [10](#). Combining the two results Alice has a gain  $\max\{g_1, g_2\} \geq g_1/2 + g_2/2 = (a + c + d + e + f)/2 + b/4 \geq (a + b + c)/4 + (d + e + f)/2$ .  $\square$

### 2.4 A Two-Jump Strategy

Throughout this subsection we assume that  $p(V) < |V|/2$  and that  $V$  is partitioned into six arcs  $A, \dots, F$  in the same way as in the previous subsection.

In this subsection we describe a strategy satisfying the following claim.

**Claim 14.** *Alice has a two-jump strategy for  $V$  with gain  $b/2 + e/4 + \min\{c + d, f + a\}$  if  $p(V) < |V|/2$ .*

**Two Phases of the Game.** Let  $B = v_i v_{i+1} \dots v_{i+\Delta}$ . Then  $E = v_j v_{j+1} \dots v_{j+\Delta+1}$ , where  $j = i + (n - 1)/2$ . Consider the circular sequence  $V' = v_i v_{i+1} \dots v_{i+\Delta} v_j v_{j+1} \dots v_{j+\Delta+1}$  obtained by concatenating the arcs  $B$  and  $E$ .

Let  $H$  be a half-circle of  $V'$  containing  $v_j$ . Then its size is not smaller than the size of  $E$ , since otherwise the half-circle  $CDE$  of  $V$  could be replaced in the minimal triple  $T$  by a half-circle of smaller size—namely by the half-circle formed by the slices contained in  $CD$  and in  $H$ .

Similarly, if  $H$  is a half-circle of  $V'$  containing  $v_{j+\Delta+1}$ , then its size is also not smaller than the size of  $E$ . Since each half-circle of  $V'$  contains  $v_j$  or  $v_{j+\Delta+1}$ , it follows that  $E$  is a half-circle of  $V'$  of minimum size.

If  $p(V') \geq |V'|/2$  then Alice has a zero-jump strategy  $\Sigma$  for  $V'$  with gain  $p(V') \geq |V'|/2 \geq b/2 + e/4$ . Otherwise, by Corollary 13 (applied on  $V'$ ), Alice has a one-jump strategy  $\Sigma$  for  $V'$  with gain  $b/2 + e/4$  (we use the fact that  $E$  is a half-circle of  $V'$  of minimum size, and therefore it is contained in a minimal triple yielding a partition of  $V'$  into six arcs  $A', B', \dots, F'$  such that  $E = A'B'C'$  and  $B = D'E'F'$ ).

Briefly speaking, Alice’s strategy on  $V$  follows the strategy  $\Sigma$  as long as it is possible, then Alice makes one jump and after that she makes only shifts till the end of the game.

In the rest of this subsection (Subsect. 2.4), we consider a game  $G$  on  $V$ . We divide the turns of  $G$  into two phases. *The first phase of  $G$*  is the phase when Alice follows the strategy  $\Sigma$  and it ends with Bob’s turn. Alice’s first turn that does not follow (and actually cannot follow) the strategy  $\Sigma$  is the first turn of *the second phase of  $G$* . It is always a jump and all the other turns of Alice in the second phase are shifts.

We now describe Alice’s strategy in each of the two phases of  $G$  in detail.

**Alice’s Strategy in the First Phase.** As mentioned above, Alice has a one-jump strategy  $\Sigma$  for  $V'$  with gain  $b/2 + e/4$ . We now distinguish two cases.

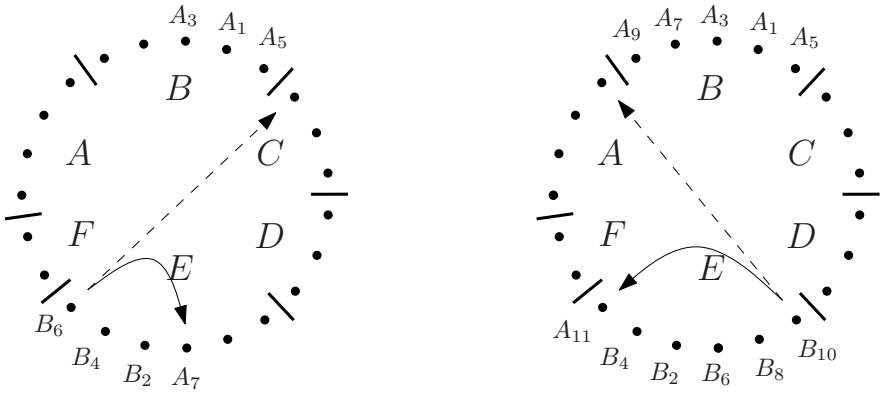
*Case 1: The strategy  $\Sigma$  is a zero-jump strategy.* Let the first turn in the zero-jump strategy  $\Sigma$  be on a slice  $q \in V'$ . The first turn could be also on any other point of  $V'$  with the same or larger potential. Observe that the potentials of the slices in  $V'$  are  $e$  on  $E$  and at least  $e$  on  $B$ . Therefore we may assume that  $q$  lies in  $B$ .

In the game  $G$ , Alice makes her first turn also on  $q$ . In the second turn Bob can choose between two slices in  $E$ . In the subsequent turns Alice makes shifts as long as Bob’s previous turn was neither on the first nor on the last slice of  $E$ . Consider all slices taken by Bob up to any fixed moment during the first phase of the game  $G$ . They always form a subarc of  $E$  according to Observation 7. The first turn in which Bob takes the first or the last slice of  $E$  is the last turn of the first phase. Note that after that Alice’s shift would be either on the last slice of  $A$  or on the first slice of  $C$  (see Fig. 6). But Alice makes a jump and this jump is the first turn of the second phase. Note that this jump is in  $E$  (see Fig. 6).

*Case 2: The strategy  $\Sigma$  is not a zero-jump strategy.* Following the proof of Corollary 13, we may suppose that  $\Sigma$  is the strategy that we describe below.

By Claim 9, the half-circle  $E$  of minimum size is contained in some minimal triple  $T'$  of half-circles of  $V'$ . The triple  $T'$  determines a partition of  $V'$  into six arcs  $A', B', \dots, F'$  in the same way as  $T$  determined a partition of  $V$  into  $A, B, \dots, F$ . We may suppose that  $E = A'B'C'$  and  $B = D'E'F'$ .

We may suppose that the size of  $B'$  is positive, since otherwise one of the half-circles  $C'D'E'$  and  $E'F'A'$  has size at least  $b/2 + e/2$  and thus Alice has



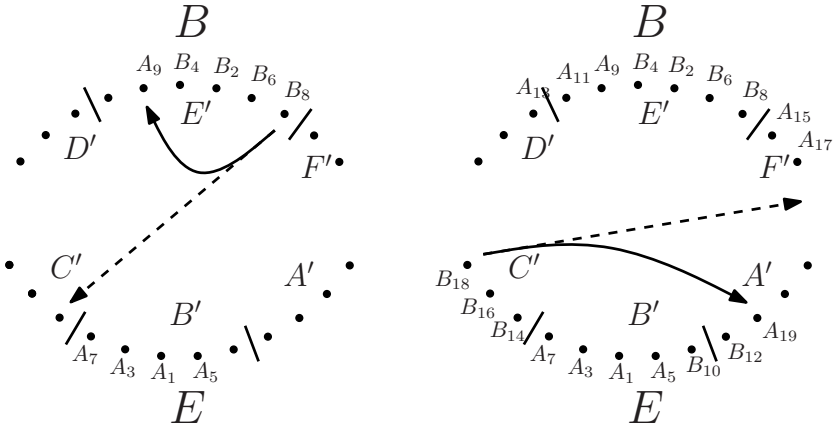
**Fig. 6.** Case 1: After the end of first phase, Alice chooses a jump rather than a shift (two examples shown)

a zero-jump strategy for  $V'$  with gain  $b/2 + e/2$ , allowing us to use the above Case 1.

In the first turn Alice takes a median slice of  $B'$ . Then in the second turn Bob can choose between two slices of  $E'$ . In the subsequent turns Alice makes shifts as long as Bob’s previous turn was neither on the first nor on the last slice of  $E'$ . In each moment in this part of the game Bob’s turns form a subarc of  $E'$ . At the first instance when Bob takes the first or the last slice of  $E'$ , Alice makes a jump, which is always in  $E'$  (see Fig. 7). Note that so far the game was an analogue of the first phase in Case 1, with  $B'$  and  $E'$  in place of  $B$  and  $E$ , respectively. After her first jump Alice makes shifts as long as Bob’s previous turn was neither on the first nor on the last slice of  $E$ . Note that Bob’s turns in this part of the game are in  $E$  (see Fig. 7). At the first instance when Bob takes the first or the last slice of  $E$ , Alice makes a jump, which is already the first turn of the second phase. This jump is necessarily in  $E$  (see Fig. 7).

**Alice’s Strategy in the Second Phase.** Alice’s strategy in the second phase is very simple. Above we describe the first phase and also the first turn of the second phase, which is always a jump done by Alice. In the rest of the second phase Alice makes only shifts.

**Analysis of Alice’s Gain.** Since the first phase of  $G$  ends by Bob’s turn on the first or on the last slice of  $E$ , we may suppose without loss of generality that it ends with Bob’s turn on  $v_j$ . Then the part of  $V$  removed in the first phase of  $G$  is a union of some initial subarc  $B_0$  of  $B$  and some initial subarc  $E_0$  of  $E$ . Let  $E_1$  be the arc formed by the slices of  $E$  not taken in the first phase of  $G$ , thus  $E = E_0E_1$ , and let  $e_1 := |E_1|$ . In her jump at the beginning of the second phase of  $G$ , Alice takes the first slice of  $E_1$ .



**Fig. 7.** Case 2: During the first phase, Alice makes a jump rather than a shift (left) and then she makes another jump after the end of the first phase (right)

All the slices taken by Alice in the second phase of  $G$  form two arcs  $X_1$  and  $X_2$  such that  $X_1E_0X_2$  is a half-circle of  $V$  (see Fig. 5). Since none of the half-circles  $CDE$  and  $EFA$  can be replaced in the triple  $T$  by a half-circle of a strictly smaller size, the sum  $|X_1| + |X_2|$  achieves its minimum either for  $X_1 = CD$  and  $X_2 = E_1$ , or for  $l(X_1) = 0$  and  $X_2 = E_1FA$ . Thus, the portion collected by Alice in the second phase of  $G$  is at least  $e_1 + \min\{c + d, f + a\}$ .

Now, consider an auxiliary game  $G'$  on  $V'$  consisting of two phases, such that *the first phase of  $G'$*  is exactly the same as the first phase of  $G$  (recall that all turns in the first phase of  $G$  are in  $B \cup E$ ) and that both Alice and Bob make only shifts in *the second phase of  $G'$* .

Observe that Alice actually follows the strategy  $\Sigma$  in the whole game  $G'$ .

By Observation 8, Alice collects exactly the slices of  $E_1$  in the second phase of  $G'$ . Thus, if  $g$  denotes the portion collected by Alice in the first phase of  $G'$  then  $g + e_1$  is her portion in the whole game  $G'$ . Since the strategy  $\Sigma$  guarantees gain  $b/2 + e/4$  to Alice, we get  $g + e_1 \geq b/2 + e/4$ .

Alice's portion in the whole game  $G$  is at least  $g + (e_1 + \min\{c + d, f + a\}) \geq b/2 + e/4 + \min\{c + d, f + a\}$ , which completes the proof of Claim 14.

### 2.5 Proof of the Lower Bound

*Proof (of Theorem 7).* If  $p(V) \geq |V|/2$  then Alice has a zero-jump strategy with gain  $|V|/2 = |P|/2$ .

Suppose now that  $p(V) < |V|/2$ . Then  $V$  may be partitioned into six arcs  $A, \dots, F$  as in Claim 11. Without loss of generality, we may assume that  $a + b + c \leq c + d + e \leq e + f + a$ . Thus,  $a + b \leq d + e$  and  $c + d \leq f + a$ . By Observation 10, Alice has a zero-jump strategy with gain

$$g_1 := e + f + a.$$

By Claim [14](#), Alice has a two-jump strategy with gain

$$g_2 := b/2 + e/4 + \min\{c + d, f + a\} = b/2 + e/4 + c + d.$$

By an analogue of Claim [14](#), Alice also has a two-jump strategy with gain

$$g_3 := f/2 + c/4 + \min\{a + b, d + e\} = f/2 + c/4 + a + b.$$

One of the three strategies gives gain

$$\begin{aligned} \max\{g_1, g_2, g_3\} &\geq (3g_1 + 4g_2 + 2g_3)/9 \\ &= (5a + 4b + 9c/2 + 4d + 4e + 4f)/9 = (4|P| + a + c/2)/9 \geq 4|P|/9. \quad \square \end{aligned}$$

## Acknowledgments

We thank Peter Winkler for posing this problem and for the discussions. The third author thanks Dirk Oliver Theis for the discussions about the upper bound during Fete of Combinatorics and Computer Science in Keszthely, August 11-16, 2008. We also acknowledge the use of the LPSolve IDE [3](#) software, which made our computations much more efficient.

## References

1. Cibulka, J., Kynčl, J., Mészáros, V., Stolař, R., Valtr, P.: Solution of Peter Winkler's Pizza Problem. In: Building Bridges: Between Mathematics and Computer Science, vol. 2. Springer, New York (to appear)
2. Cibulka, J., Kynčl, J., Mészáros, V., Stolař, R., Valtr, P.: Generalizations of Peter Winkler's Pizza Problem (in preparation)
3. Gourvest, H.: LPSolve IDE v5, <http://www.progdigy.com>; Berkelaar, M., Dirks, J., Eikland, K., Notebaert, P.: lp\_solve 5.5, <http://lpsolve.sourceforge.net/5.5>
4. Knauer, K., Micek, P., Ueckerdt, T.: How to eat 4/9 of a pizza (manuscript)

# An $O(n)$ -Time Algorithm for the Paired-Domination Problem on Permutation Graphs

Evangelos Lappas, Stavros D. Nikolopoulos, and Leonidas Palios

Department of Computer Science, University of Ioannina  
GR-45110 Ioannina, Greece  
{elappas,stavros,palios}@cs.uoi.gr

**Abstract.** A vertex subset  $D$  of a graph  $G$  is a dominating set if every vertex of  $G$  is either in  $D$  or is adjacent to a vertex in  $D$ . The paired-domination problem on  $G$  asks for a minimum-cardinality dominating set  $S$  of  $G$  such that the subgraph induced by  $S$  contains a perfect matching; motivation for this problem comes from the interest in finding a small number of locations to place pairs of mutually visible guards so that the entire set of guards monitors a given area. The paired-domination problem on general graphs is known to be NP-complete.

In this paper, we consider the paired-domination problem on permutation graphs. We define an embedding of permutation graphs in the plane which enables us to obtain an equivalent version of the problem involving points in the plane, and we describe a sweeping algorithm for this problem; if the permutation over the set  $N_n = \{1, 2, \dots, n\}$  defining a permutation graph  $G$  on  $n$  vertices is given, our algorithm computes a paired-dominating set of  $G$  in  $O(n)$  time, and is therefore optimal.

**Keywords:** permutation graphs, paired-domination, domination, algorithms, complexity.

## 1 Introduction

A subset  $D$  of vertices of a graph  $G$  is a *dominating set* if every vertex of  $G$  either belongs to  $D$  or is adjacent to a vertex in  $D$ ; the minimum cardinality of a dominating set of  $G$  is called the *domination number* of  $G$  and is denoted by  $\gamma(G)$ . The problem of computing the domination number of a graph has received and keeps receiving considerable attention by many researchers (see [11] for a long bibliography on domination). The problem finds many applications, most notably in relation to area monitoring problems by the minimum number of guards: the potential guard locations are vertices of a graph in which two locations are adjacent if a guard in one of them monitors the other; then, the minimum dominating set of the graph determines the locations to place the guards.

The domination problem admits many variants; the most basic ones include: domination, edge domination, weighted domination, independent domination,



connected domination, total/open domination, locating domination, and paired-domination [11,12,13,14,18,30]. Among these, we will focus on paired-domination: a vertex subset  $S$  of a graph  $G$  is a *paired-dominating set* if it is a dominating set and the subgraph induced by the set  $S$  has a perfect matching; the minimum cardinality of a paired-dominating set in  $G$  is called the *paired-domination number* and is denoted by  $\gamma_p(G)$ . Paired-domination was introduced by Haynes and Slater [13]; their motivation came from the variant of the area monitoring problem in which each guard has another guard as a backup (i.e., we have pairs of guards protecting each other). Haynes and Slater noted that every graph with no isolated vertices has a paired-dominating set (on the other hand, it easily follows from the definition that a graph with isolated vertices does not have a paired-dominating set). Additionally, they showed that the paired-domination problem is NP-complete on arbitrary graphs; thus, it is of theoretical and practical importance to find classes of graphs for which this problem can be solved in polynomial time and to describe efficient algorithms for its solution.

Trees have been one of the first targets of researchers working on paired-domination: Qiao *et al.* [23] presented a linear-time algorithm for computing the paired-domination number of a tree and characterized the trees with equal domination and paired-domination number; Henning and Plummer [16] characterized the set of vertices of a tree that are contained in all, or in no minimum paired-dominating sets of the tree. Kang *et al.* [17] considered “inflated” graphs (for a graph  $G$ , its inflated version is obtained from  $G$  by replacing each vertex of degree  $d$  in  $G$  by a clique on  $d$  vertices), gave an upper and lower bound for the paired-domination number of the inflated version of a graph, and described a linear-time algorithm for computing a minimum paired-dominating set of an inflated tree. Bounds for the paired-domination number have been established also for claw-free cubic graphs [9], for Cartesian products of graphs [3], and for generalized claw-free graphs [7]. An  $O(n + m)$ -time algorithm for computing a paired-dominating set of an interval graph on  $n$  vertices and  $m$  edges, when an interval model for the graph with endpoints sorted is available has been given by Cheng *et al.* [5]; they also extended their result to circular-arc graphs giving an algorithm running in  $O(m(m + n))$  time in this case. Very recently, Cheng *et al.* [6] gave an  $O(mn)$ -time algorithm for the paired domination problem on permutation graphs.

We too consider the paired domination problem on the class of permutation graphs, a well-known subclass of perfect graphs. Given a permutation  $\pi = (\pi_1, \pi_2, \dots, \pi_n)$  over the set  $N_n = \{1, 2, \dots, n\}$ , we define the  $n$ -vertex graph  $G[\pi]$  with vertex set  $V(G[\pi]) = N_n$  and edge set  $E(G[\pi])$  such that  $ij \in E(G[\pi])$  if and only if  $(i - j)(\pi_i^{-1} - \pi_j^{-1}) < 0$ , for all  $i, j \in V(G[\pi])$ , where  $\pi_i^{-1}$  is the index of the element  $i$  in  $\pi$ . A graph  $G$  on  $n$  vertices is a *permutation graph* if there exists a permutation  $\pi$  on  $N_n$  such that  $G$  is isomorphic to  $G[\pi]$  (the graph  $G[\pi]$  is also known as the inversion graph of  $G$  [10]). Therefore, in this paper, we assume that a permutation graph  $G[\pi]$  is represented by the corresponding permutation  $\pi$ . A lot of research work has been devoted to the study of permutation graphs, and several algorithms have been proposed for recognizing

permutation graphs and for solving combinatorial and optimization problems on them both for sequential computation (see for example [22,26,19,28,21]) as well as for parallel (see [15,20,24]). Moreover, in addition to the above mentioned result of Cheng *et al.* [6] on paired domination, several variants of the domination problem have been considered on permutation graphs; see [8,21,29,25,4,27].

In this paper, we study the paired-domination problem on permutation graphs following an approach different from that of Cheng *et al.* [6]. We define an embedding of permutation graphs in the plane and show that every permutation graph  $G$  with no isolated vertices admits a minimum-cardinality paired-dominating set of a particular form in the embedding of  $G$ . We take advantage of this property to describe an algorithm which “sweeps” the vertices of the embedding from left to right and computes a minimum cardinality paired-dominating set if such a set exists; if the permutation over the set  $N_n = \{1, 2, \dots, n\}$  defining a permutation graph on  $n$  vertices is given, our algorithm runs in  $O(n)$  time using  $O(n)$  space. Since for a permutation graph, a defining permutation can be computed in  $O(n + m)$  time [19], our algorithm is optimal.

## 2 Theoretical Framework

We consider finite undirected graphs with no loops or multiple edges; for a graph  $G$ , we denote its vertex and edge set by  $V(G)$  and  $E(G)$ , respectively.

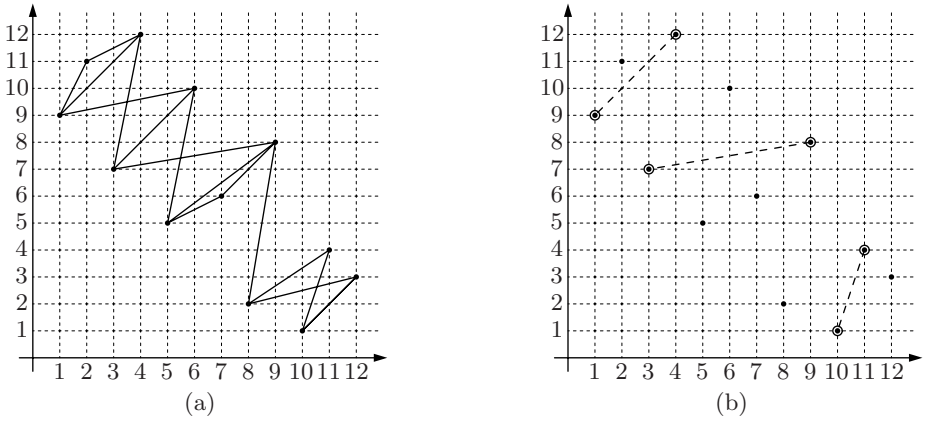
Let  $\pi = (\pi_1, \pi_2, \dots, \pi_n)$  be a permutation over the set  $N_n = \{1, 2, \dots, n\}$ . A *subsequence* of  $\pi$  is a sequence  $\alpha = (\pi_{i_1}, \pi_{i_2}, \dots, \pi_{i_k})$  such that  $i_1 < i_2 < \dots < i_k$ . If, in addition,  $\pi_{i_1} < \pi_{i_2} < \dots < \pi_{i_k}$ , then we say that  $\alpha$  is an *increasing subsequence* of  $\pi$ .

A *left-to-right maximum* of  $\pi$  is an element  $\pi_i$ ,  $1 \leq i \leq n$ , such that  $\pi_i > \pi_j$  for all  $j < i$ . The first element in every permutation is a left-to-right maximum. If the largest element is the first, then it is the only left-to-right maximum; otherwise there are at least two (the first and the largest). The increasing subsequence  $\alpha = (\pi_{i_1}, \pi_{i_2}, \dots, \pi_{i_k})$  is called a *left-to-right maxima subsequence* if it consists of all the left-to-right maxima of  $\pi$ ; clearly,  $\pi_{i_1} = \pi_1$ . For example, the left-to-right maxima subsequence of the permutation  $(4, 2, 6, 1, 9, 3, 7, 5, 11, 12, 8, 10)$  is  $(4, 6, 9, 11, 12)$ .

The *right-to-left minima subsequence* of  $\pi$  is defined analogously:  $\alpha' = (\pi_{j_1}, \pi_{j_2}, \dots, \pi_{j_{k'}})$  is called a right-to-left minima subsequence if it is an increasing subsequence and consists of all the right-to-left minima of  $\pi$ , where an element  $\pi_i$ ,  $1 \leq i \leq n$ , is a *right-to-left minimum* if  $\pi_i < \pi_j$  for all  $j > i$ . The last element in every permutation is a right-to-left minimum, and thus  $\pi_{j_{k'}} = \pi_n$ . For the permutation  $(4, 2, 6, 1, 9, 3, 7, 5, 11, 12, 8, 10)$ , the right-to-left minima subsequence is  $(1, 3, 5, 8, 10)$ .

We will also be considering points in the plane. For such a point  $p$ , we denote by  $x(p)$  and  $y(p)$  the  $x$ - and  $y$ -coordinate of  $p$ , respectively.

**An Embedding of Permutation Graphs.** Given a permutation  $\pi$  over the set  $N_n = \{1, 2, \dots, n\}$ , we define and use an embedding of the vertices of the permutation graph  $G[\pi]$  in the plane based on the mapping:



**Fig. 1.** (a) The embedding of the permutation graph corresponding to the per-mutation (4, 2, 6, 1, 9, 3, 7, 5, 11, 12, 8, 10); (b) A minimum paired-dominating set

$$\text{vertex corresponding to integer } i \longrightarrow \text{point } p_i = (i, n + 1 - \pi_i^{-1}). \quad (1)$$

We note that similar representations have been used by other authors as well; see [1, 21]. In our representation, all the points  $p_i$ ,  $1 \leq i \leq n$ , are located in the first quadrant of the Cartesian coordinate system and no two such points have the same  $x$ - or the same  $y$ -coordinate (see Figure 1(a)). Let  $P_\pi = \{p_1, p_2, \dots, p_n\}$ . The adjacency condition  $ij \in E(G[\pi])$  iff  $(i - j)(\pi_i^{-1} - \pi_j^{-1}) < 0$  (for all  $i, j \in N_n$ ) for the permutation graph  $G[\pi]$  implies that two points  $p_i$  and  $p_j$  are adjacent iff  $(x(p_i) - x(p_j)) \cdot (y(p_i) - y(p_j)) > 0$ , i.e., the one of the points is below and to the left of the other. Thus, all the edges have a down-left to up-right direction (Figure 1(a)).

Due to the bijection between the vertices of the permutation graph and the points  $p_i$ , with a slight abuse of notation, in the following, we will regard *the points  $p_i$  as the vertices of the permutation graph*.

In terms of the above embedding, a point  $p_i$  dominates all points  $p \in P_\pi$  such that  $(x(p) - x(p_i)) \cdot (y(p) - y(p_i)) \geq 0$ , i.e.,  $p$  is either below and to the left or above and to the right of  $p_i$  (the shaded area in Figure 2 (left)). Then,

**Definition 1.** For any edge  $e = p_i p_j$ , where  $p_i, p_j \in P_\pi$ , the portion of the plane covered by  $e$  is the portion of the plane

$$\{q \in \mathbb{R}^2 \mid (x(q) - x(p_i)) \cdot (y(q) - y(p_i)) \geq 0 \text{ or } (x(q) - x(p_j)) \cdot (y(q) - y(p_j)) \geq 0\}$$

dominated by  $p_i$  or  $p_j$ .

The part of the plane not covered by  $e$  consists of two disjoint open quadrants, one occupying the upper left corner and the other the bottom right corner.

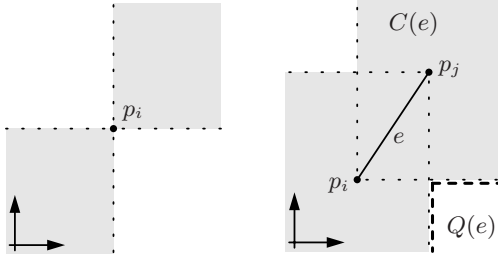


Fig. 2.

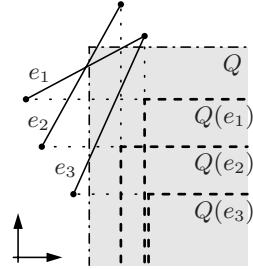


Fig. 3.

To simplify our description, we introduce the following notation (see Figure 2 (right)):

**Notation 1.** We denote by

- $C(e)$  the portion of the plane covered by the edge  $e$  and
- $Q(e)$  the bottom right quadrant not covered by  $e$ .

Moreover, a left-to-right maximum of a permutation  $\pi$  defining a permutation graph is mapped to a point  $p \in P_\pi$  that is a vertex of the upper envelope of the point set  $P_\pi$  (i.e., there does not exist a point  $q \in P_\pi - \{p\}$  for which  $x(p) \leq x(q)$  and  $y(p) \leq y(q)$ )<sup>1</sup>. For example, the 5 left-to-right maxima of the permutation defining the graph of Figure 1(a) correspond to the points (4, 12), (6, 10), (9, 8), (11, 4), and (12, 3). Similarly, a right-to-left minimum is mapped to a point  $p \in P_\pi$  that is a vertex of the lower envelope of the point set  $P_\pi$  (i.e., there does not exist a point  $q \in P_\pi - \{p\}$  for which  $x(p) \geq x(q)$  and  $y(p) \geq y(q)$ ); the 5 right-to-left minima of the graph of Figure 1(a) correspond to the points (1, 9), (3, 7), (5, 5), (8, 2), and (10, 1) of the lower envelope of  $P_\pi$ . For convenience, each point in  $P_\pi$  corresponding to a left-to-right-maximum (right-to-left minimum, resp.) of a permutation  $\pi$  will be called a left-to-right-maximum (right-to-left minimum, resp.) as well.

Finally, the following result helps us focus on solutions to the paired-domination problem on permutation graphs which are of a particular form, thus enabling us to obtain an efficient algorithm.

**Lemma 1.** Let  $G$  be an embedded permutation graph with no isolated vertices,  $P_\pi = \{p_1, p_2, \dots, p_n\}$  the corresponding point set (determined by the mapping in Eq. (1)), and  $u_1, u_2, \dots, u_\ell$  ( $v_1, v_2, \dots, v_{\ell'}$ , resp.) be the left-to-right maxima (right-to-left minima, resp.) of  $P_\pi$  in order from left to right. Then, for any set  $A$  of edges of  $G$  whose endpoints dominate the entire point set  $P_\pi$ , there exists a matching  $M$  of edges of  $G$  such that

<sup>1</sup> When such inequalities hold for the coordinates of two points  $p$  and  $q$ , it is often said that  $q$  dominates  $p$ ; however, we will avoid using this term so that there is no confusion with the notion of vertex domination which is central to our work.

- the endpoints of the edges in  $M$  dominate the entire  $P_\pi$ ,
- $|M| \leq |A|$ , and
- $M = \{v_{s_1}u_{t_1}, v_{s_2}u_{t_2}, \dots, v_{s_{|M|}}u_{t_{|M|}}\}$  where  $s_1 < s_2 < \dots < s_{|M|} \leq \ell'$  and  $t_1 < t_2 < \dots < t_{|M|} \leq \ell$  (i.e.,  $M$  is a matching which dominates  $P_\pi$  and consists of at most  $|A|$  non-crossing edges each of which connects a left-to-right maximum to a right-to-left minimum of  $P_\pi$ ).

Lemma 1 readily implies the following corollary.

**Corollary 1.** *Let  $G$  be an embedded permutation graph with no isolated vertices, and  $P_\pi = \{p_1, p_2, \dots, p_n\}$  the corresponding point set. Then,  $G$  has a paired-dominating set of minimum cardinality whose induced subgraph admits a perfect matching consisting of non-crossing edges of  $G$  each of which connects a left-to-right maximum to a right-to-left minimum.*

Such a matching is of the form shown in Figure 1(b). As the edges in such a matching do not cross, they exhibit an ordering from up-left to bottom-right.

### 3 The Algorithm

Corollary 1 implies that for every permutation graph with no isolated vertices there exists a minimum-cardinality paired-dominating set whose induced embedded subgraph admits a perfect matching of the form shown in Figure 1(b); for a permutation graph  $G$ , our algorithm precisely computes a minimum matching  $M$  of (the embedded)  $G$  of this form whose endpoints dominate all the vertices of  $G$ . As the edges in such a matching exhibit an ordering from left to right, our algorithm works by identifying candidates for each edge in  $M$  in order from left to right.

In particular, regarding the *leftmost edge* in  $M$ , we need to have that

- for each candidate  $e$  for the leftmost edge, every point in  $P_\pi$  either is dominated by the endpoints of  $e$  or lies in the bottom-right non-covered quadrant  $Q(e)$  of  $e$ , i.e.,

$$P_\pi \text{ lies in } C(e) \cup Q(e). \tag{2}$$

Furthermore, in order to obtain a minimum-size set  $M$ ,

- we maintain only the “usefull” partial solutions.

In order to formalize the latter condition, we give the following definition of redundant edges.

**Definition 2.** *Let  $G$  be an embedded permutation graph,  $Q$  an open quadrant (bounded only from above and left) which we wish to cover, and  $X = \{e \in E(G) \mid Q \cap P_\pi = (C(e) \cup Q(e)) \cap P_\pi\}$  (i.e., all the points of  $P_\pi$  belonging to  $Q$  lie either in  $C(e)$  or in  $Q(e)$ ). Then, we say that an edge  $d \in X$  is redundant if there exists another edge  $d' \in X$  such that  $Q(d') \subset Q(d)$ .*

For example, in Figure 3, the edges  $e_1$  and  $e_2$  are redundant in light of  $e_3$ .

We note that we are interested in minimizing the non-covered part of the plane rather than minimizing the number of points that are not dominated. In light of Definition 2, the fact that we are interested in edges  $e$  that minimize the non-covered part  $Q(e)$  of the plane is rephrased into that *we are interested in edges  $e$  that are not redundant*. The following lemma enables us to identify redundant edges among edges incident on a left-to-right maximum and a right-to-left minimum (see Figure 3):

**Lemma 2.** *Let  $G$  be an embedded permutation graph and let  $u_1, u_2, \dots, u_\ell$  ( $v_1, v_2, \dots, v_{\ell'}$ , resp.) be the left-to-right maxima (right-to-left minima, resp.) of  $G$  in order from left to right. Moreover, let  $A$  be a subset of edges of  $G$  which cover the plane except for an open quadrant  $Q$  (bounded only from above and left), and  $X = \{e \in E(G) - A \mid Q \cap P_\pi = (C(e) \cup Q(e)) \cap P_\pi\}$ . Then, if  $X$  contains an edge  $d = v_i u_j$ , any edge  $v_{i'} u_{j'} \in X - \{d\}$  such that  $i' \leq i$  and  $j' \leq j$  is redundant.*

Lemma 2 implies that for two edges  $v_i u_j, v_{i'} u_{j'} \in X$  to be non-redundant, it has to be the case that  $(i' - i) \cdot (j' - j) < 0$ , that is, the non-redundant edges form a *crossing pattern* like the one shown in Figure 4.

Here is an outline of our algorithm for computing a minimum matching  $M$  such that the edges in  $M$  are of the form shown in Figure 1(b) and their endpoints dominate all the vertices of the given permutation graph  $G$ : The algorithm identifies the non-redundant candidates for the leftmost edge of  $M$  and constructs a set  $E_1 = \{e_{1,1}, e_{1,2}, \dots, e_{1,h_1}\}$  of all these candidates. In the general step, we have a set  $E_i = \{e_{i,1}, e_{i,2}, \dots, e_{i,h_i}\}$  of candidates for the  $i$ -th edge of the matching  $M$ . Then, the algorithm constructs the set  $E_{i+1}$  of candidates for the  $(i + 1)$ -st edge by selecting the non-redundant edges among the edges in

$$\{e \in (E(G) - \bigcup_{r=1}^i E_r) \mid \exists j \text{ such that } Q(e_{i,j}) \cap P_\pi = (C(e) \cup Q(e)) \cap P_\pi\}$$

(i.e., among the edges  $e$  such that each of the points that belong to the uncovered quadrant  $Q(e_{i,j})$  of an edge  $e_{i,j} \in E_i$  is either covered by  $e$  or lies in the quadrant  $Q(e)$  of  $e$ ). This is repeated until for some  $i'$  and  $j'$ , the quadrant  $Q(e_{i',j'})$  contains no points of  $P_\pi$ . We also ensure that each collected candidate edge  $e \in E_{i+1}$  ( $i > 1$ ) has a pointer *back* which points to an edge  $e' \in E_i$  such that  $Q(e') \cap P_\pi = (C(e) \cup Q(e)) \cap P_\pi$ ; then, starting from  $e_{i',j'}$  (whose quadrant  $Q(e_{i',j'})$  contains no points of  $P_\pi$ ), we follow *back*-pointers collecting the edges we visit, thus constructing the matching  $M$  that we seek.

The correctness of the algorithm is established by induction on the size of any solution to the paired-domination problem on the input permutation graph  $G$  and follows from the correctness of the procedures to compute the set  $E_1$  of candidate edges for the leftmost edge of a solution and to compute the set  $E_{i+1}$  of candidates from the corresponding set  $E_i$ . We give details on these two procedures in the following paragraphs. For simplicity, we introduce the following additional notation:

**Notation 2.** *For a point  $p \in P_\pi$ , we denote by*

- $lrm_{\text{max\_above}}[p]$  *the lowest left-to-right maximum above  $p$  and*
- $rlmin_{\text{left}}[p]$  *the rightmost right-to-left minimum to the left of  $p$ .*

### 3.1 Computing the Set $E_1$

The goal in the construction of the set  $E_1$  is that each edge  $e \in E_1$  is incident on a right-to-left-minimum and a left-to-right maximum, is not redundant, and satisfies Eq. (2). Let  $v_i$  be a right-to-left minimum. The other endpoint of an edge in  $E_1$  incident on  $v_i$  has to be adjacent to  $v_i$  and to all the points in  $P_\pi$  to the left of  $v_i$  (which are not dominated by  $v_i$ ); therefore, it needs to be above and to the right of the highest point, say  $p$ , among  $v_i$  and all the points to the left of  $v_i$ . Then, if  $u_{q_i}$  is the lowest left-to-right maximum above  $p$ , each of the left-to-right maxima  $u_1, \dots, u_{q_i}$  will do, whereas none other will do. Yet, among the edges  $v_i u_1, \dots, v_i u_{q_i}$ , all but the last one are redundant.

More formally, our observations are summarized in the following lemma:

**Lemma 3.** *Let  $G$  be an embedded permutation graph with no isolated vertices,  $P_\pi = \{p_1, p_2, \dots, p_n\}$  the corresponding point set, and let  $u_1, u_2, \dots, u_\ell$  ( $v_1, v_2, \dots, v_\ell$ , resp.) be the left-to-right maxima (right-to-left minima, resp.) in  $P_\pi$  in order from left to right. If  $v_r = \text{rlmin\_left}[u_1]$ , we have:*

- (i) *For each  $v_i$ ,  $i = 1, 2, \dots, r$ , let  $p(v_i)$  be the highest among the points in  $P_\pi$  with  $x$ -coordinate  $\leq x(v_i)$ , and let  $u_{q_i} = \text{lrmx\_above}[p(v_i)]$ . Then, for any edge  $e_q = v_i u_q$  with  $1 \leq q \leq q_i$ , it holds that  $P_\pi$  lies in  $C(e_q) \cup Q(e_q)$  (i.e., Eq. (2) holds); this does not hold for any edge  $e_q = v_i u_q$  with  $q > q_i$ .*
- (ii) *Among the edges referred to in the statement (i) of the lemma, the edges  $v_i u_q$  (where  $1 \leq q < q_i$ ) are all redundant in light of the existence of the edge  $v_i u_{q_i}$ .*
- (iii) *No edge  $e$  incident on a right-to-left minimum to the right of  $v_r$  satisfies Eq. (2).*

In Figure 1(a),  $v_1 = (1, 9)$ ,  $v_2 = (3, 7)$ , and  $v_r = v_2$ ; so, the edges considered are  $v_1 u_1, v_1 u_2, v_2 u_1$  (where  $u_1 = (4, 12)$  and  $u_2 = (6, 10)$ ), among which  $v_1 u_1$  is redundant. We give below the outline of this procedure: in Step 1, we use Lemma 3 to construct a list  $L$  of edges satisfying Eq. (2) where  $L$  contains exactly the single non-redundant edge incident on each right-to-left minimum to the left of  $u_1$  (see statement (ii) of Lemma 3); in Step 2, we obtain the final set  $E_1$  by removing any redundant edges from  $L$ .

Procedure Compute\_ $E_1$

1.  $\text{highest\_}p \leftarrow p_1$ ;      {the highest point seen so far is the leftmost point}  
 $L \leftarrow$  a list containing a single node storing the edge connecting  $p_1$  to  $\text{lrmx\_above}[p_1]$ ;  
 $i \leftarrow 2$ ;      {process the points by increasing  $x$ -coordinate}  
**while**  $p_i$  does not coincide with the leftmost left-to-right maximum  $u_1$  **do**  
     **if**  $y(p_i) > y(\text{highest\_}p)$   
     **then**  $\text{highest\_}p \leftarrow p_i$ ;      {update highest point seen so far}  
     **if**  $p_i$  is a right-to-left minimum  
     **then** insert at the end of  $L$  the edge connecting  $p_i$  to  $\text{lrmx\_above}[p_i]$ ;  
      $i \leftarrow i + 1$ ;

2.  $E_1 \leftarrow \emptyset$ ;

let the list  $L$  contain the edges  $e_1, e_2, \dots, e_{|L|}$  in order and suppose that  $e_i = v_{s_i} u_{t_i}$ , where  $v_{s_i}$  is a right-to-left minimum and  $u_{t_i}$  is a left-to-right maximum;

$i \leftarrow 1$ ;  $\{i \text{ indicates position in } L \text{ of edge checked for inclusion in } E_1\}$

**while**  $i < |L|$  **do**

$j \leftarrow i + 1$ ;

$\{ \text{ignore all edges incident on the same left-to-right maximum...} \}$

$\{ \text{...except for the last one} \}$

**while**  $j \leq |L|$  **and**  $u_{t_j} = u_{t_i}$  **do**

$j \leftarrow j + 1$ ;

add the edge  $e_{j-1}$  in  $E_1$  with its *back-pointer* pointing to **NIL**;

$i \leftarrow j$ ;

**if**  $i = |L|$

**then**  $\{i = |L| \iff e_{|L|-1} \text{ is last edge included in } E_1 \text{ and } u_{t_{|L|-1}} \neq u_{t_{|L|}}\}$

add the edge  $e_{|L|}$  in  $E_1$  with its *back-pointer* pointing to **NIL**;

The correctness of Step 1 follows from Lemma 3, statement (ii): for each  $v_i$ , we consider only the edge  $v_i u_{q_i}$  where  $u_{q_i} = \text{lrmax\_above}[p(v_i)]$ . The correctness of Step 2 follows from Lemma 2: for the correctness of Step 2, it is important to note that because the  $y$ -coordinate of point *highest $_x$*  never decreases during the execution of Step 1, the edges  $v_{s_i} u_{t_i}$  and  $v_{s_j} u_{t_j}$  located in the  $i$ -th and  $j$ -th node of the list  $L$  (for any  $i < j$ ) have  $s_i < s_j$  and  $t_i \geq t_j$ . The edges in the resulting set  $E_1$  form a crossing pattern like the one shown in Figure 4.

### 3.2 Computing the Set $E_{i+1}$ from $E_i$

Let  $E_i = \{e_{i,1}, e_{i,2}, \dots, e_{i,h}\}$  be the set of candidate edges for the  $i$ -th edge in a minimum matching  $M$  such that the edges in  $M$  are of the form shown in Figure 1(b) and their endpoints dominate all the vertices of the given permutation graph  $G$ . As shown in Figure 4, the quadrants from left to right and from bottom to top are  $Q(e_{i,h}), Q(e_{i,h-1}), \dots, Q(e_{i,1})$ , respectively.

For the construction of  $E_{i+1}$ , we are interested in non-redundant edges  $e$  incident on a right-to-left minimum and on a left-to-right maximum such that there exists  $e_{i,j} \in E_i$  for which all the points in  $Q(e_{i,j}) \cap P_\pi$  are either covered by  $e$  or lie in the bottom right uncovered quadrant  $Q(e)$ , i.e.,

$$Q(e_{i,j}) \cap P_\pi = (C(e) \cup Q(e)) \cap P_\pi. \tag{3}$$

This case is a generalization of the case for  $E_1$ ; this time, however, we are dealing with a number of quadrants  $Q(e_{i,j})$ . The following lemma gives a complete coverage of all cases.

**Lemma 4.** *Let  $G$  be an embedded permutation graph with no isolated vertices,  $P_\pi = \{p_1, p_2, \dots, p_n\}$  the corresponding point set, and let  $u_1, u_2, \dots, u_\ell$  ( $v_1, v_2, \dots, v_\ell$ , resp.) be the left-to-right maxima (right-to-left minima, resp.) in  $P_\pi$  in order from left to right. Suppose further that the set  $E_i$  contains the edges  $e_{i,1}$ ,*



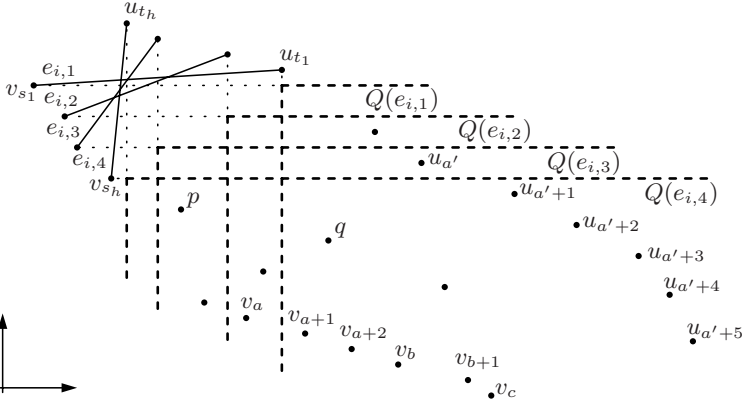


Fig. 4.

