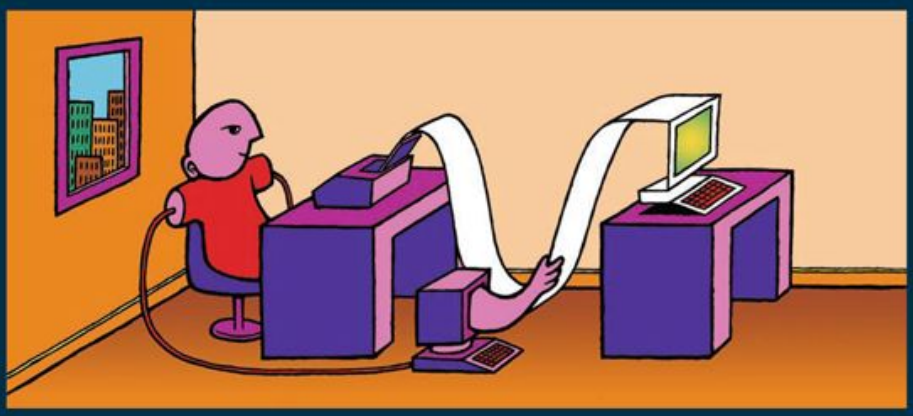
Festschrift

LNCS 5420

Marina Lipshteyn
Vadim E. Levit
Ross M. McConnell (Eds.)

Graph Theory, Computational Intelligence and Thought

Essays Dedicated to Martin Charles Golumbic on the Occasion of His 60th Birthday



Springer

Lecture Notes in Computer Science 5420

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

Marina Lipshteyn Vadim E. Levit
Ross M. McConnell (Eds.)

Graph Theory, Computational Intelligence and Thought

Essays Dedicated to Martin Charles Golumbic on the Occasion of His 60th Birthday

Springer

Volume Editors

Marina Lipshteyn
University of Haifa, The Caesarea Rothschild Institute
Mount Carmel, Haifa 31905, Israel
E-mail: marinal@cs.haifa.ac.il

Vadim E. Levit
Ariel University Center of Samaria
Department of Computer Science and Mathematics
Ariel 40700, Israel
E-mail: levitv@ariel.ac.il

Ross M. McConnell
Colorado State University, Department of Computer Science
Fort Collins, CO 80523-1873, USA
E-mail: rmm@cs.colostate.edu

The illustration appearing on the cover of this book is the work of Daniel Rozenberg (DADARA)

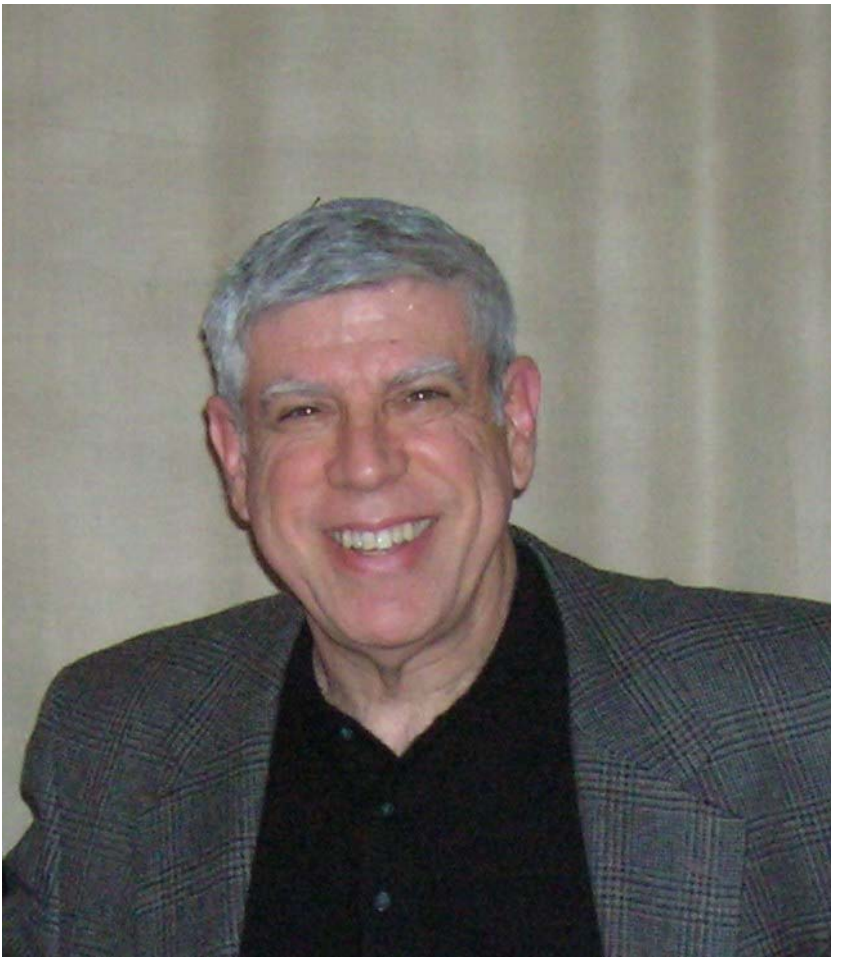
Library of Congress Control Number: 2009930965

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

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

ISSN 0302-9743
ISBN-10 3-642-02028-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-02028-5 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12684137 06/3180 5 4 3 2 1 0



Martin Charles Golumbic

Dedications in honor of Martin Golumbic in the occasion of his 60th birthday

"In 1985, as a newcomer in graphs and algorithms, I became aware of Marty's wonderful monograph "Algorithmic Graph Theory and Perfect Graphs", and in the same year I was happy enough to get a copy of his monograph (which was not so easy in East Germany at that time) which tremendously influenced my work and interests on graph classes and algorithms. A variety of Marty's papers such as results on chordal bipartite graphs and variants, on intersection graphs, on clique-width of distance-hereditary graphs and of unit interval graphs, on induced matchings and other topics directly and also indirectly inspired and motivated a great number of my papers. Marty has set milestones in various fields of research."

Andreas
Brandstädt

Marty Golumbic

Like many others, I first knew of Marty through his book, "Algorithmic Graph Theory and Perfect Graphs" which I read as a graduate student. It is a remarkable book for its organization, the clarity of exposition and its shelf life. It is still a useful reference today. The book is memorable in part because it is sprinkled with just the right amount of Marty's quirky sense of humor. What is especially impressive is how young Marty was when he wrote this book; he was only 32 when it first appeared in print.

It was a great honor and privilege for me to work with Marty on our book "Tolerance Graphs," published in 2004. We began work in 1999 when I made a three-week visit to Israel. After this we worked together three or four times a year in North America (Ithaca, Wellesley, New York City, Rutgers, Boca Raton, Toronto) with Marty cheerfully taking on much more than his fair share of travel. Marty's good nature helped make these intense work visits a real pleasure.

I learned so much from working with Marty. He has the patience to painstakingly work through background papers and to rewrite sections for added clarity. Yet he also is pragmatic enough to know when it is time to move on and declare a project finished. He has an abundance of good sense and I have benefited immensely from his advice and encouragement. Working with Marty has been one of the highlights of my professional career. I am so pleased I was able to join Marty and his family, friends and colleagues at his 60th birthday celebration.

Ann Trenk

A Tribute to Marty

I have known Marty for many years, since before he came on Aliya to Israel. In 2001 Marty asked me if I would be the scientific coordinator in the new Interdisciplinary

Research Institute in the University of Haifa, headed by him. I gladly accepted the offer. The Caesarea Edmonds Benjamin de Rothschild Foundation Institute for Interdisciplinary Applications of Computer Science (yes, it is a mouth full- CRI for short) was flourishing in the period 2001-2008. We had more than 100 conferences and workshops, dozens of new interdisciplinary courses, hundreds of visitors, and many research projects dealing with interdisciplinary research applications of computer science. Most projects were within the university, many touched other universities in Israel and abroad, and some touched the community as a whole. I enjoyed working with Marty, and I learned a lot from him. I feel indebted to Marty for giving me this opportunity of contributing to CRI, serving many faculty members, and learning from them. Heading such an institution requires an open mind, genuine interest in cross-disciplinary subjects, vision, political skills, hard work, and more. Marty has all these traits, and he made CRI what it is. Kol- Hakavod!

Irith Hartman

Marty Golumbic's 1980 book, Algorithmic Graph Theory and Perfect Graphs, has been largely responsible for fostering an entire community of researchers who work on the rich field of problems suggested by the fascinating work he describes there. I came across the book in 1992 when I was getting my doctorate, and I have used the open questions suggested by the book as a roadmap for my entire subsequent research career, as have many other researchers. This work, and other papers and monographs of his on structured classes of graphs have been enormously influential and remain so.

When my copy of his 1980 book fell to tatters a few years ago because of many years of reading, carrying it in suitcases, and lending it to friends and students, I was dismayed to find that it was out of print. People were selling used copies on the Internet for many times the original selling price. Fortunately, printing of this classic text is back by popular demand.

Marty has also cultivated the interest of many people in the field by organizing conferences and taking a long-term and selfless interest in the success of newcomers and the vitality of our research community. His talk at the conference about the highlights of his career brought back many memories for those of us who have known him for many years.

Ross McConnell

I first met Marty as an undergraduate student in his graph theory course. I was charmed by the way Marty taught graph theory. It inspired my curiosity and interest and challenged me intellectually. His influence gave me the motivation to complete the doctoral studies which he advised me to pursue a few years later. Even though I worked very hard, it felt like playing a mathematical game. Graph theory is not all I learned from Martin; I also learned precious life values – broad vision, approaches to problem solving, team work and multidicplinarity. Marty's guidance will continue to influence my life and career for many years to come.

Marina Lipshteyn

For me it all began when Martin invited me to participate in one of his seminars in Bar-Ilan University about nine years ago. Back then it was to be invited by "the Professor" who wrote "the book": Algorithmic Graph Theory and Perfect Graphs.