$e_{i,2}, \dots, e_{i,h}$ , each incident on a right-to-left minimum and a left-to-right maximum. If  $v_a = \text{rlmin\_left}[u_{t_1}]$ ,  $u_{a'} = \text{lrmax\_above}[v_{s_h}]$ ,  $v_b = \text{rlmin\_left}[u_{a'}]$ , and  $v_c = \text{rlmin\_left}[u_{a'+1}]$  (see Figure 4), we have:

- (i) The edge connecting  $v_a$  to  $\text{lrmax\_above}[v_a]$  satisfies Eq. (3) for  $j = 1$ .
- (ii) Consider  $v_k$ , where  $k = a+1, a+2, \dots, b$ . Let  $Q(e_{i,r})$  be the rightmost<sup>2</sup> (i.e., its left side is to the right of the left sides of the other quadrants) among the quadrants that do not contain points  $p \in P_\pi$  such that  $x(p) < x(v_k)$  and  $y(p) > y(v_{s_h})$ , and let  $u_{q_k} = \text{lrmax\_above}[p(v_k)]$  where  $p(v_k)$  is the highest point in  $P_\pi$  which belongs to  $Q(e_{i,r})$  and is not to the right of  $v_k$ . Then, Eq. (3) is satisfied for  $Q(e_{i,j}) = Q(e_{i,r})$  and the edge  $e = v_k u_{q_k}$ ; this does not hold for any edge  $e = v_k u_q$  with  $q > q_k$ .
- (iii) Consider  $v_k$ , where  $k = b+1, b+2, \dots, c$ . Suppose that there exists a quadrant  $Q(e_{i,r})$  that contains no points  $p \in P_\pi$  such that  $y(p) > y(u_{a'+1})$ , and let  $u_{q_k} = \text{lrmax\_above}[p(v_k)]$  where  $p(v_k)$  is the highest point in  $P_\pi$  which belongs to  $Q(e_{i,r})$  and is not to the right of  $v_k$ . Then, Eq. (3) is satisfied for  $Q(e_{i,j}) = Q(e_{i,r})$  and the edge  $e = v_k u_{q_k}$ ; this does not hold for any edge  $e_q = v_k u_q$  with  $q > q_k$ .
- (iv) Each edge incident on a right-to-left minimum to the left of  $v_a$  is redundant. Moreover, for any edge  $e$  incident on a right-to-left minimum to the right of  $v_c$ , there does not exist  $e_{i,j} \in E_i$  that satisfies Eq. (3) with  $e$ ; in fact, the same holds for any edge  $e$  incident on a right-to-left minimum  $v_k$ , where  $k = b, b+1, \dots, c$ , if every quadrant  $Q(\ )$  contains points  $p \in P_\pi$  such that  $y(p) > y(u_{a'+1})$ .

As an example for statement (ii), consider  $v_k = v_{a+2}$  in Figure 4; then all 4 quadrants  $Q(e_{i,1}), \dots, Q(e_{i,4})$  contain no points  $p \in P_\pi$  such that  $x(p) < x(v_k)$

<sup>2</sup> The quadrant  $Q(e_{i,r})$  is well defined, since the quadrant  $Q(e_{i,h})$  does not contain points  $p \in P_\pi$  such that  $x(p) < x(v_k)$  and  $y(p) > y(v_{s_h})$ .

and  $y(p) > y(v_{s_h})$ ; the rightmost quadrant  $Q(e_{i,r})$  is  $Q(e_{i,1})$ ,  $p(v_k) = q$ , and  $u_{q_k} = u_{a'+2}$ . On the other hand, in the case of  $v_k = v_b$ , the quadrants  $Q(e_{i,1})$  and  $Q(e_{i,2})$  contain a point  $p \in P_\pi$  such that  $x(p) < x(v_k)$  and  $y(p) > y(v_{s_h})$ ; the rightmost quadrant  $Q(e_{i,r})$  is  $Q(e_{i,3})$ ,  $p(v_k) = p$ , and  $u_{q_k} = u_{a'+1}$ . As an example for statement (iii), we may consider  $v_k = v_{b+1}$  or  $v_c$  in Figure 4 in either case,  $Q(e_{i,r}) = Q(e_{i,4})$ ,  $p(v_k) = p$ , and  $u_{q_k} = u_{a'+1}$ .

Our procedure for computing  $E_{i+1}$  takes advantage of Lemma 4. Similarly to Procedure Compute\_E<sub>1</sub>, it works in two steps: in the first step, it constructs a list  $L$  containing at most one edge incident on each of the right-to-left minima from  $v_a$  (inclusive) to  $v_b$  (inclusive), and potentially to  $v_c$  (inclusive) depending on whether the conditions of statement (iii) of the lemma hold; next, in a 2nd step, it selects only the non-redundant edges among the edges in  $L$ . In more detail, the procedure processes the points in  $P_\pi$  to the right of  $u_{t_h}$  up to  $v_b$  or  $v_c$  from left to right, and maintains in a stack only the quadrants that do not contain any point of  $P_\pi$  above the line  $y = y(v_{s_h})$  and stores with each of them its highest point so far. Then, for each right-to-left minimum encountered starting with  $v_a$ , it applies statement (i), (ii) or (iii) of Lemma 4.

For the case shown in Figure 4, at the end of the first step, the list  $L$  contains the edges  $v_a u_{a'+4}$ ,  $v_{a+1} u_{a'+4}$ ,  $v_{a+2} u_{a'+2}$ ,  $v_b u_{a'+1}$ ,  $v_{b+1} u_{a'+1}$ , and  $v_c u_{a'+1}$ . Among them, the edges  $v_a u_{a'+4}$ ,  $v_b u_{a'+1}$ , and  $v_{b+1} u_{a'+1}$  are redundant, so that the final set is  $\{v_{a+1} u_{a'+4}, v_{a+2} u_{a'+2}, v_c u_{a'+1}\}$ .

### 3.3 Time and Space Complexity of the Algorithm

Regarding the complexity of our algorithm, we can show the following theorem:

**Theorem 1.** *Let  $G$  be a permutation graph with no isolated vertices determined by a permutation  $\pi$  over the set  $N_n$ . Then, given  $\pi$ , our algorithm computes a minimum-cardinality paired-dominating set of  $G$  in  $O(n)$  time using  $O(n)$  space.*

## References

1. Atallah, M.J., Manacher, G.K., Urrutia, J.: Finding a minimum independent dominating set in a permutation graph. *Discrete Appl. Math.* 21, 177–183 (1988)
2. Brandstadt, A., Kratsch, D.: On domination problems for permutation and other graphs. *Theoret. Comput. Sci.* 54, 181–198 (1987)
3. Brešar, B., Henning, M.A., Rall, D.F.: Paired-domination of Cartesian products of graphs and rainbow domination. *Electr. Notes in Discrete Math.* 22, 233–237 (2005)
4. Chao, H.S., Hsu, F.R., Lee, R.C.T.: An optimal algorithm for finding the minimum cardinality dominating set on permutation graphs. *Discrete Appl. Math.* 102, 159–173 (2000)
5. Cheng, T.C.E., Kang, L.Y., Ng, C.T.: Paired domination on interval and circular-arc graphs. *Discrete Appl. Math.* 155, 2077–2086 (2007)
6. Cheng, T.C.E., Kang, L., Shan, E.: A polynomial-time algorithm for the paired-domination problem on permutation graphs. *Discrete Appl. Math.* 157, 262–271 (2009)

7. Dorbec, P., Gravier, S., Henning, M.A.: Paired-domination in generalized claw-free graphs. *J. Comb. Optim.* 14, 1–7 (2007)
8. Farber, M., Keil, J.M.: Domination in permutation graphs. *J. Algorithms* 6, 309–321 (1985)
9. Favaron, O., Henning, M.A.: Paired-domination in claw-free cubic graphs. *Graphs and Combinatorics* 20, 447–456 (2004)
10. Golombic, M.C.: *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, Inc., New York (1980)
11. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: *Fundamentals of Domination in Graphs*. Marcel Dekker, New York (1998)
12. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: *Domination in Graphs: Advanced Topics*. Marcel Dekker, New York (1998)
13. Haynes, T.W., Slater, P.J.: Paired-domination in graphs. *Networks* 32, 199–206 (1998)
14. Hedetniemi, S.T., Laskar, R. (eds.): *Topics on Domination*. *Ann. Discrete Math.*, vol. 48. North-Holland, Amsterdam (1991)
15. Helmbold, D., Mayr, E.W.: Applications of parallel algorithms to families of perfect graphs. *Computing* 7, 93–107 (1990)
16. Henning, M.A., Plummer, M.D.: Vertices contained in all or in no minimum paired-dominating set of a tree. *J. Comb. Optim.* 10, 283–294 (2005)
17. Kang, L., Sohn, M.Y., Cheng, T.C.E.: Paired-domination in inflated graphs. *Theor. Comput. Sci.* 320, 485–494 (2004)
18. Lu, C.L., Ko, M.-T., Tang, C.Y.: Perfect edge domination and efficient edge domination in graphs. *Discrete Appl. Math.* 119, 227–250 (2002)
19. McConnell, R.M., Spinrad, J.P.: Modular decomposition and transitive orientation. *Discrete Math.* 201, 189–241 (1999)
20. Nikolopoulos, S.D.: Coloring permutation graphs in parallel. *Discrete Appl. Math.* 120, 165–195 (2002)
21. Nikolopoulos, S.D., Papadopoulos, C.: On the performance of the First-Fit coloring algorithm on permutation graphs. *Inform. Process. Lett.* 75, 265–273 (2000)
22. Pnueli, A., Lempel, A., Even, S.: Transitive orientation of graphs and identification of permutation graphs. *Canadian J. Math.* 23, 160–175 (1971)
23. Qiao, H., Kang, L., Gardei, M., Du, D.-Z.: Paired-domination of trees. *J. of Global Optimization* 25, 43–54 (2003)
24. Reif, J. (ed.): *Synthesis of Parallel Algorithms*. Morgan Kaufmann Publishers, Inc., San Mateo (1993)
25. Rhee, C., Liang, Y.D., Dhall, S.K., Lakshmivarahan, S.: An  $O(n + m)$ -time algorithm for finding a minimum-weight dominating set in a permutation graph. *SIAM J. Comput.* 25, 404–419 (1996)
26. Spinrad, J.: On comparability and permutation graphs. *SIAM J. Comput.* 14, 658–670 (1985)
27. Srinivasan, A., Madhukar, K., Nagavamsi, P., Rangan, C.P., Chang, M.-S.: Edge domination on bipartite permutation graphs and cotriangulated graphs. *Inform. Process. Lett.* 56, 165–171 (1995)
28. Supowit, K.J.: Decomposing a set of points into chains, with applications to permutation and circle graphs. *Inform. Process. Lett.* 21, 249–252 (1985)
29. Tsai, K.H., Hsu, W.L.: Fast algorithms for the dominating set problem on permutation graphs. *Algorithmica* 9, 601–614 (1993)
30. Yannakakis, M., Gavril, F.: Edge domination sets in graphs. *SIAM J. Appl. Math.* 38, 364–372 (1980)

# Simpler Parameterized Algorithm for OCT

Daniel Lokshтанov<sup>1</sup>, Saket Saurabh<sup>1,\*</sup>, and Somnath Sikdar<sup>2</sup>

<sup>1</sup> The University of Bergen, Norway

{daniello,saket.saurabh}@ii.uib.no

<sup>2</sup> The Institute of Mathematical Sciences, India

somnath@imsc.res.in

**Abstract.** We give a simple and intuitive fixed parameter tractable algorithm for the ODD CYCLE TRANSVERSAL problem, running in time  $O(3^k \cdot k \cdot |E| \cdot |V|)$ . Our algorithm is best viewed as a reinterpretation of the classical Iterative Compression algorithm for ODD CYCLE TRANSVERSAL by Reed, Smith and Vetta [8].

## 1 Introduction

ITERATIVE COMPRESSION is a tool that has recently been used successfully to solve a number of problems in Parameterized Complexity. This technique was first introduced by Reed, Smith and Vetta in order to solve the ODD CYCLE TRANSVERSAL problem. In this problem we are given a graph  $G$  together with an integer  $k$ . The objective is to find a set  $S$  of at most  $k$  vertices whose deletion makes the graph bipartite [8], and a set  $S$  such that  $G \setminus S$  is bipartite is called an odd cycle transversal of  $G$ . The method of Iterative Compression was used in obtaining faster fixed parameter tractable (FPT) algorithms for FEEDBACK VERTEX SET, EDGE BIPARTIZATION, CHORDAL DELETION and CLUSTER VERTEX DELETION on undirected graphs [2,3,6,4]. The technique was also used by Chen et al. [1] to show that the DIRECTED FEEDBACK VERTEX SET problem is FPT, resolving a long standing open problem in Parameterized Complexity.

While the algorithm of Reed, Smith and Vetta for ODD CYCLE TRANSVERSAL was a breakthrough for parameterized algorithms, the algorithm and correctness proof is quite hard to understand. In an attempt to remedy this, Hüffner [5] provided an alternative algorithm for the problem. In this paper we give yet another algorithm for OCT. We believe that our algorithm is simpler and more intuitive than the previous versions.

## 2 The Method of Iterative Compression

The method of Iterative Compression was introduced by Reed et al. [8] in order to solve the ODD CYCLE TRANSVERSAL (OCT) problem. The idea is to reduce the problem in question to a modified version, where we are also given as input a

---

\* Supported by the Norwegian Research Council.

solution that is almost good enough, but not quite. For the case of ODD CYCLE TRANSVERSAL, we are given an odd cycle transversal  $S'$  of  $G$  of size  $k + 1$ . We call this problem the *compression* version of ODD CYCLE TRANSVERSAL. The crux of the Iterative Compression method is that often the compression version of a problem is easier to solve than the original one.

Suppose we could solve the compression version of the problem in  $O(f(k)n^c)$  time. We show how to solve the original problem in  $O(f(k)n^{c+1})$  time. Order the vertices of  $V(G)$  into  $v_1v_2 \dots v_n$  and define  $V_i = \{v_1 \dots v_i\}$  for every  $i$ . Notice that if  $G$  has an odd cycle transversal  $S$  of size  $k$  then  $S \cap V_i$  is an odd cycle transversal of  $G[V_i]$  for every  $i \leq n$ . Furthermore, if  $S$  is an odd cycle transversal of  $G[V_i]$  then  $S \cup \{v_{i+1}\}$  is an odd cycle transversal of  $G[V_{i+1}]$ . Finally,  $V_k$  is an odd cycle transversal of size  $k$  of  $G[V_k]$ . These three facts together with the  $f(k)n^c$  algorithm for the compression version of OCT give a  $f(k)n^{c+1}$  time algorithm for OCT as follows. Call the algorithm for the compression version with input  $(G[V_{k+1}], V_{k+1}, k)$ . The algorithm will either report that  $(G[V_{k+1}], k)$  has no odd cycle transversal of size  $k$  or return such an odd cycle transversal, call it  $S_{k+1}$ . In the first case  $G$  has no  $k$ -sized odd cycle transversal. In the second, call the algorithm for the compression version with input  $(G[V_{k+2}], S_{k+1} \cup \{v_{k+2}\}, k)$ . Again we either receive a “no” answer or a  $k$ -sized odd cycle transversal  $S_{k+2}$  of  $G[V_{k+2}]$  and again, if the answer is negative then  $G$  has no  $k$ -sized odd cycle transversal. Otherwise we call the compression algorithm with input  $(G, S_{k+2} \cup \{v_{k+3}\}, k)$  and keep going on in a similar manner. If we receive a negative answer at some step we answer that  $G$  has no  $k$ -sized odd cycle transversal. If we do not receive a negative answer at any step, then after  $n - k$  calls to the compression algorithm we have a  $k$ -sized odd cycle transversal of  $G[V_n] = G$ . Thus we have resolved the input instance in time  $O(f(k)n^{c+1})$ .

We refer to [7] for a more thorough introduction to Iterative Compression.

### 3 An Algorithm for Odd Cycle Transversal

We now show how to solve the compression version of ODD CYCLE TRANSVERSAL in time  $O(3^k \cdot k \cdot |E|)$ . From the discussion in Section 2 it will follow that OCT can be solved in time  $O(3^k \cdot k \cdot |E| \cdot |V|)$ . For two vertex subsets  $V_1$  and  $V_2$  of  $V(G)$  a walk from  $V_1$  to  $V_2$  is a walk with one endpoint in  $V_1$  and the other in  $V_2$ , or a single vertex in  $V_1 \cap V_2$ . The following is a simple fact about bipartite graphs.

**Fact 1.** *Let  $G = (V_1 \uplus V_2, E)$  be a bipartite graph with vertex bipartition  $V_1 \uplus V_2$ . Then*

1. *For  $i \in \{1, 2\}$ , no walk from  $V_i$  to  $V_i$  has odd length.*
2. *No walk from  $V_1$  to  $V_2$  has even length.*

A walk in a graph is an alternating sequence of vertices and edges  $v_0, e_1, v_1, e_2, \dots, v_n$ , such that  $e_i = (v_{i-1}, v_i)$  is an edge for  $i \in \{1, \dots, n\}$ . The length  $l$  of a walk is the number of edges used in the sequence.

Recall that we are given a graph  $G$  and an odd cycle transversal  $S'$  of  $G$  of size  $k + 1$  and we have to decide whether  $G$  has an odd cycle transversal of size at most  $k$ . If such an odd cycle transversal  $S$  exists then there exists a partition of  $S'$  into  $L \uplus R \uplus T$ , where  $T = S' \cap S$  and  $L$  and  $R$  are subsets of the left and right bipartitions of the resulting graph. The algorithm iterates over all  $3^k$  partitions of  $S$  into  $L \uplus R \uplus T$ . For each partition we run an algorithm that takes as input a partition of  $S'$  into  $L \uplus R \uplus T$ , runs in  $O(k \cdot |E|)$  time and decides whether there exists a set of vertices  $T'$  of size at most  $k - |T|$  in  $G \setminus S'$  such that  $G \setminus (T \cup T')$  is bipartite with bipartitions  $V_L$  and  $V_R$  such that  $L \subseteq V_L$  and  $R \subseteq V_R$ . In the remainder of this section we give such an algorithm. This algorithm together with the outer loop over all partitions of  $S'$  yields the  $O(3^k \cdot k \cdot |E|)$  time algorithm for the compression step.

Before proceeding we do a simple “sanity check”. If there is an edge in  $G[L]$  or  $G[R]$  it is clear that  $X$  can not exist since then either  $V_L$  or  $V_R$  can not be an independent set. Hence if there is an edge in  $G[L]$  or  $G[R]$  we can immediately skip to the next partition of  $S'$ . Now, since  $G \setminus S'$  is bipartite, let  $A \uplus B$  be a bipartition of  $G \setminus S'$ . Let  $A_l$  and  $B_l$  be the neighbors of  $L$  in  $A$  and  $B$  respectively. Similarly let  $A_r$  and  $B_r$  be the neighbours of  $R$  in  $A$  and  $B$  respectively.

**Lemma 1.** *Let  $(G, S', k)$  be an instance of the compression version of ODD CYCLE TRANSVERSAL and let  $S' = L \uplus R \uplus T$ . If  $X \subseteq (V(G) \setminus S')$  is a set of vertices such that  $G \setminus (T \cup X)$  is bipartite with bipartitions  $V_L$  and  $V_R$  such that  $L \subseteq V_L$  and  $R \subseteq V_R$ , then in  $G \setminus (S' \cup X)$ , there are no paths between  $A_l$  and  $B_l$ ;  $B_l$  and  $B_r$ ;  $B_r$  and  $A_r$ ; and,  $A_r$  and  $A_l$ .*

*Proof.* Any path from  $A_l$  to  $B_l$  in  $G \setminus (S' \cup X)$  has odd length and can be extended to a walk from  $L$  to  $L$  of odd length in  $G' \setminus (T \cup X)$ , contradicting Fact  $\square$ . A symmetric argument shows that there are no paths between  $B_r$  and  $A_r$  in  $G \setminus (S' \cup X)$ . Any path from  $B_l$  to  $B_r$  in  $G \setminus (S' \cup X)$  must be of even length and can be extended to a walk in  $G \setminus (T \cup X)$  from  $L$  to  $R$  of even length, again contradicting Fact  $\square$ . A symmetric argument yields that there are no paths between  $A_r$  and  $A_l$ .

**Lemma 2.** *Let  $(G, S', k)$  be an instance of the compression version of ODD CYCLE TRANSVERSAL and let  $S = L \uplus R \uplus T$  such that  $G[L]$  and  $G[R]$  are independent sets. Let  $X$  be a set of vertices in  $V(G) \setminus S'$  such that in  $G \setminus (S' \cup X)$ , there are no paths between  $A_l$  and  $B_l$ ;  $B_l$  and  $B_r$ ;  $B_r$  and  $A_r$ ; and,  $A_r$  and  $A_l$ . Then  $G \setminus (T \cup X)$  is bipartite with bipartitions  $V_L$  and  $V_R$  such that  $L \subseteq V_L$  and  $R \subseteq V_R$ .*

*Proof.* Notice that every path from a vertex in  $L$  to another vertex in  $L$  with inner vertices only in  $V(G) \setminus (S' \cup X)$  must have even length. Similarly every path from a vertex in  $R$  to another vertex in  $R$  with inner vertices only in  $V(G) \setminus (S' \cup X)$  must have even length and every path from a vertex in  $L$  to a vertex in  $R$  with inner vertices only in  $V(G) \setminus (S' \cup X)$  must have odd length. Since  $G[L]$  and  $G[R]$  are independent sets it follows that if  $G \setminus (T \cup X)$  is bipartite then it has bipartitions  $V_L$  and  $V_R$  such that  $L \subseteq V_L$  and  $R \subseteq V_R$ . We now prove that  $G \setminus (T \cup X)$  is bipartite.

Consider a cycle in  $G \setminus (T \cup X)$ . If  $C$  does not contain any vertices of  $(L \cup R)$  then  $|E(C)|$  is even since  $G \setminus S'$  is bipartite. Let  $v_1, v_2, \dots, v_t$  be the vertices of  $(L \cup R) \cap C$  in their order of appearance along  $C$ . Let  $v_0 = v_t$ , then we have that  $|E(C)| = \sum_{i=0}^{t-1} d_c(v_i, v_{i+1})$ . But then  $E(C)$  must be even since the number of indices  $i$  such that  $v_i \in L$  and  $v_{i+1} \in R$  is equal to the number of indices  $j$  such that  $v_j \in R$  and  $v_{j+1} \in L$ . This concludes the proof.

To check whether  $G \setminus T$  has an odd cycle transversal  $X$  such that  $G \setminus (T \cup X)$  is bipartite with bipartitions  $V_L$  and  $V_R$  such that  $L \subseteq V_L$  and  $R \subseteq V_R$  we proceed as follows. Construct an auxiliary graph  $\tilde{G}$  from  $G \setminus S'$  by introducing two special vertices  $s, t$  and connecting  $s$  to each vertex in  $A_l \cup B_r$  and  $t$  to each vertex in  $A_r \cup B_l$ . Lemmas 1 and 2 show that it is sufficient to check whether there is an  $st$ -separator in  $\tilde{G}$  of size at most  $k - |T \cup T'|$ . This can be done using max flow in time  $O(k \cdot |E|)$ . These discussions together with Lemmata 1 and 2 bring us to the following theorem.

**Theorem 2.** *There is an algorithm that given a graph  $G = (V, E)$  and integer  $k$  decides whether  $G$  has an OCT of size at most  $k$  in time  $O(3^k \cdot k \cdot |E| \cdot |V|)$ .*

## 4 Concluding Remarks

In this paper we gave an alternate algorithm for ODD CYCLE TRANSVERSAL based on the Iterative Compression technique. Traditionally, algorithms that use Iterative Compression partition the given  $k+1$ -sized solution into two parts. Our algorithm is the first to partition this set in three parts. This is a key element in deriving our algorithm. We believe that partitioning the given  $k+1$ -sized solution into more than two parts will be useful in designing Iterative Compression based algorithms.

**Acknowledgments.** The authors would like to thank Michael Fellows, Fedor V. Fomin, Rolf Niedermeier, Venkatesh Raman and Frances Rosamond for helpful discussions and for suggesting to put this article in print.

## References

1. Chen, J., Liu, Y., Lu, S., Razgon, I., O'Sullivan, B.: A Fixed-Parameter Algorithm for the Directed Feedback Vertex Set Problem. *Journal of the ACM* 55(5) (2008)
2. Dehne, F., Fellows, M., Langston, M., Rosamond, F., Stevens, K.: An  $O(2^{O(k)} n^3)$  FPT algorithm for the undirected feedback vertex set problem. *Theory of Comput. Syst.* 41(3), 479–492 (2007)
3. Guo, J., Gramm, J., Hüffner, F., Niedermeier, R., Wernicke, S.: Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *J. Comput. Syst. Sci.* 72(8), 1386–1396 (2006)
4. Hüffner, F., Komusiewicz, C., Moser, H., Niedermeier, R.: Fixed-parameter algorithms for cluster vertex deletion. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) *LATIN 2008*. LNCS, vol. 4957, pp. 711–722. Springer, Heidelberg (2008)

5. Hüffner, F.: Algorithm engineering for optimal graph bipartization. *Journal of Graph Algorithms and Applications* 13(2), 77–98 (2009)
6. Marx, D.: Chordal deletion is fixed-parameter tractable. In: Fomin, F.V. (ed.) *WG 2006*. LNCS, vol. 4271, pp. 37–48. Springer, Heidelberg (2006)
7. Niedermeier, R.: *An Invitation to Fixed-Parameter Algorithms*. Oxford University Press, Oxford (2006)
8. Reed, B., Vetta, A., Smith, K.: Finding Odd Cycle Transversals. *Operations Research Letters* 32, 229–301 (2004)



# Bipartite Graphs of Large Clique-Width<sup>\*</sup>

Nicholas Korpelainen and Vadim V. Lozin

DIMAP and Mathematics Institute, University of Warwick, Coventry CV4 7AL, UK

**Abstract.** Recently, several constructions of bipartite graphs of large clique-width have been discovered in the literature. In the present paper, we propose a general framework for developing such constructions and use it to obtain new results on this topic.

**Keywords:** Clique-width, Tree-width, Hereditary class.

## 1 Introduction

Clique-width is a relatively young notion generalizing another important graph parameter, *tree-width*, studied in the literature for decades. The notion of clique-width generalizes that of tree-width in the sense that graphs of bounded tree-width have bounded clique-width.

The importance of these graph invariants is due to the fact that numerous problems that are NP-hard in general admit polynomial-time solutions when restricted to graphs of bounded tree- or clique-width (see e.g. [16]). The celebrated result due to Robertson and Seymour [13] states that for any planar graph  $H$  there is an integer  $N$  such that the tree-width of graphs containing no  $H$  as a minor is at most  $N$ . In other words, the planar graphs constitute a unique minimal minor-closed class of graphs of unbounded tree-width. A special role in this class is assigned to rectangle grids, because every planar graph is a minor of some large enough grid and grids can have arbitrarily large tree-width. Therefore, grids form the only “unavoidable minors” in graphs of large tree-width. In the study of the notion of tree-width, the restriction to the graph minor relation is justified by the fact that the tree-width of a graph cannot be less than the tree-width of its minor. This is not the case with respect to the notion of clique-width. Therefore, in the study of this notion the restriction to the graph minor relation is not valid anymore. Instead, we restrict ourselves to the induced subgraph relation, because the clique-width of a graph cannot be less than the clique-width of any of its induced subgraphs [7]. The family of graph classes closed under taking induced subgraphs is much richer than that of minor-closed graph classes, and we believe that the set of “unavoidable induced subgraphs” of large clique-width is more diverse than the set of “unavoidable minors” of large tree-width. In this paper, we restrict ourselves to bipartite graphs.

---

<sup>\*</sup> Research supported by the Centre for Discrete Mathematics and Its Applications (DIMAP), University of Warwick.

Recently, several constructions of bipartite graphs of large clique-width have been discovered in the literature (see e.g. [3,10,12]). In the present paper, we propose a general framework for developing such constructions and use it to obtain new results on this topic.

## 2 Preliminaries

All graphs in this paper are undirected, without loops and multiple edges. For a graph  $G$ , we denote by  $V(G)$  and  $E(G)$  the vertex set and the edge set of  $G$ , respectively. The neighborhood of a vertex  $v \in V(G)$  is the set of vertices adjacent to  $v$  and the degree of  $v$  is the size of its neighborhood. For a bipartite graph  $G = (V_1, V_2, E)$ , the bipartite complement of  $G$  is the bipartite graph  $\bar{G} = (V_1, V_2, V_1 \times V_2 - E)$ . Given a graph  $G$  and a subset of vertices  $U \subseteq V(G)$ , two vertices of  $U$  will be called  $U$ -similar if they have the same neighborhood outside  $U$ . Clearly,  $U$ -similarity is an equivalence relation. The number of equivalence classes of  $U$  will be denoted  $\mu(U)$ . Also, by  $G[U]$  we will denote the subgraph of  $G$  induced by  $U$ , i.e., the subgraph of  $G$  with vertex set  $U$  and two vertices being adjacent in  $G[U]$  if and only if they are adjacent in  $G$ . We say that a graph  $H$  is an induced subgraph of  $G$  if  $H$  is isomorphic to  $G[U]$  for some  $U \subseteq V(G)$ .

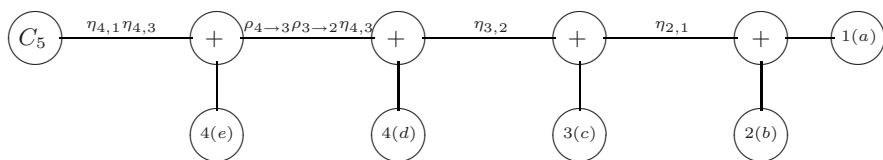
The notion of clique-width of a graph was introduced in [5] and is defined as the minimum number of labels needed to construct the graph by means of the four graph operations: creation of a new vertex  $v$  with label  $i$  (denoted  $i(v)$ ), disjoint union of two labeled graphs  $G$  and  $H$  (denoted  $G \oplus H$ ), connecting vertices with specified labels  $i$  and  $j$  (denoted  $\eta_{i,j}$ ) and renaming label  $i$  to label  $j$  (denoted  $\rho_{i \rightarrow j}$ ). The clique-width of a graph  $G$  will be denoted  $\text{cwd}(G)$ .

Every graph can be defined by an algebraic expression using the four operations above. This expression will be called a  $k$ -expression if it uses  $k$  different labels. For instance, the cycle  $C_5$  on vertices  $a, b, c, d, e$  (listed along the cycle) can be defined by the following 4-expression:

$$\eta_{4,1}(\eta_{4,3}(4(e) \oplus \rho_{4 \rightarrow 3}(\rho_{3 \rightarrow 2}(\eta_{4,3}(4(d) \oplus \eta_{3,2}(3(c) \oplus \eta_{2,1}(2(b) \oplus 1(a))))))))).$$

Alternatively, any algebraic expression defining  $G$  can be represented as a rooted tree, whose leaves correspond to the operations of vertex creation, the internal nodes correspond to the  $\oplus$ -operations, and the root is associated with  $G$ . The operations  $\eta$  and  $\rho$  are assigned to the respective edges of the tree. Figure 1 shows the tree representing the above expression defining a  $C_5$ .

As we mentioned earlier, in the study of the notion of clique-width we may be restricted to graph classes that are closed under taking induced subgraphs. Such classes are known in the literature as *hereditary classes*. If a class of graphs  $X$  is not hereditary, we can extend it to a hereditary class by adding to it all induced subgraphs of graphs in  $X$ . The hereditary closure of  $X$  will be denoted  $[X]$ . It is known that a class of graphs is hereditary if and only if it can be characterized in terms of forbidden induced subgraphs. More formally, given a set of graphs  $M$ , we say that a graph  $G$  is  $M$ -free if  $G$  does not contain induced subgraphs



**Fig. 1.** The tree representing the expression defining a  $C_5$

from the set  $M$ . The class of all  $M$ -free graphs will be denoted  $Free(M)$ . Then a class  $X$  is hereditary if and only if  $X = Free(M)$  for a certain set  $M$ .

Recently, the clique-width has been shown to be unbounded in several hereditary classes of bipartite graphs, such as chordal bipartite [2], bipartite permutation [3],  $P_7$ -free bipartite [12], bipartite graphs of bounded vertex degree and large girth [10]. Our goal is to identify minimal hereditary classes of graphs of unbounded clique-width. From this perspective, the class of chordal bipartite graphs is of no interest, because it properly contains another class of unbounded clique-width, namely, bipartite permutation graphs. On the contrary, in any proper hereditary subclass of bipartite permutation graphs the clique-width is bounded by a constant (see [8] for a related result), i.e., the role of bipartite permutation graphs in the family of hereditary classes is analogous to the role of planar graphs in the family of minor-closed graph classes. This makes the class of bipartite permutation graphs critically important in the study of the notion of clique-width. In the attempt to identify more critical classes with respect to this notion, we propose in the next section a general framework for constructing bipartite graphs of large clique-width.

### 3 Building Blocks and Building Operations

In our framework for constructing bipartite graphs of large clique-width we distinguish three basic building blocks and two building operations.

Building blocks:

$B_n$  : the graph  $B_n$  has  $2n$  vertices  $x_1, \dots, x_n$  and  $y_1, \dots, y_n$  and edges connecting, for each  $i = 1, \dots, n$ , vertex  $x_i$  to vertices  $y_j$  with  $j \geq i$ .

$M_n$  : the graph  $M_n$  has  $2n$  vertices  $x_1, \dots, x_n$  and  $y_1, \dots, y_n$  and edges connecting vertex  $x_i$  to  $y_i$  for each  $i = 1, 2, \dots, n$ , i.e.,  $M_n$  is a collection of  $n$  disjoint edges.

$F_n$  : the graph  $F_n$  is the bipartite complement of  $M_n$ .

Let  $X_n$  denote any of the building blocks described above. Notice that the clique-width of  $X_n$  is at most 3 regardless of the choice of the block. Now we define two operations by means of which we will create graphs of large clique-width out of  $X_n$ . In the description of the operations, we use the following terminological convention: given a set  $C = \{c_{i,j} \mid 1 \leq i, j \leq n\}$  of  $n^2$  elements, we call the

elements  $c_{i,1}, \dots, c_{i,n}$  the  $i$ -th row of  $C$ , and we call the elements  $c_{1,j}, \dots, c_{n,j}$  the  $j$ -th column of  $C$ .

Building operations:

- \*  $n$ -concatenation  $n * X_n$  is the graph with  $n^2$  vertices  $C = \{c_{i,j} \mid 1 \leq i, j \leq n\}$  such that any two consecutive rows of  $C$  induce a copy of  $X_n$ , and there are no other edges in the graph.
- \* orthogonal concatenation  $X_n^{(2)}$  is the graph with  $2n+n^2$  vertices  $A = \{a_1, \dots, a_n\}$ ,  $B = \{b_1, \dots, b_n\}$  and  $C = \{c_{i,j} \mid 1 \leq i, j \leq n\}$  such that
  - o  $A \cup B$  and  $C$  are independent sets;
  - o in the subgraph of  $X_n^{(2)}$  induced by  $A$  and  $C$ , the vertices of the same row of  $C$  have the same neighborhood, and by contracting each row of  $C$  to a single vertex we obtain an  $X_n$ ;
  - o in the subgraph of  $X_n^{(2)}$  induced by  $B$  and  $C$ , the vertices of the same column of  $C$  have the same neighborhood, and by contracting each column of  $C$  to a single vertex we obtain an  $X_n$ .

*Examples*

1. The graph  $n * B_n$  was studied in [3,11]. In particular, in [3] it was shown that the clique-width of  $n * B_n$  is at least  $n/6$  and that  $n * B_n$  is a bipartite permutation graph. Moreover, in [11] it was proved that  $n * B_n$  is an  $n$ -universal bipartite permutation graph, i.e., it contains every bipartite permutation graph with  $n$  vertices as an induced subgraph. In other words, the role of the graph  $n * B_n$  in the class of bipartite permutation graphs is analogous to the role of the grids in the class of planar graphs. We also repeat that the role of bipartite permutation graphs in the family of hereditary classes is analogous to the role of planar graphs in the family of minor closed graph classes, as  $\{n * B_n : n \geq 1\}$  is a minimal hereditary class of unbounded clique-width.

2. The graph  $B_n^{(2)}$  was introduced in [4] and was shown there to have clique-width at least  $n$ . Therefore, the clique-width of graphs in the class  $\{B_n^{(2)} : n \geq 1\}$  is unbounded. However, whether this is a minimal hereditary class of unbounded clique-width is an open question.

In the next two sections we will study more constructions obtained by means of the above operations. Not each of them leads to a graph of large clique-width. For instance,  $n * M_n$  is the disjoint union of  $n$  chordless paths and therefore the clique-width of  $n * M_n$  is at most 3. However,  $n$ -concatenation of the bipartite complement of  $M_n$ , i.e., the graph  $n * F_n$ , has large clique-width, as we show in Section 4. Moreover, in the same section we show that  $\mathcal{F} := \{n * F_n : n \geq 1\}$  is a minimal hereditary class of unbounded clique-width.

In Section 5, we study the class  $\mathcal{M} := \{M_n^{(2)} : n \geq 1\}$  and show that it is also of unbounded clique-width. However, this class is not a minimal hereditary class of unbounded clique-width. Moreover, we discover an infinite decreasing sequence  $\mathcal{M}_1 \supset \mathcal{M}_2 \supset \dots$  of subclasses of  $\mathcal{M}$  of unbounded clique-width. We also show that the limit class of this sequence, i.e., the class  $\bigcap_{i \geq 1} \mathcal{M}_i$ , is unique

in the sense that by excluding any graph from this class we obtained a subclass of  $\mathcal{M}$  of bounded clique-width.

### 4 The Class $\mathcal{F}$

Recall that  $\mathcal{F}$  is the hereditary closure of the set  $\{n * F_n : n \geq 1\}$ . Throughout the section we denote the set of vertices of the graph  $n * F_n$  by  $V = \{v_{i,j} : 1 \leq i, j \leq n\}$ . Also, the subgraph of  $n * F_n$  induced by any  $k$  consecutive rows of  $V$  will be denoted  $k * F_n$ , i.e.  $2 * F_n = F_n$ .

**Theorem 1.** *The clique-width of the graph  $n * F_n$  is at least  $\lfloor n/2 \rfloor$ .*

*Proof.* Let  $\text{cwd}(n * F_n) = t$ . Denote by  $\tau$  a  $t$ -expression defining  $n * F_n$  and by  $\text{tree}(\tau)$  the rooted tree representing  $\tau$ . The subtree of  $\text{tree}(\tau)$  rooted at a node  $x$  will be denoted  $\text{tree}(x, \tau)$ . This subtree corresponds to a subgraph of  $n * F_n$ , which will be denoted  $F(x)$ . The label of a vertex  $v$  of the graph  $n * F_n$  at the node  $x$  is defined as the label that  $v$  has immediately prior to applying the operation  $x$ .

Let  $a$  be a lowest  $\oplus$ -node in  $\text{tree}(\tau)$  such that  $F(a)$  contains a full row of  $V$ . Denote the children of  $a$  in  $\text{tree}(\tau)$  by  $b$  and  $c$ . Let us color all vertices in  $F(b)$  blue and all vertices in  $F(c)$  red, and the remaining vertices of  $n * F_n$  yellow. Note that by the choice of  $a$  the graph  $n * F_n$  contains a non-yellow row (i.e. a row each vertex of which is non-yellow), but none of its rows is entirely red or blue. We denote a non-yellow row of  $n * F_n$  by  $r$ . Without loss of generality we assume that  $r \leq \lfloor n/2 \rfloor$  and that the row  $r$  contains at least  $n/2$  red vertices, since otherwise we could consider the rows in reverse order and swap colors red and blue.

Observe that edges of  $n * F_n$  between different colored vertices are not present in  $F(a)$ . Therefore, if a non-red vertex distinguishes two red vertices  $u$  and  $v$ , then  $u$  and  $v$  must have different labels at the node  $a$ . We will use this fact to show that  $F(a)$  contains a set  $U$  of at least  $\lfloor n/2 \rfloor$  vertices with pairwise different labels at the node  $a$ . Such a set can be constructed by the following procedure.

1. Set  $i = r$ ,  $U = \emptyset$  and  $J = \{j : v_{r,j} \text{ is red}\}$ .
2. Set  $K = \{j \in J : v_{i+1,j} \text{ is non-red}\}$ .
3. If  $K \neq \emptyset$ , add the vertices  $\{v_{i,k} : k \in K\}$  to  $U$ . Remove members of  $K$  from  $J$ .
4. If  $J = \emptyset$ , terminate the procedure.
5. Increase  $i$  by 1. If  $i = n$ , choose an arbitrary  $j \in J$ , put  $U = \{v_{m,j} : r \leq m \leq n - 1\}$  and terminate the procedure.
6. Go back to Step 2.

It is not difficult to see that this procedure must terminate. To complete the proof, it suffices to show that whenever the procedure terminates, the size of  $U$  is at least  $\lfloor n/2 \rfloor$  and the vertices in  $U$  have pairwise different labels at the node  $a$ .

First, suppose that the procedure terminates in Step 5. Then  $U$  is a subset of red vertices from at least  $\lfloor n/2 \rfloor$  consecutive rows of column  $j$ . Consider two vertices  $v_{l,j}, v_{m,j} \in U$  with  $l < m$ . According to the above procedure,  $v_{m+1,j}$  is red. Since  $n * F_n$  does not contain an entirely red row, the vertex  $v_{m,j}$  must have a non-red neighbor  $w$  in row  $m + 1$ . But  $w$  is not a neighbor of  $v_{l,j}$ , trivially. We conclude that  $v_{l,j}$  and  $v_{m,j}$  have different labels. Since  $v_{l,j}$  and  $v_{m,j}$  have been chosen arbitrarily, the vertices of  $U$  have pairwise different labels.

Now suppose that the procedure terminates in Step 4. By analyzing Steps 2 and 3, it is easy to deduce that  $U$  is a subset of red vertices of size at least  $\lfloor n/2 \rfloor$ . Suppose that  $v_{l,j}$  and  $v_{m,k}$  are two vertices in  $U$  with  $l \leq m$ . The procedure certainly guarantees that  $j \neq k$  and that both  $v_{l+1,j}$  and  $v_{m+1,k}$  are non-red. If  $m \in \{l, l+2\}$ , then it is clear that  $v_{l+1,j}$  distinguishes vertices  $v_{l,j}$  and  $v_{m,k}$ , and therefore these vertices have different labels. If  $m \notin \{l, l+2\}$ , we may consider vertex  $v_{m-1,k}$  which must be red. Since  $n * F_n$  does not contain an entirely red row, the vertex  $v_{m,k}$  must have a non-red neighbor  $w$  in row  $m - 1$ . But  $w$  is not a neighbor of  $v_{l,j}$ , trivially. We conclude that  $v_{l,j}$  and  $v_{m,k}$  have different labels, and therefore, the vertices of  $U$  have pairwise different labels. The proof is complete.  $\square$

By Theorem 1, the clique-width of graphs in  $\mathcal{F}$  is unbounded. Now let us show that  $\mathcal{F}$  is a minimal hereditary class of unbounded clique-width. To this end, we will employ the following technical lemma proved in [8].

**Lemma 1.** *If the vertices of a graph  $G$  can be partitioned into subsets  $V_1, V_2, \dots$  in such a way that for each  $i$ ,*

- (1)  $\text{cwd}(G[V_i]) \leq l$  with  $l \geq 2$ ,
- (2)  $\mu(V_i) \leq m$  and  $\mu(V_1 \cup \dots \cup V_i) \leq m$ ,

*then  $\text{cwd}(G) \leq lm$ .*

In particular, Lemma 1 implies the following corollary.

**Corollary 1.** *The clique-width of  $k * F_n$  is at most  $2k$ .*

*Proof.* Denote by  $V_i$  the  $i$ -th column of  $k * F_n$ . Since each column induces an independent set, it is clear that  $\text{cwd}(G[V_i]) \leq 2$  for every  $i$ . Trivially,  $\mu(V_i) \leq k$ , since  $|V_i| = k$ . Also, denoting  $W_i := V_1 \cup \dots \cup V_i$ , it is not difficult to see that  $\mu(W_i) \leq k$  for every  $i$ , since the vertices of the same row in  $W_i$  are  $W_i$ -similar. Now the conclusion follows from Lemma 1.  $\square$

Now we use Lemma 1 and Corollary 1 to prove the following result.

**Lemma 2.** *For any fixed  $k \geq 1$ , the clique-width of  $k * F_k$ -free graphs in the class  $\mathcal{F}$  is bounded by a function of  $k$ .*

*Proof.* Let  $k$  be a fixed number and  $G$  be a  $k * F_k$ -free graph in  $\mathcal{F}$ . By definition of  $\mathcal{F}$ , the graph  $G$  is an induced subgraph of  $n * F_n$  for some  $n$ . For convenience, assume that  $n$  is a multiple of  $k$ , say  $n = tk$ . The vertices of  $n * F_n$  that induce

$G$  will be called *black* and the remaining vertices of  $n * F_n$  will be called *white*. Also, we will refer to the set of vertices of  $G$  in the same row of  $n * F_n$  as a layer of  $G$ .

For  $1 \leq i \leq t$ , let us denote by  $W_i$  the subgraph of  $n * F_n$  induced by the  $k$  consecutive rows  $(i - 1)k + 1, (i - 1)k + 2, \dots, ik$ . For simplicity, we will use the term ‘row  $r$  of  $W_i$ ’ when referring to the row  $(i - 1)k + r$  of  $n * F_n$ . We partition the vertices of  $G$  into subsets  $V_1, V_2, \dots, V_t$  according to the following procedure:

1. Set  $V_j = \emptyset$  for  $1 \leq j \leq t$ . Add every black vertex of  $W_1$  to  $V_1$ . Set  $i = 2$ .
2. For  $j = 1, \dots, n$ ,
  - if column  $j$  of  $W_i$  is entirely black, then add the first vertex of this column to  $V_{i-1}$  and the remaining vertices of the column to  $V_i$ .
  - otherwise, add the (black) vertices of column  $j$  preceding the first white vertex to  $V_{i-1}$  and add the remaining black vertices of the column to  $V_i$ .
3. Increase  $i$  by 1. If  $i = t + 1$ , terminate the procedure.
4. Go back to Step 2.

Let us show that the partition  $V_1, V_2, \dots, V_t$  given by the procedure satisfies the assumptions of Lemma 1 with  $l$  and  $m$  depending only on  $k$ .

The procedure clearly assures that each  $G[V_i]$  is an induced subgraph of  $W_i \cup W_{i+1}$ . By Corollary 1, we have  $\text{cwd}(W_i \cup W_{i+1}) = \text{cwd}(2k * F_n) \leq 4k$ . Since the clique-width of an induced subgraph cannot exceed the clique-width of the parent graph, we conclude that  $\text{cwd}(G[V_j]) \leq 4k$ , which shows condition (1) of Lemma 1.