Martin introduced me to his students and also to his colleagues. I felt honored and I still do. At these seminars I learned so much and grew fond of the subjects of VPT, EPT and intersection graphs in general. Then Martin moved to the University of Haifa to be the director of CRI. When I finished my Ph.D., Martin took me into CRI under his wide wing. It became a warm and welcoming place to come to and work once a week. I treasure the long hours we work together. Thank you so much, with all my heart, Happy Birthday.

Michal Stern

I was delighted to be able to participate in the celebration of Martin Charles Golumbic's sixtieth birthday, held in September, 2008, in Jerusalem, Tiberias and Haifa. It was a wonderful celebration of a man who has succeeded spectacularly on a number of fronts. When I first met Marty thirty years ago, he had completed a doctorate in mathematics under one of the leading twentieth century mathematicians, Sammy Eilenberg, and had published a book in a prestigious research series, Algorithmic Graph Theory and Perfect Graphs, which was on the cusp of mathematics and computer science, and the first of several important works. I met him when he attended the Southeastern International Conference on Combinatorics, Graph Theory and Computing at Florida Atlantic University. He has been back many times since then, with repeated appearances as an invited plenary speaker, and as an organizer of special sessions.

Marty is an extremely creative and versatile mathematical scientist. He has advanced established research areas, and begun new ones. He has served as an editor for highly respected journals, and is the founder of a new journal. I have been privileged to work with him as a member of the editorial board of his Annals of Mathematics and Artificial Intelligence, and as a co-organizer of the series of biennial International Symposia on Artificial Intelligence and Mathematics. He is a delightful person to work with, and an excellent leader. Marty has been successful in both industry and academics, and as the founder and director of an outstanding research institute. The administration of our university was excited to be able to enter into a cooperative arrangement with the Caesarea Edmond Benjamin de Rothschild Institute for Interdisciplinary Applications of Computer Science.

Marty is a great person, with a wonderful, loving family. He is a true friend, and a brilliant thinker. We look forward to many more accomplishments from him as he enters the prime of his life.

Fred Hoffman

Preface

This volume is dedicated to Martin Charles Golumbic on the occasion of his 60th birthday. Professor Golumbic has been making seminal contributions to algorithmic graph theory and artificial intelligence throughout his career. He is universally admired as a long-standing pillar of the discipline of computer science. To honor this event, many of Martin's graduate students, research collaborators, and computer science colleagues gathered in Israel for a conference on subjects related to Martin's manifold contributions in the field.

The conference, "Graph Theory, Computational Intelligence and Thought" was held in Jerusalem, Tiberias and Haifa, Israel during September 19–25, 2008. The conference was organized by Irith Ben-Arroyo Hartman, Shlomo Kipnis and Michal Stern. Local arrangements were coordinated by CRI staff members Rona Perkis, Avital Berkovich, Orly Ross, George Karapetyan and graduate students Hananel Hazan and Elad Cohen. Their help was instrumental in the success of the event and is most gratefully acknowledged.

The meeting received generous support from the following institutions:

- Hadassah College, Jerusalem
- Caesarea Edmond Benjamin de Rothschild Institute, University of Haifa

The 19 refereed papers of this volume have been drawn from the proceedings of the event. A few lectures are not represented; some varied somewhat from the subsequent written contributions; and some contributors to this volume were unfortunately unable to attend the event. All papers have undergone the review process.

January 2009

Marina Lipshteyn
Vadim E. Levit
Ross M. McConnell

Table of Contents

Landmarks in Algorithmic Graph Theory: A Personal Retrospective

Martin Charles Golumbic

Abstract. This is an edited version of the conference lecture delivered by the author in celebration of his 60th birthday. It is intended to be an autobiographical tour through stories, pictures and theorems, suitable for both mathematicians and non-scientists.

1 When I Was Seventeen, It Was a Very Good Year…

Erie, Pennsylvania, on Lake Erie, where I was born and grew up, could be pronounced in Hebrew as "Ir-i" עירי —which coincidentally means "my city". I graduated from Academy High School in 1966; there is a picture of me from the Erie Morning News on June 10th one of five students in a graduating class of 500 chosen to be a commencement speaker.

> *Mankind has thrilled at the prospect of operating in boundless space. But with this thrill has come dread that the advance of nations into space will precipitate a new cycle of conflict. To combat this threat, nations have sought to cooperate with each other, to ensure peace. …*

This is the opening of my high school commencement address, "Cooperation in Space". In many ways, the speech reflects some of the big issues that matter to me—those I speak and write about today—that are not so different from the way they were when I was 17.