To show condition (2) of Lemma 1, let us call a vertex  $v_{m,j}$  of  $V_i$  boundary if either  $v_{m-1,j}$  belongs to  $V_{i-1}$  or  $v_{m+1,j}$  belongs to  $V_{i+1}$  (or both). It is not difficult to see that a vertex of  $V_i$  is boundary if it belongs either to the second row of an entirely black column of  $W_i$  or to the first row of an entirely black column of  $W_{i+1}$ . Since the graph  $G$  is  $k * F_k$ -free, the number of columns of  $W_i$  which are entirely black is at most  $k - 1$ . Therefore, the boundary vertices of  $V_i$  introduce at most  $2(k - 1)$  equivalence classes in  $V_i$ .

Now consider two non-boundary vertices  $v_{m,j}$  and  $v_{m,p}$  in  $V_i$  from the same row. It is not difficult to see that  $v_{m,j}$  and  $v_{m,p}$  have the same neighborhood outside  $V_i$ . Therefore, the non-boundary vertices of the same row of  $V_i$  are  $V_i$ -similar, and hence the non-boundary vertices give rise to at most  $2k$  equivalence classes in  $V_i$ . Thus,  $\mu(V_i) \leq 4k - 2$  for all  $i$ .

An identical argument shows that  $\mu(V_1 \cup \dots \cup V_i) \leq 3k - 1 \leq 4k - 2$  for all  $i$ . Therefore, by Lemma 1, we conclude that  $\text{cwd}(G) \leq c(k) := 16k^2 - 8k$ , which completes the proof. □

**Theorem 2.**  $\mathcal{F}$  is a minimal hereditary class of graphs of unbounded clique-width.

*Proof.* Let  $X$  be a proper hereditary subclass of  $\mathcal{F}$  and  $H \in \mathcal{F} - X$ . Since  $H$  is an induced subgraph of  $k * F_k$  for some  $k$ , each graph in  $X$  is  $k * F_k$ -free. Therefore, by Lemma 2, the clique-width of graphs in  $X$  is bounded by a constant. □

### 5 The Class $\mathcal{M}$

Recall that  $\mathcal{M}$  is the hereditary closure of the set  $\{M_n^{(2)} : n \geq 1\}$ . By analogy with Theorem 1 one can prove the following result.

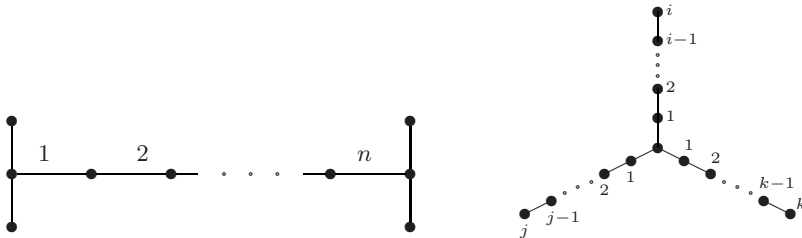
**Theorem 3.** *The clique-width of  $M_n^{(2)}$  is at least  $n/4$ .*

Theorem 3 shows that the clique-width of graphs in the class  $\mathcal{M}$  is unbounded. However, unlike the class  $\mathcal{F}$  studied in the previous section,  $\mathcal{M}$  is not a minimal hereditary class of unbounded clique-width. To show this, let us provide an alternative definition of the graph  $M_n^{(2)}$ . Let  $K_{n,n}$  be the complete bipartite graph with vertices  $a_1, \dots, a_n$  in one part and vertices  $b_1, \dots, b_n$  in the other part. Denote by  $M_{n,n}$  the graph obtained from the  $K_{n,n}$  by subdividing each edge  $a_i b_j$  by a new vertex  $c_{ij}$  (i.e., by introducing vertex  $c_{ij}$  on the edge  $a_i b_j$ ). It is not difficult to see that  $M_{n,n}$  coincides with  $M_n^{(2)}$ . Therefore, every graph in  $\mathcal{M}$  is obtained from a bipartite graph by subdividing each of its edges exactly once (or an induced subgraph of such a graph). We will call the vertices of type  $a_i$  or  $b_i$  in  $M_{n,n}$  *black* and the vertices of type  $c_{i,j}$  *white*.

In order to see that  $\mathcal{M}$  is not a minimal hereditary class of unbounded clique-width, let us introduce the following notations: denote by  $C_n$  the chordless cycle of length  $n$  and by  $H_n$  the graph in Figure 2(left). Also, denote by  $\mathcal{S}_k$  be the class of  $(C_3, \dots, C_k, H_1, \dots, H_k)$ -free bipartite graphs of vertex degree at most 3 and by  $\mathcal{M}_k$  the intersection  $\mathcal{S}_k \cap \mathcal{M}$ .

**Lemma 3.** *For any natural  $k$ , the clique-width of graphs in  $\mathcal{M}_k$  is unbounded.*

*Proof.* It is known that both the clique-width and tree-width are unbounded in the class  $\mathcal{S}_k$  for any value of  $k$  [10]. Since subdivision of an edge does not change the tree-width of a graph (see e.g. [10]), by subdividing each edge of graphs in  $\mathcal{S}_k$  exactly once we obtain a class of graphs  $X$  of unbounded tree-width. Moreover, it is known that for graphs of bounded vertex degree, the tree-width is bounded if and only if the clique-width is bounded [7]. Therefore, the clique-width of graphs in  $X$  also is unbounded, and obviously  $X \subseteq \mathcal{S}_k \cap \mathcal{M}$ , which proves the lemma. □



**Fig. 2.** Graphs  $H_n$  (left) and  $S_{i,j,k}$  (right)



Lemma 3 shows that  $\mathcal{M}$  is not a minimal hereditary class of unbounded clique-width. Indeed, for any  $k$ , the class  $\mathcal{M}_k$  is a subclass of  $\mathcal{M}$  simply because in  $\mathcal{M}_k$  the vertex degree is bounded by 3, while in  $\mathcal{M}$  it is not. Moreover, it is not difficult to see that  $M_{2,2} = M_2^{(2)}$  is a  $C_8$ , i.e.,  $\mathcal{M}_8$  is a subclass of  $M_{2,2}$ -free graphs in  $\mathcal{M}$ .

Let us denote the limit class of the sequence  $\mathcal{S}_1 \supset \mathcal{S}_2 \supset \mathcal{S}_3 \dots$  by  $\mathcal{S}$ , i.e.,  $\mathcal{S} = \bigcap_{k \geq 1} \mathcal{S}_k$ . It is not difficult to see that  $\mathcal{S}$  is the class of graphs every connected

component of which is of the form  $S_{i,j,k}$  represented in Figure 2(right). Obviously,  $\mathcal{S}$  is a subclass of  $\mathcal{M}_k$  for each  $k$ . Therefore,  $\mathcal{S}$  is a limit class of the sequence  $\mathcal{M}_1 \supset \mathcal{M}_2 \supset \mathcal{M}_3 \dots$  as well. In the rest of the section, we show that  $\mathcal{S}$  is a minimal limit subclass of  $\mathcal{M}$ , i.e., for any graph  $H \in \mathcal{S}$ , the clique width of graphs in  $\mathcal{M} \cap \text{Free}(H)$  is bounded by a constant. This will be done through a sequence of auxiliary lemmas. The first lemma in this sequence was proved in 9.

**Lemma 4.** 9 For a class of graphs  $X$  and an integer  $\rho$ , let  $[X]_\rho$  be the class of graphs  $G$  such that  $G - U$  belongs to  $X$  for some subset  $U \subseteq V(G)$  of cardinality at most  $\rho$ , and let  $[X]_B$  be the class of graphs every block of which belongs to  $X$ . If the clique of graphs in  $X$  is bounded by  $p$ , then the clique of graphs in  $[X]_\rho$  is bounded by  $2^\rho(p + 1)$ , and the clique-width in  $[X]_B$  is bounded by  $p + 2$ .

In the proofs of the next lemmas we will frequently use the following obvious observation.

**Observation 1.** Any cycle in any graph  $G \in \mathcal{M}$  is chordless.

**Lemma 5.** For each  $k \geq 3$ , the clique width of graphs in  $L_k := \mathcal{M} \cap \text{Free}(C_k, C_{k+1}, \dots)$  is bounded by a function of  $k$ .

*Proof.* For  $k = 3$ , the proposition follows from the fact that every graph in  $L_3$  is a forest. For  $k > 3$ , we use the induction on  $k$ .

Let  $G$  be a graph in  $L_{k+1}$ . By Lemma 4 we can assume without loss of generality that  $G$  is 2-connected. If  $G$  contains no cycles of length  $k$ , then  $G \in L_k$  in which case the lemma follows by induction. Now let  $C$  be a cycle of length  $k$  in  $G$ . We will show that any other cycle  $C'$  of length  $k$  in  $G$  (if any) has a common vertex with  $C$ . Assume the contrary:  $C$  and  $C'$  are vertex disjoint. Consider two edges  $e \in C$  and  $e' \in C'$ . Since  $G$  is 2-connected, there is a cycle containing both  $e$  and  $e'$ . In this cycle, one can distinguish two disjoint paths  $P$  and  $Q$ , each of which contains the endpoints in  $C$  and  $C'$ , and the remaining vertices outside the cycles. The endpoints of the paths  $P$  and  $Q$  partition each of the cycles  $C$  and  $C'$  into two parts. The larger parts in both cycles together with paths  $P$  and  $Q$  form a cycle of length at least  $k + 2$ , contradicting the assumption that  $G \in L_{k+1}$ . This contradiction shows that any two cycles of length  $k$  in  $G$  have a vertex in common. Therefore, removing the vertices of any cycle of length  $k$  from  $H$  results in a graph in  $L_k$ , as required. □

**Lemma 6.** For each  $k \geq 1$ , the clique width of graphs in  $\mathcal{M} \cap \text{Free}(S_{k,k,k})$  is bounded by a function of  $k$ .

*Proof.* Let  $G \in \mathcal{M} \cap \text{Free}(S_{k,k,k})$ . Consider a chordless path  $P$  of length  $2k - 2$  and a chordless cycle  $C$  of length at least  $2k + 2$  in  $G$ . If  $G$  does not contain such  $P$  or  $C$ , the the clique-width of  $G$  is bounded according to Lemma 5. Assume  $P$  and  $C$  are vertex disjoint. Since  $G$  is connected, there must exist a chordless path  $P'$  connecting  $C$  to  $P$ . Since only black vertices of  $C$  can have neighbors outside  $C$ , the vertex of  $P'$  that has a neighbor on  $C$  is white, and therefore this vertex has exactly one neighbor on  $C$ . Similarly, it is not difficult to see that the vertex of  $P'$  that has a neighbor on  $P$  is adjacent to exactly one vertex of  $P$ . But now the reader can easily find an induced  $S_{k,k,k}$ . This contradiction shows that  $P$  and  $C$  contain a vertex in common. Therefore, the graph obtained from  $G$  by deletion of the vertices of  $P$  belongs to  $L_{2k+2}$ , and the proposition follows from Lemmas 4 and 5.  $\square$

**Theorem 4.** *For any graph  $H \in \mathcal{S}$ , the clique width of graphs in  $\mathcal{M} \cap \text{Free}(H)$  is bounded by a constant.*

*Proof.* Without loss of generality we will assume that every connected component of  $H$  is of the form  $S_{k,k,k}$  for some even  $k \geq 2$  (obviously every graph in  $\mathcal{S}$  is an induced subgraph of a graph of this form). Let  $p$  be the number of connected components of  $H$ , i.e.,  $H = pS_{k,k,k}$ . We will show that the clique-width of any graph  $G$  in  $\mathcal{M} \cap \text{Free}(H)$  is bounded by a function of  $k$  and  $p$ . The proof will be given by induction on the minimum number  $m \leq p$  such that  $G$  is  $mS_{k,k,k}$ -free. If  $m = 1$ , then the clique-width of  $G$  is bounded according to Lemma 6. If  $G$  contains an induced copy of  $S_{k,k,k}$ , then by deleting this copy we obtain a graph  $G'$  which is  $(m - 1)S_{k,k,k}$ -free. Indeed, if  $G'$  contains an induced copy of  $(m - 1)S_{k,k,k}$ , then there are no edges between this copy and the deleted copy of  $S_{k,k,k}$  in  $G$ , because  $k$  is even, which means white vertices in both copies have no neighbors outside the copies. By the induction hypothesis, the clique-width of  $G'$  is bounded by a function of  $k$  and  $p$ . Therefore, by Lemma 4, the clique-width of  $G$  is bounded as well, since the number of deleted vertices is  $3k + 1$ .  $\square$

## References

1. Arnborg, S., Proskurowski, A.: Linear time algorithms for NP-hard problems restricted to partial  $k$ -trees. *Discrete Appl. Math.* 23, 11–24 (1989)
2. Boliac, R., Lozin, V.V.: On the clique-width of graphs in hereditary classes. In: Bose, P., Morin, P. (eds.) *ISAAC 2002*. LNCS, vol. 2518, pp. 44–54. Springer, Heidelberg (2002)
3. Brandstädt, A., Lozin, V.V.: On the linear structure and clique-width of bipartite permutation graphs. *Ars Combinatoria* 67, 273–289 (2003)
4. Brandstädt, A., Engelfriet, J., Le, H.-O., Lozin, V.V.: Clique-Width for Four-Vertex Forbidden Subgraphs. *Theory of Computing Systems* 34, 561–590 (2006)
5. Courcelle, B., Engelfriet, J., Rozenberg, G.: Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences* 46, 218–270 (1993)
6. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.* 33, 125–150 (2000)
7. Courcelle, B., Olariu, S.: Upper bounds to the clique-width of a graph. *Discrete Applied Math.* 101, 77–114 (2000)

8. Lozin, V.V.: From tree-width to clique-width: Excluding a unit interval graph. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 871–882. Springer, Heidelberg (2008)
9. Lozin, V.V., Rautenbach, D.: On the band-, tree- and clique-width of graphs with bounded vertex degree. *SIAM J. Discrete Mathematics* 18, 195–206 (2004)
10. Lozin, V.V., Rautenbach, D.: The tree- and clique-width of bipartite graphs in special classes. *Australasian J. Combinatorics* 34, 57–67 (2006)
11. Lozin, V.V., Rudolf, G.: Minimal Universal Bipartite Graphs. *Ars Combinatoria* 84, 345–356 (2007)
12. Lozin, V.V., Volz, J.: The clique-width of bipartite graphs in monogenic classes. *International Journal of Foundations of Computer Science* 19, 477–494 (2008)
13. Robertson, N., Seymour, P.D.: Graph minors. V. Excluding a planar graph. *J. Combinatorial Theory Ser. B* 41, 92–114 (1986)

# Kernel in Oriented Circulant Graphs

Paul Manuel<sup>1</sup>, Indra Rajasingh<sup>2</sup>, Bharati Rajan<sup>2</sup>, and Joice Punitha<sup>3</sup>

<sup>1</sup> Department of Information Science, Kuwait University, Kuwait-13060

<sup>2</sup> Department of Mathematics, Loyola college, Chennai 600 034, India

<sup>3</sup> Department of Mathematics, R.M.D Engineering College, Kavaraipeitai 601206, India

joicepunitha@rediffmail.com

**Abstract.** A kernel in a directed graph  $D(V, E)$  is a set  $S$  of vertices of  $D$  such that no two vertices in  $S$  are adjacent and for every vertex  $u$  in  $V \setminus S$  there is a vertex  $v$  in  $S$ , such that  $(u, v)$  is an arc of  $D$ . The problem of existence of a kernel is  $NP$ -complete for a general digraph. In this paper we introduce the strong kernel problem of an undirected graph  $G$  and solve it in polynomial time for circulant graphs.

**Keywords:** oriented graph, kernel, strong kernel number,  $NP$ -complete, strongly connected.

## 1 Introduction

The concept of kernel is widespread and appears in diverse fields such as logic, computational complexity, artificial intelligence, graph theory, game theory, combinatorics and coding theory [3], [4]. Efficient routing among a set of mobile hosts is one of the most important functions in ad hoc wireless networks. Dominating-set-based routing to networks with unidirectional links is proposed in [1], [9]. A few years ago a new interest for these studies arose due to their applications in finite model theory. Indeed variants of kernel are the best properties to provide counter examples of 0 – 1 laws in fragments of monadic second order logic [8].

A *kernel* [6] in a directed graph  $D(V, E)$  is a set  $S$  of vertices of  $D$  such that no two vertices in  $S$  are adjacent and for every vertex  $u$  in  $V \setminus S$  there is a vertex  $v$  in  $S$ , such that  $(u, v)$  is an arc of  $D$ . The concept of kernels in digraphs was introduced in different ways [10], [14]. Von Neumann and Morgenstern [14] were the first to introduce kernels when describing winning positions in 2 person games. They proved that any directed acyclic graph has a unique kernel. Not every digraph has a kernel and if a digraph has a kernel, this kernel is not necessarily unique. All odd length directed cycles and most tournaments have no kernels [3], [4]. If  $D$  is finite, the decision problem of the existence of a kernel is  $NP$ -complete for a general digraph [5], [13], and for a planar digraph with indegrees  $\leq 2$ , outdegrees  $\leq 2$  and degrees  $\leq 3$  [7]. It is further known that a finite digraph all of whose cycles have even length has a kernel [11], and that the question of the number of kernels is  $NP$ -complete even for this restricted class of digraphs [12].

In this paper we view the kernel problem from a different perspective. In the literature, only the existence of kernel of a digraph  $G$  and its applications are extensively studied. Our aim in this paper is to investigate all strong orientations of  $G$  and to determine the strong kernel number of  $G$ . This number is different from the independent domination number  $\gamma_i$  for undirected graphs where  $\gamma_i$  is the cardinality of a minimum independent dominating set [2].

## 2 Kernel in Oriented Graphs

An orientation of an undirected graph  $G$  is an assignment of exactly one direction to each of the edges of  $G$ . There are  $2^{|E|}$  orientations for  $G$ . Let  $O_x(G)$  denote the set of all orientations of  $G$ . For an orientation  $O \in O_x$ , let  $G(O)$  denote the directed graph with orientation  $O$  and whose underlying graph is  $G$ .

An orientation  $O$  of an undirected graph  $G$  is said to be an *acyclic orientation* if it contains no directed cycles. Let  $O_a(G)$  denote the set of all acyclic orientations of  $G$ .

An orientation  $O$  of an undirected graph  $G$  is said to be *strong* if for any two vertices  $x, y$  of  $G(O)$ , there are both  $(x, y)$ -path and  $(y, x)$ -path in  $G(O)$ . Let  $O_s(G)$  denote the set of all strong orientations of  $G$ .

An orientation  $O$  of an undirected graph  $G$  is said to be *weak* if for any two vertices  $x, y$  of  $G(O)$ , there is either a  $(x, y)$ -path or a  $(y, x)$ -path in  $G(O)$ . Let  $O_w(G)$  denote the set of all weak orientations of  $G$ .

So far the only known problem on this topic is the kernel problem which investigates the existence of some kernel in a digraph. We introduce kernel number ( $\kappa_x$ ), acyclic kernel number ( $\kappa_a$ ), strong kernel number ( $\kappa_s$ ), and weak kernel number ( $\kappa_w$ ) for various orientations of a given graph.

**Notation 1.** Let  $G(O)$  denote the directed graph with orientation  $O$  and whose underlying graph is  $G$ . The kernel number of  $G(O)$  to be defined below is denoted by  $\kappa(G(O))$ . For convenience we write  $\kappa(G(O))$  as  $\kappa(O)$ .

**Definition 1.** The kernel number  $\kappa_x$  of  $G$  is defined as

$$\kappa_x(G) = \min \{ \kappa(O) : O \in O_x(G) \}$$

**Definition 2.** The acyclic kernel number  $\kappa_a$  of  $G$  is defined as

$$\kappa_a(G) = \min \{ \kappa(O) : O \in O_a(G) \}$$

**Definition 3.** The strong kernel number  $\kappa_s$  of  $G$  is defined as

$$\kappa_s(G) = \min \{ \kappa(O) : O \in O_s(G) \}$$

**Definition 4.** The weak kernel number  $\kappa_w$  of  $G$  is defined as

$$\kappa_w(G) = \min \{ \kappa(O) : O \in O_w(G) \}$$

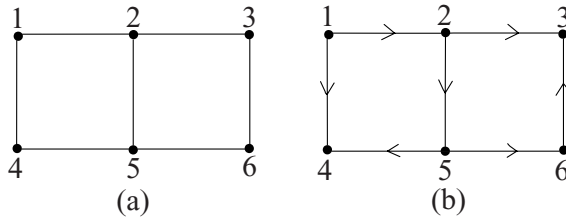
The *acyclic (strong or weak) kernel problem* of an undirected graph  $G$  is to find a kernel  $K$  of  $G(O)$  for some acyclic (strong or weak) orientation  $O$  of  $G$  such that  $|K| = \kappa_x(\kappa_s \text{ or } \kappa_w)$ .

In this section we exhibit the relationship among the parameters  $\gamma_i, \kappa_x, \kappa_s$  and  $\kappa_w$ .

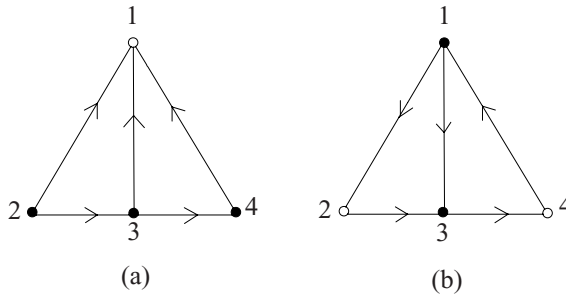
**Theorem 1.** Let  $G$  be a graph. Then  $\gamma_i = \kappa_x$ , where  $\gamma_i$  is the independent domination number.

**Proof.** It is always true that  $\gamma_i \leq k_x$ , since a kernel of a directed graph is an independent dominating set of the corresponding underlying undirected graph. We identify a minimum independent dominating set and orient all the edges incident at each of these vertices as incoming edges. This implies  $k_x \leq \gamma_i$ . Hence  $\gamma_i = \kappa_x$ .  $\square$

**Illustration 1.** For the graph  $G$  in Figure 1(a),  $\Gamma = \{3, 4\}$  is an independent dominating set. Thus  $\gamma_i = 2$ . Figure 1(b) shows an arbitrary orientation of  $G$  with the edges incident at 3, 4 as incoming edges. Hence  $\kappa_x = 2$ .



**Fig. 1.** (a): Independent domination number = 2; (b): Kernel number = 2



**Fig. 2.** (a): Weak kernel number = 1; (b): Strong kernel number = 2

**Illustration 2.** In Figure 2(a),  $K = \{1\}$  is a weak kernel. Thus  $\kappa_w = 1$ . On the other hand, in Figure 2(b)  $K = \{2, 4\}$  is a strong kernel. Thus  $\kappa_s = 2$ .

**Lemma 1.** For any graph  $G$ ,  $O_a$  and  $O_s$  are disjoint.

*Proof.* Let  $O$  be some orientation in  $O_s$  and  $G(O)$ , the corresponding oriented graph. Then there exist  $(u, v)$ -path and  $(v, u)$ -path, for all  $u, v \in V(G)$ . This shows that there exists at least one cycle in graph  $G(O)$ . Hence  $O \notin O_a$ . Thus  $O_a$  and  $O_s$  are disjoint.  $\square$

The following result is trivial as every strong orientation of a graph  $G$  is also a weak orientation.

**Lemma 2.** For any graph  $G$ ,  $O_s$  is a subset of  $O_w$ .

**Lemma 3.** Given any hamiltonian graph  $G$ ,  $O_a \cap O_w \neq \phi$ .

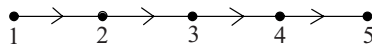
*Proof.* It is enough to find an orientation which is both acyclic and weakly oriented. Let  $C = v_1v_2\dots v_nv_1$  be a hamiltonian cycle in  $G$ . Orient the path  $v_1v_2\dots v_{n-1}v_n$  in the clockwise direction and the edge  $v_nv_1$  in the anticlockwise direction. Orient every other edge  $v_iv_j$  of  $G$  from  $v_i$  to  $v_j$  whenever  $i < j$ . The resulting orientation  $O$  on  $G$  is clearly weak, as there is always either a  $(v_i, v_j)$ -path or  $(v_j, v_i)$ -path along the orientation of  $C$ . Next we claim that  $G(O)$  is acyclic. Suppose there is a cycle  $v_{i_1}v_{i_2}\dots v_{i_k}v_{i_1}$ . By definition  $i_1 < i_2 < \dots < i_k$  and hence  $i_k < i_1$  is not possible. Thus  $v_{i_1}v_{i_2}\dots v_{i_k}v_{i_1}$  is not a directed cycle. This proves that  $O$  is an acyclic orientation.  $\square$

**Remark 1.** If  $O_a \cap O_w \neq \phi$ , then it is not necessary that  $G$  is hamiltonian. See Figure 3.

**Remark 2.** For an arbitrary graph  $G$ ,  $O_a \cap O_w$  may be empty. The graph  $G$  in Figure 4(a), has an acyclic orientation  $O$  and  $G(O)$  is not weakly connected. Thus  $O_a \cap O_w = \phi$ , whereas for the graph in Figure 4(b),  $O_a \cap O_w \neq \phi$ .

**Lemma 4.** For any noncomplete graph  $G$  on 3 or more vertices,  $O_w$  is a proper subset of  $O_x$ .

*Proof.* Since  $G$  is not complete, there exist two independent vertices  $u$  and  $v$ . Consider an orientation  $O$  such that all the edges incident at  $u$  and  $v$  are incoming edges. Then neither there is a  $(u, v)$ -path nor there is a  $(v, u)$ -path. This



**Fig. 3.**  $O \in O_a \cap O_w$  but  $G(O)$  is not hamiltonian

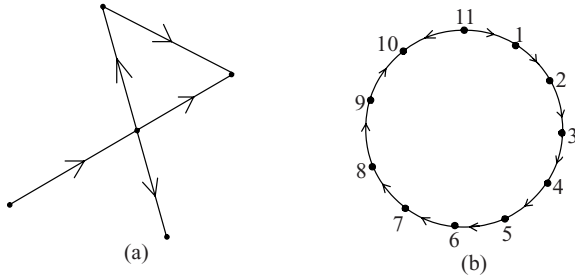


Fig. 4. (a): an acyclic orientation  $G$ ; (b):  $O_a \cap O_w \neq \phi$

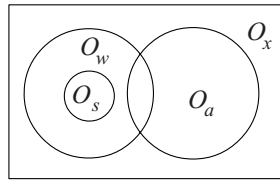


Fig. 5. Relation between different orientations for hamiltonian Graphs

implies that  $O \notin O_w$ . Since  $O_w$  is a subset of  $O_x$  and there exist an orientation  $O \in O_x \setminus O_w$ ,  $O_w$  is a proper subset of  $O_x$ .  $\square$

Figure 5 depicts the set theoretical relationship among  $O_x, O_a, O_s, O_w$ , for hamiltonian graphs.

Next we exhibit the relation among the parameters such as kernel number, acyclic kernel number, strong kernel number and weak kernel number.

The proof follows from Lemma 1 and Lemma 2.

**Theorem 2.** *Let  $G$  be an undirected graph. Then (i)  $\kappa_x \leq \kappa_w \leq \kappa_s$  (ii)  $\kappa_x \leq \kappa_a$ .*

The salient feature of this paper is the following theorem where we obtain a lower bound for the strong kernel number of regular graphs.

**Theorem 3.** *Let  $G$  be an  $r$ -regular graph on  $n$  vertices. Then  $\kappa_s \geq \lceil n/r \rceil$ .*

**Proof.** Let  $O \in O_s$  and  $K$  be some kernel of  $G(O)$ . For any vertex  $v \in K$  there are at most  $r - 1$  incoming edges. Thus  $|K| \geq \lceil n/r \rceil$ .  $\square$

### 3 Kernel in Oriented Circulant Graphs

In this section we obtain the strong kernel number for oriented circulant graphs, proving that the strong kernel problem is polynomially solvable.

We begin with the definition of circulant digraph [15].



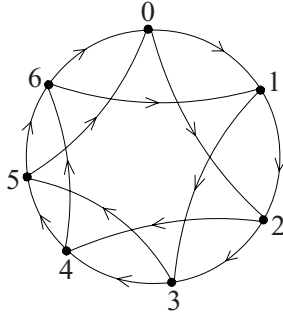


Fig. 6. Circulant digraph(DC<sub>7</sub>(1,2))

**Definition 5.** A circulant digraph  $D(C_n(S))$ , where  $S \subseteq \{1, 2, \dots, n - 1\}$ ,  $n \geq 2$ , is defined as a digraph with vertex set  $V = \{0, 1, 2, \dots, n - 1\}$  and the edge set  $E = \{(i, j)\}$ : there is  $s \in S$  such that  $j - i \equiv s \pmod{n}$ . See Figure 6.

If  $S = \{1, 2, \dots, j\}$  we write  $D(C_n(1, 2, \dots, j))$  instead of  $D(C_n(S))$ . We observe that kernel does not exist for  $D(C_7(1, 2))$  in Figure 6. Thus it is interesting to note that the directed circulant graphs do not possess kernels in certain cases. This motivates us to consider oriented circulant graphs.

**Definition 6.** [15] A circulant undirected graph  $C_n(S)$ , where  $S \subseteq \{1, 2, \dots, \lfloor n/2 \rfloor\}$ ,  $n \geq 3$ , is defined as an undirected graph with vertex set  $V = \{0, 1, 2, \dots, n - 1\}$  and the edge set  $E = \{(i, j) : \text{there is } s \in S \text{ such that } |j - i| \equiv s \pmod{n}\}$ .

Clearly  $C_n(S)$  is  $2|S|$ -regular.

**Lemma 5.** Let  $G$  be  $C_n(1, 2, \dots, s)$ ,  $1 \leq s \leq \lfloor n/2 \rfloor$ . Then  $G$  has at least  $2^{m-n}$  strong orientations where  $m$  denotes the number of edges in  $G$ .

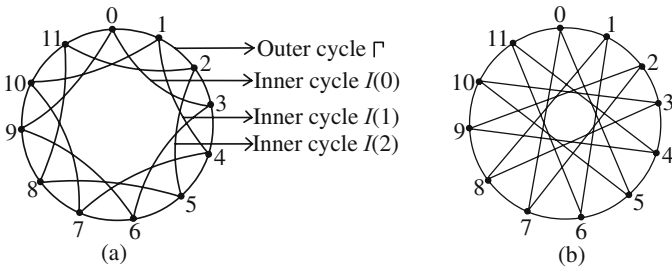


Fig. 7. (a) Cycles marked in  $C_{12}(1, 3)$ ; (b)  $C_{12}(1, 5)$  has one hamiltonian cycle  $\Gamma$  and exactly one inner hamiltonian cycle  $I(0)$

**Proof.** The circulant graph  $G$  contains the cycle  $C_n(1)$  on  $n$  vertices as a subgraph. Let it be oriented in clockwise direction. Clearly, this orientation induces a strong orientation of  $G$ . Now the remaining  $m - n$  edges can be oriented in  $2^{m-n}$  ways. Hence  $G$  has at least  $2^{m-n}$  strong orientations.  $\square$

We proceed to prove that the strong kernel problem is polynomially solvable for  $C_n(1, 2, \dots, s), 1 \leq s \leq \lfloor n/2 \rfloor$ .

**Lemma 6.** *Let  $G$  be  $C_n(1, s), 2 \leq s \leq \lfloor n/2 \rfloor$ . The edge set  $E$  of  $C_n(1, s)$  is partitioned into the following cycles:*

1. *The outer hamiltonian cycle  $\Gamma = (0, 1, 2, \dots, 0)$ .*
2.  *$l$  number of edge-disjoint inner cycles  $I(k) = (k, k + s, k + 2s, \dots, k + (\frac{n}{l} - 1)s, k), 0 \leq k \leq l - 1$ , each of length  $\frac{n}{l}$  where  $l = \text{g.c.d}(n, s)$  and vertex labels taken modulo  $n$ .*

**Proof. Case 1: ( s divides n )**

Since  $s$  divides  $n$ , we have  $l = s$ . In addition to the outer hamiltonian cycle  $\Gamma$ ,  $C_n(1, s)$  has the following inner cycles:  $I(0) = (0, s, \dots, 0); I(1) = (1, s + 1, \dots, 1); I(2) = (2, s + 2, \dots, 2); \dots I(s - 1) = (s - 1, 2s - 1, \dots, s - 1)$ . See Figure 7(a). For  $i \neq j, V(I(i)) \cap V(I(j)) = \emptyset, 1 \leq i, j \leq l$  and  $\cup V(I(k)) = \{0, 1, 2, 3, \dots, n - 1\}$ . Hence the cycles  $I(0), I(1), I(2), \dots, I(s - 1)$  are edge-disjoint and these cycles together with  $\Gamma$  partition the set  $E$ .

**Case 2: ( s does not divide n )**

If  $s$  does not divide  $n$ , then  $l = 1$ . In this case, in addition to the outer hamiltonian cycle  $\Gamma$ , there is one more hamiltonian cycle which is  $I(0) = 0, s, 2s, \dots, 0$ . See Figure 7(b).  $\square$

**Notation 2.** *Let  $C_n^{sub}(1, s)$  be a subgraph of  $C_n(1, s)$  where the edge set of  $C_n^{sub}(1, s)$  consists of : (i) the edges of the outer cycle  $\Gamma$  (ii) the edges of the inner cycle  $I(0)$ .*

**Notation 3.** *A segment  $\Pi^+(x)$  of  $C_n^{sub}(1, s)$  is a path of consecutive vertices  $x, x + 1, \dots, x + l$  of length  $l$ . Similarly a segment  $\Pi^-(x)$  is a path of consecutive vertices  $x, x - 1, \dots, x - l$  of length  $l$ , where  $l = \text{g.c.d}(n, s)$ . See Figure 8.*

The following Lemma gives a strong orientation of  $C_n(1, 2, \dots, s), 2 \leq s \leq \lfloor n/2 \rfloor$ .

**Lemma 7.** *Let  $C_n^{sub}(1, s)$  have an orientation  $O$  satisfying the following conditions.*

1. *Inner cycle  $I(0)$  is oriented clockwise*
2.  *$\Pi^+(il)$  is oriented either in clockwise or in anticlockwise direction where  $i = 0, 1, \dots, \lceil n/l \rceil - 1$ . Then  $O$  is a strong orientation.*

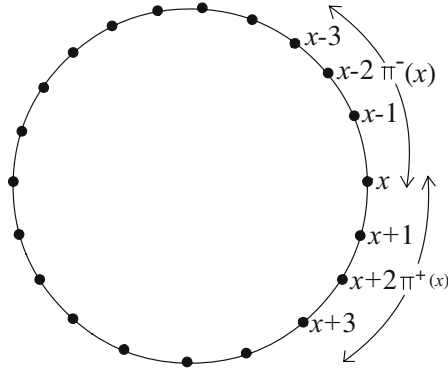


Fig. 8.  $l = 3, \Pi^+(x) = x, x + 1, x + 2, x + 3; \Pi^-(x) = x, x - 1, x - 2, x - 3$

**Proof.** Let  $u, v$  be any two vertices of  $C_n^{sub}(1, s)$  and let  $u$  be in  $\Pi^+(il)$  such that  $u = il + t$  and  $v$  be in  $\Pi^+(jl)$  such that  $v = jl + h$ . Both  $\Pi^+(il)$  and  $\Pi^+(jl)$  are oriented either clockwise or anticlockwise. Thus there are four possible cases.

**Case 1: ( $\Pi^+(il)$  and  $\Pi^+(jl)$  are Oriented Clockwise)**

Here is a directed path from  $u$  to  $v$  :  $(il + t, il + t + 1, \dots, (i + 1)l, (i + 1)l + s, \dots, jl, jl + 1, \dots, jl + h) \bmod n$ . In the same way we trace out a directed path from  $v$  to  $u$  :  $(jl + h, jl + h + 1, \dots, (j + 1)l, (j + 1)l + s, \dots, il, il + 1, il + 2, \dots, il + t) \bmod n$ . See Figure 9.

**Case 2: ( $\Pi^+(il)$  is Oriented Clockwise and  $\Pi^+(jl)$  is Oriented Anticlockwise)**

There is a directed path from  $u$  to  $v$  :  $(il + t, il + t + 1, \dots, (i + 1)l, (i + 1)l + s, \dots, (j + 1)l, (j + 1)l - 1, \dots, jl + h) \bmod n$  and there exist a directed path from  $v$  to  $u$  :  $(jl + h, jl + h - 1, \dots, jl, jl + s, \dots, il, il + 1, \dots, il + t) \bmod n$ .

**Case 3: ( $\Pi^+(il)$  is oriented anticlockwise and  $\Pi^+(jl)$  is oriented clockwise)**

Here is a directed path from  $u$  to  $v$  :  $(il + t, il + t - 1, \dots, il, il + s, \dots, jl, jl + 1, \dots, jl + h) \bmod n$ . In the same way we exhibit a directed path from  $v$  to  $u$  :  $(jl + h, jl + h + 1, \dots, (j + 1)l, (j + 1)l + s, \dots, (i + 1)l, (i + 1)l - 1, \dots, il + t) \bmod n$ .

**Case 4: ( $\Pi^+(il)$  and  $\Pi^+(jl)$  are oriented anticlockwise)**

Here is a directed path from  $u$  to  $v$  :  $(il + t, il + t - 1, \dots, il, il + s, \dots, (j + 1)l, (j + 1)l - 1, \dots, jl + h) \bmod n$ . In the same way we trace out a directed path from  $v$  to  $u$  :  $(jl + h, jl + h - 1, \dots, jl, jl + s, \dots, (i + 1)l, (i + 1)l - 1, \dots, il + t) \bmod n$ .  $\square$

**Theorem 4.** Let  $G$  be  $C_n(1, 2, \dots, s), 1 \leq s \leq \lfloor n/2 \rfloor$ . Then  $\kappa_s = \lceil n/2s \rceil$ .

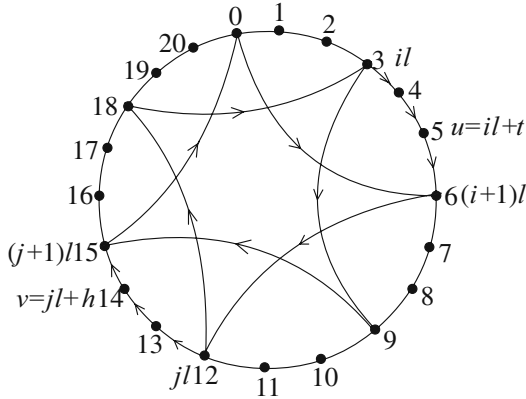


Fig. 9.  $C_{21}(1, 6)$ ;  $\Pi^+(il)$  and  $\Pi^+(jl)$  are oriented clockwise

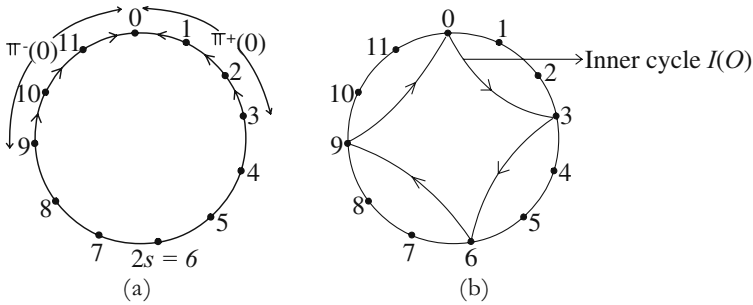


Fig. 10. (a)  $\Pi^-(0)$  is oriented in clockwise direction and  $\Pi^+(0)$  is oriented in anticlockwise direction in  $C_{12}(1, 2, 3)$ ; (b) Inner cycle  $I(0)$  is oriented clockwise in  $C_{12}(1, 2, 3)$

**Proof.** By Theorem 3,  $\kappa_s \geq \lceil n/2s \rceil$ . We claim  $\kappa_s \leq \lceil n/2s \rceil$ . Two cases arise.

**Case 1: ( $s$  divides  $n$ )**

**Subcase 1: ( $n/s$  is even)**

We claim that  $K = \{0, 2s, 4s, \dots, (n/2s - 1)2s\}$  is a kernel of  $C_n(1, 2, \dots, s)$ .

Step 1: For  $0 \leq k \leq n/2s$ ,  $\Pi^+(2ks)$  is oriented anticlockwise and  $\Pi^-(2ks)$  is oriented clockwise. See Figure 10(a).

Step 2: Orient the inner cycle  $I(0)$  as clockwise. See Figure 10(b).

Step 3: The remaining edges at  $0, 2s, 4s, \dots, (n - 2s)$  are oriented as incoming edges into the respective vertices. See Figure 11.

Let  $u \in V \setminus K$  such that  $2k_0s < u < 2(k_0 + 1)s$  for some  $k_0$ . Then either  $u$  is adjacent to  $2k_0s$  or adjacent to  $2(k_0 + 1)s$  in  $C_n(1, 2, \dots, s)$ . By the construction of strong orientation of  $C_n(1, 2, \dots, s)$ , either  $\overleftarrow{(u, 2k_0s)}$  or  $\overleftarrow{(u, 2(k_0 + 1)s)}$  is an arc. Also  $K$  is an independent set of  $C_n(1, 2, \dots, s)$ . Clearly  $|K| = \lceil n/2s \rceil$ .

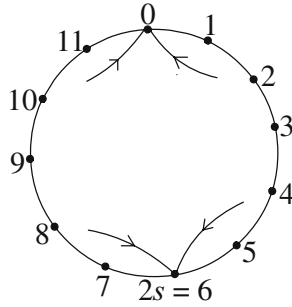


Fig. 11. Edges at  $0, 2s, 4s, \dots, n - 2s$  are oriented as incoming edges in  $C_{12}(1,2,3)$

**Subcase 2: ( $n/s$  is odd)**

We claim that  $K = \{0, 2s, 4s, \dots, (\lfloor n/2s \rfloor - 1)2s, n - s - 1\}$  is a kernel of  $C_n(1, 2, \dots, s)$ .

- Step 1: For  $0 \leq k < \lfloor n/2s \rfloor$ ,  $\Pi^+(2ks)$  is oriented anticlockwise and  $\Pi^-(2ks)$  is oriented clockwise.
- Step 2: All other segments  $\Pi(i)$  are oriented in the clockwise direction.
- Step 3: Orient the inner cycle  $I(0)$  as clockwise.
- Step 4: The remaining edges at  $0, 2s, 4s, \dots, (\lfloor n/2s \rfloor - 1)2s, n - s - 1$  are oriented as incoming edges into the respective vertices.
- Step 5: The remaining unoriented edges of  $C_n(1, 2, \dots, s)$  are oriented arbitrarily.

When  $k = \lfloor n/2s \rfloor$ , the vertex  $2s \lfloor n/2s \rfloor$  is adjacent to 0. Hence we choose the vertex  $n - s - 1$  which is not adjacent to 0 and  $n - s - 1 \in (n - 2s, n - s)$ .

Let  $u \in V \setminus K$  such that  $2k_0s < u < 2(k_0 + 1)s$  for some  $k_0$ . Then either  $u$  is adjacent to  $2k_0s$  or adjacent to  $2(k_0 + 1)s$  in  $C_n(1, 2, \dots, s)$ . By the construction of strong orientation of  $C_n(1, 2, \dots, s)$ , either  $(\overleftarrow{u, 2k_0s})$  or  $(\overleftarrow{u, 2(k_0 + 1)s})$  is an arc. Also  $K$  is an independent set of  $C_n(1, 2, \dots, s)$ . Clearly  $|K| = \lceil n/2s \rceil$ .

**Case 2: ( $s$  does not divide  $n$ )**

**Subcase 1: ( $\lfloor n/2s \rfloor 2s < n - s$ )**

We claim that  $K = \{0, 2s, \dots, 2s \lfloor n/2s \rfloor\}$  is a kernel of  $C_n(1, 2, \dots, s)$ .

- Step 1: For  $0 \leq k \leq \lfloor n/2s \rfloor$ ,  $\Pi^+(2ks)$ , is oriented anticlockwise and  $\Pi^-(2ks)$  is oriented clockwise.
- Step 2: All other segments  $\Pi(i)$  are oriented in the clockwise direction.
- Step 3: Orient the inner cycle  $I(0)$  as clockwise.
- Step 4: The remaining edges at  $0, 2s, \dots, 2s \lfloor n/2s \rfloor$  are oriented as incoming edges into the respective vertices.
- Step 5: The remaining unoriented edges of  $C_n(1, 2, \dots, s)$  are oriented arbitrarily.

Let  $u \in V \setminus K$  such that  $2k_0s < u < 2(k_0 + 1)s$  for some  $k_0$ . Then either  $u$  is adjacent to  $2k_0s$  or adjacent to  $2(k_0 + 1)s$  are adjacent in  $C_n(1, 2, \dots, s)$ . By the construction of strong orientation of  $C_n(1, 2, \dots, s)$ , either  $(\overleftarrow{u, 2k_0s})$  or  $(\overleftarrow{u, 2(k_0 + 1)s})$  is an arc. Also  $K$  is an independent set of  $C_n(1, 2, \dots, s)$ . Clearly  $|K| = \lceil n/2s \rceil$ .

**Subcase 2:**  $(n - s < \lfloor n/2s \rfloor 2s \leq n - 1)$

**Subcase 2(a):**  $(n - s < \lfloor n/2s \rfloor 2s < n - 1)$

We claim that  $K = \{0, 2s, 4s, \dots, (\lfloor n/2s \rfloor - 1)2s, n - s - 1\}$  is a kernel of  $C_n(1, 2, \dots, s)$ .

Step 1: For  $0 \leq k < \lfloor n/2s \rfloor$ ,  $\Pi^+(2ks)$  is oriented anticlockwise and  $\Pi^-(2ks)$  is oriented clockwise.

Step 2: All other segments  $\Pi(i)$  are oriented in the clockwise direction.

Step 3: Orient the inner cycle  $I(0)$  as clockwise.

Step 4: The remaining edges at  $0, 2s, 4s, \dots, (\lfloor n/2s \rfloor - 1)2s, n - s - 1$  are oriented as incoming edges into the respective vertices.

Step 5: The remaining unoriented edges of  $C_n(1, 2, \dots, s)$  are oriented arbitrarily.

The proof is similar to that of subcase 1 of case 2.