I would like to start by thanking my coauthors, whom I have listed here. This has been a long journey, and we will see landmarks in this talk of what I have been involved in over the years. I would like to thank all of these people who have worked and published with me:

> Alexander Belfer, Zeev Ben-Porat, David Bernstein, Anne Berry, Andreas Brandstadt, Mark Buckingham, Elad Cohen, Ido Dagan, Ronen Feldman, Dina Q. Goldin, Clinton F. Goss, Dennis Grinberg, Vladimir Gurvich, Peter L. Hammer, Irith Hartman, Tirza Hirst, Robert Jamison, Haim Kaplan, Hugo Krawczyk, Renu Laskar, Vadim E. Levit, Van Bang Le, Moshe Lewenstein, Marina Lipshteyn, Frederic Maffray, Yishai Mansour, Moshe Markovich, Aviad Mintz, Clyde Monma, Gregory Morel, Ido Nahshon, Uri Peled, Yehoshua Perl, Thomas K. Philips, Ron Pinter, Nicholas Pippenger, Vladimir Rainish, Doron Rotem, Udi Rotics, Edward R. Scheinerman, Uri N. Schild, Ron Shamir, Michal Shindler,

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 1–14, 2009.

Shimon Shrem, Assaf Siani, Michal Stern, Michael Tiomkin, Ann N. Trenk, William T. Trotter, Shalom Tsur, Jorge Urrutia, Elad Verbin, Amir Wassermann

And so here is the first theorem of the day.

Theorem. *The people on Marty's coauthor list have Erdős Number 1, 2 or 3.*

Most of you know that Paul Erdős was a very famous Hungarian mathematician; legendary in our discipline. If you've written a paper with Erdős, your number is 1; if you've written a paper with a person who wrote a paper with Erdős, you're number is 2, and so forth. Since I have Erdős number 2, it follows that everyone on my list will have numbers 1, 2 or 3. That's a proof.

Just one story about Erdős—there are so many stories about him. When our family was visiting in Budapest about 12 years ago on a roots trip, I spent one day—I got permission from Lynn, my wife, to spend one day—at the Math Institute. After returning from lunch with Erdős, Vera Sós and Tamás Turán (György's brother and Vera's son) Erdős was kind of tired, and he drifted off a little bit in the coffee room. It was a Friday, and when I got back to our apartment about an hour before Shabbat, the phone rang and Erdős was on the phone—he apologized for not saying goodbye to me, and he called to wish us a Shabbat Shalom. I think that this says something very special about the character of this man, and the character of a number of mathematicians we have encountered over the years—those who have a very special personal connection with the people in our field.

Frank Sinatra, on his 50th birthday, produced a record album called "September of My Years". We are in September, and I am in the September of my years. This is part of the text of the song that gives its name to the album.

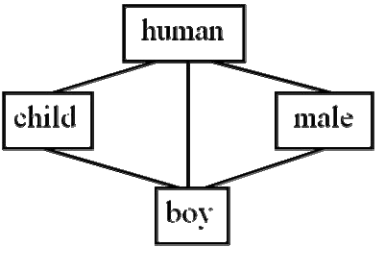
As a man, who has always had the wandering ways
I keep looking back to yesterdays
'til a long forgotten love appears
And I find, I'm sighing softly as I near
September, the warm September of my years.

Lattices and other hierarchies

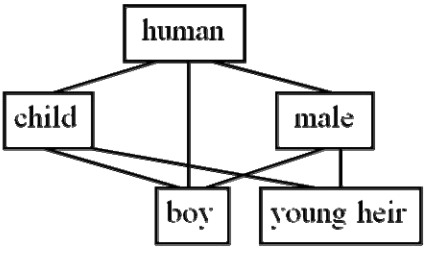
As an undergraduate at the Pennsylvania State University, I learned what mathematics really was. In high school you don't really know. Maybe I still don't know. I did a Bachelor's and a Master's degree simultaneously in my four years at Penn State. My Master's research paper was on the topic "Congruence Relations in Lattices". Let me tell you what a lattice is. A lattice is a kind of hierarchy, like this one: a boy is a male, a boy is a child, a child is a human, and (many would say that) a male is a human. These kinds of ontologies of words you may see in dictionaries, encyclopedias and so on. Mathematicians are also interested in hierarchies like this.

What is a lattice? For the mathematicians, a precise definition: a hierarchy (i.e. a poset) where every pair of nodes has a unique lowest ancestor above it and a unique highest descendant below it.

Consider the two diagrams above. The first one is a lattice but the second one is not. The reason the second is not a lattice is because you can go down from child and



This is a lattice.

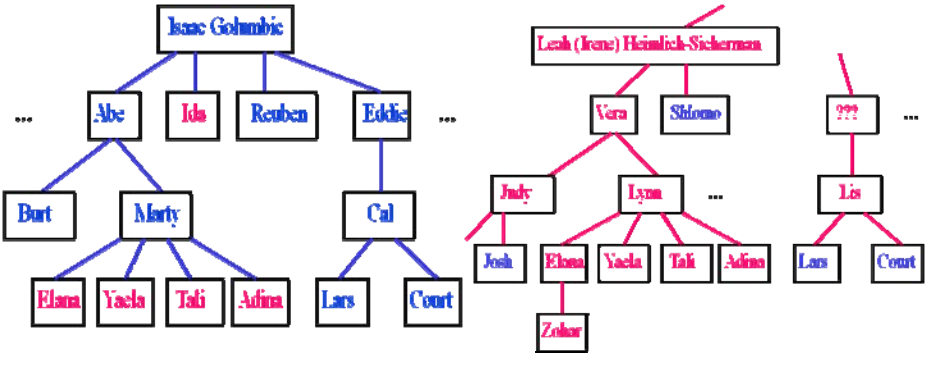


This is not a lattice.

male in two ways, to boy and also to young heir. You might say there's an error in this second diagram, because it is not a lattice, that your hierarchy should really be a little bit different. Maybe the young heir should appear over boy or, in the case of our family, a young heir might be female, then one would really need change the lattice. However, there is a serious side to this type of reasoning, namely, that it may be important either semantically or computationally for you to insist that the structure being used is a lattice.

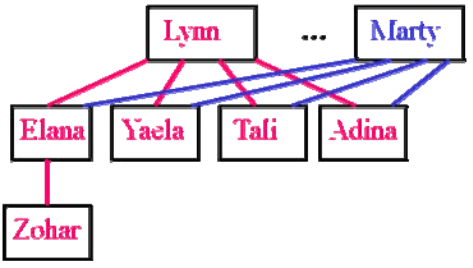
There are other kinds of hierarchies that we are familiar with and one of them is the hierarchy "*Father-of* ". This is a hierarchy whose structure is a rooted tree, as in the figure below, where Isaac Golumbic is the root and under him are four nodes (actually seven if it were complete) where one of those is Abe and another is Eddie. Under Abe you have Burt and Marty; Marty is the father of Elana, Yaela, Tali and Adina. Eddie has one son Cal, and he has two sons named Lars and Court.

There's another hierarchy, that's the "*Mother-of* " hierarchy. Here we have a mother tree with matriarch Leah, mother of Vera and Shlomo. Vera has daughters Judy, Sharona (not visible here) and Lynn; Lis is the mother of Lars and Court, etc. In my lecture, I colored the girls in pink and the boys in blue, and there is a lot of pink in the slide.



What are family trees?

I now want to ask a serious mathematical question, and that is: What is the structure of family trees? It is a question for people who work in posets. For one thing family trees

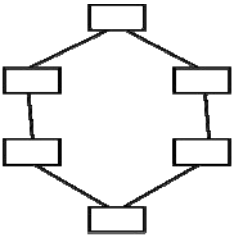


A family tree

are not trees, because they have cycles in them. Each node has two parents—one from the father tree and one from the mother tree. It is still a hierarchy but it is not a tree.

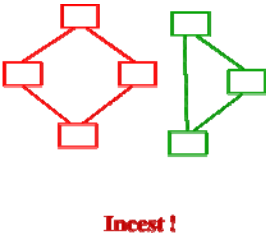
What exactly do I mean by this open question: *What are family trees*? The mathematicians know what I mean, but the non-mathematicians probably don't really get it. Yet!

A mathematician solving this problem will look for "meaningful" patterns. For example, this next pattern, which looks like a cycle of six boxes and might occur in your family tree, says something—it has some *semantics*. And what is it?

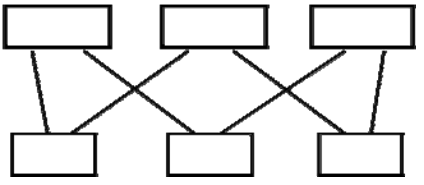


This hierarchy says that 'first cousins marry and have a child'. So if you see this pattern, it has meaning, even without filling in the names in each box.

Here are two others, which I will call 'Incest'. On the left there are a brother and sister that have a child, and on the right, *G-d forbid*, a parent and a child that have a child. These are biologically possible, but culturally unlikely.

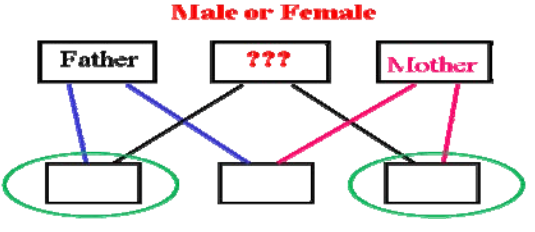


We will now see an example of a hierarchy that is biologically impossible. If this pattern occurs in your family tree, you have an error in your data, and actually there's a lot of value to that. The value is that it focuses your attention on where you have to go back to your notebooks, to see who was mother of whom, figure out where the mistake came from, and fix it in your database.



A hierarchy that is biologically impossible!

Why is this hierarchy biologically impossible? I will give you a proof. Look at the middle child. He/she has a mother and father. Let's say the father is on the left and the mother is on the right. Now we have a dilemma. On the one hand, the child on the left has its male parent on the left, so the middle parent must be female. On the other hand, the child on the right has its female parent on the right, so the middle parent must be male. Since this cannot happen biologically (in humans), this cannot happen in your family tree. Therefore, our conclusion is that this particular hierarchy cannot be in a family tree. We have a mathematical proof of something that we call a "forbidden" form or configuration.