**Subcase 2(b):**  $(\lfloor n/2s \rfloor 2s = n - 1)$

We claim that  $l = g.c.d(n, s) = 1$ . If possible let  $l > 1$ . This implies that  $n = k_1l$  and  $s = k_2l$  for some integers  $k_1$  and  $k_2$ . But  $n - 1 = ks$ , where  $k = 2 \lfloor n/2s \rfloor$ . Therefore  $n - ks = 1$ . Hence  $g.c.d(n, s) = 1$ .

Therefore  $\lfloor n/2s \rfloor 2s = n - 1 \implies l(k_1 - 2k_2 \lfloor k_1/2k_2 \rfloor) = 1$ .

Since  $l > 1$ ,  $(k_1 - 2k_2 \lfloor k_1/2k_2 \rfloor) = \frac{1}{l}$  is not an integer, a contradiction.

We next claim that the vertex  $(\lfloor n/2s \rfloor - 1)2s$  is adjacent to the vertex  $n - s - 1$ . Consider

$$\begin{aligned} (n - s - 1) - (\lfloor n/2s \rfloor - 1)2s &= n - s - 1 - \lfloor n/2s \rfloor 2s + 2s \\ &= n + s - 1 - \lfloor n/2s \rfloor 2s = s \end{aligned}$$

Since the distance between the two vertices  $(\lfloor n/2s \rfloor - 1)2s$  and  $n - s - 1$  is  $s$ , they are adjacent to each other. Hence we choose the vertices  $n - s - 1$  and  $(\lfloor n/2s \rfloor - 1)2s - 1$  in  $K$ .

We claim that  $K = \{0, 2s - 1, 4s - 1, \dots, (\lfloor n/2s \rfloor - 1)2s - 1, n - s - 1\}$ .

Step 1: Mark the vertices  $0, 2ks - 1, k = 1, 2, \dots, \lfloor n/2s \rfloor - 1$  and  $n - s - 1$ .

Step 2:  $\Pi^+(0)$ ,  $\Pi^+(2ks - 1)$  and  $\Pi^+(n - s - 1)$  are oriented anticlockwise and  $\Pi^-(0)$ ,  $\Pi^-(2ks - 1)$  and  $\Pi^-(n - s - 1)$  are oriented clockwise.

Step 3: All other segments  $\Pi(i)$  are oriented in the clockwise direction.

Step 4: Orient the inner cycle  $I(0)$  as clockwise.

Step 5: The remaining edges at  $0, 2s - 1, 4s - 1, \dots, (\lfloor n/2s \rfloor - 1)2s - 1, n - s - 1$  are oriented as incoming edges into the respective vertices.

Step 6: The remaining unoriented edges of  $C_n(1, 2, \dots, s)$  are oriented arbitrarily.

The proof is similar to that of subcase 1 of case2. □  
 Thus we have the following Theorem.

**Theorem 5.** *The strong kernel problem for  $C_n(1, 2, \dots, s)$ ,  $1 \leq s \leq \lfloor n/2 \rfloor$  is polynomially solvable.*

## 4 Conclusion

We have introduced variations of kernel number for oriented graphs and have exhibited relations among them. We have estimated the lower bound for the strong kernel number for regular graphs. We have also proved that the strong kernel problem is polynomially solvable for circulant graphs. Further the various parameters introduced in Section 2 have opened new avenues for further research in this field.

## References

1. Alzoubi, K.M., Wan, P.J., Frieder, O.: New Distributed Algorithm for Connected Dominating Set in Wireless Ad Hoc Networks. In: Proc. 35th Hawaii Int. Conf. System Sciences, pp. 1–7 (2002)
2. Armugam, S., Velammal, S.: Maximum size of a connected graph with given domination parameters. *Ars Combin.* 52, 221–227 (1992)
3. Bang-Jensen, J., Gutin, G.: *Digraphs: Theory, Algorithms and Applications*. Springer, London (2000)
4. Berge, C., Duchet, P.: Recent problems and results about kernels in directed graphs. *Discrete Mathematics* 86, 27–31 (1990)
5. Chvátal, V.: On the computational complexity of finding a kernel. Report No. CRM-300, Centre de Recherches Mathématiques, Université de Montreal (1973)
6. Berge, C.: *Graphs*, vol. 6. North Holland Publishing Co., Amsterdam (1985)
7. Fraenkel, A.S.: Planar kernel and Grundy with  $d \leq 3$ ,  $d^+ \leq 2$ ,  $d^- \leq 2$  are NP-complete. *Discrete Applied Mathematics* 3, 257–262 (1981)
8. Bars, J.-M.L.: Counter example of the 0–1 Law for fragments of existential second-order logic: an overview. *The Bulletin of Symbolic Logic* 6(1), 67–82 (2000)
9. Wu, J.: Extended Dominating-Set-Based Routing in Ad Hoc Wireless Networks with Unidirectional Links. *IEEE Transactions on Parallel and Distributed Systems* 13(9), 866–881 (2002)
10. Kswánsnik, M.: The generalisation of Richardson theorem. *Discussiones Math.* IV, 11–14 (1981)
11. Richardson, M.: Solutions of irreflexive relations. *Ann. of Math.* 58, 573–590 (1953)
12. Szwarcfiter, J.L., Chaty, G.: Enumerating the kernels of a directed graph with no odd circuits. *Inform. Process. Lett.* 51, 149–153 (1994)
13. Van Leeuwen, J.: Having a Grundy-numbering is NP-complete. Report No. 207, Computer Science Dept., Pennsylvania State University, University Park, PA (1976)
14. Von Neumann, J., Morgenstern, O.: *Theory of Games and Economic Behaviour*. Princeton University Press, Princeton (1944)
15. Xu, J.: *Topological Structure and Analysis of Interconnection Networks*. Kluwer Academic Publishers, Dordrecht (2001)

# Randomized Postoptimization of Covering Arrays

Peyman Nayeri, Charles J. Colbourn, and Goran Konjevod

Computer Science and Engineering, Arizona State University,  
P.O. Box 878809, Tempe, AZ 85287, U.S.A.  
{peyman.nayeri, charles.colbourn, goran}@asu.edu

**Abstract.** The construction of covering arrays with the fewest rows remains a challenging problem. Most computational and recursive constructions result in extensive repetition of coverage. While some is necessary, some is not. By reducing the repeated coverage, metaheuristic search techniques typically outperform simpler computational methods, but they have been applied in a limited set of cases. Time constraints often prevent them from finding an array of competitive size. We examine a different approach. Having used a simple computation or construction to find a covering array, we employ a postoptimization technique that repeatedly adjusts the array in order to (sometimes) reduce its number of rows. At every stage the array retains full coverage. We demonstrate its value on a collection of previously best known arrays by eliminating, in some cases, 10% of their rows. In the well-studied case of strength two with twenty factors having ten values each, postoptimization produces a covering array with only 162 rows, improving on a wide variety of computational and combinatorial methods. We identify certain important features of covering arrays for which postoptimization is successful.

## 1 Introduction

Covering arrays are combinatorial models of test suites used to detect faulty interactions among components in software, hardware, and networked systems. They are intimately related to orthogonal arrays and related experimental designs; to surjective codes; and to qualitatively independent partitions. As a consequence of these and other connections, the construction of covering arrays has been a topic of substantial interest. See [1,2] for surveys that are now somewhat dated. Despite the extensive effort expended, finding the smallest test suites for given testing scenarios remains an unsolved problem in general. We first introduce a purely combinatorial formulation.

Let  $N$ ,  $k$ ,  $t$ , and  $v$  be positive integers. An  $N \times k$  array, each column of which contains  $v$  distinct symbols, is a *covering array*  $\text{CA}(N; t, k, v)$  of *strength*  $t$  when, for every way to select  $t$  columns, each of the  $v^t$  possible tuples of symbols arises in at least one row. When used for testing, columns of the array form *factors*, and the symbols in the column form *values* or *levels* for the factor. Each row specifies the values to which to set the factors for an experimental run. A *t-tuple*



or *t*-way interaction is a set of *t* of the factors, and an admissible level for each. The array is ‘covering’ in the sense that every *t*-way interaction is represented by at least one run. Now  $CAN(1, k, v) = v$  and  $CAN(t, k, 1) = 1$ . So to avoid trivial cases, we suppose that  $k \geq t \geq 2$  and  $v \geq 2$ . In this paper, we always take the value set of each factor to be  $\{0, \dots, v - 1\}$ .

11120211122100120202122221  
 00011021212221100112101122  
 10212221220201211010200011  
 01222111111121002001020002  
 12110110210000022022221111  
 21020120021102212111201120  
 02001022202101202000222210  
 10102200011011222201102102  
 1\*000212111220221102011000  
 01002002020010001110121211  
 20220202100101101120012102  
 22202101002012110022110020  
 12121010202212001211002001  
 2011112010222011200022220  
 02121220121022020110010112  
 0\*010022120\*\*0210221200202  
 2121100020122012222\*211211  
 \*0\*10\*2\*1\*\*1101\*\*\*121\*\*1\*

At left is shown a  $CA(18;2,26,3)$ . The entries shown as \* can be chosen arbitrarily, and every pair of columns contains each of the nine possible pairs.

In testing applications, *the fundamental problem is to determine*  $CAN(t, k, v)$ . Evidently,  $CAN(t, k, v) \geq v^t$ , and when equality holds the CA is an *orthogonal array*  $OA(v^t; t, k, v)$ ; see [3] for a textbook treatment. Such orthogonal arrays can exist only when  $k \leq \max(v + 2, t + 1)$  [3], and hence they provide no examples beyond ‘small’ values of *k*. For fixed *v* and *t*, probabilistic methods establish that  $CAN(v, k, t) = \Theta(\log k)$  [4]. Nevertheless, only in the case when  $t = v = 2$  is this function of *k* known exactly [5,6]. When  $CAN(t, k, v)$  is not known exactly, most effort has been invested in *producing ‘good’ upper bounds*. This is the problem considered here.

Explicit constructions of covering arrays are needed in concrete testing applications. *Recursive* methods build larger covering arrays from smaller ones. Some recursive methods are product constructions; see, for example, [7] for  $t = 2$ , [8,9] for  $t = 3$ , [9] for  $t = 4$ , and [10,11] for  $t \geq 5$ . Although these all rely on a similar strategy, their use of numerous smaller covering arrays can result in substantial duplication of coverage; the specific variants result from efforts to reduce this duplication, and have been most successful to date when  $t \in \{2, 3\}$ . A second class of recursive methods are column replacement constructions, which use a second array as a pattern for selecting columns from a covering array; see [12] for the most general one at present. Again these suffer from substantial repetition of coverage. Every recursive method also requires that ingredient covering arrays be known.

*Direct* methods construct covering arrays without recourse to smaller ingredient covering arrays. Some methods employ geometric, algebraic, or number-theoretic properties. The orthogonal arrays constructed from the finite fields [3] provide a prototype for these. By exploiting the structure of automorphisms of the OAs, compact representations of covering arrays accelerate computational search [13, 14, 15, 16]. Recently, cyclotomic classes in the finite field have been shown to provide examples of binary covering arrays, and more generally examples are provided by certain Hadamard matrices [17]. Block designs have been used to make a few specific covering arrays [18]. Other easily constructed examples are

provided by taking all vectors of specified weights to form the rows of a covering array [19,20,21]. Each of these constructions provides useful examples of covering arrays, but each is quite restricted in its application. Therefore by far the most popular general methods are computational techniques.

Exhaustive computation has proved ineffective except in a handful of small cases. Therefore heuristic and metaheuristic strategies have been the norm. Techniques such as simulated annealing [22], tabu search [23], and constraint satisfaction [24] are very effective for small existence problems, but the time taken for convergence to a solution has limited their range of application. As a consequence, the most prevalent computational methods have been greedy. AETG [25] popularized greedy methods that generate one row of a covering array at a time, attempting to select a best possible next row; since that time, TCG [26] and density algorithms [27,28] have developed useful variants of this approach. For strength two, IPO [29] instead adds a factor (column) at a time, adding rows as needed to ensure coverage; the generalization to  $t$ -way coverage in [30,31] is the method that has been run on the largest set of parameters to date. When the arrays to be produced are very large, just checking the properties of the array is challenging; therefore, random methods have also been examined [32].

Unfortunately, at the present time, based on the current best known upper bounds for  $\text{CAN}(t, k, v)$  for  $2 \leq t \leq 6$ ,  $2 \leq v \leq 25$ , and  $t \leq k \leq 10000$  at [33], no single construction can be applied generally while yielding the best, or close to the best, known results. This leaves the tester with the problem of how to generate a covering array quickly that is not ‘far’ from optimum. We examine a new approach, that of improving a covering array after it is constructed; we call this process *postoptimization*. To the best of our knowledge, the only previous effort to improve an existing covering array is the elimination of redundant rows in CATS [34].

## 2 Postoptimization

In any covering array  $\text{CA}(N; t, k, v)$ , the number of  $t$ -way interactions to be covered is  $\binom{k}{t}v^t$ , while the number actually covered is  $N\binom{k}{t}$ . Except possibly when  $k \leq \max(v+2, t+1)$ , some duplication of coverage is necessary. All of the recursive and direct techniques attempt to limit this duplication, but cannot hope to eliminate it completely. Our objective is to eliminate some of the duplication, if possible. Every entry of a  $\text{CAN}(t, k, v)$  participates in  $\binom{k-1}{t-1}$   $t$ -way interactions. Some of these interactions may be covered elsewhere, while others may be covered only in this row. In principle, a specific  $t$ -way interaction could be covered as many as  $N - v^t + 1$  times or as little as once. When all of the  $\binom{k-1}{t-1}$   $t$ -way interactions involving a specific entry are covered more than once, the entry can be changed arbitrarily, or indeed omitted in the determination of coverage, and the array remains a covering array. Hence such an entry is a *possible don't care position*. If we replace such an entry by  $\star$  to indicate that  $t$ -way interactions involving this entry are not to be used for coverage, we select it as a *don't care position*. This replacement can cause other possible don't care positions to appear in  $t$ -way interactions that are now covered only once – such positions are

no longer possible don't care positions. On the other hand, replacing a  $\star$  by an element from  $\{0, \dots, v-1\}$  can result in new positions for which all of their  $t$ -way interactions are covered more than once, i.e. new possible don't care positions.

Our strategy is to exploit the presence of don't care positions in covering arrays. By choosing a specific set of such positions to change to  $\star$ , and then replacing these by elements again, we form a new covering array with a possibly different collection of possible don't care positions. By itself this is of no use other than to produce many covering arrays with the same parameters. However, in some cases we can form an entire row containing only don't care positions. When this occurs, the row is not needed and can be deleted. This is the sense in which the covering array is improved, by the deletion of rows.

## 2.1 Finding Don't Care Positions

To find possible don't care positions, it suffices to determine the numbers of times that the  $\binom{k}{t}v^t$   $t$ -way interactions are covered. For each of the  $Nk$  entries, check whether the entry appears in any  $t$ -way interaction that is covered only once. If not, it is a possible don't care position. While conceptually simple, this requires space proportional to  $\binom{k}{t}v^t$ , which is too much in practice. Instead initially mark each of the  $Nk$  entries as a possible don't care. Then for each of the  $\binom{k}{t}$  sets of columns in turn, use a vector of length  $v^t$  to record the number of times each of the  $t$ -way interactions arises in the  $t$  chosen columns. Then for each that arises only once, mark all  $t$  positions in it to be no longer don't care. This requires only  $Nk + v^t$  space, but still requires time proportional to  $tN\binom{k}{t}$ . At the same time, one can verify that the array is in fact a covering array, by ensuring that every  $t$ -way interaction is seen at least once. Unfortunately, if we change any one of the possible don't care positions to  $\star$ , some recomputation is then needed.

To find a set of don't care positions that can all be simultaneously changed to  $\star$ , we use the fact that rows are recorded in a specific order. For every set of  $t$  columns we consider the rows of the CA in order; when a  $t$ -tuple is covered for the first time we mark its  $t$  positions as *necessary*. After every possible set of  $t$  factors is treated, all positions that are not necessary can be changed to  $\star$ . This can be done in the same time and space as the identification of all possible don't care positions.

Once done, each row may have any number of  $\star$  entries from 0 to  $k-t$  or may consist entirely of don't care positions. When the latter occurs, this row can be removed without reducing the strength of the CA.

## 2.2 Choosing a Row to Eliminate

In some cases, simply identifying don't care positions enables us to remove a row, but this is atypical unless the CA is very far from optimal. Therefore we attempt to produce more don't care positions in one row by using don't care positions in others, with the objective of generating an entire row of don't care positions. Thus we wish to select a row that can be 'easily' removed. A natural selection is a row that has the most don't care positions already. Perhaps a more

appropriate selection would be the row in which the number of multiply covered  $t$ -tuples is largest. When  $\star$  entries are present, however, replacing the  $\star$  by a value results in a substantial change in this statistic. For this reason, one should calculate, for a row with  $\ell$   $\star$  entries, the quantity  $\sum_{i=1}^{\ell} \binom{\ell}{i} \binom{k-\ell}{t-i}$  plus the number of multiply covered  $t$ -tuples, and select a row that maximizes this quantity. This would require substantially more computation, so a simple count of don't care positions is used here.

### 2.3 Algorithm

Having nominated a row for possible elimination, we move the nominated row to be the last row of the CA. We now use don't care positions in other rows in an attempt to introduce (eventually) further don't care positions in the nominated row. A simple strategy is to consider each entry of the nominated row that is not  $\star$ , locate all  $\star$  positions in the same column, and replace each by the entry in the nominated row. This can result in  $t$ -way interactions that were covered only in the last row also being covered earlier, and can therefore result in new don't care positions in the last row. In our experiments we found this simple strategy to be too restrictive; while it can produce new don't care positions in the last row, it often fails to produce much change in the pattern of don't care positions in the rest of the array. We therefore adopt a less restrictive approach. For each  $\star$  position, if the nominated row does not contain  $\star$  in that column, we replace the  $\star$  with the value from the nominated row; otherwise we select a value at random to replace the  $\star$ .

One iteration typically produces a different covering array from the one given as input. However, if we simply find don't care positions again, often the set is very similar to the one just used, and consequently the method stalls quickly. Instead we randomly reorder all rows except the last. Then finding don't care positions typically yields a different set – but in the last row, all positions that were don't care positions remain so. Of course, another row that previously had fewer don't care positions than the nominated row may now have more; if it does, it becomes the nominated row and is moved to the bottom.

Arguably, one should be more clever in filling the don't care positions, and in reordering the rows. Perhaps this is so, but in our experience the randomness of these two choices is crucial. Whatever choices are made, it can happen that the same row is nominated at each step, but no row reordering of the remaining rows yields a set of  $\star$  positions that result in an improvement of the nominated row (i.e., more  $\star$  positions).

### 2.4 Escaping Local Optima

The decision that the CA is unlikely to be improved from its current state can be done by monitoring the total number of don't care positions in the array, or the number in the nominated row, and abandoning the nominated row when it is 'too long' since the number has improved. We use the number in the nominated row, and set a threshold on the number of iterations permitted without improvement. When we exceed the threshold, we take this as evidence that the search has

converged to a local optimum. We employ a simple method of escaping. We move the nominated row along with any other row that contains a don't care position to the top of the CA, fill all the  $\star$  positions with random values and start with this revised array. This could result in a major change in the state of the CA, and indeed the next row nominated may have substantially fewer don't care positions than the one just abandoned.

## 2.5 Implementation and Scalability

The escape from local optima permits us to start from one CA and produce a very different one. Therefore multiple processes can execute simultaneously, all working from a single start point and exploring different areas of the search space. Once an improvement has been made by one of the tasks the result can be shared with the others as the new starting point. An effective way to check for improvements among all processes uses an 'Allgather' operation, in which every process shares its current number of rows with the others. If there is a difference between the minimum and maximum of the values then the best result is broadcast from the lowest ranking process with the best result. A reasonable amount of time, at least sufficient for one iteration to complete, must be dedicated to searching for an improvement before communicating with other processes. We have implemented the method both in a sequential setting and in a parallel one as outlined.

## 3 Results

Perhaps the biggest surprise is that the algorithm works at all. Previously the best result for  $CAN(6,8,5)$  is the upper bound 32822 from IPOG-F [31]. Starting with this array, our method eliminates 4034 rows to show that  $CAN(6,8,5) \leq 28788$  in *one minute* of computation; in ten minutes it reduces to 27909 rows; in one hour to 27772; and in five hours to 27717. While five hours may be longer than one wishes to spend, one minute to remove 12.3% of the rows appears well worth the effort! (All times reported here are for an 8-core Intel Xeon processor clocked at 2.66GHz with 4MB of cache, bus speed 1.33GHz, and 16GB of memory. Only one core is used when timing is reported. The program is coded in C++.)

A striking example is the well studied case  $CA(N; 2, 20, 10)$ . In the announcement of AETG [25],  $CAN(2, 20, 10) \leq 180$  is stated, but no explicit description is given. Yet the commercial implementation of AETG reports 198 rows. A recent paper by Calvagna and Gargantini [35] reports bounds on  $CAN(2,20,10)$  from ten methods; other than the bound of 180 reported by AETG [25], the remaining methods give bounds of 193, 197, 201, 210, 210, 212, 220, 231, and 267. Metaheuristic search using simulated annealing [22] yields 183 rows. Two combinatorial constructions both using a 1-rotational automorphism [13, 14] yield 181 rows. Finally it was shown that  $CAN(2, 20, 10) \leq 174$  using a double projection technique [13]. In Table 1 we apply postoptimization to seven covering arrays; we give the method used to produce a  $CA(N_{old}; 2, 20, 10)$ , the number

**Table 1.** Postoptimization on  $CA(N; 2, 20, 10)$ s

Method	$N_{old}$	$N_{new}$	Poss. $\star$	$\star$
TCG	217	198	444	256
IPO	212	196	449	285
density	203	195	170	79
AETG	198	190	195	132
annealing	183	183	13	3
1-rotational	181	181	0	0
double projection	178	162	415	146

$N_{new}$  of rows after postoptimization, and the numbers of possible don't care and don't care positions. The best establishes that  $CAN(2, 20, 10) \leq 162$ ; five of the seven improve, but those from simulated annealing and the 1-rotational solution see no improvement. The improvement on  $CAN(2, 20, 10)$  is remarkable, given the variety of methods that have been previously applied to try to improve this bound.

We therefore consider projection further. In [13], a construction of Stevens, Ling, and Mendelsohn [36] is generalized to a projection technique that produces a  $CA(q^2 - x; 2, q + 1 + x, q - x)$  from an  $OA(q^2; 2, q + 1, q)$  when  $q$  is a prime power and  $x \geq 0$ . It is so named because  $x$  symbols of the  $OA$  are 'projected' to form  $x$  new columns. (See [13] for details.) There it is observed that  $x$  symbols can be projected to form  $2x$  new columns (a 'double projection'), but the result is no longer a covering array. Rather it is a partial covering array that leaves many pairs uncovered, but also contains many don't care positions. A general pattern to complete this partial array while adding few rows is elusive, if indeed one exists at all. We therefore employ this partial covering array as a 'seed' and complete it using the density algorithm [27]. We found that treating all uncovered pairs equally, as density does, results in the addition of many rows (for example, for the partial  $CA(166; 2, 20, 10)$ , as many as 50 new rows). Therefore we modified the greedy selection in density to weight uncovered pairs on columns  $\{q + 1, \dots, q + 2x\}$  highest, pairs with one column from  $\{q + 1, \dots, q + 2x\}$  next, and pairs with neither column from  $\{q + 1, \dots, q + 2x\}$  least; then density selects the largest total weight of uncovered pairs. This remains a greedy heuristic; nevertheless, it adds as few as 12 rows to complete the partial  $CA(166; 2, 20, 10)$ .

Using projection and double projection on the  $OA(q^2; 2, q + 1, q)$  for  $q \in \{13, 16, 17, 19\}$  and completing with the weighted density method, we formed numerous covering arrays and applied postoptimization to each. When  $x > 1$ , each saw a reduction in the number of rows, sometimes dramatic. In Table 2 we report the new bounds obtained. The value in parentheses is the number of rows of the  $CA$  prior to postoptimization.

One expects that the rows added by density are less effective in the coverage of pairs than the rows of the  $OA$  to which double projection are applied. Surprisingly, postoptimization can succeed in eliminating so many rows that at the end fewer than  $q^2 - x$  remain!

**Table 2.** Covering Arrays from Double Projection

$t$	$v$	$k$	$\text{CAN}(t, k, v) \leq$	Old Bound	$t$	$v$	$k$	$\text{CAN}(t, k, v) \leq$	Old Bound
2	10	17	152 (166)	154 [14]	2	10	18	155 (178)	163 [14]
2	10	19	159 (178)	172 [14]	2	10	20	162 (178)	174 [13]
2	10	21	171 (189)	190 [14]	2	10	22	184 (195)	191 [7]
2	11	18	180 (193)	181 [14]	2	12	16	192 (219)	199 [14]
2	13	20	246 (253)	253 [14]	2	14	19	253 (254)	254 [13]
2	14	21	279 (286)	286 [13]	2	14	24	310 (387)	313 [14]
2	15	24	343 (357)	357 [13]	2	16	23	353 (358)	358 [13]

**Table 3.** Covering Arrays from Density

CA( $N; 4, k, 3$ )											
$k$	New	Old	$k$	New	Old	$k$	New	Old	$k$	New	Old
11	211	230	17	300	312	24	377	389	31	440	446
33	454	461	34	462	468	40	499	504	41	506	510
43	518	522	44	522	526	45	526	530	46	530	534
48	542	546	52	560	562	53	565	567	54	568	572
56	578	581	57	581	584	58	585	588	59	589	592
63	604	607	64	612	614	66	618	620	70	627	629
CA( $N; 5, k, 2$ )											
11	82	86	12	89	92	13	95	103	14	103	110
16	117	123	18	127	135				15	110	115
CA( $N; 6, k, 2$ )											
9	118	120	10	144	150	11	167	178	12	184	190
18	294	309	19	309	323	20	327	337	21	341	352
23	371	377	33	496	503	34	502	508	35	510	516
37	534	541							36	525	529

Now we consider arrays from the density method [28, 37]. We treat a few specific values of  $t$  and  $v$ . In Table 3, each input array  $\text{CA}(N; 4, k, 3)$ ,  $\text{CA}(N; 5, k, 2)$ , and  $\text{CA}(N; 6, k, 2)$  is from density [28, 37], and postoptimization is run for 10 minutes (on a single core). The wall clock time limit results in many more iterations being completed when  $k$  is small; we expect that this is the primary reason for the larger improvements for few factors. Two of the ‘old’ bounds ( $\text{CAN}(5, 12, 2) \leq 92$  and  $\text{CAN}(5, 14, 2) \leq 110$ ) are from [16]. For  $t = 6$ , the ‘old’ bounds are from [20] when  $k = 9$ , a greedy method of Kuliamin [38] when  $k = 10$ , PaintBall [32] when  $k \in \{11, 12, 16\}$ , and density [28, 37] otherwise. All of the new bounds are obtained by postoptimization of CAs from density.

It appears that postoptimization is applicable to covering arrays from a number of sources, but there are cases where it has no effect. Indeed we applied postoptimization to all of the arrays found by Nurmela [23] using tabu search, and none improved. We applied postoptimization to numerous arrays found by Cohen [22] using simulated annealing, and none improved.



We report one more successful application next. Colbourn and Kéri [17] recently employed Hadamard matrices to establish that  $\text{CAN}(4, 20, 2) \leq 40$ ,  $\text{CAN}(4, 32, 2) \leq 64$ , and  $\text{CAN}(4, 36, 2) \leq 72$ ; previously the best known bounds were  $\text{CAN}(4, 20, 2) \leq 55$  [9],  $\text{CAN}(4, 32, 2) \leq 73$  [38], and  $\text{CAN}(4, 36, 2) \leq 95$  [9]. Applying postoptimization to the Hadamard matrix solutions improve these to establish that  $\text{CAN}(4, 20, 2) \leq 39$ ,  $\text{CAN}(4, 32, 2) \leq 59$ , and  $\text{CAN}(4, 36, 2) \leq 66$ .

## 4 Using Postoptimization in Practice

Arguably the success of postoptimization is evidence of our limited understanding of covering arrays. Indeed the restrictions on applicability of combinatorial constructions have forced us to consider computational search for ‘small’ covering arrays both to provide best known small arrays, and to serve as ingredient arrays in recursions. However our ability to carry out computations is limited. To illustrate this, consider strength  $t = 4$  using [33]. Among the best known arrays, only the bounds  $\text{CAN}(4, 13, 2) \leq 34$  [39],  $\text{CAN}(4, 6, 3) \leq 111$  [39],  $\text{CAN}(4, 7, 3) \leq 126$  [39],  $\text{CAN}(4, 8, 3) \leq 153$  [22],  $\text{CAN}(4, 6, 4) \leq 375$  [22],  $\text{CAN}(4, 7, 6) \leq 1893$  [39], and  $\text{CAN}(4, 8, 6) \leq 2068$  [39] are produced by simulated annealing. None have been produced by tabu search, constraint satisfaction, or other metaheuristic search techniques. The workhorses of computation are the greedy methods; both density [28] and IPO [30,31] yield numerous best known covering arrays of strength four. IPO, for example, yields the best known  $\text{CA}(207; 4, 599, 2)$ ,  $\text{CA}(1050; 4, 445, 3)$ ,  $\text{CA}(3170; 4, 308, 4)$ ,  $\text{CA}(7145; 4, 208, 5)$ , and  $\text{CA}(13983; 4, 163, 6)$ , along with many arrays with fewer columns. Some direct constructions that limit or eliminate the computation provide sporadic results, but the rest of our knowledge rests on recursions.

What explains the prevalence of greedy computations among the best known results? It is very unlikely that simulated annealing or tabu search would not yield better results, if either could be run for an adequate period of time. That is precisely the problem, however. Neither has been implemented so as to find competitive solutions starting from scratch within a time frame that anyone is willing to invest. Yet neither is configured so as to take an existing covering array and improve it by removing rows. Indeed both have been devised to improve a partial covering array to make it cover more and more  $t$ -way interactions within a specified number of rows. Hence if the time allocated is insufficient, these metaheuristic search methods end with an array that is still not a covering array. The fundamental difference in postoptimization is that at every stage we are dealing with a covering array, not a partial one. This focuses the search much more than is typically done with simulated annealing or tabu search.

This suggests the main merit of using postoptimization. In using a greedy approach, or a recursion that may have poor ingredients, we do not expect to produce a covering array whose size is close to the minimum. But we can produce such an array quickly for a wide range of parameters. And having produced it, we can invest time in postoptimizing the array, stopping at any time with the assurance that a covering array is produced. This appears to be a practical



solution to the problem of balancing the time to produce a test suite (covering array) and the time to execute the tests. Within a total time budget for testing, it suggests the feasibility of investing less time in the initial construction of the tests while exploiting the (relatively) fast operation of postoptimization to reduce the time for test execution.

Postoptimization also plays a role in producing the smallest arrays known, as we have seen. Naturally it would be of interest to be able to predict the extent to which postoptimization will be successful. This could help us decide when to try postoptimization. Perhaps more importantly, it would suggest criteria to construct covering arrays that are amenable to postoptimization. Consider Table 1 for the widely studied case  $CA(N; 2, 20, 10)$ . Obviously the repetition of coverage in the larger arrays is greater in total, yet the size of the input array does not serve as a good predictor of the improvement seen. In these results, the number of possible don't care positions appears to be the key. Certainly the presence of possible don't care positions is necessary for improvement. However, we believe that the distributions of possible don't care positions among the rows and columns of the array also affect the extent of improvement. Moreover, the patterns of positions that can be realized simultaneously as don't care positions may be more relevant than the pattern of possible don't care positions. Nevertheless, using the number of possible don't care positions as a preliminary indicator of the potential improvement appears worthwhile.

## 5 Conclusion

It comes as no surprise that many of the covering arrays that are best known at present are far from optimal. In these cases, postoptimization provides a relatively fast method for detecting and exploiting duplication of coverage in order to improve the arrays. More surprising are the cases in which postoptimization improves on a result that is already better than those obtained from heuristic search, as we saw with double projection and with arrays from Hadamard matrices. In these cases, the reason for success does not appear to be the poor quality of the initial array. While duplication of coverage is necessary in all arrays with  $N > v^t$ , the distributions of numbers of times that a  $t$ -way interaction is covered can vary widely from interaction to interaction. This can result in certain cells or rows being more effective in coverage than are others. By focusing on arrays in which the contributions of cells or rows are quite unbalanced, postoptimization is sometimes able to eliminate the need for a cell, and perhaps an entire row.

The main benefits of postoptimization are that it does not depend on a particular construction technique; iterations can be executed in approximately the same time as needed to check that the array is in fact a covering array; and that it can be executed for as many iterations as desired, with the assurance that whenever it is stopped, the array is a covering array. At present the main limitations are that it does not appear to be effective for certain covering arrays such as those produced by metaheuristic search; and that the extent of improvement that one can expect cannot be reliably predicted. Despite these limitations,

postoptimization has already proved to be an easy and effective means to improve a wide variety of covering arrays.

## Acknowledgements

Research of the second author is supported by DOD grant N00014-08-1-1070.

## References

1. Colbourn, C.J.: Combinatorial aspects of covering arrays. *Le Matematiche (Catania)* 58, 121–167 (2004)
2. Hartman, A.: Software and hardware testing using combinatorial covering suites. In: Golumbic, M.C., Hartman, I.B.A. (eds.) *Interdisciplinary Applications of Graph Theory, Combinatorics, and Algorithms*, pp. 237–266. Springer, Norwell (2005)
3. Hedayat, A.S., Sloane, N.J.A., Stufken, J.: *Orthogonal Arrays*. Springer, New York (1999)
4. Godbole, A.P., Skipper, D.E., Sunley, R.A.:  $t$ -covering arrays: upper bounds and Poisson approximations. *Combinatorics, Probability and Computing* 5, 105–118 (1996)
5. Katona, G.: Two applications (for search theory and truth functions) of Sperner type theorems. *Periodica Math.* 3, 19–26 (1973)
6. Kleitman, D., Spencer, J.: Families of  $k$ -independent sets. *Discrete Math.* 6, 255–262 (1973)
7. Colbourn, C.J., Martirosyan, S.S., Mullen, G.L., Shasha, D.E., Sherwood, G.B., Yucas, J.L.: Products of mixed covering arrays of strength two. *Journal of Combinatorial Designs* 14(2), 124–138 (2006)
8. Cohen, M.B., Colbourn, C.J., Ling, A.C.H.: Constructing strength three covering arrays with augmented annealing. *Discrete Math.* 308, 2709–2722 (2008)
9. Colbourn, C.J., Martirosyan, S.S., van Trung, T., Walker II, R.A.: Roux-type constructions for covering arrays of strengths three and four. *Designs, Codes and Cryptography* 41, 33–57 (2006)
10. Martirosyan, S.S., Colbourn, C.J.: Recursive constructions for covering arrays. *Bayreuther Math. Schriften* 74, 266–275 (2005)
11. Martirosyan, S.S., van Trung, T.: On  $t$ -covering arrays. *Des. Codes Cryptogr.* 32, 323–339 (2004)
12. Colbourn, C.J.: Distributing hash families and covering arrays. *J. Combin. Inf. Syst. Sci.* (to appear)
13. Colbourn, C.J.: Strength two covering arrays: Existence tables and projection. *Discrete Math.* 308, 772–786 (2008)
14. Meagher, K., Stevens, B.: Group construction of covering arrays. *J. Combin. Des.* 13(1), 70–77 (2005)
15. Sherwood, G.B., Martirosyan, S.S., Colbourn, C.J.: Covering arrays of higher strength from permutation vectors. *J. Combin. Des.* 14(3), 202–213 (2006)
16. Walker II, R.A., Colbourn, C.J.: Tabu search for covering arrays using permutation vectors. *J. Stat. Plann. Infer.* 139, 69–80 (2009)
17. Colbourn, C.J., Kéri, G.: Covering arrays and existentially closed graphs. In: Xing, C., et al. (eds.) *IWCC 2009. LNCS*, vol. 5557, pp. 22–33. Springer, Heidelberg (2009)

18. Chateaufneuf, M.A., Kreher, D.L.: On the state of strength-three covering arrays. *J. Combin. Des.* 10(4), 217–238 (2002)
19. Johnson, K.A., Entringer, R.: Largest induced subgraphs of the  $n$ -cube that contain no 4-cycles. *J. Combin. Theory Ser. B* 46(3), 346–355 (1989)
20. Johnson, K.A., Grassl, R., McCanna, J., Székely, L.A.: Pascalian rectangles modulo  $m$ . *Quaestiones Math.* 14(4), 383–400 (1991)
21. Tang, D.T., Chen, C.L.: Iterative exhaustive pattern generation for logic testing. *IBM Journal Research and Development* 28(2), 212–219 (1984)
22. Cohen, M.B.: Designing test suites for software interaction testing. PhD thesis, The University of Auckland, Department of Computer Science (2004)
23. Nurmela, K.: Upper bounds for covering arrays by tabu search. *Discrete Applied Mathematics* 138(9), 143–152 (2004)
24. Hnich, B., Prestwich, S., Selensky, E., Smith, B.M.: Constraint models for the covering test problem. *Constraints* 11, 199–219 (2006)
25. Cohen, D.M., Dalal, S.R., Fredman, M.L., Patton, G.C.: The AETG system: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering* 23(7), 437–444 (1997)
26. Tung, Y.W., Aldiwan, W.S.: Automating test case generation for the new generation mission software system. In: *Proc. 30th IEEE Aerospace Conference*, pp. 431–437. IEEE, Los Alamitos (2000)
27. Bryce, R.C., Colbourn, C.J.: The density algorithm for pairwise interaction testing. *Software Testing, Verification, and Reliability* 17, 159–182 (2007)
28. Bryce, R.C., Colbourn, C.J.: A density-based greedy algorithm for higher strength covering arrays. *Software Testing, Verification, and Reliability* 19, 37–53 (2009)
29. Tai, K.C., Yu, L.: A test generation strategy for pairwise testing. *IEEE Transactions on Software Engineering* 28(1), 109–111 (2002)
30. Lei, Y., Kacker, R., Kuhn, D.R., Okun, V., Lawrence, J.: IPOG: A general strategy for  $t$ -way software testing. In: *Fourteenth Int. Conf. Engineering Computer-Based Systems*, pp. 549–556 (2007)
31. Forbes, M., Lawrence, J., Lei, Y., Kacker, R.N., Kuhn, D.R.: Refining the in-parameter-order strategy for constructing covering arrays. *J. Res. Nat. Inst. Stand. Tech.* 113(5), 287–297 (2008)
32. Kuhn, D.R., Lei, Y., Kacker, R., Okun, V., Lawrence, J.: Paintball: A fast algorithm for covering arrays of high strength. Internal Tech. Report, NISTIR 7308 (2007)
33. Colbourn, C.J.: Covering array tables, <http://www.public.asu.edu/~ccolbou/src/tabby>, 2005–present
34. Sherwood, G.: Effective testing of factor combinations. In: *Proc. 3rd Int'l Conf. Software Testing, Analysis and Review, Software Quality Eng.* (1994)
35. Calvagna, A., Gargantini, A.: IPO-s: incremental generation of combinatorial interaction test data based on symmetries of covering arrays. In: *Proc. Fifth Workshop on Advances in Model Based Testing* (to appear, 2009)
36. Stevens, B., Ling, A., Mendelsohn, E.: A direct construction of transversal covers using group divisible designs. *Ars Combin.* 63, 145–159 (2002)
37. Linnemann, D., Frewer, M.: Computations with the density algorithm (private communication by e-mail) (October 2008)
38. Kuliainin, V.V.: Private communication by e-mail (February 2007)
39. Soriano, P.P.: Private communication by e-mail (March 2008)

# New Word-Based Adaptive Dense Compressors\*

Petr Procházka and Jan Holub

Department of Computer Science and Engineering,  
FEE Czech Technical University in Prague,  
Karlovo náměstí 13, 121 35 Praha 2, Czech Republic  
{prochp5,holub}@fel.cvut.cz  
<http://www.fel.cvut.cz>

**Abstract.** In the last two decades the natural language compression made a great progress. The main step in this evolution was the introduction of word-based compression by Moffat. The word-based statistical compression algorithms are able to achieve 35% improvement in the compression ratio in comparison with character-based ones. We present two new word-based statistical compression algorithms based on dense coding idea: *Two Byte Dense Code (TBDC)* and *Self-Tuning Dense Code (SCDC)*. *TBDC* uses the codewords with maximal size 2 bytes and must be implemented with some pruning technique. *STDC* is able to tune its code space during the compression process and so achieve better compression. Our algorithms improve the compression ratio and are considerate to smaller files which are very often omitted. We present also a generalized concept of dense coding called *Open Dense Code (ODC)* which provides a frame for definition of these two and many other dense code schemas.

**Keywords:** Natural language compression, Word-based compression, Dense Code.

## 1 Introduction

The amount of information is rapidly growing up. Data compression helps us to reduce two most expensive resources: time needed to transmit data and space needed to store data. Word-based text compression is a novel approach to text data compression proposed by A. Moffat in [1]. It belongs to the family of lossless data compression methods where the compression process is fully reversible and decompressed data is identical to the original data. Word-based algorithms use words instead of characters as the symbols of alphabet. It means that they work with larger units (words instead of characters) and they can exploit longer correlation in the text. Due to work with the larger units the word-based algorithms are able to achieve better results in compression ratio but they have

---

\* This research has been partially supported by the Ministry of Education, Youth and Sports under research program MSM 6840770014, and by the Czech Science Foundation as project No. 201/09/0807.

higher memory requirements. Word-based algorithms consider the text like a strictly alternating sequence of *words* (string composed of alphanumeric characters) and *non-words* (string composed of non-alphanumeric characters). This is a necessary condition for input data because of the ability to split the data stream into single symbols.

The word-based approach takes advantage of the fact that words in a natural language are not created as a random combination of characters so the words contain a possible coding redundancy in the text. We can use this redundancy and implement context character-based compression methods of Markov model of higher order or we can eliminate the redundancy by raising our approach from character-based to word-based. In fact the set of words related to certain context is very limited because of Heaps Law [4], which implies that we can easily eliminate the redundancy when we will represent the words in some more space-saving way than like a sequence of characters. Following a Zipfs Law [5] we can see that the distribution of the words is biased, which means that the words can be efficiently compressed using the statistical compression methods.

The rest of this paper is organized as follows. In Section 2 we discuss related work. We present idea of Open Dense Code in Section 3. Definitions of *TBDC* and *SCDC* are presented in Section 4 and Section 5. Our experiments are discussed in Section 6. Finally in Section 7 we present our conclusions and ideas for future work.

## 2 Related Work

The statistical data compression methods are based on knowledge of probability of each symbol. The probability of symbol is defined as a number of occurrences of the symbol. Suppose Compressor and Decompressor performing data compression and decompression. They consist of two basic modules: Model and Coder. In statistical compression methods the Model module is responsible for collecting all necessary statistics and transforming them into the compression model. The Coder simply uses the compression model to output a bit stream representing the symbol.

### 2.1 End-Tagged Dense Code

*End-Tagged Dense Code (ETDC)* is a word-based compression method proposed by Brisaboa et al. in [2]. *ETDC* was designed to be very fast in both compression and decompression. Especially the fact that *ETDC* is byte-oriented improves substantially the speed of compression and decompression. Byte-oriented means that the lowest coding unit is one byte. So the coder encodes each symbol as a sequence of bytes instead of bits as it is usual.

The idea of *ETDC* is very simple. The Model module collects frequency of each word. It uses this information not to define the probability, but just to define the rank of each word. So the main issue of the Model module is to store all words in descending order by their frequency.

The Coder module uses the rank information to output a codeword of each word. It assigns shorter codeword to the word of higher rank. The structure of code is very simple again. Codeword is represented as a sequence of blocks. Size of the block is usually eight, which ensures the byte-oriented approach. A combination of all bits except the most important bit of each block defines the rank of word. The most important bit is used to define whether the current block is the last or not. This property ensures that the code is a prefix code which is necessary condition to be unambiguously decodable. It brings still another advantage, that the semi-static version of *ETDC* allows direct searching the compressed data as the single codewords are easy recognizable. The searched pattern is just compressed according to dictionary and then any classical string matching algorithm (like Boyer-Moore) is performed. Only small modification of the algorithm is needed. The algorithm needs to check, whether the highest bit of preceding block is set to one, and only then it can submit the hit.

**Definition 1.** *Given source symbols with decreasing probabilities  $\{p_i\}_{0 \leq i < n}$  the corresponding codeword using the End-Tagged Dense Code is formed by a sequence of symbols of  $b$  bits, all of them representing digits in base  $2^{b-1}$ , except the last one which has a value between  $2^{b-1}$  and  $2^b - 1$ , and the assignment is done in a sequential fashion.*

The main goal of *ETDC* is to achieve substantially better compression ratio in comparison with standard character-based algorithms while the compression and decompression times remain approximately the same. To achieve this goal it is necessary to use some efficient data structure to store the compression model. Such a structure was proposed by Brisaboa et al. in [2] and is shown in Figure 1.

The dictionary data structure is formed of a hash table and other two arrays. The hash table is composed of three arrays which are indexed by hash value of each word. Array *word* stores textual representation of the word, array *posInVoc* stores pointer to vocabulary, which is in fact rank of the word. Array *freq* stores frequency of given word. There are two other arrays. The first *top* array is

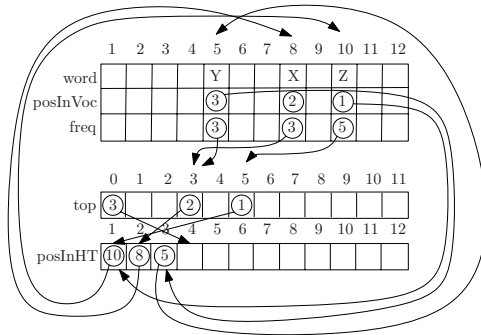


Fig. 1. Dictionary data structure

indexed by frequency and stores the first position of some word of given frequency in vocabulary. The last is *posInHT* array (vocabulary itself), which stores the pointers to words sorted by decreasing frequency.

The point of the dictionary data structure is that the update operation can be performed very fast (in constant time) by swapping of just occurred word with the top word of its original frequency. The update function is described as Algorithm 1. Both the vocabulary data structure and the update function are used also in our implementations (*TBDC* and *STDC*).