Now, my question to the mathematicians here is, 'Can we characterize family trees by a complete set of forbidden forms?' And if you would like to work on this problem with me, that would be great, especially someone who is not yet on the list.

So what is this all about?

It is *to look for important patterns* that might indicate errors in your database or that might help find anomalies or interesting or suspicious patterns. This is also part of what people do in data mining or in link analysis of emails—finding phenomena that have meaning or scrutinizing who is sending emails to whom. They are also looking for patterns that are suspicious, maybe terrorists or perhaps friends, by characterizing those patterns to look at. It's being able to determine and locate those things that are important.

2 When I Was Twenty One, It Was a Very Good Year…

I left Penn State and had a summer job in Fort Belvoir, Virginia as a junior mathematician, then moved to New York City. The years 1970 to 1975 were spent earning my doctorate at Columbia University. Incidentally, Sammy Eisenberg, my Ph.D. advisor, and I were both born on September 30th. I worked on comparability graphs (transitively oriented graphs) for my thesis, wrote a paper on inducibility of graphs with Nick Pippinger (he did most of it) and another paper on combinatorial merging. That last one was actually the first paper that I wrote but the third paper to be published, because it took me about two and a half years until I understood how to write mathematics. It was a very good lesson. Took me a while but I guess I learned.

This was also a very exciting time for the field of computational complexity. It was the years when the theory of NP-completeness was just being discovered and used, and I didn't know what that was. Until then, I thought NPC was my uncle, Norman P. Cohen.

But very quickly I learned what it was and I started teaching it.

Now a comment for my son-in-law Avishai: A few weeks ago you asked us a question and I would like to answer that question now. The period 1970 through 1975, both professionally and personally, was the defining period of my life. Everything that I had done till then was preparation for my doctoral years at Columbia and everything that I have done since is the result of those five years: my professional career, my personal and family life, my character and my values, my returning to live in the homeland of the Jewish people.

My first academic position was with the Courant Institute at New York University as Assistant Professor of Computer Science for the five years 1975-1980. Clint Goss, an undergraduate at the time, came to me and wanted to work on a research problem. It turned out he was a very smart guy. Together, we introduced the family of chordal bipartite graphs and wrote the initial paper on that family of graphs.

I took one year off to do a postdoc. We went to Paris. Lynn had said, "Why should you use your postdoc and stay in New York? Let's go someplace else, someplace interesting." So we chose Paris so I could join the group of Claude Berge, and it was an experience. We spent a little over a half a year in Paris and then four months at the Weizmann Institute. I came back to Courant and finished writing the chapters for my book on algorithmic graph theory. Just around 1979, the book was finished, and the page proofs were coming. I remember walking our first daughter, Elana, in a baby carriage, and when she would fall asleep I could quickly take out the page proofs and go over a few pages of corrections until she woke up.

Now those of you who are familiar with my book *Algorithmic Graph Theory*, know that I like to tell stories. In fact, Ann Trenk just mentioned how my book is full of stories. It is part of my style in whatever I do. I was influenced in part by the mystery story by Claude Berge, whose book was one of the early inspirations for my own work in graph theory. So I would like to run through a new story with you which has a mathematical theme.

A Mathematical Story

Two groups of high school students, the girls of Evelina and the boys of Hartman, visited the Hecht Museum on the same day. Take out your pencils and paper because

we have to take notes on these rules: *The students came on their own, arriving and leaving at different times. Each school gathered their students together to tour as a group. The Hecht Museum is rather small, so if a boy and girl were in the museum at the same time, certainly they met each other.* Those are the rules.

Here is my Question:

> *Adina's boyfriend is Eitan and her brother is Doron.*
> *Yael's boyfriend is Doron but her brother is Eitan.*
> *Is it possible that the girls met their boyfriends but not their brothers?*

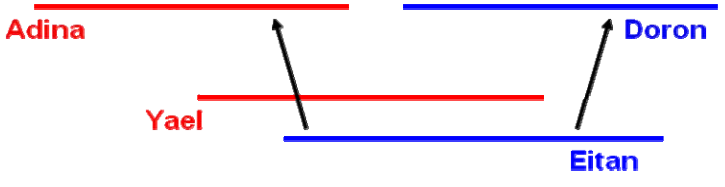
As my daughter Tali knows, now I am going to exit while you figure out the answer.

I'm back.

Show of hands: how many say 'Yes, it is possible'? How many say 'No, it is impossible'? Remember there are impossible things.

The answer is, 'No, it is not possible'. The reason, the same kind of reasoning as in the family tree argument, is the following:

If Adina did not meet her brother Doron, it follows that their time periods did not overlap—Adina came and she left and Doron came and he left. They didn't see each other. Now Yael and Adina toured together because the rules say that the girls from Evelina toured together, so they saw each other, and thus their time intervals must overlap. But Yael also saw her boyfriend Doron, so their intervals also overlap. So this is the configuration of the time intervals, with Yael "spanning the gap" between Adina and Doron. Now what about Eitan?