---

**Algorithm 1.** Update vocabulary algorithm

---

```

1: function UPDATEVOC(index)
2:   posInVocInd  $\leftarrow$  posInVoc[index]
3:   posInVocTop  $\leftarrow$  top[freq[index]]
4:   posInHTInd  $\leftarrow$  index
5:   posInHTTop  $\leftarrow$  posInHT[top[freq[index]]]
6:   posInVoc[posInHTInd]  $\leftarrow$  posInVocTop
7:   posInVoc[posInHTTop]  $\leftarrow$  posInVocInd
8:   posInHT[posInVocTop]  $\leftarrow$  posInHTInd
9:   posInHT[posInVocInd]  $\leftarrow$  posInHTTop
10:  top[freq[index]]  $\leftarrow$  top[freq[index]] + 1
11:  freq[index]  $\leftarrow$  freq[index] + 1
12: end function

```

---

**2.2 (s,c) - Dense Code**

Another word-based compression method proposed by Brisaboa et al. in [3] is (*s,c*)-Dense Code (*SCDC*). The main drawback of *ETDC* is that it cannot adjust its coding schema to word distribution of compressed text. *SCDC* unlike *ETDC* does not use most important bit of each block to mark the end of codeword. It distinguishes so called *stoppers* and *continuers*, *s* stands for number of *stoppers*, *c* stands for number of *continuers* and  $s + c = 2^b$  where *b* is a size of block. The *SCDC* codeword is then designed as a sequence of zero or more *continuers* closed by one *stopper*. *SCDC* is in fact generalization of *ETDC* as *ETDC* is (*128,128*)-Dense Code. *SCDC* is in [3] exactly defined as follows.

**Definition 2.** Given source symbols with decreasing probabilities  $\{p_i\}_{0 \leq i < n}$  an (*s,c*) stop-cont code (where *c* and *s* are integers larger than zero) assigns to each source symbol *i* a unique target code formed by a base-*c* digit sequence terminated by a digit between *c* and  $c + s - 1$ .

Adaptive version of *SCDC* is called by the authors *Dynamic (s,c) - Dense Code DSCDC*. Compressor processes the input text and when new word occurs, a special escape symbol is transmitted followed by plain text form of the word. When already known word occurs, compressor transmits codeword assigned to the word and updates the vocabulary (see Algorithm 1). Compressor and decompressor are also adjusting the values *s* and  $c = 2^b - s$  according to word

distribution as the compression continues. However this tuning technique does not work for each block separately and so it tunes the values regardless the single blocks. Another evident drawback is that this technique is very unfriendly to small files (with the size lower than 1 MB).

### 3 Open Dense Code

Let us introduce a novel concept of dense coding called *Open Dense Code (ODC)*. *ODC* attempts to cover *ETDC*, *SCDC* and other codes based on dense coding idea. The basic motivation is to allow to define some formalized prescript of codewords, which could be followed in combination with dense coding idea.

**Definition 3.** *The b-ary Open Dense Code (ODC) is a couple  $\langle b, G \rangle$  where  $b$  is a size of block and  $G = (N, T, P, S)$  is a grammar defining syntax of the code. *ODC* assigns to the  $r$ -th most frequent symbol (starting with  $r = 0$ ) a codeword  $c_r$  of  $k$  blocks, which satisfies following conditions:*

- (1)  $c_r \in L(G)$ ,
- (2)  $c_r$  is not prefix of any other codeword  $c_i \in L(G)$ ,
- (3)  $\sum_{i=1}^{k-1} \prod_{j=1}^i v_{ji} \leq r < \sum_{i=1}^k \prod_{j=1}^i v_{ji}$ , where  $v_{ji}$  is number of combinations which can occur as a  $j$ -th block in a codeword of length  $i$ .

The condition number one ensures that the codeword  $c_r$  is defined by the grammar  $G$ . The next condition ensures that the code is a prefix code, which means that it is unambiguously decodable. The last condition provides the dense coding property.

Accordance with Definition 3 we can define *ETDC* and *SCDC* as follows:

$b = 8; G(N, T, P, S) :$
$N = \{ \text{Codeword} \},$
$T = \{ s, c \},$
$P$ is defined in Table 1.
$S = \text{Codeword},$

**Table 1.** *ETDC*, *SCDC*: set of rules  $P$

#	Rule
1	<i>Codeword</i> $\rightarrow c$ <i>Codeword</i>
2	<i>Codeword</i> $\rightarrow s$

where symbol  $c$  represents *continuer* symbol,  $c \in \{0, \dots, 127\}$  for *ETDC* and  $c \in \{0, \dots, \text{cont} - 1\}$  for *SCDC* and symbol  $s$  represents *stopper* symbol,  $s \in \{128, \dots, 255\}$  for *ETDC* and  $s \in \{\text{cont}, \dots, 255\}$  for *SCDC*.

### 4 Two Byte Dense Code

*TBDC* is an adaptive compressor based on dense coding idea. The basic motivation of *TBDC* is to improve compression ratio and keep very good compression and decompression speed of all dense compressors which is very close to



character-based statistical compression methods (e.g. Huffman code). Suppose that we compress only natural portions of the text. Then the set of unique words occurring in single portion is very limited. Following Heaps Law [4] the English version of Bible has size approximately 4 MB and contains only 13 413 unique words. Performing *ETDC* on any natural language text the Compressor never uses the codewords with size greater than 3 bytes because 1B, 2B and 3B codewords provide code space for  $128 + 128^2 + 128^3 = 2\,113\,664$  unique words which satisfies almost all natural language texts.

The facts mentioned above lead us to the idea of a code which is focused on natural portions of the text (with size between 1 and 10 MB) and which uses only 1B and 2B codewords. This change allows the Compressor to leave marking of each block in any form (*stopper*, *continuer* as it is in *SCDC* or the highest bit of each block as it is in *ETDC*). Instead of that the Compressor needs to mark only whether the codeword has size 1B or 2B. It means that only the first block is affected by this marking and so the 1B and 2B codewords can cover more words and the algorithm can achieve better compression ratio. On the other hand the implementation of *TBDC* must be implemented with some pruning technique which can prune the vocabulary in case that the input text contains more words than the code space can cover.

To mark 1B or 2B codewords *TBDC* uses analogous to *SCDC* the idea of *stoppers* and *continuers*. The codewords of size 1B are composed only of one *stopper* and the codewords of size 2B are composed of one *continuer* followed by another block in which any combination of bits is allowed. Using *stoppers* and *continuers* in the first byte of the codeword ensures that *TBDC* is a prefix code. Suppose two codewords  $X$  and  $Y$  where  $|X| = 1$  and  $|Y| = 2$ . The codeword  $X$  is composed of one *stopper* and the codeword  $Y$  is composed of one *continuer* followed by another byte. It means that the codewords  $X$  and  $Y$  differ already in the first byte and so  $X$  cannot be a prefix of  $Y$ .

**Definition 4.** *Accordance with Definition [3] we can define TBDC as follows:*

---


$$\begin{aligned}
 &b = 8; G(N, T, P, S) : \\
 &N = \{\text{Codeword}\}, \\
 &T = \{s, c, b\}, \\
 &P \text{ is defined in Table [2]}, \\
 &S = \text{Codeword},
 \end{aligned}$$


---

**Table 2.** *TBDC*: set of rules  $P$

#	Rule
1	Codeword $\rightarrow c\ b$
2	Codeword $\rightarrow s$

where symbol  $s$  represents stopper symbol,  $s \in \{1, \dots, s_i\}$ ; symbol  $c$  represents continuer symbol,  $c \in \{s_i + 1, \dots, 255\}$ ; symbol  $b$  represents byte in which any combination of bits is allowed,  $b \in \{0, \dots, 255\}$ . Symbol  $s_i$  represents the number of stoppers in  $i$ -th step of compression. Similarly symbol  $c_i$  represents number of continuers in  $i$ -th step of compression. In every step  $i$  of the compression it must hold:  $s_i + c_i = 255$ . Codeword 0 is reserved for a special escape symbol.

From Definition 4 it is evident that the number of stoppers  $s_i$  can change as the compression proceeds. We adjust the number of stoppers  $s_i$  to actual number of unique words in the vocabulary. In every step  $i$  of the compression it must hold that  $s_i + (255 - s_i) * 256 \geq top[0]$ . Data structure of vocabulary is the same as in Figure 1 and  $top[0]$  represents the number of unique words stored in the vocabulary. Always when the previous condition is broken, the number of stoppers  $s_i$  must be decremented. To avoid degradation of coding schema we need to state a lower bound for  $s_i$ . When the Compressor achieves this lower bound, the number of stoppers  $s_i$  is no more decremented, but some pruning technique is applied.

We have implemented two different pruning techniques. *Least Frequently Used (LFU)* is very simple and very fast as the vocabulary is sorted by frequency but there is greater negative effect on the compression ratio as we prune the words which were recently added and are more connected with actual context. This negative effect can be eliminated by the other known technique *Least Recently Used (LRU)*. On the other hand *LRU* is a little bit more time-consuming.

## 5 Self-Tuning Dense Code

*STDC* is another adaptive compressor based on the dense coding idea. *STDC* unlike *TBDC* allows codewords of arbitrary size. The code uses again the idea of *stoppers* and *continuers* like *SCDC*. In fact the only difference between *STDC* and *SCDC* is that *SCDC* sets the same number of *stoppers* and *continuers* for all blocks, while *STDC* allows different number of *stoppers* and *continuers* for different blocks. This seemingly insignificant change can bring interesting improvement in compression ratio. This change allows us to change the number of *stoppers* and *continuers* of single blocks as necessary.

Suppose that in the  $i$ -th step of the compression the codeword with size  $k$  blocks is assigned to the last word in the vocabulary. The code space of  $(k - 1)$ -th block is tuned by similar technique like in *TBDC*. In every step  $i$  of the compression it must hold condition  $\sum_{a=1}^k s_{ai} * \prod_{b=1}^{a-1} c_{bi} \geq top[0]$ , where  $s_{ai}$  represents the number of *stoppers* of block  $a$  in step  $i$ ,  $c_{bi}$  represents the number of *continuers* of block  $b$  in step  $i$  and finally  $top[0]$  represents the number of unique words stored in the vocabulary. Always when the mentioned condition is broken, the number of *stoppers* of the  $(k - 1)$ -th block must be decremented. Again to avoid degradation of coding schema we need to state some lower bound for  $s_{k-1}$ . After exceeding of the bound, the number of blocks of last word in the vocabulary  $k$  must be incremented. The code space of blocks  $l$  where  $0 \leq l < k - 1$  are tuned independently using tuning technique proposed by Brisaboa et al. in [6]. For each  $l$  there exists a unique minimum of the function which expresses dependency between compression ratio and  $s_l$ . This fact is exploited in the tuning technique. Compressor and Decompressor store the size of encoded part of file in three variables *prev*, *curr* and *next*. The variable *curr* stores the size of encoded part of file using  $s_l$ , the variable *prev* stores the size using  $s_l - 1$  and the variable *next* stores the size using  $s_l + 1$ . When the difference  $curr - next$  or

$curr - prev$  exceeds some threshold, the  $s_l$  is incremented or decremented and  $curr$ ,  $prev$  and  $next$  are set to zero.

**Definition 5.** *Accordance with Definition 3 we can define STDC as follows:*

$b = 8; G(N, T, P, S) :$
$N = \{Codeword\},$
$T = \{s_a, c_a\},$
$P$ is defined in Table 3.
$S = Codeword,$

**Table 3.** STDC: set of rules  $P$

#	Rule
1	$Codeword \rightarrow c_a \ Codeword$
2	$Codeword \rightarrow s_a$

where symbol  $s_a$  represents stopper symbol, for  $a = 1 : s_a \in \{1, \dots, s_{ai}\}$ ; for  $a > 1 : s_a \in \{0, \dots, s_{ai} - 1\}$ ; symbol  $c_a$  represents continuer symbol, for  $a = 1 : c_a \in \{s_{ai} + 1, \dots, 255\}$ ; for  $a > 1 : c_a \in \{s_{ai}, \dots, 255\}$ . Symbol  $s_{ai}$  represents number of stoppers in block  $a$  in  $i$ -th step of compression. Similarly symbol  $c_{ai}$  represents number of continuers in block  $a$  in  $i$ -th step of compression. In every step  $i$  of the compression it must hold: for  $a = 1 : s_{ai} + c_{ai} = 255$ ; for  $a > 1 : s_{ai} + c_{ai} = 256$ . Codeword 0 is reserved for a special escape symbol.

## 6 Experiments

The test set of compression algorithms is very diverse in order to be able to show advantages and disadvantages of our implementations in comparison with the other algorithms. We have chosen both word-based and character-based, statistical and dictionary types of compression algorithms. All tested statistical compression algorithms use the adaptive compression model. Table 4 provides overview of tested compression algorithms.

We performed the algorithms on various files with English natural language content. Tested files come especially from standard corpuses (Calgary and Canterbury corpus) and from Project Gutenberg<sup>1</sup>. We created four larger corpora (*all1.txt*, *all2.txt*, *all3.txt* and *all4.txt*) by concatenating many files from standard corpuses and especially from Project Gutenberg. All tested files are stated in Table 5.

We used the spaceless word model [7] in our implementations. It means that the vocabulary is common for alphanumeric and non-alphanumeric words. The model takes a single space as a default separator. When alphanumeric word is followed by a space, the Compressor encodes just the alphanumeric word. When alphanumeric word is followed by another non-alphanumeric word (separator), the Compressor encodes both alphanumeric and non-alphanumeric word.

We performed out our tests on an AMD Athlon<sup>TM</sup> 64 Processor 3200+, 2518 MB RAM with Fedora Linux and kernel version 2.6.23.15-80.fc7. We used compiler *gcc* version 3.4.6 with compiler optimization -O3.

<sup>1</sup> Project Gutenberg (www.gutenberg.org) is the first and largest single collection of free electronic books.

Table 4. Tested algorithms

Algorithm	Notation	Approach	Family	Proposed	Implemented
Huffman Coding	<i>huff</i>	cb <sup>a</sup>	statistical	J. S. Vitter [9]	D. Scott [14]
LZ77	<i>lz77</i>	cb	dictionary	A. Lempel, J. Ziv [10]	M. Geelnard [13]
Arithmetic Coding	<i>cac</i>	cb	statistical	Moffat et al. in [8]	Moffat et al. [11]
Two Byte Dense Code(LFU)	<i>tbdc1</i>	wb <sup>b</sup>	statistical	our proposal	our implementation
Two Byte Dense Code(LRU)	<i>tbdc2</i>	wb	statistical	our proposal	our implementation
Self-Tuning Dense Code	<i>stdc</i>	wb	statistical	our proposal	our implementation
Dyn. End-tagged Dense Code	<i>detdc</i>	wb	statistical	Brisaboa et al. in [2]	Brisaboa et al. [12]
Dyn. (s,c)-Dense Code	<i>dscdc</i>	wb	statistical	Brisaboa et al. in [3]	Brisaboa et al. [12]

<sup>a</sup> character-based approach.

<sup>b</sup> word-based approach.

Table 5. Tested files

File	Notation	Source	Size [B]	# words	# unique words
bible.txt	<i>canL</i>	Large Canterbury	4,047,389	889,575	13,413
alice29.txt	<i>can1</i>	Canterbury	148,460	34,040	3,210
plrabn12.txt	<i>can2</i>	Canterbury	471,161	102,773	10,937
book1	<i>cal1</i>	Calgary	768,770	175,853	13,497
book2	<i>cal2</i>	Calgary	610,855	133,338	10,420
paper1	<i>cal3</i>	Calgary	53,160	11,143	2,175
paper2	<i>cal4</i>	Calgary	82,198	17,281	2,669
wrnpcl1.txt	<i>gut1</i>	Gutenberg	3,217,389	697,342	19,740
clarissa.txt	<i>gut2</i>	Gutenberg	5,233,126	1,209,613	22,109
all1.txt	<i>gut3</i>	Gutenberg	12,483,578	2,793,686	38,211
all2.txt	<i>gut4</i>	Gutenberg	19,352,946	4,280,943	58,475
all3.txt	<i>gut5</i>	Gutenberg	28,727,290	6,366,654	70,794
all4.txt	<i>gut6</i>	Gutenberg	48,610,669	10,799,288	96,351

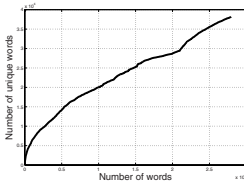
## 6.1 Compression Ratio

The results are summarized in Table 6. Our implementation of *STDC* achieved the best compression ratio in all tested files. It can adjust the coding schema to actual distribution of the alphabet and so achieve better compression ratio.

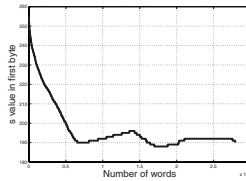
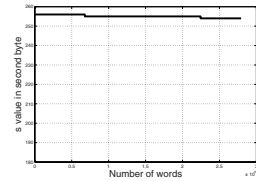
Performed on small and medium files (with size lower than approximately 4 MB) *STDC* and both *TBDC* variants achieve the same compression ratio. Both *STDC* and *TBDC* apply the same tuning technique while they are using only first two bytes to encode a symbol. In the moment when the size of two bytes is exceeded the algorithms apply different approaches to encode a symbol. *TBDC(LFU)* and *TBDC(LRU)* need to use some pruning technique because their codespace is limited. *TBDC(LFU)* uses Least Frequently Used pruning technique and *TBDC(LRU)* uses Least Recently Used pruning technique. On the other hand *STDC* can continue in compression without pruning of the vocabulary and it involves the third byte in its coding schema. Then the algorithm starts to tune the second byte and the first byte is tuned by technique proposed by Brisaboa et al. in [6].

**Table 6.** Compression ratio in %

File/Alg.	huff	lz77	cac	tbdc1	tbdc2	stdc	detdc	dscdc
canL	54.81	<b>45.85</b>	54.36	<b>29.75</b>	<b>29.75</b>	<b>29.75</b>	31.70	30.33
can1	57.72	57.97	<b>56.70</b>	<b>43.00</b>	<b>43.00</b>	<b>43.00</b>	47.92	67.69
can2	57.22	66.97	<b>56.10</b>	<b>46.22</b>	<b>46.22</b>	<b>46.22</b>	49.97	55.59
cal1	57.04	65.53	<b>56.71</b>	<b>42.72</b>	<b>42.72</b>	<b>42.72</b>	46.41	49.35
cal2	60.32	<b>52.25</b>	60.04	<b>42.61</b>	<b>42.61</b>	<b>42.61</b>	45.01	48.64
cal3	62.94	<b>54.80</b>	62.81	<b>57.74</b>	<b>57.74</b>	<b>57.74</b>	62.36	121.57
cal4	58.07	58.25	<b>57.91</b>	<b>49.29</b>	<b>49.29</b>	<b>49.29</b>	54.54	92.54
gut1	56.89	59.86	<b>55.94</b>	<b>33.51</b>	<b>33.51</b>	<b>33.51</b>	35.89	34.92
gut2	57.39	60.89	<b>56.28</b>	33.13	33.17	<b>33.03</b>	35.17	34.04
gut3	56.77	<b>55.73</b>	55.78	32.78	32.44	<b>32.10</b>	34.01	33.03
gut4	56.92	57.31	<b>55.94</b>	33.97	33.19	<b>32.41</b>	34.27	33.41
gut5	56.98	58.13	<b>55.99</b>	33.97	33.24	<b>32.20</b>	33.95	33.10
gut6	57.25	59.14	<b>56.25</b>	34.52	33.02	<b>32.12</b>	33.73	32.93



(a) Unique words

(b)  $s$  in first byte(c)  $s$  in second byte**Fig. 2.** *STDC*: Evolution of various parameters as the number of words grows

We can observe evolution of some tuning parameters of *STDC* in Figure 2. The number of unique words is growing exactly according Heaps Law [4]. The number of *stoppers* in first byte is rapidly falling down during the first phase when only first two bytes are used. During the next phase the number of *stoppers* is oscillating about the value 192. Similarly the number of *stoppers* in second byte is constant during the first phase and its falling down during the second phase. However this fall is not so fast as in the previous case because the coding space is larger and also the number of unique words is not growing so quickly in this phase.

Our implementations (*TBDC* and *STDC*) are friendly to smaller files and they achieve outstanding improvement in compression ratio in comparison with *ETDC* or *SCDC* while performed on files like *cal3*, *cal4*, *can1*. *SCDC* is very unfriendly to small files and sometimes even achieves compression ratio higher than 100%. Performed on larger files both variants of *TBDC* drag behind. Thanks to pruning of the vocabulary they loose precise information about the alphabet distribution. On the other hand *STDC* achieves still better compression ratio: approximately 1% improvement in comparison with *SCDC* and approximately 2% improvement in comparison with *ETDC*.

## 6.2 Compression and Decompression Speed

To achieve fairer comparison among the dense compressors we used the parameters for dictionary data structure optimization and so we avoided useless memory

**Table 7.** Compression speed in MB/s

File/Alg.	huff	lz77	cac	tbdc1	tbdc2	stdc	detdc	dscdc
canL	29.99	0.03	7.15	<b>29.69</b>	27.57	20.32	27.18	24.59
can1	29.50	0.03	7.08	<b>28.32</b>	<b>28.32</b>	20.23	20.23	20.23
can2	29.18	0.02	8.99	<b>24.96</b>	<b>24.96</b>	22.47	19.54	18.73
cal1	29.21	0.02	7.33	<b>24.44</b>	<b>24.44</b>	18.33	20.37	9.40
cal2	<b>28.70</b>	0.03	7.28	<b>25.33</b>	<b>25.33</b>	19.42	21.58	7.77
cal3	<b>26.68</b>	0.08	7.72	<b>25.35</b>	<b>25.35</b>	<b>25.35</b>	16.91	1.37
cal4	<b>29.03</b>	0.05	7.83	<b>26.13</b>	<b>26.13</b>	<b>26.13</b>	19.60	19.60
gut1	<b>29.67</b>	0.02	7.45	21.92	20.46	18.05	<b>23.79</b>	21.61
gut2	<b>29.12</b>	0.02	7.13	21.70	20.79	16.64	<b>24.00</b>	21.42
gut3	<b>29.85</b>	0.02	7.30	22.05	20.18	14.88	<b>23.72</b>	20.78
gut4	<b>29.80</b>	0.02	7.10	20.51	17.92	14.09	<b>22.71</b>	19.83
gut5	<b>29.78</b>	0.02	7.21	20.60	18.45	13.69	<b>22.46</b>	19.53
gut6	<b>29.56</b>	0.02	7.29	19.90	18.55	13.10	<b>21.88</b>	19.06

**Table 8.** Decompression speed in MB/s

File/Alg.	huff	lz77	cac	tbdc1	tbdc2	stdc	detdc	dscdc
canL	25.80	<b>214.44</b>	6.23	41.96	39.63	38.60	<b>55.95</b>	48.26
can1	24.84	<b>202.21</b>	7.08	47.19	47.19	47.19	<b>47.21</b>	23.60
can2	24.96	<b>172.93</b>	6.42	37.44	34.56	34.56	<b>40.86</b>	34.57
cal1	24.85	<b>178.88</b>	6.11	36.66	34.91	33.33	<b>45.83</b>	31.88
cal2	24.07	<b>208.01</b>	5.83	38.84	38.84	36.41	<b>48.55</b>	25.33
cal3	24.14	<b>253.49</b>	7.53	39.00	39.00	36.21	<b>50.70</b>	-
cal4	24.50	<b>261.30</b>	7.83	35.63	32.66	35.63	<b>39.20</b>	15.68
gut1	25.07	<b>191.77</b>	6.23	38.35	38.35	27.89	<b>47.95</b>	43.22
gut2	27.01	<b>191.95</b>	6.24	38.39	31.46	27.72	<b>46.65</b>	39.61
gut3	29.07	<b>198.42</b>	6.14	34.02	32.27	23.81	<b>45.62</b>	38.41
gut4	28.78	<b>196.35</b>	6.13	31.74	30.21	21.46	<b>43.53</b>	36.34
gut5	28.52	<b>194.29</b>	6.12	31.50	30.42	20.75	<b>41.83</b>	36.29
gut6	26.34	<b>191.57</b>	6.14	29.75	29.29	19.48	<b>40.74</b>	34.78

allocation. The results of compression and decompression speed are summarized in Table 7 and Table 8. *TBDC(LFU)* and *ETDC* are the fastest algorithms among the dense compressors in compression. *TBDC(LFU)* is faster than *STDC* because it uses only two bytes coding space and doesn't need to care about tuning all bytes except the first one. It is also faster than *TBDC(LRU)* because Least Frequently Used pruning technique is faster as the vocabulary is sorted by frequency.

In decompression the fastest algorithm at all is *lz77*, which is very asymmetrical. Among the dense compressors the fastest algorithm is *ETDC* which can use bitwise operations in decompression. All our implementations are behind when they are performed on larger files.

## 7 Conclusions

We introduced concept *Open Dense Code* which helped us to define two dense compressors: *TBDC* and *STDC*. *TBDC* proved that it is very friendly to smaller files and is able to achieve better compression ratio than existing dense compressors *SCDC* and *ETDC*. *STDC* is in fact variant of *SCDC* which uses special tuning technique and so achieves better compression ratio. It is in fact extended *TBDC* and so it preserves all advantages of *TBDC*. All our implementations proved to be excellent in compression speed but were a little bit behind in decompression speed.

In our future work we want to improve decompression performance. With the aid of linguistics we want to focus more on many properties of natural language and try to exploit them in compression.

## References

1. Moffat, A.: Word-based Text Compression. *Software - Practise and Experience* 19(2), 185–198 (1989)
2. Brisaboa, N., Iglesias, E., Navarro, G., Paramá, J.: An efficient compression code for text databases. In: Sebastiani, F. (ed.) *ECIR 2003*. LNCS, vol. 2633, pp. 468–481. Springer, Heidelberg (2003)

3. Brisaboa, N., Fariña, A., Navarro, G., Esteller, M.F.: (S,C)-dense coding: An optimized compression code for natural language text databases. In: Nascimento, M.A., de Moura, E.S., Oliveira, A.L. (eds.) SPIRE 2003. LNCS, vol. 2857, pp. 122–136. Springer, Heidelberg (2003)
4. Heaps, H.S.: Information Retrieval: Computational and Theoretical Aspects. Academic Press, London (1978)
5. Zipf, G.K.: Human Behaviour and the Principle of Least Effort. Addison-Wesley, Reading (1949)
6. Brisaboa, N.R., Fariña, A., Navarro, G., Parama, J.R.: New Adaptive Compressors for Natural Language Text. *Software - Practice & Experience* 38(13), 1429–1450 (2008)
7. de Moura, E., Navarro, G., Ziviani, N., Baeza-Yates, R.: Fast searching on compressed text allowing errors. In: Proceedings 21st SIGIR, pp. 298–306 (1998)
8. Moffat, A.: Arithmetic coding revisited. *ACM Trans. on Inf. Systems* 16, 256–294 (1998)
9. Vitter, J.S.: Algorithm 673: Dynamic Huffman coding. *ACM Transactions on Mathematical Software (TOMS)* 15(2), 158–167 (1989)
10. Ziv, J., Lempel, A.: A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory* 23(3), 337–343 (1977)
11. Moffat, A.: The Arithmetic Coding Page, <http://www.cs.mu.oz.au/~alistair/>
12. Brisaboa, et al.: Family of Dense Compressors, <http://vios.dc.fi.udc.es/codes/>
13. Geelnard, M.: Basic Compression Library, <http://bcl.comli.eu/>
14. Scott, D.: Vitter Adaptive Compression, <http://bijective.dogma.net>

# Rainbow Connection in Graphs with Minimum Degree Three

Ingo Schiermeyer

Institut für Diskrete Mathematik und Algebra, TU Bergakademie Freiberg, D-09596  
Freiberg, Germany  
Ingo.Schiermeyer@tu-freiberg.de

**Abstract.** An edge-coloured graph  $G$  is *rainbow connected* if any two vertices are connected by a path whose edges have distinct colours. The *rainbow connection number* of a connected graph  $G$ , denoted  $rc(G)$ , is the smallest number of colours that are needed in order to make  $G$  rainbow connected. In this paper we prove that  $rc(G) < \frac{3n}{4}$  for graphs with minimum degree three, which was conjectured by Caro et al. [Y. Caro, A. Lev, Y. Roditty, Z. Tuza, and R. Yuster, *On rainbow connection*, The Electronic Journal of Combinatorics 15 (2008), #57.]

## 1 Introduction

We use [1] for terminology and notation not defined here and consider finite and simple graphs only.

An edge-coloured graph  $G$  is called *rainbow-connected* if any two vertices are connected by a path whose edges have different colours. This concept of rainbow connection in graphs was recently introduced by Chartrand et al. in [4]. The rainbow connection number of a connected graph  $G$ , denoted  $rc(G)$ , is the smallest number of colours that are needed in order to make  $G$  rainbow connected. An easy observation is that if  $G$  has  $n$  vertices then  $rc(G) \leq n - 1$ , since one may colour the edges of a given spanning tree of  $G$  with different colours, and colour the remaining edges with one of the already used colours.

Chartrand et al. computed the precise rainbow connection number of several graph classes including complete multipartite graphs [4]. It is also known that  $rc(G) = 1$  if and only if  $G$  is a clique, that  $rc(G) = n - 1$  if and only if  $G$  is a tree and that a cycle with  $k \geq 3$  vertices has rainbow connection number  $\lceil \frac{k}{2} \rceil$ . Also notice that  $rc(G) \geq \text{diam}(G)$ , where  $\text{diam}(G)$  denotes the diameter of the graph  $G$ .

Motivated by the fact that there are graphs with minimum degree 2 and with  $rc(G) = n - 3$  (just take two vertex-disjoint triangles and connect them by a path of length  $n - 5$ ), and by the fact that cliques have  $rc(G) = 1$ , it is interesting to study the behaviour of  $rc(G)$  with respect to the minimum degree  $\delta(G)$ . In [3] Caro et al. have shown the following theorem.

**Theorem 1.** *If  $G$  is a connected graph with  $n$  vertices and  $\delta(G) \geq 3$  then  $rc(G) < \frac{5n}{6}$ .*

They also made the following conjecture.



*Conjecture 1.* If  $G$  is a connected graph with  $n$  vertices and  $\delta(G) \geq 3$  then  $rc(G) < \frac{3n}{4}$ .

For 2-connected graphs Conjecture 1 is true. This follows from the following proposition in [3].

**Proposition 1.** *If  $G$  is a 2-connected graph with  $n$  vertices then  $rc(G) \leq \frac{2n}{3}$ .*

**Corollary 1.** *If  $G$  is a 2-connected graph with  $n$  vertices then  $rc(G) \leq \frac{3n-1}{4}$  for  $n \geq 3$ .*

In this paper we will prove this conjecture by proving the following theorem.

**Theorem 2.** *If  $G$  is a connected graph with  $n$  vertices and  $\delta(G) \geq 3$  then  $rc(G) \leq \frac{3n-1}{4}$ .*

For 2-connected graphs Theorem 2 is true by corollary 1. Hence it remains to prove it for graphs with connectivity 1.

## 2 Graphs with Connectivity 1

For connected graphs  $G$  with connectivity  $\kappa(G) = 1$  we extend the concept of rainbow connection as follows: Additionally we require that any two edges of  $G$  have different colours whenever they belong to different blocks of  $G$ . The corresponding rainbow connection number will be denoted by  $rc^*(G)$ . Then  $rc(G) \leq rc^*(G)$  for every graph  $G$  and  $rc(G) = rc^*(G)$  for every 2-connected graph  $G$ .

For graphs with connectivity  $\kappa(G) = 1$  we will show the following theorem.

**Theorem 3.** *Let  $G$  be a connected graph with  $n$  vertices, connectivity  $\kappa(G) = 1$  and  $\delta(G) \geq 3$ . Then  $rc^*(G) \leq \frac{3n-10}{4}$ .*

The bound  $\frac{3n-10}{4}$  cannot be decreased since there are 3-regular connected graphs with  $rc^*(G) = rc(G) = diam(G) = \frac{3n-10}{4}$ . One class of such graphs can be constructed as follows: Take two vertex disjoint copies of the graph  $K_5 - P_3$  and label the two vertices of degree 2 with  $w_1$  and  $w_{2k+2}$ , where  $k \geq 1$  is an integer. Next join  $w_1$  and  $w_{2k+2}$  by a path of length  $2k + 1$  and label the vertices with  $w_1, w_2, \dots, w_{2k+2}$ . Now for  $1 \leq i \leq k$  every edge  $w_{2i}w_{2i+1}$  is replaced by a  $K_4 - e$  and we identify the two vertices of degree 2 in  $K_4 - e$  with  $w_{2i}$  and  $w_{2i+1}$ . The resulting graph  $G_{4k+10}$  is 3-regular, has order  $n = 4k + 10$  and  $rc^*(G_{4k+10}) = rc(G_{4k+10}) = diam(G_{4k+10}) = 3k + 5 = \frac{3n-10}{4}$ .

For the proof of Theorem 3 we show some preparatory results. First we extend Proposition 1 as follows.

**Proposition 2.** *Let  $G$  be a 2-connected graph with  $n$  vertices and degree sequence  $2 \leq d_1 \leq d_2 \leq \dots \leq d_n$ . If  $d_3 \geq 3$ , then  $rc(G) \leq \frac{2n-2}{3}$  for  $4 \leq n \leq 7$  and  $rc(G) \leq \frac{2n-1}{3}$  for  $n \geq 8$ .*

**Corollary 2.** *Let  $G$  be a 2-connected graph with  $n$  vertices and degree sequence  $2 \leq d_1 \leq d_2 \leq \dots \leq d_n$ . If  $d_3 \geq 3$ , then  $rc(G) \leq \frac{3n-4}{4}$ .*

**Proof of Proposition 2.** Let  $H$  be a maximal connected subgraph of  $G$  having the property that  $rc(H) \leq \frac{2h}{3} - 1$ , where  $h$  is the number of vertices of  $H$ . We first claim that  $H$  exists.

Since  $G$  is 2-connected and  $d_3 \geq 3$  we have  $n \geq 4$ . By Dirac's theorem [5] the circumference  $c(G)$  of  $G$  satisfies  $c(G) \geq \min\{n, 2\delta(G)\} \geq 4$ . We now distinguish several cases.

If  $n = 4$ , then  $rc(G) \leq 2 \leq \frac{2 \cdot 4 - 2}{3}$ . If  $c(G) = 4$ , then  $G \cong K_{2,n-2}$ , which contradicts  $d_3 \geq 3$  for  $n \geq 5$ . Hence we may assume  $c(G) \geq 5$ . If  $c(G) = 5 = n$ , then  $G$  contains  $C_5 + e$  as a subgraph implying  $rc(G) \leq 2 < \frac{2 \cdot 5 - 2}{3}$ . If  $c(G) = 5 < n$ , then taking  $H$  as a  $C_5$  with an attached edge we have  $rc(G) = 3 = \frac{2 \cdot 6}{3} - 1$ . If  $c(G) \in \{6, 8\}$ , then  $rc(C_k) = \frac{k}{2} \leq \frac{2k}{3} - 1$  for  $k = 6, 8$ . If  $c(G) = 7 = n$ , then  $rc(C_7) = 4 = \frac{2 \cdot 7 - 2}{3}$ . If  $c(G) = 7 < n$ , then taking  $H$  as a  $C_7$  with an attached edge we have  $rc(H) = 4 < \frac{2 \cdot 8}{3} - 1$ . Finally, if  $c(G) = k \geq 9$ , then  $rc(C_k) = \lceil \frac{k}{2} \rceil \leq \frac{k+1}{2} \leq \frac{2k}{3} - 1$ .

We next claim that  $h \geq n - 2$ . Indeed, assume first that there are three distinct vertices outside of  $H$ , say  $w_1, w_2, w_3$ , each having two neighbours in  $H$  (the neighbours of  $w_i$  do not have to be distinct from the neighbours of  $w_j$ ). We can add  $w_1, w_2, w_3$  to  $H$  and form a larger subgraph  $H'$  with  $h + 3$  vertices. Suppose  $e_i, f_i$  are two edges connecting  $w_i$  with  $H$ . We use only two new colours to colour the 6 designated edges:  $e_1, e_2, e_3$  all get the same colour and  $f_1, f_2, f_3$  all get the same colour. We now have

$$rc(H') \leq rc(H) + 2 \leq \frac{2h}{3} - 1 + 2 = \frac{2(h+3)}{3} - 1$$

contradicting the maximality of  $H$ . It follows that if there are three vertices outside of  $H$  then at least one of these vertices, say  $w$ , has the property that a shortest path from  $H$  to  $H$  passing through  $w$  has length at least 3 (notice that there must be a path since the graph is 2-connected). Let  $uw_1w_2 \dots w_t v$  be a path with  $u, v \in V(H), w_1, \dots, w_t \notin V(H)$ , and  $t \geq 2$ . We can add  $w_1, \dots, w_t$  to  $H$  and form a larger subgraph  $H'$  with  $h + t$  vertices. If  $t$  is odd we can colour the  $t + 1$  edges of the path with  $\frac{t+1}{2}$  new colours. In the first half of the path the colours are all distinct, and the same ordering of colours is repeated in the second half of the path. It is straightforward to verify that  $H'$  is rainbow connected. If  $t$  is even, we can colour the  $t + 1$  edges of the path with  $\frac{t}{2}$  colours as follows. The middle edge  $w_{t/2}w_{t/2+1}$  receives any colour that already appears in  $H$ . The first  $t/2$  edges of the path all receive distinct new colours and in the last  $t/2$  edges of the path this colouring is repeated in the same order. Again, it is straightforward to verify that  $H'$  is rainbow connected. We now have

$$rc(H') \leq rc(H) + \lceil \frac{t}{2} \rceil \leq \frac{2h}{3} - 1 + \lceil \frac{t}{2} \rceil \leq \frac{2(h+t)}{3} - 1$$

contradicting the maximality of  $H$ .

Having proved that  $h \geq n - 2$  we now clearly have  $rc(G) \leq \frac{2(n-2)}{3} - 1 + 2 = \frac{2n-1}{3}$  as claimed. □

Next we determine the structure of endblocks. Let  $\mathcal{B} = \{K_4, K_5, K_5 - e, K_5 - P_3, K_5 - 2P_2, K_5 - (P_3 \cup P_2)\}$ . Then  $rc(K_4) = rc(K_5) = 1$  and  $rc(K_5 - e) = rc(K_5 - P_3) = rc(K_5 - 2K_2) = rc(K_5 - (P_3 \cup P_2)) = 2$ .

For  $B \in \mathcal{B}$  let  $B \cup K_2$  be an endblock with an attached  $K_2$ . If  $B \in \{K_4, K_5, K_5 - e, K_5 - 2P_2\}$ , then the  $K_2$  can be attached to any vertex of  $B$ . If  $B \in \{K_5 - P_3, K_5 - (P_3 \cup P_2)\}$ , then the  $K_2$  is attached to the vertex of degree 2 in  $K_5 - P_3$  or  $K_5 - (P_3 \cup P_2)$ . Now the following claims are easily verified.

*Claim.* Let  $G$  be a connected graph with  $\delta(G) \geq 3$ . If  $G = G_1 \cup G_2$  with  $V(G_1) \cap V(G_2) = w$  for a cut vertex  $w$  and  $|V(G_1)| \leq 6$ , then  $G_1 \cong B$  or  $G_1 \cong B \cup K_2$  for some  $B \in \mathcal{B}$ .

*Claim.* Let  $B \in \mathcal{B}$  be an endblock. Then  $rc(B) \leq \frac{3n-7}{4}$  and  $rc(B \cup K_2) \leq \frac{3n-6}{4}$ .

Now we are ready to prove Theorem 3.

**Proof of Theorem 3.** The key idea is an induction using a cut vertex of the graph  $G$ . For a subgraph  $F \subset G$ ,  $n(F)$  denotes the order of  $F$ .

*Claim.* Let  $G$  be a connected graph with a cut vertex  $w$ . Suppose  $G = G_1 \cup G_2$  with  $V(G_1) \cap V(G_2) = w$ ,  $d_{G_1}(w) \geq 2$ ,  $d_{G_2}(w) \geq 1$ ,  $|V(G_1)| \geq 6$  and  $|V(G_2)| \geq 7$ . Then  $rc^*(G) \leq \frac{3n-10}{4}$  by induction.

*Proof.* We construct two graphs  $H_1$  and  $H_2$  as follows: Let  $H_1 = (K_5 - P_3) \cup G_2$ , where we identify the vertex of degree 2 in  $K_5 - P_3$  with the vertex  $w$  of  $G_2$ . Let  $H_2 = G_1 \cup K_2 \cup (K_5 - P_3)$ , where we identify one vertex of the  $K_2$  with the vertex  $w$  of  $G_1$  and the other vertex of the  $K_2$  with the vertex of degree 2 in  $K_5 - P_3$ .

Then  $|V(H_1)| = |V(K_5 - P_3)| + |V(G_2)| - 1 < |V(G_1)| + |V(G_2)| - 1 = |V(G)|$  and  $|V(H_2)| = |V((K_5 - P_3) \cup K_2)| + |V(G_1)| - 1 < |V(G_2)| + |V(G_1)| - 1 = |V(G)|$ . Hence by induction we have  $rc^*(H_1) \leq \frac{3n(H_1)-10}{4}$  and  $rc^*(H_2) \leq \frac{3n(H_2)-10}{4}$  implying  $rc^*(G_2) \leq \frac{3(n(G_2)+4)-10}{4} - 2 = \frac{3n(G_2)-6}{4}$  and  $rc^*(G_1) \leq \frac{3(n(G_1)+5)-10}{4} - 3 = \frac{3n(G_1)-7}{4}$ . This gives  $rc^*(G) = rc^*(G_1) + rc^*(G_2) \leq \frac{3n(G_1)-7}{4} + \frac{3n(G_2)-6}{4} = \frac{3(n(G_1)+n(G_2)-1)-10}{4} = \frac{3n-10}{4}$ . □

We now consider the block tree  $T$  of  $G$ . Let  $V(T) = \{w_1, w_2, \dots, b_1, b_2, \dots\}$ , where  $w_i$  is a cut vertex of  $G$  and  $b_i$  represents the block  $B_i$  of  $G$ . Then  $w_i b_j \in E(T)$  if and only if  $w_i$  is incident with  $B_j$  in  $G$ . We make the following useful

**Observations**

1. If  $B_i$  is an endblock of  $G$  (that is  $b_i$  is a leaf of  $T$ ), then  $|V(B_i)| \geq 4$ , since  $\delta \geq 3$ .
2. If  $d_T(w) \geq 4$  for a cut vertex  $w$  of  $G$ , then  $G = G_1 \cup G_2$  with  $G_1 \cap G_2 = w$  and  $G_1$  and  $G_2$  both contain at least two nontrivial blocks. Then  $|V(G_i)| \geq 2 \cdot 4 - 1 = 7$  for  $i = 1, 2$  and we can apply induction. Hence we may assume that  $2 \leq d_T(w) \leq 3$  for every cut vertex  $w$ .

3. If a cut vertex  $w$  has degree 3, then  $G = G_1 \cup G_2 \cup G_3$ . By the previous argument (in part 2.) each  $G_i$  contains exactly one nontrivial block, which is an endblock. We have  $|V(G_i)| \leq 6$  for  $1 \leq i \leq 3$ , since otherwise we could apply induction. With  $|V(G)| = |V(G_1)| + |V(G_2)| + |V(G_3)| - 2$  we obtain  $rc^*(G) \leq \frac{3n(G_1)-6}{4} + \frac{3n(G_2)-6}{4} + \frac{3n(G_3)-6}{4} = \frac{3n-12}{4} < \frac{3n-10}{4}$ .
4. If a vertex  $b$  (corresponding to a block  $B$ ) has degree  $p \geq 3$ , then  $B$  is incident with  $p$  cut vertices  $w_1, w_2, \dots, w_p$ . Hence  $G = B \cup G_1 \cup G_2 \cup \dots \cup G_p$ . Then  $rc^*(G) \leq \frac{2n(B)}{3} + \sum_{i=1}^p \frac{3n(G_i)-6}{4} \leq \frac{3n(B)-1}{4} + \frac{3(n+p-n(B))-6p}{4} = \frac{3n-1-3p}{4} \leq \frac{3n-10}{4}$ .

Hence we may assume that  $\Delta(T) = 2$  and thus  $T$  is a path. So finally  $G$  contains at most one nontrivial inner block  $B$ , since otherwise we could apply induction. Therefore, for the path  $T$ , the following graph structures (of blocks) are possible:

1.  $B_1, B_2$   $rc^*(G) \leq \frac{3n(B_1)-7}{4} + \frac{3n(B_2)-7}{4} = \frac{3n-11}{4} < \frac{3n-10}{4}$ .
2.  $B_1, K_2, B_2$   $rc^*(G) \leq \frac{3n(B_1 \cup K_2)-6}{4} + \frac{3n(B_2)-7}{4} = \frac{3n-10}{4}$ .
3.  $B_1, B, B_2$   $rc^*(G) \leq \frac{3n(B_1)-7}{4} + \frac{3n(B)-4}{4} + \frac{3n(B_2)-7}{4} = \frac{3n-12}{4} < \frac{3n-10}{4}$ .
4.  $B_1, K_2, B, B_2$   
 $rc^*(G) \leq \frac{3n(B_1 \cup K_2)-6}{4} + \frac{3n(B)-4}{4} + \frac{3n(B_2)-7}{4} = \frac{3n-11}{4} < \frac{3n-10}{4}$ .
5.  $B_1, K_2, B, K_2, B_2$   
 $rc^*(G) \leq \frac{3n(B_1 \cup K_2)-6}{4} + \frac{3n(B)-4}{4} + \frac{3n(B_2 \cup K_2)-6}{4} = \frac{3n-10}{4}$ . □

**Proof of Theorem 2.** If  $G$  has connectivity  $\kappa(G) = 1$ , then  $rc(G) \leq rc^*(G) \leq \frac{3n-10}{4} < \frac{3n-1}{4}$  by Theorem 3. If  $G$  is 2-connected, then  $rc(G) \leq \frac{3n-1}{4}$  by Corollary 1. □

### 3 Concluding Remarks and Open Problems

- The presented proofs provide polynomial time algorithms for edge colourings of graphs such that  $rc^*(G) \leq \frac{3n-10}{4}$  for graphs with connectivity 1 and  $rc(G) \leq \frac{3n-1}{4}$  for 2-connected graphs.
- In 3 the following upper bound for  $rc(G)$  is shown.

**Theorem 4.** *Let  $G$  be a connected graph with  $n$  vertices and minimum degree  $\delta$ . Then,*

$$rc(G) \leq \min\left\{n \frac{\ln \delta}{\delta} (1 + o_\delta(1)), n \frac{4 \ln \delta + 3}{\delta}\right\}.$$

As mentioned in 3, already for  $\delta = 18$ , Theorem 4 gives a better bound 0.81 for  $rc(G)$  than the bound 0.833 from Theorem 1. It is not known how far this bound is from being tight, but in any case it cannot be improved below  $\frac{3n}{\delta+1} - \frac{\delta+7}{\delta+1}$  as there are connected graphs with minimum degree  $\delta$  and this diameter. Very recently Krivelevich and Yuster 6 have shown that a connected graph  $G$  with  $n$  vertices and minimum degree  $\delta$  has  $rc(G) < \frac{20n}{\delta}$ .

- The presented results motivate the following challenging problem.

*Problem 1.* For every  $k \geq 2$  find a minimal constant  $c_k$  with  $0 < c_k \leq 1$  such that  $rc(G) \leq c_k \cdot n$  for all graphs  $G$  with minimum degree  $\delta(G) \geq k$ . Is it true that  $c_k = \frac{3}{k+1}$  for all  $k \geq 2$ ?

This is true for  $k = 2, 3$  as shown before ( $c_2 = 1$  and  $c_3 = \frac{3}{4}$ ).

- The computational complexity of rainbow connectivity has been studied in [2]. It is proved that the computation of  $rc(G)$  is NP-hard. In fact it is already NP-complete to decide if  $rc(G) = 2$ , and in fact it is already NP-complete to decide whether a given edge-coloured (with an unbounded number of colours) graph is rainbow connected.

After the presentation of this paper at the IWOCA workshop the following questions were posed by workshop participants.

- Hajo Broersma  
What happens with the value  $rc(G)$  for graphs with higher connectivity?
- Jan Kratochvíl  
Let the edges of a graph be coloured with a fixed number of colours, say  $k$ . What is the complexity of deciding whether  $G$  is rainbow connected? Is this a FPT problem?  
The answer to the second question is YES as indicated by Travis Gagie.
- Jack Edmonds  
What is the complexity of deciding whether a given edge-coloured graph  $G$  has a rainbow spanning tree?

**Remark.** Zsolt Tuza has told us at the Hereditarnia workshop in Herlany, June 2009, that he has also verified conjecture [1]. However, his proof is still unpublished.

**Acknowledgement.** We thank the referees for some valuable comments.

## References

1. Bondy, J.A., Murty, U.S.R.: Graph Theory. Springer, Heidelberg (2008)
2. Chakraborty, S., Fischer, E., Matsliah, A., Yuster, R.: Hardness and algorithms for rainbow connectivity. In: Proceedings STACS 2009, pp. 243–254 (2009)
3. Caro, Y., Lev, A., Roditty, Y., Tuza, Z., Yuster, R.: On rainbow connection. The Electronic Journal of Combinatorics 15, #57 (2008)
4. Chartrand, G., Johns, G.L., McKeon, K.A., Zhang, P.: Rainbow connection in graphs. Math. Bohemica. 133(1), 85–98 (2008)
5. Dirac, G.A.: Some theorems on abstract graphs. Proc. London Math. Soc. 2, 69–81 (1952)
6. Krivelevich, M., Yuster, R.: The rainbow connection of a graph is (at most) reciprocal to its minimum degree (preprint, 2009)

# The Complexity of Almost Perfect Matchings in Uniform Hypergraphs with High Codegree

Edyta Szymańska\*

Faculty of Mathematics and Computer Science,  
Adam Mickiewicz University,  
Poznań, Poland  
edka@amu.edu.pl

**Abstract.** In this paper we prove that the problem  $\text{APM}(k, r, c)$  of deciding whether a given  $k$ -uniform hypergraph  $H$ , with minimum  $(k - 1)$ -wise vertex degree at least  $c|V(H)|$ , contains a matching missing exactly  $r$  vertices, that is, a set of disjoint edges of size at least  $(|V(H)| - r)/k$ , is NP-complete for  $c < \frac{1}{k}$ , while for  $c > \frac{1}{k}$ , and  $r > 0$  we provide a polynomial time algorithm for the corresponding search problem.

## 1 Introduction

A *hypergraph*  $H = (V, E)$  is a finite set of vertices  $V$  together with a family  $E$  of distinct, nonempty subsets of vertices called edges. In this paper we consider  *$k$ -uniform hypergraph* in which, for a fixed  $k \geq 2$ , each edge is of size  $k$ . A matching in a hypergraph is a set of disjoint edges. The number of edges in a matching  $M$  is called *the size* of the matching, while the number of vertices missing from  $M$ , that is, the number  $|V(H) \setminus V(M)|$  is called *the deficiency* of  $M$  in  $H$ . A matching is perfect if its deficiency is 0, or equivalently if its size is  $|V(H)|/k$ . Hence, a necessary condition for the existence of a perfect matching in  $H$  is that  $|V(H)| \equiv 0 \pmod{k}$ .

### 1.1 Background

For  $k \geq 2$ , by  $\text{PM}(k)$  we denote the problem of deciding whether a  $k$ -uniform hypergraph contains a perfect matching. The problem  $\text{PM}(2)$  is the classical problem of deciding the existence of a perfect matching in a graph, has a complete structural characterization (Tutte's and Hall's theorems), and is known to be in class  $\mathcal{P}$  since the paper by Edmonds [6].

However, like for many other graph problems, the analogous question for hypergraphs turned out to be of a far more complex nature. One reason for that is that there is no known characterization of hypergraphs containing perfect matchings. Also, there are very few sufficient conditions for the presence of a perfect matching in a hypergraph. One such result, proved in [2] (see also [9]),

---

\* Research supported by grant N206 017 32/2452.

can be viewed as an analog of Hall’s theorem. Another, Dirac-type sufficient condition will be described in Sect. 1.2.

As for the complexity, the *k-dimensional matching* problem, known to be NP-hard for every  $k \geq 3$  ([11], [15]) is a special case of  $\text{PM}(k)$ . Moreover, another optimization problem called *exact cover by k-sets*, which is equivalent to  $\text{PM}(k)$ , is known to be NP-complete for  $k \geq 3$  [7].

In fact, the latter problem is a special case of a more general, *set packing* problem, in which a family of sets is given and the task is to find a maximum size sub-family of sets that are pairwise disjoint. A version with the restriction to *k*-element sets is referred to as *k-set packing* and is equivalent to the problem of finding a largest matching in a *k*-uniform hypergraph. The *k*-set packing problem is included in Karp’s list of twenty one NP-complete problems [11].

A special instance of the decision version of the *k*-set packing problem is the following problem of deciding the existence of an almost perfect matching. Given a *k*-uniform hypergraph on *n* vertices, and a nonnegative integer *r*, a matching *M* in *H* of deficiency *r* will be called an *r-deficient matching*. An *r*-deficient matching *M* is thus a set of exactly  $(n - r)/k$  disjoint edges in  $E(H)$ . Clearly, if  $n \not\equiv r \pmod k$  then the presence of an *r*-deficient matching in *H* is impossible.

For integers  $k \geq 2$  and  $r \geq 0$ , let  $\text{APM}(k, r)$  denote the problem of deciding whether a *k*-uniform hypergraph  $H = (V, E)$  contains an *r*-deficient matching. When  $0 < r < k$ ,  $\text{APM}(k, r)$  asks for a matching in *H* which is as perfect as one can get.

### 1.2 Motivation

The main goal of this paper is to study the computational complexity of  $\text{APM}(k, r)$  on subclasses of instances. Our choice of these subclasses has been motivated by two lines of research, the first of which concentrates around the following graph theoretic problem. Given two graphs *F* and *G*, an *F-packing in G* is a collection of vertex-disjoint copies of *F* in *G*. An *F-packing* is called *perfect* if it covers all vertices of *G* (setting  $F = K_2$ , we recover the problem  $\text{PM}(2)$ ). The decision version of this problem is known to be NP-complete if and only if *F* has a component containing at least three vertices [12].

However, things look better for dense instances, in particular, with sufficiently high minimum degree  $\delta(G)$  [3]. In [13], Kühn and Osthus investigated this problem further and discovered an interesting “phase transition”. Namely, they proved algorithmically that for every non-bipartite graph *F* and every  $\gamma > 0$ , there is a constant  $\tau_F > 0$  such that the problem of finding a maximum *F-packing* is polynomially solvable for graphs *G* with  $\delta(G) \geq (\tau_F + \gamma)|V(G)|$ . On the other hand, they showed that for any  $\gamma > 0$  this decision problem becomes NP-complete for the class of graphs *G* with  $\delta(G) \geq (\tau_F - \gamma)|V(G)|$  for infinitely many non-bipartite graphs *F*.

The results of this paper deal, in some sense, with an analogue of the *F*-packing problem for *k*-uniform hypergraphs, where *F* is a single hyperedge. According to our knowledge, there are very few algorithmic results on perfect packings in *k*-uniform hypergraphs. For instance, the structural result from

[9] has been recently turned into a local search algorithm for finding perfect matchings in a class of hypergraphs [4].

For  $F$  other than a single edge, Kühn et al. showed [14] that all  $n$ -vertex, 3-uniform hypergraphs, where  $n$  is divisible by 4, in which every pair of vertices belongs to at least  $n/4 + \epsilon n$  edges, contain a perfect  $\mathcal{C}_4$ -packing, where  $\mathcal{C}_4$  is the 3-uniform hypergraph on four vertices containing two edges. Although no algorithm is mentioned there, it is plausible that their proof can be made algorithmic. Czygrinow and Nagle [5] and, more recently Pikhurko [16] studied almost perfect  $K_4^{(3)}$ -packings of 3-uniform hypergraphs satisfying similar codegree conditions.

The second source of motivation comes from recent investigations of the structural properties of  $k$ -uniform hypergraphs satisfying the so called Dirac-type conditions. Dirac in 1952 proved that the otherwise NP-complete problem of the existence of a Hamilton cycle in a graph becomes trivial for graphs with  $\delta(G) \geq |V(G)|/2$ . For a  $k$ -uniform hypergraph, the graph minimum degree  $\delta(G)$  can be replaced by the *minimum ( $l$ -wise) degree* denoted by  $\delta_l(H)$ , for  $1 \leq l \leq k - 1$ , which is the largest integer  $d$  such that every  $l$ -element set of vertices of  $H$  is contained in at least  $d$  edges of  $H$ . In [17] the authors considered the parameter  $\delta_l(H)$ , for  $l = k - 1$  and proved asymptotically a generalization of Dirac's result to  $k$ -uniform hypergraphs conjectured in [10].

But more importantly for us, recently, Rödl et al. gave in [18] a sufficient condition for the existence of a perfect matching in a  $k$ -uniform hypergraph  $H$  in terms of  $\delta_{k-1}(H)$ . Let  $t(k, l, n)$  be the smallest integer  $t$  such that every  $k$ -uniform hypergraph on  $n$  vertices and with  $\delta_l(H) \geq t$  contains a matching of size  $\lfloor n/k \rfloor$ . For large  $n$  divisible by  $k$ , they completely determined the values of  $t(k, n) := t(k, k - 1, n)$ , which tend to be very close to  $n/2 - k$ . For example, if  $k$  is odd and  $n$  is large and even, then  $t(k, n) = n/2 - k + 2$ .

A slightly weaker result but with a significantly simpler proof was given in [19]. Inspired by the idea from [1], the authors showed that for all  $k \geq 3$  and  $n$  divisible by  $k$ ,  $t(k, n) \leq n/2 + k/4$ . Further results forcing the presence of a perfect matching by a condition on  $\delta_l(H)$  were recently obtained in [8]. They showed that  $t(3, 1, n) = (\frac{5}{9} + o(1)) \binom{n}{2}$  and their result is asymptotically tight. In addition, they upper bounded  $t(k, l, n)$  by  $(\max\{\frac{1}{2}, \frac{k-l}{k}\} + o(1)) \binom{n}{k-l}$  for any  $0 < l < k$ .

In contrast to the divisible case, another result from [18] states that for  $n$  not divisible by  $k$ ,  $t(k, n) \leq n/k + O(\log n)$ , much less than before. This means that the presence of  $r$ -deficient matchings with  $0 < r < k$  is forced by a much weaker codegree condition than in the case  $r = 0$ . This has been a starting point of the research described in this paper.

### 1.3 Results

Motivated by the results in [13], [18] and [8], we will consider the following class of decision problems. Given integers  $k \geq 3$ ,  $0 < l < k$  and  $r \geq 0$  and a real  $c > 0$ , by  $\text{APM}_l(k, r, c)$  we denote a restricted version of  $\text{APM}(k, r)$ , in which we ask whether a  $k$ -uniform hypergraph  $H = (V, E)$  with minimum degree  $\delta_l(H) \geq c \binom{|V(H)|}{k-l}$  contains an  $r$ -deficient matching  $M$ , that is, a matching  $M$



of order  $|V(M)| = |V(H)| - r$ . In the case  $l = k - 1$  we will use a simplified notation  $\text{APM}(k, r, c) := \text{APM}_{k-1}(k, r, c)$ .

An immediate consequence of the results in [18] is that the decision problem  $\text{APM}(k, 0, c)$  is trivial for every  $c > \frac{1}{2}$ , while  $\text{APM}(k, r, c)$ ,  $r > 0$ , is trivial already for  $c > \frac{1}{k}$ .

In this paper we prove the following hardness result which, combined with the results in [18] establishes a “phase transition” at  $1/k$  for  $r \geq 1$ , and leaves a hardness gap of  $(1/k, 1/2)$  for  $r = 0$ .

**Theorem 1.** *For all  $k \geq 3$ ,  $r \geq 0$ , and every constant  $c < \frac{1}{k}$ ,  $\text{APM}(k, r, c)$  is NP-complete.*

Moreover, for  $r > 0$ , we solve the corresponding search problem, by providing a polynomial time algorithm, which finds an  $r$ -deficient matching in every  $k$ -uniform hypergraph  $H$  with  $\delta_{k-1}(H)$  sufficiently large. The case  $r \geq (k - 2)k$  (say, large deficiency) is slightly easier.

**Proposition 2.** *For every  $k \geq 3$  and  $r \geq (k - 2)k$  there exists an algorithm, called  $\text{LDMATCHING}$ , which in every  $k$ -uniform hypergraph on  $n \geq k$  vertices, where  $n \equiv r \pmod{k}$  and  $\delta_{k-1}(H) \geq (n - r)/k$ , finds an  $r$ -deficient matching in  $H$  in time  $O(n^2)$ .*

The remaining positive values of  $r$  (small deficiency) are covered by our next result.

**Theorem 3.** *For every  $k \geq 3$  and  $0 < r < (k - 2)k$  there exist constants  $C$  and  $n_0$ , and an algorithm, called  $\text{SDMATCHING}$ , which in every  $k$ -uniform hypergraph on  $n \geq n_0$  vertices, where  $n \equiv r \pmod{k}$  and  $\delta_{k-1}(H) \geq \frac{n}{k} + C \log n$  finds an  $r$ -deficient matching  $M$  in  $H$ , in time  $O(\log n \cdot n^{k^2 - k + 1})$ .*

Finally, let us mention that although the proofs in [18] and [19], as described there, are not constructive, they can be turned into efficient algorithms finding perfect matchings under respective degree conditions. In particular, the proof in [19] yields in a straightforward way a polynomial time algorithm which constructs a perfect matching in every  $k$ -uniform hypergraph  $H$ , with  $|V(H)| = n \equiv 0 \pmod{k}$  and  $\delta_{k-1}(H) \geq n/2 + k/4$ .

Also, the key concepts of the proof of the divisible case in [18] can be adopted to yield deterministic, polynomial time procedures, in a similar but more complex way than in the proof of Theorem 3. For completeness, we include this result here without proof. Recall that the value of  $t(k, n)$  determined in [18] is close to, but always less than  $n/2$ .

**Theorem 4.** *For every  $k \geq 3$  there exists constant  $n_0$ , and an algorithm, called  $\text{PMATCHING}$ , which in every  $k$ -uniform hypergraph on  $n \geq n_0$  vertices with  $n$  divisible by  $k$  and  $\delta_{k-1}(H) \geq t(k, n)$  finds a perfect matching  $M$  in  $H$  in time  $O(\log^4 n \cdot n^{k^2 + 2k})$ .*

All the above mentioned results are summarized in Table 1.

Finally, it is worth mentioning that our method yields a more general hardness result.

**Table 1.** The complexity of  $\text{APM}(k, r, c)$  with  $k \geq 3$ . For every *trivial* problem there exists a *polynomial* time algorithm finding an  $r$ -deficient matching.

	$c < \frac{1}{k}$	$c = \frac{1}{k}$	$c \in (\frac{1}{k}, \frac{1}{2}]$	$c > \frac{1}{2}$
$r \geq (k - 2)k$	NP-complete	trivial	trivial	trivial
$0 < r < (k - 2)k$	NP-complete	?	trivial	trivial
$r = 0$	NP-complete	?	?	trivial

**Theorem 5.** For all  $k \geq 3, 0 < l < k, r \geq 0$ , and every constant  $c < 1 - (\frac{k-1}{k})^{k-l}$ ,  $\text{APM}_l(k, r, c)$  is NP-complete.

For  $k = 3, \ell = 1$  and  $r = 0$  the above result, combined with the result of [8], has the following consequence.

**Corollary 6.**  $\text{APM}_1(3, 0, c)$  is  $\begin{cases} \text{trivial for } c > \frac{5}{9}, \\ \text{NP-complete for } c < \frac{5}{9}. \end{cases}$

In Sect. 2 we prove the hardness result (Theorem 1) and sketch the proof of Theorem 5, while Sect. 3 contains the description and analysis of the polynomial algorithms  $\text{LDMATCHING}$  from Proposition 2 and  $\text{SDMATCHING}$  from Theorem 3.

## 2 Proof of Hardness

In this section we prove the hardness result for the  $\text{APM}(k, r, c)$  problem stated in Theorem 1.

*Proof (of Theorem 7).* It is easy to see that  $\text{APM}(k, r, c) \in \text{NP}$ . The theorem will be shown by a reduction from the  $\text{PM}(k)$  problem, which is known to be NP-complete (see Sect. 1.1).

Given  $k, r$ , and  $c < 1/k$ , for every hypergraph  $H$ , which is an instance of  $\text{PM}(k)$ , we will construct a hypergraph  $H'$ , an instance of the  $\text{APM}(k, r, c)$  problem (a gadget), in such a way that  $H$  has a perfect matching if and only if  $H'$  has an  $r$ -deficient matching. The starting point for our construction is a critical hypergraph  $H_0$  (see Fig. 1(a)).

Put  $\gamma = \frac{1}{k} - c$  and note that  $\gamma > 0$ . Setting  $n = |V(H)|$ , let  $H_0 = H_0(k, r, n, \gamma)$  be a  $k$ -uniform hypergraph, in which the vertex set is the union of two disjoint sets  $A \cup B$ , such that  $|A| = \lceil \frac{n}{\gamma} \rceil$  and  $|B| = (k - 1)|A| + r$ . The edge set of  $H_0$  consists of all  $k$ -element subsets of  $A \cup B$  which have a non-empty intersection with the set  $A$ . It is easy to see that  $\delta_{k-1}(H_0) = |A| = \lceil \frac{n}{\gamma} \rceil$  and that  $H_0$  has an  $r$ -deficient matching.

The hypergraph  $H'$  is obtained by taking a vertex-disjoint union of  $H$  and  $H_0$  with some additional edges (see Fig. 1(b)). More precisely, let  $V(H') = V' =$

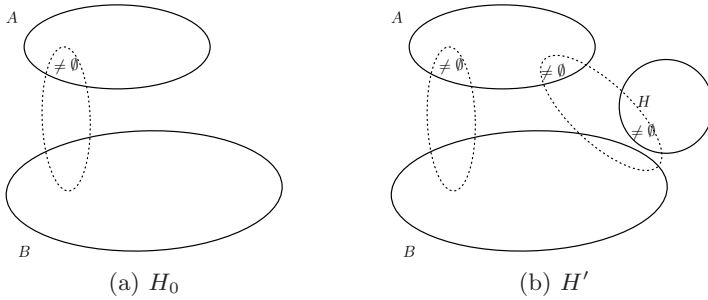


Fig. 1. Gadget construction

$A \cup B \cup V(H)$  and  $E(H') = E(H_0) \cup E(H) \cup E^*$ , where  $E^*$  is the set of all  $k$ -element subsets  $e$  of  $V'$  such that  $e \cap V(H) \neq \emptyset$  and  $e \cap A \neq \emptyset$ . This means that for every  $e \in E^*$  we have  $|e \cap B| \leq k - 2$ .

First, we will show that  $H'$  is indeed an instance of  $\text{APM}(k, r, c)$ , that is, that  $\delta_{k-1}(H') \geq c|V'|$ . To see this, assume that  $S$  is an arbitrary  $(k - 1)$ -element subset of  $V'$ . Consider the following cases with respect to the location of  $S$ :

- i)  $S \cap A \neq \emptyset$ . Then  $S \cup \{v\}$  is an edge of  $H'$  for every  $v \notin S$ , and thus  $\text{deg}_{H'}(S) = |V'| - k + 1$ .
- ii)  $S \cap A = \emptyset$ . Then  $S \cup \{v\}$  is an edge of  $H'$  only for  $v \in A$ , and, if  $S \subset V(H)$ , also for all  $v \in V(H)$  which form an edge in  $H$ . Thus, setting  $\text{deg}_H(S) = 0$  if  $S \not\subset V(H)$ , we have  $\text{deg}_{H'}(S) = |A| + \text{deg}_H(S) \geq |A|$ , and this bound is achieved by several sets  $S$ .

Consequently,

$$\delta_{k-1}(H') = |A| = \left\lceil \frac{n}{\gamma} \right\rceil \geq c \left( k \left\lceil \frac{n}{\gamma} \right\rceil + n + r \right)$$

because, recalling that  $c = \frac{1}{k} - \gamma$ , this inequality is equivalent to

$$\frac{n + r}{k} \leq \gamma k \left\lceil \frac{n}{\gamma} \right\rceil + \gamma(n + r) \tag{1}$$

and the right-hand-side of (1) is greater than  $kn$ , which, in turn, is greater than  $(n + r)/k$  with a big margin.

It remains to show that  $H$  has a perfect matching if and only if  $H'$  has an  $r$ -deficient matching. First note that if  $H$  has a perfect matching then this perfect matching, together with an  $r$ -deficient matching of  $H_0$  form an  $r$ -deficient matching of  $H'$ . Now suppose that  $H$  does not have a perfect matching, but  $H'$  has an  $r$ -deficient matching  $\mathcal{M}$ .

Let us split  $\mathcal{M} = \mathcal{M}_H \cup \mathcal{M}_0 \cup \mathcal{M}^*$ , where  $\mathcal{M}_H = \mathcal{M} \cap H$ ,  $\mathcal{M}_0 = \mathcal{M} \cap H_0$ , and  $\mathcal{M}^* = \mathcal{M} \cap E^*$ . By our assumption on  $H$ ,  $\mathcal{M}^* \neq \emptyset$ . Recall that for each edge  $e \in \mathcal{M}^*$ , we have  $e \cap A \neq \emptyset$  and  $e \cap V(H) \neq \emptyset$  and set  $|\mathcal{M}^*| = x$ .

Let  $A' = A \setminus V(\mathcal{M}^*)$  and  $B' = B \setminus V(\mathcal{M}^*)$ . Then  $|A'| \leq |A| - x$  and  $|B'| \geq |B| - x(k - 2)$ . Since  $x > 0$ , this implies that  $|B'| - (k - 1)|A'| > r$ . Note that  $V(\mathcal{M}_0) \subseteq A' \cup B'$ . Since the largest matching in the subhypergraph  $H_0[A', B']$  has size  $|A'|$ , its deficiency in  $H_0[A', B']$  is precisely  $|B'| - (k - 1)|A'| > r$ . So is the case of  $\mathcal{M}_0$ , and, since  $V(\mathcal{M}_H) \subseteq V(H)$ , this is a contradiction with the assumption that  $\mathcal{M}$  is  $r$ -deficient.  $\square$

The above construction does not work for  $c = \frac{1}{k}$ . Except for that, in view of the results in [18], it is optimal for the case  $r > 0$ , but we have a hardness gap for the case  $r = 0$ , in which the decision problem  $\text{APM}(k, 0, c)$  becomes trivial only for  $c \geq 1/2$ . To close this gap, one would need to come up with a gadget which is very sensitive to the parity of both  $n$  and  $k$  (cf. the critical hypergraphs given in [18] for the divisible case).

However, the above proof can be generalized to yield Theorem 5.

*Proof (of Theorem 5).* It is again a reduction from the  $\text{PM}(k)$  problem with the same gadget and a different parameter  $\gamma = 1 - (\frac{k-1}{k})^{k-l} - c$ .  $\square$

### 3 The Search Algorithm

In this section we will describe two polynomial time algorithms  $\text{LDMATCHING}$  and  $\text{SDMATCHING}$  which find  $r$ -deficient matchings in hypergraphs with the minimum codegree as specified in Proposition 2 and Theorem 3, respectively.

#### 3.1 Large Deficiency

Here we prove Proposition 2 and make some preparations toward the proof of Theorem 3. Proposition 2 will follow immediately from the next fact, whose existential version was proven in [18]. We do not describe the algorithm  $\text{LDMATCHING}$  explicitly.

**Fact 7.** *Let  $n \geq k \geq 2$ . For every  $k$ -uniform hypergraph  $H$  on  $n$  vertices, there exists a matching  $M$  of size at least*

$$\min \left\{ \left\lfloor \frac{n}{k} \right\rfloor - k + 2, \quad \delta_{k-1}(H) \right\}$$

*which can be constructed in time  $O(n^2)$ .*

*Proof.* We will build  $M$  by adding edges stepwise. Set  $|M| = m$  and suppose that  $m \leq \delta_{k-1}(H) - 1$  and  $m \leq \lfloor \frac{n}{k} \rfloor - k + 1$ . We will next extend  $M$  to size  $\lfloor \frac{n}{k} \rfloor - k + 2$  by the following augmenting procedure. Note that there are at least  $(k - 1)k$  vertices outside  $M$ . Arbitrarily select disjoint sets  $S_1, S_2, \dots, S_k$  of size  $|S_i| = k - 1$  from  $V(H) \setminus V(M)$ . If any of the sets  $S_i$  forms an edge  $e$  with a vertex  $v \in V(H) \setminus V(M) \setminus \bigcup_{i=1}^k S_i$  then add  $e$  to  $M$ . After exhausting all such edges consider an auxiliary bipartite graph  $B$  with a vertex set  $X \cup Y$ , where  $X$  consists of vertices  $u_j$  representing the edges  $e_j$  of  $M$ ,  $j = 1, \dots, m'$ ,  $m' \geq m$ , and  $Y$  consists of vertices  $s_i$  representing the sets  $S_i$ ,  $i = 1, \dots, k'$  with  $k' \leq k$ . There

is an edge  $(u_j, s_i)$  in  $B$  if there is a vertex  $v \in e_j$  such that  $S_i \cup \{v\} \in E(H)$ . Since each set  $S_i$  has all its neighbors in  $V(M)$  (other edges were already added to  $M$ ), the degree of  $s_i$  in  $B$ ,  $deg_B(s_i)$ , is at least  $\delta_{k-1}(H)$ . Thus, the total number of edges in  $B$  is at least  $k\delta_{k-1}(H)$ , and, by averaging and the assumption that  $m' \leq m < \delta_{k-1}(H)$ , we can find a vertex  $u_j \in Y$  such that  $deg_B(u_j) \geq k + 1$ . Constructing the graph  $B$  and finding the vertex  $u_j$  takes  $O(n)$  steps.

The choice of  $u_j$  guarantees that the  $k$  vertices of the edge  $e_j \in M$  form at least  $k + 1$  edges with the sets  $S_1, \dots, S_k$ . This, in turn, implies that  $e_j$  contains two distinct vertices  $v, v'$  such that two sets  $S'$  and  $S''$  from among  $\{S_1, \dots, S_k\}$  form with, respectively,  $v'$  and  $v''$ , two disjoint edges  $e'$  and  $e''$ . Note that  $V(M) \cap e' = e_j \cap e' = \{v'\}$ ,  $V(M) \cap e'' = e_j \cap e'' = \{v''\}$ . Hence,  $e', e_j, e''$  forms an  $M$ -alternating hyperpath and  $M$  can be enlarged by swapping  $e_j$  for  $e'$  and  $e''$ , that is, by resetting  $M := (M \setminus \{e_j\}) \cup \{e', e''\}$ . Obviously, the edges  $e'$  and  $e''$  can be found in constant time. Iterating this procedure at most  $n/k$  times, we extend the initial matching  $M$  to eventually have size  $\lfloor \frac{n}{k} \rfloor - k + 2$  edges. □

Note that as the threshold for  $\delta_{k-1}(H)$  in Theorem 3 does not depend on  $r$ , we may assume that  $0 < r \leq k$ , because if  $r_1 < r_2$  and  $r_i \equiv n \pmod k$ ,  $i = 1, 2$ , then an  $r_1$ -deficient matching contains an  $r_2$ -deficient matching.

Moreover, we can reduce the range of  $r$  even further to the single case  $r = 1$ . Indeed, suppose that algorithm SDMATCHING on every hypergraph  $H'$  with  $n' \equiv 1 \pmod k$  and  $\delta_{k-1}(H') \geq \frac{n'}{k} + C \log n'$  returns a 1-deficient matching in  $H'$ . Now consider an input  $H$  with  $n \equiv r \pmod k$ ,  $2 \leq r \leq k$  and  $\delta_{k-1}(H) \geq \frac{n}{k} + C \log n + r - 1$ , remove  $r - 1$  vertices of  $H$  obtaining a hypergraph  $H'$  with  $n' = n - r + 1$  vertices, where  $n' \equiv 1 \pmod k$ . Then,  $\delta_{k-1}(H') \geq \frac{n}{k} + C \log n > \frac{n'}{k} + C \log n'$ , and algorithm SDMATCHING returns a 1-deficient matching in  $H'$  which is, of course, an  $r$ -deficient matching in  $H$ . So, assume from now on that  $n \equiv 1 \pmod k$  and  $\delta_{k-1}(H) \geq \frac{n}{k} + C \log n$ , and that we are after a 1-deficient matching in  $H$ .

For  $n \equiv 1 \pmod k$  we have the following consequence of Fact 7.

**Corollary 8.** *If  $\delta_{k-1}(H) \geq \frac{n}{k}$ , then algorithm LDMATCHING finds a  $[(k - 2)k + 1]$ -deficient matching  $M$  in  $H$ .*

In order to reduce this deficiency to 1, that is, to enlarge the size of that matching by  $k - 2$ , a significant effort is required. Following the proof in [18], we initiate the construction by preparing a special small matching which will be able to absorb all but one vertex left after the greedy procedure given in Fact 7. This special matching will be called *powerful* and its building blocks will be called *absorbing edges*.

In the following subsection we will first define absorbing edges and show how to construct powerful matchings. Then, in the final subsection we will present the SDMATCHING algorithm.

### 3.2 Absorbing Edges and Powerful Matchings

Let  $H$  be a  $k$ -uniform hypergraph on  $n$  vertices. A matching of size  $\ell$  in  $H$  will be called an  $\ell$ -matching. Our absorbing technique benefits from the fact that as long as the matching under construction still needs to be enlarged, there are at least  $k + 1$  loose vertices available. (This is not the case when  $r = 0$ , which makes the proof of Theorem 4 so much longer and more technical.)

**Definition 9 (absorbing edge).** *Given a set  $S$  of  $k + 1$  vertices, an edge  $e \in H$  is called  $S$ -absorbing if there are two disjoint edges  $e'$  and  $e''$  in  $H$  such that  $|e' \cap S| = k - 1$ ,  $|e' \cap e| = 1$ ,  $|e'' \cap S| = 2$  and  $|e'' \cap e| = k - 2$ .*

Clearly, if the set  $S$  is outside a matching  $M$  which contains an  $S$ -absorbing edge  $e$ , then  $M$  can “absorb”  $S$  by swapping  $e$  for  $e'$  and  $e''$  (one vertex of  $e$  will become unmatched).

The key feature of the absorbing edge is that there are many of them for every set  $S$  in the input hypergraph  $H$ .

**Fact 10.** *There exists a constant  $c_k > 0$  such that for every  $(k + 1)$ -element set  $S$  of vertices of  $H$  there are at least  $c_k n^k$  of  $S$ -absorbing edges  $e$  in  $H$ .*

*Proof.* Fix  $S$ . Any initial  $k - 3$  vertices of an  $S$ -absorbing edge  $e$  can be chosen arbitrarily, and thus in  $\Omega(n^{k-3})$  ways. The last three vertices, each must close an edge with a specified set of  $k - 1$  vertices. Hence, the number of choices of each such vertex is  $\delta_{k-1} - O(1)$ . Altogether, there are  $\Omega(\delta_{k-1}^3 n^{k-3}) = \Omega(n^k)$  choices of an  $S$ -absorbing edge  $e$ . □

For the absorbing technique to work, we need a small matching  $M$  containing enough absorbing edges for each set  $S$ . It will be then altered in the absorbing procedure, extending any sufficiently large matching so that it becomes 1-deficient. We call such a matching  $M$  *powerful* and define it formally below.

**Definition 11.** *A matching  $M$  in  $H$  with  $n \not\equiv 0 \pmod k$  is called powerful if for every set  $S \subset V$  of size  $k + 1$  the number of  $S$ -absorbing edges in  $M$  is at least  $k - 2$ .*

The existence and construction of small powerful matchings in hypergraphs having large degree is implied by the following, quite general statement. The generality of this statement allows us to treat the divisible case too, in which the absorbing devices are much more complicated.

**Proposition 12.** *Let  $\ell$  and  $k$  be positive integers, let  $m = m(n)$  be a polynomial function of  $n$ , and let  $\alpha = \alpha(n)$  such that  $\frac{1}{\alpha} = o\left(\frac{\sqrt{n}}{\log n}\right)$ . There exist constants  $n_0 > 0$  and  $C' > 0$  such that the following holds. Let  $H$  be a  $k$ -uniform hypergraph with  $n \geq n_0$  vertices, and let  $\mathcal{F}_1, \dots, \mathcal{F}_m$  be families of  $\ell$ -matchings in  $H$  of sizes  $|\mathcal{F}_i| \geq \alpha n^{\ell k}$ ,  $i = 1, \dots, m$ . Then one can construct in polynomial time a matching  $M'$  in  $H$  of size  $|M'| \leq C'(\log n)/\alpha$  such that for every  $i = 1, \dots, m$  there is an  $\ell$ -matching in  $M'$  which belongs to  $\mathcal{F}_i$ .*

The original proof of an existential version of the above proposition in [18] was probabilistic, but for the purpose of this paper we will give a deterministic argument which can be translated into a  $O(\frac{\log n}{\alpha} n^{k\ell} m)$ -time algorithm.

*Proof.* Given the families of  $\ell$ -matchings, the goal is to construct a matching  $M' = \bigcup_{q=1}^s M_q$ ,  $s \leq \frac{1}{\ell} C' \frac{\log n}{\alpha}$ , which contains an  $\ell$ -matching from each family. This can be accomplished by choosing disjoint  $\ell$ -matchings  $M_q$  sequentially in such a way that  $\forall i = 1, \dots, m \exists 1 < j < s \mathcal{F}_i \ni M_j$ .

Since the number of  $\ell$ -matchings in  $H$  is at most  $n^{\ell k}$ , while each  $\mathcal{F}_i$  contains at least  $\alpha n^{\ell k}$  of them, there is an  $\ell$ -matching  $M_1$  contained in at least  $\alpha m$  families  $\mathcal{F}_i$ .

Suppose that  $\ell$ -matchings  $M_1, \dots, M_q$ ,  $q \geq 1$ , with all  $q\ell$  edges pairwise disjoint have been already selected in such a way that

$$m_q \leq (1 - \alpha/2)^q m$$

families  $\mathcal{F}_i$  contain none of  $M_1, \dots, M_q$ . Without loss of generality we assume for convenience that these are  $\mathcal{F}_1, \dots, \mathcal{F}_{m_q}$ .

Since  $M_1, \dots, M_q$  contain together  $\ell k q$  vertices of  $H$ , there are at most  $\ell k q n^{\ell k - 1}$  of  $\ell$ -matchings  $M$  in  $H$  intersecting the vertex set of at least one of  $M_1, \dots, M_q$ . We discard these matchings  $M$  and observe that, by our assumption on  $\alpha$ , each  $\mathcal{F}_i$  still contains at least

$$\alpha n^{\ell k} - \ell k q n^{\ell k - 1} \geq \frac{1}{2} \alpha n^{\ell k}$$

remaining  $\ell$ -matchings in  $H$ . In fact, here we need only to assume that  $\frac{1}{\alpha} = o\left(\sqrt{\frac{n}{\log n}}\right)$ .

Hence, as before, there is an  $\ell$ -matching  $M_{q+1}$  which is contained in at least  $\frac{1}{2} \alpha m_q$  families  $\mathcal{F}_i$  with  $1 \leq i \leq m_q$ . Consequently, there are only at most

$$(1 - \alpha/2) m_q \leq (1 - \alpha/2)^{q+1} m$$

families  $\mathcal{F}_i$  which do not contain any of  $M_1, \dots, M_{q+1}$ .

This process terminates in at most  $s$  steps, where

$$(1 - \alpha/2)^s m \leq e^{-\alpha s/2} m < 1,$$

which implies that  $s \leq 2 \log m / \alpha \leq \frac{1}{\ell} C' (\log n) / \alpha$ , for a suitably chosen  $C'$ .  $\square$

To obtain from Proposition [2] a powerful matching, number all  $(k + 1)$ -element sets in  $V(H)$  by consecutive integers,  $S_1 \dots, S_m$ , and consider the following choice of parameters:  $\ell = k - 2$ ,  $m = \binom{n}{k+1}$ ,  $\alpha = \frac{1}{2} c_k^{k-2}$ , where  $c_k$  has been defined in Fact [10] and for every  $i$ , define  $\mathcal{F}_i$  as the family of all  $(k - 2)$ -matchings consisting exclusively of  $S_i$ -absorbing edges. Note that, by Fact [10] for every  $i = 1, \dots, m$ , there are at least  $c_k n^k$  of  $S_i$ -absorbing edges, one can choose a  $(k - 2)$ -matching out of them in at least  $c^{k-2} n^{(k-2)k} (1 - O(1/n))$  ways. Thus, indeed,

$$|\mathcal{F}_i| \geq \frac{1}{2} c_k^{k-2} n^{(k-2)k} = \alpha n^{(k-2)k}.$$

Also, we have

$$|M'| \leq \frac{C'}{\alpha} \log n = \frac{2C'}{c_k^{k-2}} \log n := C'' \log n.$$

Note that, with the values of parameters specified above, the procedure finding a small powerful matching takes  $O(\log n \cdot n^{k^2+2k})$  steps.

### 3.3 The SDMatching Algorithm

Having defined the absorbing edges and the powerful matching, we are ready to formulate the algorithm. In short, the idea is the following. First, we find a powerful matching  $M'$  of a small size. This is done via the construction described in Proposition 12. At the same time, the absorbing edges contained in  $M'$  can be stored for further use. Since  $|M'| \leq C'' \log n$ , after removing it from  $H$  we are left with a hypergraph  $H'$ , still of a large degree. Indeed, with the choice of  $C = kC''$ , by the assumption that  $\delta_{k-1}(H) \geq n/k + C \log n$ , we have  $\delta(H') \geq \delta(H) - C''k \log n \geq n/k > |V(H')|/k$ .

This allows us to apply the greedy procedure from Corollary 8 and find a matching  $M''$  in  $H'$  covering all but  $(k - 2)k + 1$  vertices of  $H'$ , and thus, the matching  $M' \cup M''$  has deficiency  $(k - 2)k + 1$  in  $H$ . At the end, we absorb  $(k - 2)k$  of the remaining vertices by considering sets of  $k + 1$  vertices stepwise, in any order, and incorporate them into the current matching using absorbing edges from  $M'$ . Since this loop (cf. Step 5 of the algorithm) is performed  $k - 2$  times, we need  $k - 2$  of  $S$  absorbing edges for any set  $S$ , although a set is absorbed at most once. Indeed, in the worst scenario, a set  $S_0$  could be the last one to incorporate, and some  $k - 3$  of the  $S_0$ -absorbing edges could have been already used in earlier rounds, when absorbing other sets  $S$ .

ALGORITHM SDMATCHING

**In:** a hypergraph  $H$  with  $\delta_{k-1}(H) \geq \frac{n}{k} + kC'' \log n$  and  $n \geq n_0$ ,  $n \equiv 1 \pmod{k}$ .

**Out:** a 1-deficient matching  $M$

1. Find a powerful matching  $M'$  (as in Definition 11) using Proposition 12.
2.  $H' := H - V(M')$ , (notice that  $\delta_{k-1}(H') \geq \frac{|V(H')|}{k}$ )
3. Greedily find a  $[(k - 2)k + 1]$ -deficient matching  $M''$  in  $H'$  using procedure LDMATCHING from Corollary 8.
4.  $T := V(H) \setminus V(M' \cup M'')$
5. Repeat until  $|T| = 1$  ( $(k - 2)$  iterations)
  - (a) absorb a set  $S \subseteq T$ , such that  $|S| = k + 1$  using one  $S$ -absorbing edge  $e \in M'$  to get  $M'_S$
  - (b)  $M' := M'_S$
  - (c)  $T := V(H) \setminus V(M' \cup M'')$
6. Return  $M = M' \cup M''$ .

The running time of the above procedure is dominated by Step 1 and equals  $O(\log n \cdot n^{k^2-k+1})$ .



## References

1. Aharoni, R., Georgakopoulos, A., Sprüssel, P.: Perfect matchings in  $r$ -partite  $r$ -graphs. *European. J. Combin.* 30(1), 39–42 (2009)
2. Aharoni, R., Haxell, P.: Hall's theorem for hypergraphs. *J. Graph Theory* 35, 83–88 (2000)
3. Alon, N., Yuster, R.:  $H$ -Factors in Dense Graphs. *J. Combin. Theory, Ser. B* 66(2), 269–282 (1996)
4. Asadpour, A., Feige, U., Saberi, A.: Santa Claus Meets Hypergraph Matchings. In: Goel, A., Jansen, K., Rolim, J.D.P., Rubinfeld, R. (eds.) APPROX and RANDOM 2008. LNCS, vol. 5171, pp. 10–20. Springer, Heidelberg (2008)
5. Czygrinow, A., Nagle, B.: A note on a codegree problem for hypergraphs. *Bulletin of ICA* 32, 63–69 (2001)
6. Edmonds, J.: Paths, trees and flowers. *Canad J. Math.* 17, 449–467 (1965)
7. Garey, M.R., Johnson, D.S.: Computers and intractability. Freeman, New York (1979)
8. Han, H., Person, Y., Schacht, M.: On perfect matchings in uniform hypergraphs with large minimum vertex degree. *SIAM J. Discrete Math.* 23(2), 732–748 (2009)
9. Haxell, P.E.: A Condition for Matchability in Hypergraphs. *Graphs and Combinatorics* 11, 245–248 (1995)
10. Katona, G.Y., Kierstead, H.A.: Hamiltonian chains in hypergraphs. *J. Graph Theory* 30, 205–212 (1999)
11. Karp, R.M.: Reducibility among combinatorial problems. *Complexity of Computer Computations*, 83–103 (1972)
12. Kirkpatrick, D.G., Hell, P.: On the Complexity of General Graph Factor Problems. *SIAM J. Comput.* 12(3), 601–609 (1983)
13. Kühn, D., Osthus, D.: Critical chromatic number and the complexity of perfect packings in graphs. In: 17th ACM-SIAM SODA 2006, pp. 851–859 (2006)
14. Kühn, D., Osthus, D.: Loose Hamilton cycles in 3-uniform hypergraphs of high minimum degree. *J. Combin. Theory, Ser. B* 96(6), 767–821 (2006)
15. Papadimitriou, C.: Computational Complexity. Addison-Wesley, Reading (1994)
16. Pikhurko, O.: Perfect matchings and  $K_4^3$ -tilings in hypergraphs of large codegree. *Graphs Combin.* 24(4), 391–404 (2008)
17. Rödl, V., Ruciński, A., Szemerédi, E.: Dirac's theorem for 3-uniform hypergraphs. *Combin. Probab. Comput.* 15(1-2), 229–251 (2006)
18. Rödl, V., Ruciński, A., Szemerédi, E.: Perfect matchings in large uniform hypergraphs with large minimum collective degree. *J. Combin. Theory, Ser. A* 116, 613–636 (2009)
19. Rödl, V., Ruciński, A., Schacht, M., Szemerédi, E.: A note on perfect matchings in uniform hypergraphs with large minimum collective degree. *Commentationes Mathematicae Universitatis Carolinae* 49(4), 633–636 (2008)

# Computability of Width of Submodular Partition Functions

Petr Škoda\*

Department of Applied Mathematics,  
Faculty of Mathematics and Physics, Charles University,  
Malostranské nám. 25, 118 00 Prague, Czech Republic  
peskoj@kam.mff.cuni.cz

**Abstract.** The notion of submodular partition functions generalizes many of well-known tree decompositions of graphs. For fixed  $k$ , there are polynomial-time algorithms to determine whether a graph has tree-width, branch-width, etc. at most  $k$ . Contrary to these results, we show that there is no sub-exponential algorithm for determining whether the width of a given submodular partition function is at most two. In addition, we also develop another dual notion for submodular partition functions which is analogous to loose tangles for connectivity functions.