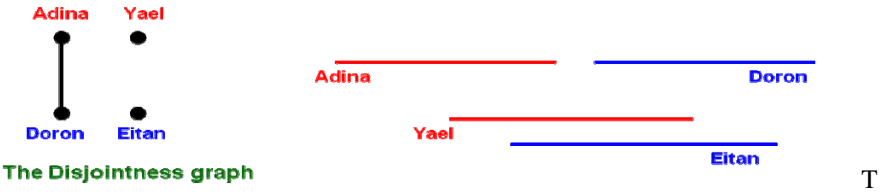


On the one hand, Eitan toured together with Doron and the other Hartman boys, on the other hand he saw his girlfriend Adina. So this gap is also covered by Eitan's time interval. Thus, Yael and Eitan would have seen each other!

Therefore, it is impossible that both girls met their boyfriends and not their brothers.

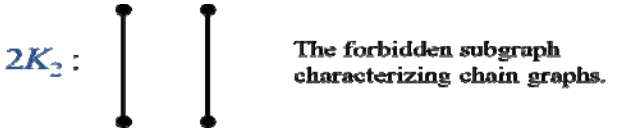
This story is an example of *reasoning about time* and it can be modeled by a graph. CSI might like to use this information the next time they are trying to solve a crime scene investigation. But there is mathematics behind this, and—work with me a little bit you non-mathematicians—I want to say a few technical things, and I hope you'll catch the gist of what I am doing.

We build a graph, with a vertex for each student and an edge connecting two vertices if the two students did not meet in the museum, that is, their time intervals were *disjoint*. We call this graph the *disjointness* graph. In our example, Adina and Doron were disjoint and everyone else intersects with everyone else.

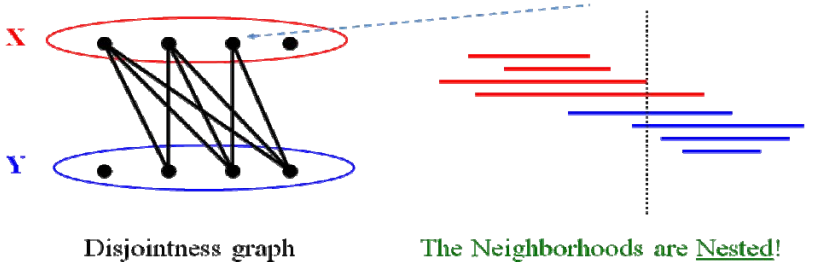


T

What our argument says mathematically is that the configuration that we call $2K_2$ is forbidden to be in the database of temporal facts. (This is just like the fact in the family tree that *up-down-up-down-up-down* was forbidden. By the way, if you go *up-down-up-down* an even number of times it's ok, but if you go *up-down-up-down* an odd number of times, it's not ok.)



So this is a forbidden graph and graph theorists call graphs that don't have these, *chain graphs*. Chain graphs are a kind of bipartite graph, a graph where you have nodes for the girls and nodes for the boys and edges between them representing disjointness, like in the figure below. And sure enough, we have a theorem that tells us about this class of graphs.



Theorem (Hammer, Peled, Simeone; others). Let $G = (X \cup Y, E)$ be a bipartite graph. The following are equivalent:

1) G is $2K_2$-free.
2) X can be ordered so that neighborhoods are nested.
3) Y can be ordered so that neighborhoods are nested.
4) Every induced subgraph of G has at most one non-singleton component, and has a universal x or a universal y vertex.[1]
5) Each vertex v of G can be assigned a weight $w(v)$, and a threshold t can be assigned, such that $(x,y) \in E$ if and only if $|w(x) - w(y)| > t$, for all x and y.

[1] A *universal* vertex in a bipartite graph is one that is adjacent to all vertices on the other side of the bipartition.

Remark. *X* or *Y* can simply be ordered by number of neighbors (vertex degree).

There are three names for this class of graphs. One is called *chain graphs* from the nested neighborhood properties (2) and (3); one is called *difference graphs* from the weights and threshold property (5); and one is simply called $2K_2$*-free bipartite graphs* from the forbidden subgraph property (1). In addition, you can recognize them in linear time by using property (4), repeatedly pulling off a universal vertex one by one, and erasing isolated vertices, just like you do in recognizing threshold graphs. In fact, difference graphs are very, very close to threshold graphs.

Remark. Who was Evelina? Evelina Gertrude de Rothschild was from the English branch of the Rothschilds. She died at a fairly young age (August 25, 1839 – December 4, 1866) and her father, Baron Lionel de Rothschild, the first Jewish member of the British House of Commons, assumed sponsorship of the first school for girls in Israel, opened in Jerusalem in 1854, renaming it the *Evelina de Rothschild School.* The Hartman High School is part of the Shalom Hartman Institute in Jerusalem.

3 When I Was Thirty Five, It Was a Very Good Year…

If Micky Rodeh were here, he would say, "That Golumbic, he always has his five-year plan." All I can say is that things somehow fit into five year periods. Now I don't know if it has really been on purpose or not, but that's what has happened.