## 1 Introduction

Graph decompositions and width-parameters play a very important role in algorithmic graph theory (as well as structural graph theory). The most well-known and studied notions include the tree-width, branch-width and clique-width of graphs. The importance of these notions lie in the fact that many NP-complete problems can be decided for classes of graphs of bounded tree-/branch-width in polynomial time. A classical result of Courcelle [5] and Arnborg, Lagergren, and Seese [2] asserts that every problem expressible in the monadic second-order logic can be decided in linear time for the class of graphs with bounded tree-/branch-width. An analogous result for matroids with bounded branch-width representable over finite fields have been established by Hliněný (see [6] and [7]) and generalized using a more specialized notion of width to all matroids by Král' [9].

Most of the algorithms for classes of graphs of bounded width require a decomposition of an input graph as part of input. Fortunately, optimal tree-decompositions of graphs can be computed in linear time [3] if the width is fixed and there are even simple efficient approximation algorithms [4]. For branch-width, Oum and Seymour [10] recently established that the branch-decompositions of a fixed width of graphs and matroids can be computed in polynomial-time (or decided that they do not exist). Their algorithm actually deals with a more general notion of connectivity functions which are given by an oracle.

---

\* This research was partially supported by the grant GACR 201/09/0197.

A fixed-parameter algorithm for computing optimal branch-decompositions for matroids represented over finite fields was designed by Hliněný and Oum [8].

In this paper, we study submodular partition functions introduced by Amini et al. [1]. This general notion includes both graph tree-width and branch-width as special cases. We postpone the formal definition to Sect. 2. In their paper, Amini et al. [1] presented a duality theorem that implies the known duality theorems for graph tree-width and graph/matroid branch-width of Robertson and Seymour [11].

Since the duality, an essential ingredient for some of the known algorithms for computing decompositions of small width, smoothly translates to this general setting, it is natural to ask whether decompositions of submodular partition functions with fixed width can be computed in polynomial-time. In this paper, we show that such an algorithm cannot be designed in general. In particular, we present an argument that every algorithm deciding whether a partition width of an  $n$ -element set is at most two must ask an oracle the number of queries exponential in  $n$ . On a positive side, we were able to develop a notion of loose tangles, a key ingredient of the algorithm of Oum and Seymour [10], for this more general concept which we hope to be of some use to design algorithms for special classes of submodular partition functions.

## 2 Notation

In this section, we introduce the notation and concepts used in this paper. A function  $f : 2^E \rightarrow \mathbb{N}$  for a finite set  $E$  is said to be *submodular* if the following holds for every pair of subsets  $X, Y \subseteq E$ :

$$f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y). \tag{1}$$

A submodular function  $f$  is *symmetric* if  $f(X) = f(\overline{X})$ , for all subsets  $X$  of  $E$ . Finally, a *connectivity function* is a submodular function that is symmetric and  $f(\emptyset) = 0$ .

For a connectivity function  $f$  on a ground set  $E$ , a *branch-decomposition* of  $f$  is a pair  $(T, \sigma)$  where  $T$  is a *ternary tree* and  $\sigma$  is a bijection between the set of leaves of  $T$  and  $E$ . Every edge  $e$  of  $T$  naturally defines a bipartition  $(A_e, \overline{A_e})$  of the ground set  $E$ , i.e.,  $A_e$  consists of all elements that corresponds to leaves of  $T$  in one of the two components of  $T \setminus e$ . The *order* of an edge  $e$  of  $T$  is the value  $f(A_e)$  and the *width* of a branch-decomposition  $(T, \sigma)$  is the maximum order of an edge of  $T$ . The *branch-width* of  $f$  is the minimum width of all branch-decompositions of  $f$ . This notion includes the notion of the usual branch-width of graphs and matroids.

There is a dual object to branch-decompositions called a *tangle*, introduced by Robertson and Seymour [11]. A set  $\mathcal{T}$  of subsets of  $E$  is called an  $f$ -tangle of *order*  $k + 1$  if  $\mathcal{T}$  satisfies the following three axioms:

- (T1) For all  $A \subseteq E$ , if  $f(A) \leq k$ , then either  $A \in \mathcal{T}$  or  $\overline{A} \in \mathcal{T}$ .
- (T2) If  $A, B, C \in \mathcal{T}$ , then  $A \cup B \cup C \neq E$ .

(T3) For all  $e \in E$ , we have  $E \setminus \{e\} \notin \mathcal{T}$ .

Robertson and Seymour [11] proved the following duality theorem between branch-decompositions and tangles.

**Theorem 1 (Robertson and Seymour [11]).** *Let  $f$  be a connectivity function on a ground set  $E$ . There is no  $f$ -tangle of order  $k + 1$  if and only if the branch-width of  $f$  is at most  $k$ .*

We now introduce the concept of submodular partition functions that provides a unified view on branch-decompositions of connectivity functions and tree-decompositions of graphs. Throughout the paper, Greek letters will be used for collections of subsets, i.e.,  $\alpha$  can stand for a collection  $A_1, \dots, A_k$  of subsets of a set  $E$ . Note, that the sets in a collection are not ordered in any way and a set can occur more than once in a collection. The collection  $\alpha$  is a *partition* if the sets  $A_i$  are mutually disjoint and their union is the whole set  $E$ .

There are shorthands for operations with collections of subsets we want to use: if  $\alpha$  is such a collection  $A_1, \dots, A_k$  and  $A$  is another subset, then  $\alpha \cap A$  stands for the collection  $A_1 \cap A, \dots, A_k \cap A$ . We use  $\alpha \setminus A$  in a similar way. Finally,  $[A, \alpha]$  stands for the collection obtained from  $\alpha$  by inserting  $A$  to the collection. Note that empty sets are allowed in the collections.

A *partition function* is a function from the set of all partitions to non-negative integers that satisfies  $\psi([\emptyset, \alpha]) = \psi(\alpha)$  for every partition  $\alpha$ , i.e., inserting an empty set to a collection does not change the value of the partition function. A partition function  $\psi$  is *submodular* if the following holds for every two partitions  $[A, \alpha]$  and  $[B, \beta]$ :

$$\psi([A, \alpha]) + \psi([B, \beta]) \geq \psi([A \cup \overline{B}, \alpha \cap B]) + \psi([B \cup \overline{A}, \beta \cap A]) \tag{2}$$

Similarly to branch-decompositions, Amini et al. [1] defined a *decomposition tree* of a partition function  $\psi$ . A *decomposition tree* on a finite set  $E$  is a tree  $T$  with a bijection  $\sigma$  between its leaves and  $E$ . Every internal node  $v$  of  $T$  corresponds to the partition of  $E$  whose parts are the leaves contained in subtrees of  $T \setminus v$ . A decomposition tree is *compatible* with a set of partitions  $\mathcal{P}$  of  $E$  if all partitions corresponding to the internal nodes of  $T$  belong to  $\mathcal{P}$ .

Let  $\mathcal{P}_k[\psi]$  denote the set of partitions  $\alpha$  of  $E$  such that  $\psi(\alpha) \leq k$ . The *width* of a submodular partition function  $\psi$  is the smallest integer  $k$  such that there exists a decomposition tree compatible with  $\mathcal{P}_k[\psi]$ . The concepts of submodular partition functions and decomposition trees include graph tree-width and branch-width as special cases.

There is a dual object to the decomposition tree called a *bramble* introduced by Amini et al. [1]. A  $\mathcal{P}$ -*bramble*  $\mathcal{B}$  on  $E$  is a set of pairwise intersecting subsets of  $E$  which contains a part of every partition of  $\mathcal{P}$ . A  $\mathcal{P}$ -bramble is called *non-principal* if it contains no singleton. The duality theorem for submodular partition functions asserts the following.

**Theorem 2 (Amini et al. [1]).** *Let  $\psi$  be a submodular partition function. There is no decomposition tree compatible with  $\mathcal{P}_k[\psi]$  if and only if there is a non-principal  $\mathcal{P}_k[\psi]$ -bramble.*

Note that Theorem 2 is proven in [1] for a larger class of *weakly submodular* partition functions. In this paper, we restrict our attention only to the class of submodular partition functions. In particular, the loose tangles defined in the next section are studied only for submodular partition functions.

### 3 Loose Tangles

A key ingredient of the algorithm of Oum and Seymour [10] for deciding whether a connectivity function has branch-width  $k$  ( $k$  is fixed) is the notion of a *loose tangle* which we now recall. For a connectivity function  $f$  on a ground set  $E$ , a *loose  $f$ -tangle* of order  $k + 1$  is a set  $\mathcal{T}$  of subsets of  $E$  satisfying the following three axioms:

- (L1)  $\emptyset \in \mathcal{T}$  and  $\{e\} \in \mathcal{T}$  for every  $e \in E$  such that  $f(\{e\}) \leq k$ .
- (L2) If  $A, B \in \mathcal{T}$ ,  $C \subseteq A \cup B$ , and  $f(C) \leq k$ , then  $C \in \mathcal{T}$ .
- (L3)  $E \notin \mathcal{T}$ .

The following theorem by Oum and Seymour [10] states that the loose  $f$ -tangles are also dual objects to branch-decompositions of connectivity functions.

**Theorem 3 (Oum and Seymour [10]).** *Let  $f$  be a connectivity function on a ground set  $E$ . Then, no loose  $f$ -tangle of order  $k + 1$  exists if and only if the branch-width of  $f$  is at most  $k$ .*

Using loose tangles Oum and Seymour [10] managed to construct an algorithm for deciding whether the branch-width of a connectivity function is at most  $k$  for a fixed  $k$  in polynomial time when  $f$  is given by an oracle.

Similarly to the loose tangles of Oum and Seymour we introduce *loose tangles* for submodular partition functions. A *loose  $\mathcal{P}$ -tangle* is a set  $\mathcal{T}$  of subsets of  $E$  closed under taking subsets satisfying the following three axioms.

- (P1)  $\emptyset \in \mathcal{T}$ ,  $\{e\} \in \mathcal{T}$ , for all  $e \in E$  such that the partition  $[\{e\}, \overline{\{e\}}]$  belongs to  $\mathcal{P}$ .
- (P2) If  $A_1, A_2, \dots, A_p \in \mathcal{T}$ ,  $C_i \subseteq A_i$ , for  $i = 1, \dots, p$ ,  $[C_1, \dots, C_p, \overline{\cup_{i=1}^p C_i}] \in \mathcal{P}$ , then  $\cup_{i=1}^p C_i \in \mathcal{T}$ .
- (P3)  $E \notin \mathcal{T}$ .

To prove the main theorem of this section, we need a lemma.

**Lemma 1.** *Let  $\psi$  be a submodular partition function and  $[A, \alpha]$  a partition. Then  $\psi([A, \alpha]) \geq \psi([A, \overline{A}])$ .*

*Proof.* Suppose that the partition  $[A, \alpha]$  has at least three non-empty parts and let  $[A, B, \beta] = [A, \alpha]$ . By submodularity,

$$\begin{aligned} \psi([A, \alpha]) + \psi([B, \overline{B}]) &\geq \psi([A \cup \overline{B}, \alpha \cap B]) + \psi([B \cup \overline{A}, \overline{B} \cap A]) \\ &= \psi([B, \overline{B}]) + \psi([A, \overline{A}]). \end{aligned}$$

The result follows. □

In the following theorem, we show that for classes of partitions of bounded width, the loose tangle is a dual object to the decomposition tree.

**Theorem 4.** *Let  $\psi$  be a submodular partition function. There is no decomposition tree compatible with  $\mathcal{P}_k[\psi]$  if and only if there is a loose  $\mathcal{P}_k[\psi]$ -tangle.*

*Proof.* Suppose there is a decomposition tree  $(T, \sigma)$  compatible with  $\mathcal{P}_k[\psi]$  and a loose  $\mathcal{P}_k[\psi]$ -tangle  $\mathcal{T}$ . We will show that  $\mathcal{T}$  violates (P3). Choose an arbitrary leaf  $x$  of  $T$  as a root. Every internal node  $v$  of  $T$  corresponds to a partition  $\alpha_v$ . Let  $C_v$  be a union of all parts of  $\alpha_v$  except the one containing  $x$ . Define  $C_v$  of a leaf  $v$  as the singleton  $\sigma(v)$ . We will show by backward induction on the distance from  $x$  that for every node  $v$  of  $T$ , the set  $C_v$  belongs to  $\mathcal{T}$ . Since  $T$  is a decomposition tree of  $E$  compatible with  $\mathcal{P}_k[\psi]$ , there is a partition  $\{\{e\}, \alpha_e\}$  in  $\mathcal{P}_k[\psi]$ , for each  $e \in E$ . By Lemma 1,  $\psi(\{\{e\}, \{e\}\}) \leq \psi(\{\{e\}, \alpha_e\})$ . Hence,  $\{\{e\}, \{e\}\}$  belongs to  $\mathcal{P}_k[\psi]$  and  $\{e\}$  is in  $\mathcal{T}$  by (P1). For an inner node  $v$ , all his children  $u_1, \dots, u_p$  are farther from  $x$  than  $v$  and therefore all  $C_{u_i}$  are in  $\mathcal{T}$ . By (P2), since  $[C_{u_i}, \overline{C_{u_i}}]$  belongs to  $\mathcal{P}_k[\psi]$ ,  $C_v \equiv \cup C_{u_i} \in \mathcal{T}$ . Finally, let  $v$  be the only child of  $x$ . Since  $C_v \in \mathcal{T}$  and  $\{\sigma(x)\} \in \mathcal{T}$ , by (P2),  $C_v \cup \{\sigma(x)\} = E$  also belongs to  $\mathcal{T}$ . (P3) is now violated.

A partial decomposition tree for  $A \subseteq E$  is a decomposition tree for a partition function  $\psi'$  on  $A \cup \{a\}$  defined as  $\psi'([B, \beta]) = \psi(((B \setminus \{a\}) \cup \overline{A}, \beta))$  for a partition  $[B, \beta]$  where  $B$  contains  $a$ . We say that a set  $A \subseteq E$  is  $k$ -branched if there is a partial decomposition tree for  $A$  compatible with  $\mathcal{P}_k[\psi]$ .

Define  $\mathcal{T}$  to be a subset of  $2^E$  closed under taking subsets, containing all singletons and all  $k$ -branched sets. We will show that  $\mathcal{T}$  is a loose tangle. (P1) trivially holds since all  $k$ -branched singletons are in  $\mathcal{T}$ . Let  $A_1, \dots, A_p \in \mathcal{T}$  and  $C_i \subseteq A_i$ ,  $i = 1, \dots, p$ , such that  $[C_1, \dots, C_p, \overline{C_i}] \in \mathcal{P}_k[\psi]$ . We can assume that  $A_i$  are  $k$ -branched (otherwise take such a superset of it instead). Let  $Y_1, \dots, Y_p$ ,  $Y_i \subseteq A_i$ , be such sets that  $\cup C_i \subseteq \cup Y_i$  and  $\psi([Y_1, \dots, Y_p, \overline{\cup Y_i}])$  is minimum. We will show that the set  $\cup Y_i$  is  $k$ -branched.

To this end, we modify the partial decomposition tree  $T_i$  for  $A_i$  to be a partial decomposition tree for  $Y_i$ . At first, we delete from  $T_i$  all leaves corresponding to elements not in  $Y_i$ . We then repeatedly contract all nodes of degree two or less until we get a ternary tree  $T'_i$ . We claim  $T'_i$  is compatible with  $\mathcal{P}_k[\psi]$ . Suppose for a contradiction that there is an internal node  $v'$  of  $T'_i$  corresponding to an internal node  $v$  of  $T_i$  such that  $\alpha_{v'} \notin \mathcal{P}_k[\psi]$ . Assume  $i = 1$  since we can relabel the parts so. Let  $[A, \alpha] = \alpha_v$  such that  $A$  is the part of  $\alpha_v$  that contains  $\overline{A_1}$ . We infer from the submodularity of the function  $\psi$  that

$$\begin{aligned} \psi([A, \alpha]) + \psi([Y_1, Y_2, \dots, Y_p, \overline{\cup Y_i}]) &\geq \psi([A \cup \overline{Y_1}, \alpha \cap Y_1]) \\ &+ \psi([Y_1 \cup \overline{A}, Y_2 \cap A, \dots, Y_p \cap A, \overline{\cup Y_i} \cap A]) \end{aligned}$$

The choice of  $Y_1, \dots, Y_p$  yields that

$$\psi([Y_1 \cup \overline{A}, Y_2 \cap A, \dots, Y_p \cap A, \overline{\cup Y_i} \cap A]) \geq \psi([Y_1, \dots, Y_p, \overline{\cup Y_i}]).$$

Hence,  $\psi([A \cup \overline{Y_1}, \alpha \cap Y_1]) \leq \psi([A, \alpha]) \leq k$  and  $T'_1$  is compatible with  $\mathcal{P}_k[\psi]$ .

Now, construct a partial decomposition tree  $T$  by connecting  $T'_i$  to a single node corresponding to a partition  $[Y_1, \dots, Y_p, \overline{\cup Y_i}]$ . This partition belongs to  $\mathcal{P}_k[\psi]$  since  $\psi([Y_1, \dots, Y_p, \overline{\cup Y_i}]) \leq \psi([C_1, \dots, C_p]) \leq k$  by the minimality of  $\psi([Y_1, \dots, Y_p, \overline{\cup Y_i}])$ . Therefore  $T$  is a partial decomposition tree for  $\cup Y_i$  compatible with  $\mathcal{P}_k[\psi]$  and thus  $\cup Y_i \in \mathcal{T}$ . Since  $\cup C_i \subseteq \cup Y_i$ , also  $\cup C_i \in \mathcal{T}$  as required.

If  $E \in T$ , then  $E$  is  $k$ -branched and the partial decomposition tree for  $E$  is actually a decomposition tree for  $\psi$ . This contradicts the fact that  $\psi$  does not have a decomposition tree compatible with  $\mathcal{P}_k[\psi]$ . Therefore,  $E \notin \mathcal{T}$  and (P3) holds. We conclude that  $\mathcal{T}$  is a loose  $\mathcal{P}_k[\psi]$ -tangle.  $\square$

### 4 Hardness of Submodular Partition Functions

We first have to define several auxiliary functions before we can establish our hardness result. Let  $g_n$  be the function  $g_n : 2^E \rightarrow \mathbb{N}$  for  $E = \{1, \dots, 2n\}$  defined as  $g_n(X) = \min\{|X|, |\overline{X}|\}$ . We start our exposition with showing that  $g_n$  is submodular.

**Lemma 2.** *The function  $g_n$  is submodular for every  $n$ .*

*Proof.* Consider two subsets  $X$  and  $Y$ . If both  $|X| \leq n$  and  $|Y| \leq n$ , then

$$\begin{aligned} g_n(X) + g_n(Y) &= |X| + |Y| = |X \cap Y| + |X \cup Y| \\ &\geq g_n(X \cap Y) + g_n(X \cup Y). \end{aligned}$$

If both  $|X| > n$  and  $|Y| > n$ , we get the same result by the symmetry of  $g$ .

$$\begin{aligned} g_n(X) + g_n(Y) &= g_n(\overline{X}) + g_n(\overline{Y}) \geq g_n(\overline{X \cap Y}) + g_n(\overline{X \cup Y}) \\ &= g_n(X \cup Y) + g_n(X \cap Y) \end{aligned}$$

So suppose that  $|X| > n$  and  $|Y| \leq n$ . We get

$$\begin{aligned} g_n(X) + g_n(Y) &= |\overline{X}| + |Y| = |\overline{X} \setminus Y| + |Y \setminus \overline{X}| + 2|\overline{X} \cap Y| \\ &\geq g_n(\overline{X} \setminus Y) + g_n(Y \setminus \overline{X}) = g_n(\overline{X} \cap \overline{Y}) + g_n(X \cap Y) \\ &= g_n(X \cup Y) + g_n(X \cap Y). \end{aligned}$$

This finishes the proof.  $\square$

The function  $g_n$  can be extended to a partition function  $\phi_n$  on the ground set  $E = \{1, \dots, 2n\}$  by setting

$$\phi_n(\alpha) = \max_{i \in I} g_n(A_i).$$

A part  $A_i$  of  $\alpha$  is *dominating* if  $g_n(A_i) = \phi_n(\alpha)$ . Note that, if  $\alpha$  has a part with at least  $n$  elements, then that part is dominating.

We proceed by showing that the function  $\phi_n$  is submodular.

**Lemma 3.** *The function  $\phi_n$  is submodular for every  $n$ .*

*Proof.* We check the following inequality for all partitions  $[A, \alpha]$  and  $[B, \beta]$ :

$$\phi_n([A, \alpha]) + \phi_n([B, \beta]) \geq \phi_n([A \cup \overline{B}, \alpha \cap B]) + \phi_n([B \cup \overline{A}, \beta \cap A]).$$

Since one of  $A, \overline{A}$  and one of  $B, \overline{B}$  has at least  $n$  elements, at least one of the parts  $A \cup \overline{B}$  or  $B \cup \overline{A}$  in this inequality has at least  $n$  elements and hence it is dominating. If both  $A \cup \overline{B}$  and  $B \cup \overline{A}$  are dominating, then the submodularity of  $\phi_n$  follows from the submodularity of  $g$ :

$$\begin{aligned} \phi_n([A, \alpha]) + \phi_n([B, \beta]) &\geq g_n(A) + g_n(B) = g_n(A) + g_n(\overline{B}) \\ &\geq g_n(A \cap \overline{B}) + g_n(A \cup \overline{B}) = g_n(\overline{A} \cup B) + g_n(A \cup \overline{B}) \\ &= \phi_n([A \cup \overline{B}, \alpha \cap B]) + \phi_n([B \cup \overline{A}, \beta \cap A]) \end{aligned}$$

Suppose that  $A \cup \overline{B}$  is not dominating, so take an  $A_i \in \alpha$  such that  $A_i \cap B$  is dominating. Since  $|B| \geq n$  and  $A_i \subseteq \overline{A}$ , it holds that  $g_n(A_i \cup B) \geq g_n(B \cup \overline{A})$ . We use this inequality to prove the submodularity as follows:

$$\begin{aligned} \phi_n([A, \alpha]) + \phi_n([B, \beta]) &\geq g_n(A_i) + g_n(B) \geq g_n(A_i \cap B) + g_n(A_i \cup B) \\ &\geq g_n(A_i \cap B) + g_n(B \cup \overline{A}) \\ &= \phi_n([A \cup \overline{B}, \alpha \cap B]) + \phi_n([B \cup \overline{A}, \beta \cap A]) \end{aligned}$$

The case when  $B \cup \overline{A}$  is not dominating follows by symmetry. □

Values of the function  $\phi_n$  range between 0 and  $n$ . We now truncate the function and define the following partition function  $\phi_{n,k}$  on  $E = \{1, \dots, 2n\}$  as follows:

$$\phi_{n,k}(\alpha) = \min\{\phi_n(\alpha), k\}.$$

Next, we show that the function  $\phi_n$  stays submodular after the truncation.

**Lemma 4.** *The function  $\phi_{n,k}$  is submodular for every  $n$  and  $k$ .*

*Proof.* Let us consider two partitions  $[A, \alpha]$  and  $[B, \beta]$  that violates the inequality (2):

$$\phi_{n,k}([A, \alpha]) + \phi_{n,k}([B, \beta]) \geq \phi_{n,k}([A \cup \overline{B}, \alpha \cap B]) + \phi_{n,k}([B \cup \overline{A}, \beta \cap A]).$$

Since  $\phi_{n,k}(\gamma) \leq \phi_n(\gamma)$  for all partitions  $\gamma$ , at least one of  $\phi_n([A, \alpha])$  or  $\phi_n([B, \beta])$  is larger than  $k$ . If both of them are, then the inequality trivially holds. Suppose that  $\phi_n([A, \alpha]) < k$ . We will show that at least one of  $\phi_n([A \cup \overline{B}, \alpha \cap B])$  or  $\phi_n([B \cup \overline{A}, \beta \cap A])$  is smaller or equal to  $\phi_n([A, \alpha])$ .

If  $|A| \geq n$ , then  $\phi_n([A \cup \overline{B}, \alpha \cap B]) \leq \phi_n([A, \alpha])$  since  $A \cup \overline{B}$  is the dominating part and  $g_n(A \cup \overline{B}) \leq g_n(A) \leq \phi_n([A, \alpha])$ . If  $|A| < n$ , then  $\phi_n([B \cup \overline{A}, \beta \cap A]) \leq \phi_n([A, \alpha])$  since  $B \cup \overline{A}$  is the dominating part and  $g_n(B \cup \overline{A}) \leq g_n(\overline{A}) \leq \phi_n([A, \alpha])$ . This finishes the proof. □



Now, we use the function  $\phi_{n,3}$  to construct partition functions  $\phi_n^*$  and  $\phi_{n,\beta}^*$  which appear in our hardness result. The function  $\phi_n^*$  is defined as

$$\phi_n^*(\alpha) = \begin{cases} \phi_{n,3}(\alpha) & \text{if } \alpha \text{ has at most three non-empty parts, and} \\ 3 & \text{otherwise.} \end{cases}$$

For a partition  $\beta$  of  $\{1, \dots, 2n\}$  into  $n$  two-element subsets, the function  $\phi_{n,\beta}^*$  is then defined as

$$\phi_{n,\beta}^*(\alpha) = \begin{cases} \phi_{n,3}(\alpha) & \text{if } \alpha \text{ has at most three non-empty parts,} \\ 2 & \text{if } \alpha = \beta, \text{ and} \\ 3 & \text{otherwise.} \end{cases}$$

First, we show that these functions are submodular.

**Lemma 5.** *The function  $\phi_n^*$  is submodular for every  $n$ .*

*Proof.* Observe the following:

- If  $\phi_{n,3}(\alpha) = 0$ , then also  $\phi_n^*(\alpha) = 0$ .
- If  $\phi_{n,3}(\alpha) = 1$ , then  $\phi_n^*(\alpha) = 1$  unless  $\alpha$  is a set of singletons where  $\phi_n^*(\alpha) = 3$ .
- If  $\phi_{n,3}(\alpha) = 2$ , then  $\phi_n^*(\alpha) = 2$  unless  $\alpha$  has more than three non-empty parts. In this case, every part of  $\alpha$  is a pair or a singleton.

Therefore the functions  $\phi_{n,3}$  and  $\phi_n^*$  differ only on partitions consisting of singletons and pairs.

Let us assume for a contradiction that  $\phi_n^*$  is not submodular. Since  $\phi_n^*(\alpha) \geq \phi_{n,3}(\alpha)$  for all partitions  $\alpha$ , the violation of the submodularity is caused by an increase on the right-hand side of (2). Consider partitions  $[A, \alpha]$  and  $[B, \beta]$  violating (2). Hence, say,  $\gamma = [A \cup \overline{B}, \alpha \cap B]$  is that partition containing only singletons and pairs. Since  $\gamma$  has all parts of size at most two,  $|\overline{B}| \leq 2$ . If  $\overline{A} \cap \overline{B} = \emptyset$ , then  $\overline{B} \subseteq A$  and  $\overline{A} \subseteq B$ . Therefore  $\gamma = [A, \alpha]$ ,  $[B \cup \overline{A}, \beta \cap A] = [B, \beta]$  and the inequality trivially holds. So we can assume that  $|B \cup \overline{A}| > |B|$  and since  $2n - 2 \leq |B| < 2n$ , by the definition of  $\phi_n^*$

$$\phi_n^*([B, \beta]) > \phi_n^*([B \cup \overline{A}, \beta \cap A]). \tag{3}$$

Since the number of non-empty parts of  $\gamma$  is at least 4, the number of non-empty parts of  $[A, \alpha]$  is at least 3 and therefore  $\phi_n^*([A, \alpha]) \geq 2$  by the definition of  $\phi_n^*$ . The submodularity follows from (3) and the fact that  $\phi_n^*(\gamma) \leq 3 \leq \phi_n^*([A, \alpha]) + 1$ . □

**Lemma 6.** *The function  $\phi_{n,\beta}^*$  is submodular for every  $n \geq 4$  and for every partition  $\beta$  consisting only of two-element sets.*

*Proof.* Since  $\phi_n^*$  and  $\phi_{n,\beta}^*$  differ only on the partition  $\beta$  where  $\phi_n^*(\beta) \geq \phi_{n,\beta}^*(\beta)$ ,  $\beta$  has to be on the left-hand side of the inequality (2) to violate it. Let  $[A, \alpha]$  and  $\beta = [C, \gamma]$  be the partitions violating (2):

$$\phi_{n,\beta}^*([A, \alpha]) + \phi_{n,\beta}^*([C, \gamma]) \geq \phi_{n,\beta}^*([A \cup \overline{C}, \alpha \cap C]) + \phi_{n,\beta}^*([C \cup \overline{A}, \gamma \cap A])$$

Since  $|C| = 2$ ,  $\phi_{n,\beta}^*([A \cup \overline{C}, \alpha \cap C]) \leq 2$ . Hence  $\phi_{n,\beta}^*([A, \alpha]) \leq 2$ . If  $|A| \leq 2$ , then  $|C \cup \overline{A}| \geq 2n - |A|$  and  $\phi_{n,\beta}^*([C \cup \overline{A}, \gamma \cap A]) \leq \phi_{n,\beta}^*([A, \alpha])$ , contradicting the assumption. Therefore  $A$  has to have at least  $2n - 2$  elements and  $\phi_{n,\beta}^*([A \cup \overline{C}, \alpha \cap C]) \leq \phi_{n,\beta}^*([A, \alpha])$ .

If  $\overline{C} \subseteq A$ , then  $\overline{A} \subseteq C$  and  $\phi_{n,\beta}^*([C \cup \overline{A}, \gamma \cap A]) = \phi_{n,\beta}^*([C, \gamma])$ , contradicting the assumption. Therefore  $|A \cup \overline{C}| > |A|$  giving  $\phi_{n,\beta}^*([A, \alpha]) > \phi_{n,\beta}^*([A \cup \overline{C}, \alpha \cap C])$ . Since  $\phi_{n,\beta}^*(\beta) + 1 = 3 \geq \phi_{n,\beta}^*([C \cup \overline{A}, \gamma \cap A])$ , the inequality (2) holds — a contradiction.  $\square$

In the proof of the main theorem we will use the fact that the width of the function  $\phi_n^*$  is three while the width of the modified function  $\phi_{n,\beta}^*$  is two. To see that width of  $\phi_{n,\beta}^*$  is at most two, just consider the following decomposition tree  $T$  of  $\phi_{n,\beta}^*$ .  $T$  has a root  $x$  with  $n$  children  $v_1, \dots, v_n$  each  $v_i$  connected to two leaves corresponding to the two elements in  $\beta_i$ . Since  $\phi_{n,\beta}^*(\alpha_x) = \phi_{n,\beta}^*(\beta) = 2$  and  $\phi_{n,\beta}^*(\alpha_{v_i}) = 2$ , for  $i = 1, \dots, n$ , the decomposition tree  $T$  has width two. In the next lemma, we show that the width of  $\phi_n^*$  is three.

**Lemma 7.** *For  $n \geq 4$ , the width of  $\phi_n^*$  is three.*

*Proof.* Let  $T$  be a decomposition tree of  $\phi_n^*$  of width smaller than three. We assume there are no nodes of degree two in  $T$  since we can contract them obtaining a smaller decomposition tree of the same width. Since every internal node  $v$  of  $T$  of degree larger than three corresponds to a partition  $\alpha_v$  of  $E$  with more than three parts (thus  $\phi_n^*(\alpha_v) = 3$ ), there are no such vertices in  $T$  and  $T$  is a ternary tree. Consider an arbitrary internal node  $v$  of  $T$  with less than two leaves as neighbors. There have to be such a vertex  $v$  since there are at most  $n$  vertices with two leaves as neighbors but there are  $2(n - 1)$  internal nodes. For such a vertex  $v$ ,  $\alpha_v$  contains a part with at least three elements and at most  $2n - 3$  elements implying  $\phi_n^*(\alpha_v) = 3$ . This finishes the proof.  $\square$

We are now ready to establish our hardness result. We assume the existence of an algorithm and show that it cannot discover a small discrepancy between a submodular partition function having width three and two.

**Theorem 5.** *There is no sub-exponential algorithm for determining whether the width of an oracle-given submodular partition function on a set with  $2n$  elements is at most two.*

*Proof.* Assume that there exists such a sub-exponential algorithm  $\mathcal{A}$  and run  $\mathcal{A}$  for the submodular partition function  $\phi_n^*$ . The algorithm  $\mathcal{A}$  must clearly output that the width  $\phi_n^*$  is at least three. Since the running time of the algorithm is sub-exponential, for  $n$  sufficiently large, there exists a partition  $\beta$  of  $\{1, \dots, 2n\}$  into  $n$  two-element subsets such that  $\mathcal{A}$  never queries  $\beta$  since the number of such partitions is

$$\frac{(2n)!}{n!2^n} = (2n - 1)(2n - 3) \cdots 3 \cdot 1 \geq n!$$

and  $\mathcal{A}$  cannot query all of them because of its running time. However, the algorithm  $\mathcal{A}$  for  $\phi_{n,\beta}^*$  performs the same steps and thus it outputs that the width of  $\phi_{n,\beta}^*$  is at least three which is not correct.  $\square$

Using Yao's principle, Theorem 5 also implies the following lower bound for randomized algorithms:

**Corollary 1.** *For every randomized algorithm determining whether the width of an oracle-given submodular partition function on a set with  $2n$  elements is at most two, there exists a submodular partition function  $\psi$  such that the expected running time of the algorithm for  $\psi$  is exponential in  $n$ .*

## Acknowledgement

The author would like to thank Daniel Král' for suggesting the problem, inspiring comments on the topic, and giving me a big blue elephant as a birthday present.

## References

1. Amini, O., Mazoit, F., Nisse, N., Thomassé, S.: Submodular Partition Functions. Accepted to Discrete Mathematics (2008)
2. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree decomposable graphs. *J. of Algorithms* 12, 308–340 (1991)
3. Bodlaender, H.: A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Computing* 25, 1305–1317 (1996)
4. Bouchitté, V., Kratsch, D., Müller, H., Todinca, I.: On treewidth approximations. *Discrete Appl. Math.* 136(2-3), 183–196 (2004)
5. Courcelle, B.: The monadic second-order logic of graph I, Recognizable sets of finite graphs. *Information and Computation* 85, 12–75 (1990)
6. Hliněný, P.: A parametrized algorithm for matroid branch-width. *SIAM J. Computing* 35(2), 259–277 (2005)
7. Hliněný, P.: Branch-width, parse trees and monadic second-order logic for matroids. *J. Comb. Theory Series B* 96, 325–351 (2006)
8. Hliněný, P., Oum, S.: Finding branch-decomposition and rank-decomposition. *SIAM J. Computing* 38(3), 1012–1032 (2008)
9. Král', D.: Width of matroids (in preparation)
10. Oum, S., Seymour, P.: Testing branch-width. *J. Comb. Theory Series B* 97(3), 385–393 (2007)
11. Robertson, N., Seymour, P.: Graph minors. X. Obstructions to tree-decomposition. *J. Comb. Theory Series B* 52, 153–190 (1991)

# The Guarding Problem – Complexity and Approximation

Thirumala Reddy T.V., Sai Krishna D., and Pandu Rangan C.

Department of Computer Science and Engineering,  
Indian Institute of Technology Madras,  
Chennai 600036, India  
{tiru114,dsaikris86,prangan55}@gmail.com

**Abstract.** Let  $G = (V, E)$  be the given graph and  $G_R = (V_R, E_R)$  and  $G_C = (V_C, E_C)$  be the sub graphs of  $G$  such that  $V_R \cap V_C = \emptyset$  and  $V_R \cup V_C = V$ .  $G_C$  is referred to as the cops region and  $G_R$  is called as the robber region. Initially a robber is placed at some vertex of  $V_R$  and the cops are placed at some vertices of  $V_C$ . The robber and cops may move from their current vertices to one of their neighbours. While a cop can move only within the cops region, the robber may move to any neighbour. The robber and cops move alternatively. A vertex  $v \in V_C$  is said to be *attacked* if the current turn is the robber's turn, the robber is at vertex  $u$  where  $u \in V_R$ ,  $(u, v) \in E$  and no cop is present at  $v$ . The guarding problem is to find the minimum number of cops required to guard the graph  $G_C$  from the robber's attack. We first prove that the decision version of this problem when  $G_R$  is an arbitrary undirected graph is PSPACE-hard. We also prove that the complexity of the decision version of the guarding problem when  $G_R$  is a wheel graph is NP-hard. We then present approximation algorithms if  $G_R$  is a star graph, a clique and a wheel graph with approximation ratios  $H(n_1)$ ,  $2H(n_1)$  and  $(H(n_1) + \frac{3}{2})$  respectively, where  $H(n_1) = 1 + \frac{1}{2} + \dots + \frac{1}{n_1}$  and  $n_1 = |V_R|$ .

**Keywords:** Approximation Algorithms, PSPACE-complete, QBF (Quantified Boolean Formula), QSAT (Quantified Satisfiability).

## 1 Introduction

Let  $G = (V, E)$  be a graph with  $G_R = (V_R, E_R)$  and  $G_C = (V_C, E_C)$  being induced subgraphs of  $G$  where  $V_R \cap V_C = \emptyset$ ,  $V_R \cup V_C = V$ . We call  $G_R$  the robber's region and  $G_C$  the cops region. Initially a robber is placed at one of the vertices in  $G_R$  and cops are placed at the vertices of  $G_C$ . A move of the cop or the robber consists of moving from the current vertex position to a neighbouring vertex. While the robber can move to any of his neighbour, the cops can move only to their neighbours in  $G_C$ . The game starts with the robber making the first move, followed by the cops and this process repeats. In their respective turns, the cop or the robber may stay where they are or move to a neighbour. Both the cops and the robber know each others positions. Multiple cops are allowed per vertex. A vertex  $v \in V_C$  is said to be *attacked* if the current turn is the robber's

turn, the robber is at vertex  $u$  where  $u \in V_R$ ,  $(u, v) \in E$  and no cop is present at vertex  $v$ . The robber always tries to move in such a way that at some point of time he can attack some vertex of  $V_C$ . The guarding game is called a *robber-win* game if there exists a strategy for the robber where he can attack a vertex in  $V_C$ . The guarding game is called a *cop-win* game if the cops are able to guard  $G_C$  for all strategies of the robber. The state of the game is represented by the current positions of the cops and robber.

Given a graph  $G$  (along with subgraphs of  $G$ ,  $G_R$  and  $G_C$ ) and a positive integer  $c$ , deciding whether  $c$  cops can guard  $G_C$  or not is the decision version of the guarding problem. In the optimization version, we have to find the minimum number of cops required to guard the graph  $G_C$ .

The Guarding problem was introduced by Fomin [1]. They have given a polynomial time solution for the guarding problem if  $G_R$  is a path graph. They proved that the decision version of the guarding problem will become NP-complete if  $G_R$  is a tree graph. They also proved that this problem will become PSPACE-complete if the robber is allowed to move in a Directed Acyclic Graph (DAG). Deciding whether guarding the graph  $G_C$  is PSPACE-hard when  $G_R$  is an arbitrary undirected graph is left open in [1].

In this paper we prove that the complexity of the decision version of the guarding problem when  $G_R$  is an arbitrary undirected graph is PSPACE-hard by reducing Quantified Satisfiability (QSAT) problem to the guarding problem. We also prove that the complexity of the decision version of the guarding problem when  $G_R$  is a wheel graph is NP-hard by giving reduction from SET COVER. We also present approximation algorithms for the guarding problem when  $G_R$  is a star graph, a clique or a wheel graph with approximation ratios  $H(n_1)$ ,  $2H(n_1)$  and  $(H(n_1) + \frac{3}{2})$  respectively, where  $H(n_1) = 1 + \frac{1}{2} + \dots + \frac{1}{n_1}$  and  $n_1 = |V_R|$ .

The results reported in [1] and [2] are the only known results on this problem prior to our work.

## 2 Complexity Results

We now show that the decision version of the guarding problem, when robber region is an arbitrary undirected graph, is PSPACE-hard by reducing QSAT problem to the guarding problem.

**Theorem 1.** *The problem of deciding whether  $c$  cops can guard the graph  $G_C$  when robber region is an arbitrary undirected graph is PSPACE-hard.*

*Proof.* We now give a reduction from QSAT. A Quantified Boolean Formula (QBF) is a Boolean Formula in which both existential quantifiers and universal quantifiers are applied to variables in the formula.

Following well known forms used in the literature [3], we assume that an arbitrary QBF is of the form  $\exists x_1 \forall x_2 \exists x_3 \dots \exists x_n \phi(x_1, x_2, \dots, x_n)$  where the formula starts with  $\exists$  and alternatively we have  $\exists$  and  $\forall$ ,  $n$  is odd and  $\phi$  is an  $n$  variable boolean function over  $x_1, x_2, \dots, x_n$ . We further assume that  $\phi$  is in Conjunctive Normal form (CNF) with  $k$  clauses.

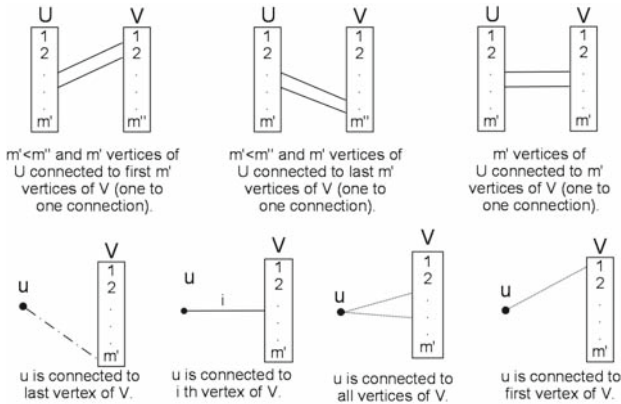


Fig. 1. Notations

We first provide the construction and details of the graph  $G$ . After that we explain the separation of the cops and robber region in the graph  $G$ . After that in lemma 1 we give a winning strategy for robber if  $QBF$  is *false*. Finally in lemma 2, if  $QBF$  is *true*, we provide the strategy that the cops follow for guarding  $G_C$  against all strategies of the robber.

In our construction, we encounter certain patterns of connections so often. We introduce some simple notations to refer these patterns of connectivity. This helps in providing simple description of a complex structure. First we describe the basic building blocks used in construction of the graph  $G$ . They are labeled as  $S$  block,  $\alpha_i$  block and  $\beta_i$  block, where  $1 \leq i \leq n$ . The notations used in the construction of these three blocks are shown in figure 1.

The  $S$  block,  $\alpha_i$  block and  $\beta_i$  block are shown in figures 2 and 3. The connectivity between these blocks is shown in figures 4 and 5.

We now specify the cop and robber regions associated with  $S$ ,  $\alpha_i$  and  $\beta_i$  blocks. In the  $S$  block, the vertex  $r_s$  belongs to robber region and the set of  $n$  independent vertices  $N_0$  belongs to cop region. In the  $\beta_i$  block the vertices  $\beta_a^i, \beta_b^i, \beta_c^i$ , and  $\beta_d^i$ , and the set of  $n$  independent vertices  $A_i$  belongs to robber region and other vertices belong to the cop region. In the  $\alpha_i$  block the vertices  $\alpha_a^i, \alpha_b^i$ , and  $\alpha_c^i$  and the set of  $n$  independent vertices  $A_i$  belongs to robber region and other vertices belongs to cop region. In the final block the vertex  $r_f$ , the set of  $k$  independent vertices  $Z$  and the set of  $n$  independent vertices  $A_{n+1}$  belongs to robber region and other vertices belongs to cop region.

Given  $QBF$  with  $n$  variables, we have to prove that  $QBF$  is satisfiable if and only if  $n$  cops can guard  $G$ . For this we are proving if  $QBF$  is satisfiable then  $n$  cops can successfully guard  $G_C$  for all strategies of the robber otherwise there exists a strategy for the robber such that he can attack some vertex of  $V_C$ . Now we look at the robber and cops strategies if  $QBF$  is *false* and if  $QBF$  is *true* respectively.

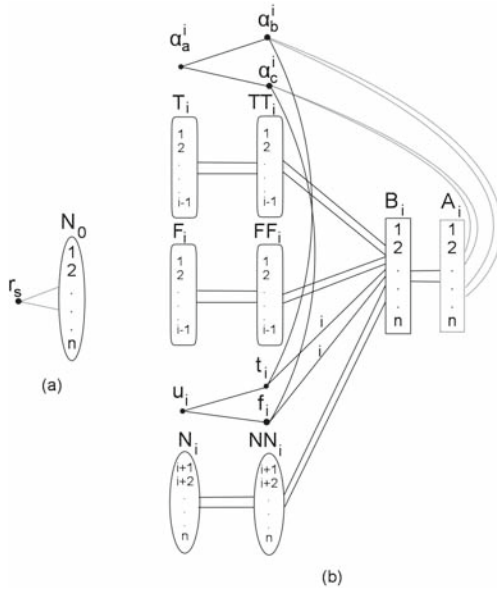


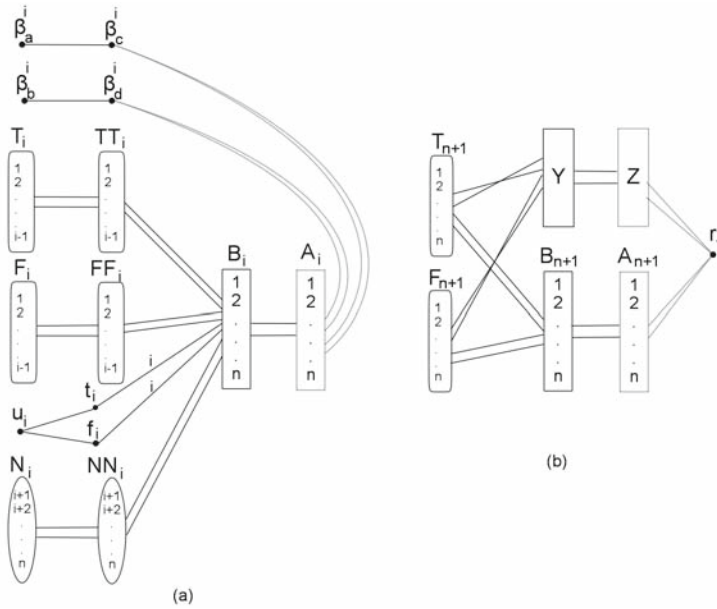
Fig. 2. (a) S block (b)  $\alpha_i$  block

**Lemma 1.** *If the QBF is false then there exists a winning strategy for the robber when the number of cops is  $n$ .*

*Proof.* Let  $QBF = \exists x_1 \forall x_2 \exists x_3 \dots \exists x_n \phi$  is false. As QBF is false there is a truth assignment for boolean variables  $x_1, x_2, \dots, x_n$  such that at least one of the clauses of QBF contain all false literals. Let us denote this assignment as  $D$ .

Now we provide a winning strategy for the robber. Let the robber be at vertex  $r_s$  of  $S$  block, the only one vertex available for him in  $S$ . When the robber is at  $r_s$  then  $n$  cops must be there at  $N_0$ . Now the robber may move forward and reach other blocks. First, the robber may move to any one of  $\beta_a^1$  or  $\beta_b^1$  from  $r_s$ . When the robber moves to  $\beta_a^1$ , one cop moves to  $u_1$  from the first vertex of  $N_0$  and the remaining cops must move to  $N_1$  from  $N_0$ . If cops do not move in this way, then robber can attack corresponding vertex of  $B_1$ . Thus the cops are constrained to move from  $N_0$  to  $u_1$  and  $N_1$ . In a similar way we can handle the case when robber moves to  $\beta_b^1$ . Also, the robber may choose to visit  $\beta_a^1$  or  $\beta_b^1$  in an arbitrary way. In the  $\beta_1$  block, the robber moves to  $\beta_c^1$  ( $\beta_d^1$ ) from  $\beta_a^1$  ( $\beta_b^1$ ). When the robber moves to  $\beta_c^1$ , the cop at  $u_1$  moves to  $t_1$  or  $f_1$  and the cops at  $N_0$  move to  $NN_1$ . If cops do not move in this way then robber can attack corresponding vertex of  $B_1$ . Thus the cops are constrained to move  $t_1$  or  $f_1$  from  $u_1$  and  $NN_1$  from  $N_1$ . In a similar way we can handle the case when robber moves to  $\beta_d^1$ .

Now the robber moves from  $\beta_c^1$  ( $\beta_d^1$ ) to  $\alpha_a^2$  and enters  $\alpha_2$  block. When the robber moves from  $\beta_c^1$  ( $\beta_d^1$ ) to  $\alpha_a^2$ , if there is a cop at  $t_1$  ( $f_1$ ) then this cop moves



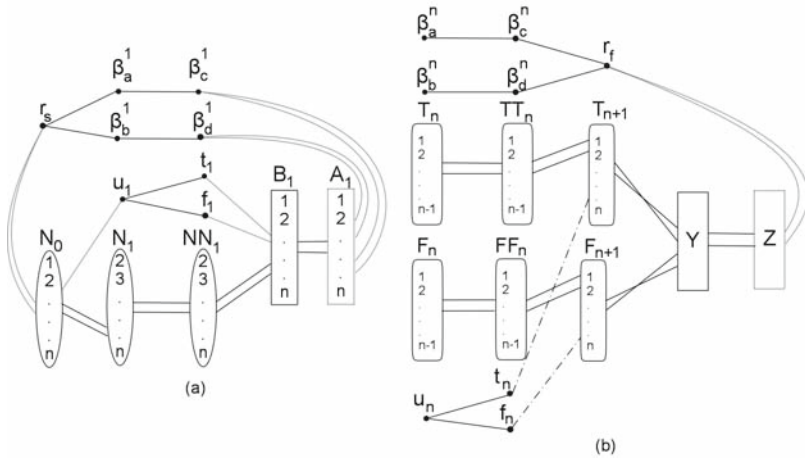
**Fig. 3.** (a)  $\beta_i$  block (b) Final block

to last vertex of  $T_2$  ( $F_2$ ), the cop at first vertex of  $NN_1$  moves to  $u_2$  and the remaining cops at  $NN_1$  move to  $N_2$ . If cops do not move then robber can attack corresponding vertex of  $B_2$ . In  $\alpha_2$  block the robber makes a move depending on the value of  $x_2$  in  $D$ . If  $x_2$  is set to *true* in  $D$ , then the robber moves from  $\alpha_a^2$  to  $\alpha_b^2$ . If  $x_2$  is *false* in  $D$ , then the robber moves from  $\alpha_a^2$  to  $\alpha_c^2$ . This constrains the cop to move from  $u_2$  to  $t_2$  or  $f_2$  depending on  $x_2$  is *true* or *false*. We now describe cops movement when robber has moved from  $\alpha_a^2$  to  $\alpha_b^2$ . In this case one cop moves from  $u_2$  to  $t_2$  as mentioned earlier and other cops in  $T_2$ ,  $F_2$  and  $N_2$  move to  $TT_2$ ,  $FF_2$  and  $NN_2$  respectively. If cops do not move in this way then robber can attack corresponding vertex of  $B_2$ . When robber moves from  $\alpha_a^2$  to  $\alpha_c^2$ , one cop moves from  $u_2$  to  $f_2$  as mentioned earlier and other cops in  $T_2$ ,  $F_2$  and  $N_2$  move to  $TT_2$ ,  $FF_2$  and  $NN_2$  respectively. If cops do not move in this way then robber can attack corresponding vertex of  $B_2$ .

Inductively assume that the robber is at  $r_f$  and cops are constrained to be at  $T_{n+1}$  and  $F_{n+1}$  in the final block. A cop at  $j^{th}$  vertex of  $T_{n+1}$  represents setting *true* to  $x_j$  in  $\phi$ . Similarly a cop at  $j^{th}$  vertex of  $F_{n+1}$  represents setting *false* to  $x_j$  in  $\phi$ . Remember that robber followed the assignment  $D$ . If the variable  $x_i$ ,  $\forall x_i$ , set to *true* in  $D$  then there is a cop at  $i^{th}$  vertex of  $T_{n+1}$ . Similarly if the variable  $x_i$ ,  $\forall x_i$ , set to *false* in  $D$  then there is a cop at  $i^{th}$  vertex of  $F_{n+1}$ .

So if the robber is at vertex  $r_f$  then the current positions of  $n$  cops represent the truth assignment of boolean variables  $x_1, x_2, \dots, x_n$ . As we already discussed above  $Z$  and  $Y$  contain a vertex for every clause of  $\phi$  and  $i^{th}$  vertex of  $Y$  is





**Fig. 4.** (a) shows the connectivity between S block and  $\beta_i$  block and (b) shows the connectivity between  $\beta_{i-1}$  block and Final block

connected to  $i^{th}$  vertex of  $Z$ . Let the  $j^{th}$  clause of  $QBF$  has all literals assigned *false*. Now the robber will move from the vertex  $r_f$  to the  $j^{th}$  vertex of  $Z$ . However, in the cops turn no cop can move to the  $j^{th}$  vertex of  $Y$  as the  $j^{th}$  clause of  $\phi$  has all literals assigned *false*. So the  $j^{th}$  vertex of  $Y$  is under *attack* and the robber wins the game.  $\square$

**Lemma 2.** *If the  $QBF$  is true then the  $n$  cops can guard the graph for any strategy of the robber.*

*Proof.* Please refer extended version [4].

From lemma 1 and lemma 2 the theorem holds.  $\square$

**Theorem 2.** *The decision version of the guarding problem when robber region is a wheel graph is NP-hard.*

*Proof.* Please refer extended version [4].

### 3 Approximation Algorithms

We now provide approximation algorithms for guarding against a star graph, a clique and a wheel graph with approximation ratios  $H(n_1)$ ,  $2H(n_1)$  and  $(H(n_1) + \frac{3}{2})$  respectively, where  $H(n_1) = 1 + \frac{1}{2} + \dots + \frac{1}{n_1}$ . Before going to details of Algorithms we first look at the details of the notations and definitions used in these algorithms.

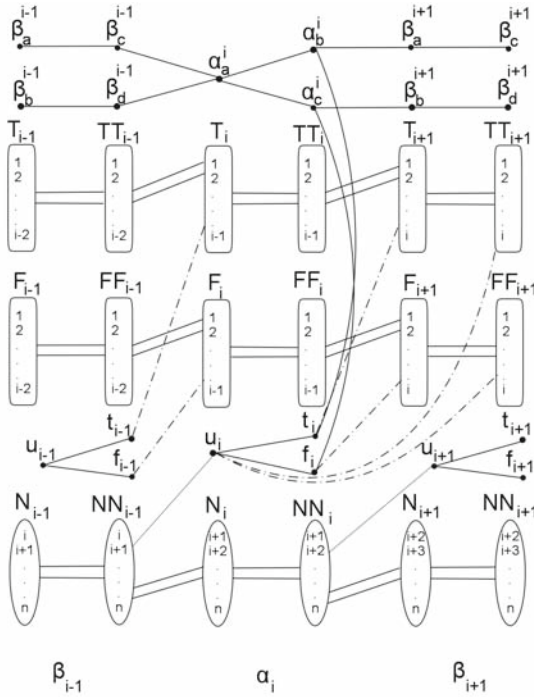


Fig. 5. Connectivity between  $\beta_{i-1}$  block,  $\alpha_i$  block and  $\beta_{i+1}$  block

Let  $G = (V, E)$  be the given graph with  $G_R = (V_R, E_R)$  and  $G_C = (V_C, E_C)$  be induced subgraphs of  $G$  where  $V_R \cap V_C = \emptyset$ ,  $V_R \cup V_C = V$ . Let  $|V_R| = n_1$  and  $\{v_1, v_2, v_3, \dots, v_{n_1}\}$  be the vertices of  $G_R$ . For any vertex  $v \in V$ ,  $N_R(v)$  is the open neighbourhood of  $v$  in  $G_R$  i.e.,  $N_R(v) = N(v) \cap V_R$ , similarly  $N_C(v)$  is the open neighbourhood of  $v$  in  $G_C$  i.e.,  $N_C(v) = N(v) \cap V_C$ ,  $N_R[v]$  is the closed neighbourhood of  $v$  in  $G_R$  i.e.,  $N_R[v] = N_R(v) \cup \{v\}$  and  $N_C[v]$  is the closed neighbourhood of  $v$  in  $G_C$  i.e.,  $N_C[v] = N_C(v) \cup \{v\}$ .  $G(S)$  is the sub graph of  $G$  with vertex set  $S$ , where  $S \subseteq V_R \cup V_C$ .

**Definition 1.** Given  $G = (V, E)$  and let the graph  $G_R$  be a star graph or a wheel graph. Define a set  $S_i$  for every vertex  $v_i \in V_R$  as  $S_i = N_C(v_i)$ , where  $1 \leq i \leq n_1$ .

**Definition 2.** Let  $G_R$  be a star graph or a wheel graph and let the vertex  $v_1$  be the center vertex. Let  $x$  be a fixed but arbitrary element in  $S_j \cap N_C[v]$  when  $S_j \cap N_C[v]$  is nonempty. Define  $GU(v)$  for all  $v \in V_C$  as  $GU(v) = \{x_j \mid x \in S_j \cap N_C[v] \text{ and } 2 \leq j \leq n_1\}$ . Remember that  $x \in S_j \cap N_C[v]$  but  $x_j \in GU(v)$  i.e., in  $GU(v)$  each element is suffixed by its set name.

**Definition 3.** Let  $G(V_R)$  be a wheel graph with the center vertex  $v_1$ . The graph  $G(V_R \setminus \{v_1\})$  is a cycle graph. Consider a vertex  $v_j$ ,  $v_j \in V_R \setminus \{v_1\}$ . Let the

neighbours of the vertex  $v_j$  in the graph  $G(V_R \setminus \{v_1\})$  be  $v_a$  and  $v_b$ . Let  $x$  be a fixed but arbitrary element in  $S_k \cap N_C[v]$  when  $S_k \cap N_C[v]$  is non empty. Define  $WGU_j(u)$  for all  $u \in V_C$  as  $WGU_j(u) = \{x_k \mid x \in S_k \cap N_C[u] \text{ and } k \in \{2, 3, \dots, n_1\} \setminus \{a, b\}\}$ . Remember that  $x \in S_k \cap N_C[u]$  but  $x_k \in WGU_j(u)$  i.e., in  $WGU_j(u)$  each element is suffixed by its set name.

### 3.1 Guarding against Star

Guarding the graph  $G_C$  when the graph  $G_R$  is a star graph is NP-Complete [1]. We now present a  $H(n_1)$  approximation algorithm for guarding against a star graph, where  $n_1 = |V_R|$ .

---

**Algorithm 1.** Algorithm to compute the number of cops required to guard  $G_C$  if  $G_R$  is a star graph.

---

**Require:** A graph  $G = (V, E)$ .

```

c ← 0
if  $S_1 \neq \emptyset$  then
  let  $S_1 = \{u_1, u_2, \dots, u_{|S_1|}\}$ 
  for  $i = 1$  to  $|S_1|$  do
    for  $j = 2$  to  $n_1$  do
      if  $S_j \cap N_C[u_i] \neq \emptyset$  then
        choose an element  $x_j$  from the set  $S_j \cap N_C[u_i]$ 
         $S_j \leftarrow S_j - \{x_j\}$ 
      end if
       $j \leftarrow j + 1$ 
    end for
     $i \leftarrow i + 1$ 
  end for
   $c \leftarrow |S_1|$ 
end if
while  $S_2 \neq \emptyset$  or  $S_3 \neq \emptyset$  or  $S_4 \neq \emptyset \dots$  or  $S_{n_1} \neq \emptyset$  do
   $\forall v \in V_C$  compute  $m_v$  as in Algorithm2 given below.
   $m \leftarrow \underset{v \in V_C}{Max} \{m_v\}$ 
  place a cop any one of the vertex  $u$  whose  $m_u$  is  $m$ 
  for  $j = 2$  to  $n_1$  do
    if  $S_j \cap N_C[u] \neq \emptyset$  then
      choose an element  $x_j$  from the set  $S_j \cap N_C[u]$ 
       $S_j \leftarrow S_j - \{x_j\}$ 
    end if
  end for
   $c \leftarrow c + 1$ 
end while
return c           {c is the number of cops needed to guard the graph}

```

---

**Algorithm 2.** Algorithm to compute  $m_v$ .

**Require:** A graph  $G = (V, E)$  and a vertex  $v$ .

$m_v \leftarrow 0$

**for**  $j = 2$  to  $n_1$  **do**

**if**  $S_j \cap N_C[v] \neq \emptyset$  **then**

$m_v = m_v + 1$

**end if**

**end for**

**return**  $m_v$        $\{m_v$  is the number of sets in  $\{S_2, S_3, \dots, S_{n_1}\}$  that has non empty intersection with  $N_C[v]\}$

**Theorem 3.** *The number of cops return by Algorithm 1,  $c$ , is sufficient for guarding  $G_C$  against a star graph.*

*Proof.* First we prove that  $c$  cops can guard  $G_C$  for all strategies of the robber starting from the center vertex. Then we show that how these  $c$  cops are sufficient for guarding  $G_C$  for all strategies of the robber. Let  $u_1, u_2, u_3, \dots, u_c$  be the vertex positions chosen in Algorithm 1 to place cops. Let us suppose that initially the robber is at the center vertex. Place  $c$  cops, one cop per vertex, at  $u_1, u_2, \dots, u_c$ . Now the robber can move from the center vertex to any of its neighbour vertices. Let us suppose that the robber moved from the center vertex to one of the neighbour vertex, say  $v_k$ . Consider the removal of the set  $S_k$  elements in the while loop. Let  $u'_1, u'_2, \dots, u'_{|S_k|}$  be the vertices belongs to  $V_C$  and let  $u'_i$  deleted  $\alpha_i \in S_k$ . Now the strategy for the cops is : the cop at  $u'_i$  move to  $\alpha_i$ . Now the robber cannot move to any vertex other than the center vertex. If the robber moves back to the center vertex then the cops also moves back to previous positions and the process repeats. As all the sets  $S_j$ , where  $2 \leq j \leq n_1$ , are empty after execution of Algorithm 1,  $c$  cops can guard  $G_C$  for any strategy of the robber starting from the center vertex.

We have given the scheme for defending the attacks when the robber starts at  $v_1$ . This scheme has cops configuration for every vertex reachable from  $v_1$  by robber. Let  $C_i$  be the configuration of cops for vertex  $v_i \in V_R$  in the above scheme. If the robber starts at a vertex, say  $v_i$ , other than the center vertex then cops set their initial configuration to  $C_i$ . □

**Theorem 4.** *Algorithm 1 is a  $H(n_1)$  approximation algorithm for guarding  $G_C$  against a star graph.*

*Proof.* Let us suppose that the while loop of Algorithm 1 is executed for  $c$  times. If the robber is at  $r_1$  and if the set  $S_1$  is not empty then one cop should be present at every vertex of  $S_1$  in any solution. We are considering the worst case that the set  $S_1$  is empty.

Let  $u_1, u_2, u_3, \dots, u_c$  be the vertices chosen in while loop for placing cops. When the cop position  $u_i$  is chosen, as many as  $m_{u_i}$  elements are deleted from  $S_2, S_3, \dots, S_{n_1}$  and these elements form the set  $GU(u_i)$ . For each deleted element  $x$  we assign a value called  $val(x)$ , where  $val(x) = \frac{1}{m_{u_i}}$ . Each element of  $S_i$  is

deleted once and only once and we keep on deleting until all  $S_i$  sets become empty. Let  $X$  be the set of vertices chosen to place cops in *Algorithm 1* i.e.,  $X = \{u_1, u_2, \dots, u_c\}$  and let  $X^*$  be the set of vertex positions of optimal solution when robber is at center vertex of  $G_R$ .

Now we describe the procedure to compute the set  $GU(v), \forall v \in X^*$ . Let  $v_1$  be the center vertex of  $G_R$  and the robber moves from  $v_1$  to  $v_j$ . Now some of the cops at positions  $X^*$  move to positions  $S_j$ . If the cop at vertex  $v \in X^*$  moves to  $x \in S_j$  then add  $x_j$  to  $GU(v)$ .

$|X|$  is the solution given by *Algorithm 1* and  $|X^*|$  is the optimal solution. We have,

$$\sum_{i=2}^{n_1} \sum_{x \in S_i} val(x) = c = |X|. \tag{1}$$

Now consider the optimal solution  $X^*$ . For all  $x \in S_j$ , where  $2 \leq j \leq n_1$ , there exists a cop position  $v \in X^*$  such that  $x \in GU(v)$ , so

$$\sum_{v \in X^*} \sum_{x \in GU(v)} val(x) \geq \sum_{i=2}^{n_1} \sum_{x \in S_i} val(x) = |X|. \tag{2}$$

We will prove that  $\sum_{x \in GU(v)} val(x)$  is upper bounded by  $H(n_1)$ .

Define  $\alpha_i$  as: for any vertex  $v \in V_C$ , consider any  $GU(v)$ ,  $\alpha_i$  is the number of elements left unguarded in  $GU(v)$  after  $i^{th}$  cop position has been selected by *Algorithm 1*. Let  $\alpha_i^*$  is the  $\alpha_i$  value of  $v \in X^*$ . So for every  $v \in X^*$  we have

$$\sum_{x \in GU(v)} val(x) \leq \sum_{i=1}^k (\alpha_{i-1}^* - \alpha_i^*) \frac{1}{|GU(u_i)|} \leq \sum_{i=1}^k (\alpha_{i-1}^* - \alpha_i^*) \frac{1}{\alpha_{i-1}^*}$$

since the algorithm chooses  $u_i$  such that  $|GU(u_i)|$  is maximum; in particular,  $|GU(u_i)| \geq \alpha_{i-1}^*$ . By using the proof similar to the proof of approximation bounds for SET COVER problem given in [5], we obtain

$$|X| \leq |X^*| \cdot H(n_1) \tag{3} \quad \square$$

By modifying *Algorithm1* we can obtain Approximation algorithms for guarding against clique and wheel. For detailed algorithms and proofs please refer [4].

### Acknowledgments

The authors thank the anonymous referees for their suggestions. We also thank Ananth R and Srivatsan Narayanan for several useful discussions.

## References

1. Fomin, F.V., Golovach, P.A., Hall, A., Mihalák, M., Vicari, E., Widmayer, P.: How to guard a graph? In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 318–329. Springer, Heidelberg (2008)
2. Nagamochi, H.: Cop-robber guarding game with cycle robber region. In: FAW 2009. LNCS, vol. 5598, pp. 74–84. Springer, Heidelberg (2009)
3. Papadimitriou, C.M.: Computational complexity. Addison-Wesley, Reading (1994)
4. Reddy, T., Krishna, S., Rangan, P.: The guarding problem - complexity and approximation (2009), [http://www.cse.iitm.ac.in/~tiru/tiru/Publications\\_files/iwoca-1.pdf](http://www.cse.iitm.ac.in/~tiru/tiru/Publications_files/iwoca-1.pdf)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)

# Antibandwidth of $d$ -Dimensional Meshes

Lubomír Török<sup>1,\*</sup> and Imrich Vrto<sup>2,\*\*</sup>

<sup>1</sup> Institute of Mathematics and Computer Science  
Slovak Academy of Sciences and Matej Bel University  
Dúmbierska 1, 974 11, Banská Bystrica, Slovak Republic  
torok@savbb.sk

<sup>2</sup> Institute of Mathematics, Slovak Academy of Sciences  
Dúbravská 9, 841 04 Bratislava, Slovak Republic  
vrto@savba.sk

**Abstract.** The antibandwidth problem is to label vertices of a graph  $G = (V, E)$  bijectively by  $0, 1, 2, \dots, |V| - 1$  such that the minimal difference of labels of adjacent vertices is maximised. In this paper we discuss the antibandwidth of  $d$ -dimensional meshes. We provide labelling algorithm giving antibandwidth value matching the upper bound up to the third order term. This work is a continuation of our previous results for antibandwidths of two and three-dimensional meshes and hypercubes.

## 1 Introduction

The antibandwidth problem is to label vertices of a graph  $G = (V, E)$  bijectively by  $0, 1, 2, \dots, |V| - 1$  such that the minimal difference of labels of adjacent vertices is maximised. The problem was originally introduced in [15] in a connection with multiprocessor scheduling problems. It can be understood as a dual problem to the well known bandwidth problem [10] in which the maximum distance of adjacent vertices in the linear layout is minimised. Another motivation comes from the area of radio frequencies assignment problem [13] and obnoxious facility location problem [7]. According to a survey [11] this problem is another labelling problem. The problem was originally studied under the term *separation number* [15]. In the meantime, this name was also used for another linear layout problem [10]. Lin and Yuan called it *dual bandwidth*. In our older paper [18] we proposed a new term for the problem (the most appropriate according to our opinion), *antibandwidth*.

The antibandwidth problem is NP-complete [15]. So far it is polynomially solvable for 3 classes of graphs: the complements of interval, arborescent comparability and threshold graphs [9,14]. Known results include simple relations of the antibandwidth invariant to the minimum, maximum degree, chromatic index and powers of hamiltonian paths in the complement graph [15,16,17]. Exact results are known for paths, cycles, special trees, complete and complete bipartite

---

\* Supported by the the VEGA grant No. 1/0722/08 and APVV grant No. 51-009605.

\*\* Supported by the APVV grant No. 0433/06 and the VEGA grant No. 2/0111/09.

graphs [16,17,22]. The class of  $n$ -vertex forests with  $\text{ab}(F) = \lfloor n/2 \rfloor$  is characterized in [17], which covers, e.g., complete binary trees. The invariant for complete  $k$ -ary trees is discussed in [6,19].

This paper extends our previous results for 2- and 3-dimensional meshes and hypercubes [18,20] where we proved for the two-dimensional mesh  $M_2$  of type  $n \times n$

$$\text{ab}(M_2) = \frac{n^2}{2} - \frac{n}{2}.$$

For the three-dimensional mesh  $M_3$  of type  $n \times n \times n$

$$\text{ab}(M_3) = \frac{n^3}{2} - \frac{3n^2}{8} + O(n),$$

and for the  $d$ -dimensional hypercube  $Q_d$

$$\text{ab}(Q_d) = 2^{d-1} - \frac{2^d}{\sqrt{\pi d}}(1 + o(1)).$$

Note that this value has been recently improved [21] to

$$\text{ab}(Q_d) = 2^{d-1} - \sum_{m=0}^{d-2} \binom{m}{\lfloor \frac{m}{2} \rfloor}.$$

For the  $d$ -dimensional mesh  $M_d$  of type  $n \times n \times \dots \times n$  we show:

**Theorem 1.** *For  $n \rightarrow \infty$  and  $d \geq 3$  fixed*

$$\text{ab}(M_d) = \frac{n^d}{2} - c_d \frac{n^{d-1}}{2\sqrt{d}}(1 + o(1))$$

where  $c_d = \frac{\sqrt{d}}{2^{d-1}(d-1)!} \sum_{j=0}^{\lfloor d/2 \rfloor} (-1)^j \binom{d}{j} (d-2j)^{d-1}$ , and  $1/2 \leq c_d \leq 2\sqrt{\pi}$ .

## 2 Preliminaries

Let  $P_n$  be an  $n$  vertex path on vertices  $\{0, 1, 2, \dots, n-1\}$  with edges  $\{(i, i+1) | i = 0, 1, 2, \dots, n-1\}$ . Define the  $d$ -dimensional mesh  $M_d = \prod_{i=1}^d P_n$  by means of cartesian product. The diameter of  $M_d$  is  $d(n-1)$  and it has  $n^d$  vertices.

For  $r = 0, 1, 2, \dots, d(n-1)$ , let  $B(r)$  denote the set of vertices of  $M_d$  in the distance  $r$  from  $(0, 0, 0, \dots, 0)$ . It is known [2] that for  $r = 0, 1, 2, \dots, d(n-1)$

$$|B(r)| = |B(d(n-1) - r)|. \tag{1}$$

Moreover for fixed  $d$  and  $n$ ,  $|B(r)|$  is strictly increasing for  $r \leq \lfloor d(n-1)/2 \rfloor$  and strictly decreasing for  $r \geq \lceil d(n-1)/2 \rceil$ . Further we claim that

$$\sum_{r=0}^{d(n-1)} (-1)^r |B(r)| = (n \bmod 2). \tag{2}$$



First observe that

$$V_1 = \bigcup_{\substack{r=0 \\ r \text{ even}}}^{d(n-1)} B(r).$$

Hence (2) is equivalent to  $|V_1| - |V_2| = n \pmod 2$ . We have  $V_1 = \cup_{i=0}^{n-1} Z_i$ , where  $Z_i = \{(x_1, x_2, \dots, x_{d-1}, i) : (x_1 + x_2 + \dots + x_{d-1} + i) \pmod 2 = 0\}$ . Realize, that  $|Z_i| = \lceil n^{d-1}/2 \rceil$ , for even  $i$  and  $\lfloor n^{d-1}/2 \rfloor$ , otherwise. Then

$$|V_1| = \left\lceil \frac{n}{2} \right\rceil \left\lceil \frac{n^{d-1}}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{n^{d-1}}{2} \right\rfloor = \left\lceil \frac{n^d}{2} \right\rceil,$$

which proves the claim.

The number of vertices in the "central slice", i.e.  $|B(\lfloor d(n-1)/2 \rfloor)|$  is of special interest. It was intensively studied in the past, see [8]. For our purposes we use an estimation mentioned in Akhtar et al. [1]: For  $n \rightarrow \infty$  and  $d \geq 3$  fixed

$$|B(\lfloor d(n-1)/2 \rfloor)| = c_d \frac{n^{d-1}}{\sqrt{d}} + O(n^{d-2}) \tag{3}$$

where  $c_d$  is defined by

$$c_d = \frac{\sqrt{d}}{2^{d-1}(d-1)!} \sum_{j=0}^{\lfloor d/2 \rfloor} (-1)^j \binom{d}{j} (d-2j)^{d-1}, \text{ and } 1/2 \leq c_d \leq 2\sqrt{11}.$$

Define a simplicial order on  $M_d$  according to [4]. Let  $x = (x_1, x_2, x_3, \dots, x_d)$  and  $y = (y_1, y_2, y_3, \dots, y_d)$ . Then  $x < y$  if either  $\sum x_i < \sum y_i$ , or  $\sum x_i = \sum y_i$ , and for some  $j$  we have  $x_j > y_j$  and  $x_i = y_i$  for all  $i < j$ .

Let  $V_1, V_2$  be a partitions of a vertex set of a bipartite graph  $G = (V, E)$  such that  $V_1 \cup V_2 = V, V_1 \cap V_2 = \emptyset$ . Minimal bipartite vertex boundary  $\partial_b(A)$  of a set  $A \subseteq V_1$  is a set of all vertices from  $V_2$  having neighbour in  $V_1$ .

### 3 Upper Bound Proof

Assume  $n \equiv 1 \pmod 4$ . Other cases are similar. Denote  $t = d(n-1)/4$ . In this case (1) and (2) imply

$$|B(0)| + |B(2)| + |B(4)| + \dots + |B(2t-2)| + \frac{1}{2}|B(2t)| = \frac{n^d}{4}. \tag{4}$$

Consider an optimal linear layout of  $M_d$ . Denote  $k = \text{ab}(M_d)$ . If

$$k \leq \frac{n^d - |B(2t)|}{2} + 2,$$

then we are done. Suppose indirectly that

$$k \geq \frac{n^d - |B(2t)|}{2} + 3,$$

We know that  $k < n^d/2$ . Let  $S$  be a set of consecutive  $k$  vertices on the line. Denote  $A_i = V_i \cap S$ , for  $i = 1, 2$ . Denote by  $J = [L, R]$ , an interval with

$$L = \frac{n^d}{4} - \frac{1}{2}|B(2t)| + 1, R = \frac{n^d}{4}.$$

Distinguish two cases.

**Case 1.** Assume that there exist  $S$  such that the corresponding  $A_1$  satisfies  $|A_1| \in J$ . Observe that  $\partial_b(A_1) \cap A_2 = \emptyset$  and  $\partial_b(A_1) \cup A_2 \subseteq V_2$ . Then

$$|\partial_b(A_1)| + |A_2| = |\partial_b(A_1)| + k - |A_1| \leq |V_2|.$$

Hence

$$k \leq \frac{n^d - 1}{2} - (|\partial_b(A_1)| - |A_1|).$$

In what follows we will show that  $|\partial_b(A_1)| - |A_1| \geq |B(2t)|/2 - 2$ , which immediately gives a contradiction. The equation (4) implies that

$$|A_1| = |B(0)| + |B(2)| + |B(4)| + \dots + |B(2t - 2)| + \alpha|B(2t)|,$$

for some constant  $0 < \alpha \leq 1/2$ . Let  $I_1$  be the set of the first  $|A_1|$  vertices from  $V_1$  in the simplicial order. Bezrukov and Piotrowski [3] proved that

$$|\partial_b(A_1)| \geq |\partial_b(I_1)|. \tag{5}$$

Let  $A \in V_1 \cup V_2$  be a set of cardinality

$$|A| = |B(0)| + |B(1)| + |B(2)| + \dots + |B(2t - 1)| + \alpha|B(2t)|.$$

Let  $I$  be the set of the first  $|A|$  vertices from  $V_1 \cup V_2$  in the simplicial order. Bollobás and Leader [4] proved

$$|\partial_b(A)| \geq |\partial_b(I)|. \tag{6}$$

From the definitions of  $I_1$  and  $I$  we have

$$|\partial_b(I_1)| = |B(1)| + |B(3)| + |B(5)| + \dots + |B(2t - 1)| + |\partial(I)| - (1 - \alpha)|B(2t)| \tag{7}$$

We claim that:

$$|\partial(I)| \geq (1 - \alpha)|B(2t)| + \alpha|B(2t + 1)|. \tag{8}$$

Denote  $P = \lceil (1 - \alpha)|B(2t)| \rceil, Q = \lceil \alpha|B(2t + 1)| \rceil$ . To prove the claim we have to show that there exists  $P$  vertices in  $B(2t)$  having at least one neighbour in  $B(2t - 1)$  and that there exists  $Q$  vertices in  $B(2t + 1)$  having at least one neighbour in  $B(2t)$ . We find the neighbours using the following construction. Consider the simplicial order of vertices in  $B(2t)$ . For every  $v(x_1, x_2, \dots, x_d) \in B(2t)$  find  $j = \min(i) : x_i > 0$ . Then its neighbour in  $B(2t - 1)$  has coordinates  $(0, 0, \dots, x_j - 1, \dots, x_d)$ . We find the neighbours of vertices in  $B(2t + 1)$  in a similar way. From the definition of the mesh for every  $v \in B(k), k = 1, 2, \dots, d(n - 1)$  there exists at least one neighbour in  $B(k - 1)$ ; what proves the claim.

Combining (5), (6), (7) and (8) we obtain

$$\begin{aligned}
 |\partial_b(A_1)| - |A_1| &\geq |\partial_b(I_1)| - (|B(0)| + |B(2)| + \dots + |B(2t-2)| + \alpha|B(2t)|) \\
 &\geq \sum_{i=0}^{2t-1} (-1)^{i+1} |B(i)| - \alpha(|B(2t)| - |B(2t+1)|) \\
 &= -\frac{1}{2} + \frac{1}{2}|B(2t)| - \alpha(|B(2t)| - |B(2t+1)|) \\
 &\geq \frac{1}{2}|B(2t)| - \frac{3}{2}.
 \end{aligned}$$

**Case 2.** Assume that for all sets  $S$ ,  $|A_1| \notin J$ . We show that this leads to a contradiction. Distinguish 3 cases.

1. There exist  $S, S'$  such that  $|A_1| < L$  and  $|A'_1| > R$ . Then one can easily compute that  $|A_1| < |A_2|$  and  $|A'_1| > |A'_2|$ . This forces the existence of  $S''$  such that  $||A''_1| - |A''_2|| \leq 1$  which gives

$$\frac{k-1}{2} \leq |A''_1| \leq \frac{k+1}{2}.$$

Comparing this interval with  $J$  we conclude that  $|A''_1| \in J$ , a contradiction.

2. Assume that for all  $S$ ,  $|A_1| < L$ . Then  $|A_2| = k - |A_1| > k - L$ . Take two disjoint  $S$  and  $S'$  with  $|A_2| > k - L$  and  $|A'_2| > k - L$ . We get  $|A_2| + |A'_2| > 2k - 2L \geq \frac{n^d}{2} + 4$  a contradiction.
3. Assume that for all  $S$ ,  $|A_1| > R$ . This case is symmetric to the previous case.

Finally, for  $|B(\lfloor d(n-1)/2 \rfloor)|$  use the estimation from (3).

### 4 Lower Bound

The labeling of vertices of  $M_d$  proceeds in two phases. In the first phase we label sets  $B(r)$ , for even  $r$  as follows: The single vertex in  $B(0)$  gets label 0, the vertices of  $B(2)$  are labelled by  $1, 2, 3, \dots, |B(2)|$ , the vertices in  $B(4)$  by  $|B(2)| + 1, \dots, |B(2)| + |B(4)|$  and so on. The labelling in every  $B(r)$  is given by the induced simplicial order, i.e.,  $x = (x_1, x_2, \dots, x_d)$  precedes  $y = (y_1, y_2, \dots, y_d)$  if for some  $j$  we have  $x_j > y_j$  and  $x_i = y_i$  for all  $i < j$ . In the second phase we label  $B(r)$ , for odd  $r$ , similarly. Now we analyze the labeling algorithm.

Let  $d(n-1)$  be divisible by four. The remaining cases are similar. Let  $r$  be even, the odd case is similar. Let  $v_1$  and  $v_2$  be two neighbouring vertices, such that  $v_1 \in B(r)$  and  $v_2 \in B(r-1)$ . Due to symmetry, we may assume that  $r \leq d(n-1)/2$ . Then

$$\begin{aligned}
 f(v_1) &= |B(0)| + |B(2)| + \dots + |B(r-2)| + c(v_1) \\
 f(v_2) &= \frac{n^d}{2} + |B(1)| + |B(3)| + \dots + |B(r-3)| + c(v_2)
 \end{aligned}$$

where  $c(v_1)$  and  $c(v_2)$  stand for ranks of  $v_1$  and  $v_2$  in simplicial orders in  $B(r)$  and  $B(r - 1)$  respectively. Clearly  $f(v_2) > f(v_1)$ . Compute

$$f(v_2) - f(v_1) = \frac{n^d}{2} - B(0) + |B(1)| - |B(2)| + \dots + |B(r - 3)| - |B(r - 2)| + c(v_2) - c(v_1). \tag{9}$$

Further, we estimate the term  $c(v_2) - c(v_1)$ . Let  $g$  be the simplicial labelling of vertices of  $M_d$ . Then

$$g(v_1) = |B(0)| + |B(1)| + \dots + |B(r - 1)| + c(v_1),$$

$$g(v_2) = |B(0)| + |B(1)| + \dots + |B(r - 2)| + c(v_2),$$

which gives

$$g(v_2) - g(v_1) = -|B(r - 1)| + c(v_2) - c(v_1)$$

We use the following fact proved in [1] (Lemma 4.4, last line of the proof):

$$|g(v_2) - g(v_1)| \leq 2|B_{d-1}(\lfloor (d - 1)(n - 1)/2 \rfloor)| - 1 + |B(r)|$$

where  $|B_{d-1}(r)|$  stands for the value of  $|B(r)|$  in the mesh of dimension  $d - 1$  and it is known [1] that  $|B_{d-1}(\lfloor (d - 1)(n - 1)/2 \rfloor)| = O(n^{d-2}/\sqrt{d-1})$ . Hence

$$c(v_2) - c(v_1) \geq g(v_2) - g(v_1) + |B(r - 1)|$$

$$\geq -2|B_{d-1}(\lfloor (d - 1)(n - 1)/2 \rfloor)| + 1 - |B(r)| + |B(r - 1)|$$

$$= O\left(\frac{n^{d-2}}{\sqrt{d-1}}\right) - |B(r)| + |B(r - 1)|.$$

Substituting into (9) we get

$$f(v_2) - f(v_1) = \frac{n^d}{2} - |B(0)| + |B(1)| - |B(2)| + \dots + |B(r - 1)| - |B(r)| + O\left(\frac{n^{d-2}}{\sqrt{d-1}}\right).$$

Because  $|B(i)|$  is increasing till  $d(n - 1)/2$  and then decreasing, the alternating sum of  $|B(i)|$ 's achieves its minimum when  $r = r^* = d(n - 1)/2$ . We have

$$2(-|B(0)| + |B(1)| - |B(2)| + \dots + |B(r^* - 1)| - |B(r^*)|)$$

$$= - \sum_{r=0}^{d(n-1)} (-1)^r |B(r)| - |B(r^*)| = -n \bmod 2 - |B(r^*)|$$

Substituting this back we get the minimal difference of two labels in this labelling:

$$f(v_2) - f(v_1) = \frac{n^d}{2} - \frac{B(\lfloor d(n - 1)/2 \rfloor)}{2} + O\left(\frac{n^{d-2}}{\sqrt{d-1}}\right).$$

Where the estimation for  $B(\lfloor d(n - 1)/2 \rfloor)$  is given by (3).

## References

1. Akhtar, R., Jiang, T., Miller, Z.: Asymptotic determination of edge-bandwidth of multidimensional grids and Hamming graphs. *SIAM Journal on Discrete Mathematics* 22, 425–449 (2008)
2. Anderson, I.: *Combinatorics of Finite Sets*. Oxford University Press, New York (1987)
3. Bezrukov, S.L., Piotrowski, V.P.: Shadows and isoperimetry in bipartite graphs. Presented at SIAM Conference on Discrete Mathematics, Minneapolis (2000)
4. Bollobás, B., Leader, I.: Compressions and isoperimetric inequalities. *Journal of Combinatorial Theory A* 56, 47–62 (1991)
5. Bollobás, B., Leader, I.: Matchings and paths in the cube. *Discrete Applied Mathematics* 75, 1–8 (1997)
6. Calmoneri, T., Massini, A., Török, L., Vrt'ó, I.: Antibandwidth of complete k-ary trees. *Electronic Notes in Discrete Mathematics* 24, 259–266 (2006)
7. Cappanera, P.: A survey on obnoxious facility location problems. Technical Report TR-99-11, Dipartimento di Informatica, Uni. di Pisa (1999)
8. Chakerian, D., Logothetti, D.: Cube slices, pictorial triangles, and probability. *Mathematics Magazine* 64, 219–241 (1991)
9. Donnelly, S., Isaak, G.: Hamiltonian powers in threshold and arborescent comparability graphs. *Discrete Mathematics* 202, 33–44 (1999)
10. Díaz, J., Petit, J., Serna, M.: A survey of graph layout problems. *ACM Computing Surveys* 34, 313–356 (2002)
11. Gallian, J.A.: Graph labeling. *Electronic Journal on Combinatorics*, DS6 (2005)
12. Göbel, F.: The separation number. *Ars Combinatoria* 37, 262–274 (1994)
13. Hale, W.K.: Frequency assignment: theory and applications. *Proceedings of IEEE* 60, 1497–1514 (1980)
14. Isaak, G.: Powers of Hamiltonian paths in interval graphs. *J. Graph Theory* 28, 31–38 (1998)
15. Leung, J.Y.-T., Vornberger, O., Witthoff, J.D.: On some variants of the bandwidth minimization problem. *SIAM J. Computing* 13, 650–667 (1984)
16. Yixun, L., Jiang, Y.J.: The dual bandwidth problem for graphs. *J. Zhengzhou Univ. Nat. Sci. Ed.* 35, 1–5 (2003)
17. Miller, Z., Pritikin, D.: On the separation number of a graph. *Networks* 19, 651–666 (1989)
18. Raspaud, A., Schröder, H., Sýkora, O., Török, L., Vrt'ó, I.: Antibandwidth and cyclic antibandwidth of meshes and hypercubes. *Discrete Mathematics* 309, 3541–3552 (2009)
19. Jiang, T., Miller, Z., Pritikin, D.: Separation numbers of trees. *Theoretical Computer Science* (to appear)
20. Török, L., Vrt'ó, I.: Antibandwidth of three-dimensional meshes. *Discrete Mathematics* (to appear)
21. Wang, X., Wu, X., Dumitrescu, S.: On explicit formulas for bandwidth and antibandwidth of hypercubes. *Discrete Applied Mathematics* 157, 1947–1952 (2009)
22. Weili, Y., Ju, Z., Xiaoxu, L.: Dual bandwidth of some special trees. *J. Zhengzhou Univ. Nat. Sci. Ed.* 35 (2003)
23. Yu, P.: The separation numbers of two classes of graphs. *J. Changsha Univ. Electr. Power Nat. Sci. Ed.* 12, 421–426 (1997)

# Author Index

- Amir, Amihood 36  
Apollonio, Nicola 48  
Aravind, N.R. 60
- Bannai, Hideo 312  
Bessy, Stéphane 72  
Bourgeois, Nicolas 83  
Broersma, Hajo 95, 105  
Bui-Xuan, Binh-Minh 113
- Campanelli, Matteo 230  
Cantone, Domenico 230  
Caramia, Massimiliano 48  
Cibulka, Josef 356  
Cichacz, Sylwia 125  
Colbourn, Charles J. 408  
Coudert, David 134  
Crespelle, Christophe 146  
Crochemore, Maxime 158
- D'Angelo, Gianlorenzo 24  
D., Sai Krishna 460  
Darrasse, Alexis 170  
Deguchi, Satoshi 312  
Dimitrov, Darko 183  
Di Stefano, Gabriele 24  
Drmot, Michael 194, 206  
Dvořák, Tomáš 183
- Edmonds, Jack 1  
Ekim, Tinaz 218
- Faro, Simone 230  
Feierl, Thomas 242  
Fellows, Michael 2  
Fomin, Fedor V. 95  
Fredriksson, Kimmo 254  
Fronček, Dalibor 125
- Gagie, Travis 278  
Gambette, Philippe 146  
Ganian, Robert 266  
Gawrychowski, Paweł 278  
Giroire, Frédéric 134  
Gittenberger, Bernhard 206
- Golovach, Petr A. 95  
Gotthilf, Zvi 36  
Grabowski, Szymon 254  
Gregor, Petr 183  
Gutwenger, Carsten 289
- Heggernes, Pinar 218  
Hliněný, Petr 266  
Holub, Jan 420  
Hung, Ling-Ju 301
- I, Tomohiro 312  
Ilie, Lucian 158  
Iliopoulos, Costas S. 158  
Inenaga, Shunsuke 312
- Klešč, Marián 324  
Kloks, Ton 301  
Konjevod, Goran 408  
Korpelainen, Nicholas 385  
Kovář, Petr 125, 334  
Kratsch, Dieter 105  
Kubesa, Michael 334  
Kubica, Marcin 158, 345  
Kutzelnigg, Reinhard 206  
Kynčl, Jan 356
- Lappas, Evaggelos 368  
Lee, Chuan Min 301  
Lokshtanov, Daniel 380  
Lozin, Vadim V. 385  
Lucarelli, Giorgio 83
- Manuel, Paul 396  
Manzini, Giovanni 11  
Meister, Daniel 218  
Mészáros, Viola 356  
Milis, Ioannis 83  
Mutzel, Petra 289
- Navarra, Alfredo 24  
Nayeri, Peyman 408  
Nikolopoulos, Stavros D. 368
- Palios, Leonidas 368  
Paschos, Vangelis Th. 83

- Paul, Christophe 72  
Paulusma, Daniël 95  
Perez, Anthony 72  
Procházka, Petr 420  
Punitha, Joice 396
- Radoszewski, Jakub 345  
Rajan, Bharati 396  
Rajasingh, Indra 396  
Rangan C., Pandu 460  
Reddy T.V., Thirumala 460  
Rytter, Wojciech 158, 345
- Sau, Ignasi 134  
Saurabh, Saket 380  
Schiermeyer, Ingo 432  
Shalom, B. Riva 36  
Sikdar, Somnath 380  
Škoda, Petr 450  
Škrekovski, Riste 183
- Soria, Michèle 170  
Stolař, Rudolf 356  
Subramanian, C.R. 60  
Szymańska, Edyta 438
- Takeda, Masayuki 312  
Telle, Jan Arne 113  
Thomas, Robin 19  
Török, Lubomír 471
- Valtr, Pavel 356  
Vatshelle, Martin 113  
Vrto, Imrich 471
- Waleń, Tomasz 158, 345  
Whitesides, Sue 20  
Woeginger, Gerhard J. 105
- Zey, Bernd 289