And the next five years, from '80 to '85, was a transition time—first, my moving from academia at NYU to industrial research at Bell Labs and then at IBM, and second, our moving from the USA to Israel.

And Yaela was born. Now look at her name very carefully. In English or in Hebrew: Yaela — יעלה and compare it to Aliya — עליה (*moving up*, the word traditionally used for immigration to Israel). You don't have to be dyslexic to note the similarity.

During this period of time I introduced *tolerance graphs* with Clyde Monma. Then Tom Trotter joined us and we wrote what might be called the fundamental paper starting the study of tolerance graphs. I also initiated the study of EPT graphs with Robert Jamison during that period. And then when I moved to Israel, I started working on projects of expert systems and prolog, getting into artificial intelligence, scheduling and CSP constraint satisfiability problems.

I remained working in Haifa and here's another play on words. You remember the first slide about Erie "my city". Haifa is spelled like this: חיפה. But if you parse it or split it in the middle, pull it apart, it says something interesting in Hebrew, which the city of Haifa has been popularizing on billboards this summer: אני חי פה. "I live here". So I moved from Erie (עירי) to חי פה. And during that period Tali and Adina were both born.

While still working at IBM, I became an adjunct professor at Bar-Ilan University in 1985, the start of my next five year plan. I travelled there once a week to keep up my pure research activities, working with graduate students both in graph theory and in artificial intelligence. David Bernstein and Ron Pinter mentioned in their talks that at IBM, I began to work with them (and others) on compilers. I worked on register

allocation and instruction scheduling, and with Vladimir Rainish we had a patent. In graph theory, Ron Pinter, Ido Dagan and I initiated the work on trapezoid graphs, which we introduced in this period.

There were three other very important landmark series that started towards the end of that period. The first was founding the *Bar-Ilan Symposium on the Foundations of Artificial Intelligence* (BISFAI) which has taken place every two years since that time. I owe Micky Rodeh and Joe Raviv a little credit for my learning indirectly that if you want to go to more international conferences or if you want Israelis to be able to go to more international conferences, you bring the international conference to Israel. And I think that this has been a driving motivation for the 105 workshops and conferences in Israel that Irith Hartman mentioned in her opening remarks about my professional activities. The second was founding the *Annals of Mathematics and Artificial Intelligence*, of which I am still the editor-in-chief. I owe a lot of credit to Peter Hammer for having brought me in to do that and for much advice and friendship over the years. His tragic death a year ago was a terrible blow to the discrete mathematics community. And the third landmark was the founding of the *International Symposium on AI & Mathematics* series in Fort Lauderdale together with Fred Hoffman, who is here with us.

Artificial intelligence and reasoning about graphs

I would like to add something now about artificial intelligence. A central issue in AI is dealing with missing data, and having to deduce consistency with only partial information. For example, suppose at the Hecht Museum we don't know about the arrival times and the departure times of the students. We only have some information about when some of the Evelina girls and Hartman boys came and left. Even with that partial information, can we still construct a consistent set of intervals? Can we find new facts based on existing facts? Look at the example we just saw. If you know that Adina met Eitan, and you know that she did not meet Doron, and you know that Yael met Doron, then you can conclude, without having to know anything else, that it *must be* that Yael met Eitan. Otherwise, you'd have a contradiction. So you could deduce the fourth piece of information, having known only the three.

In graph theory we also have a kind of problem where we have missing information, and must reason about how to complete it. I would call this guessing and filling in missing edges from a partially specified graph. One such problem is called the Graph Sandwich Problem. The other is called the Probe Problem.

Again I am going to ask the non-mathematicians to stay with me for a little bit, while I address the mathematicians. I am sure you're going to appreciate at least some of what I am doing. You don't need to know the details, but rather watch my process of thinking.

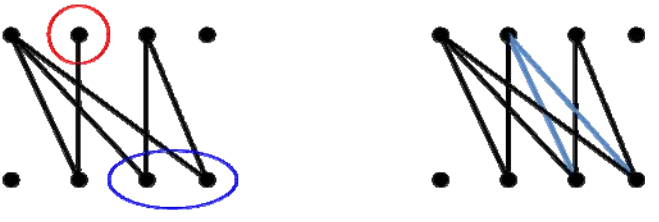
Let's talk about the *graph sandwich problem for chain graphs*: A bipartite graph $G = (X \cup Y, E)$ is given to you and a set E_0 of *optional edges* (pairs of vertices in $X \times Y$) which you could add to your graph. The algorithmic question is: Is there a subset $F \subseteq E_0$ of the op tional edges that you could add to the graph, such that the filled-in graph $G' = (X \cup Y, E \cup F)$ is a chain graph? In fact, there is a theorem that says that the chain graph sandwich problem can be solved in linear time: Repeatedly, remove either an isolated vertex in G or a universal vertex in $H = (X \cup Y, E \cup E_0)$.

That was an easy problem. Now there is a harder problem.

The chain probe graph problem

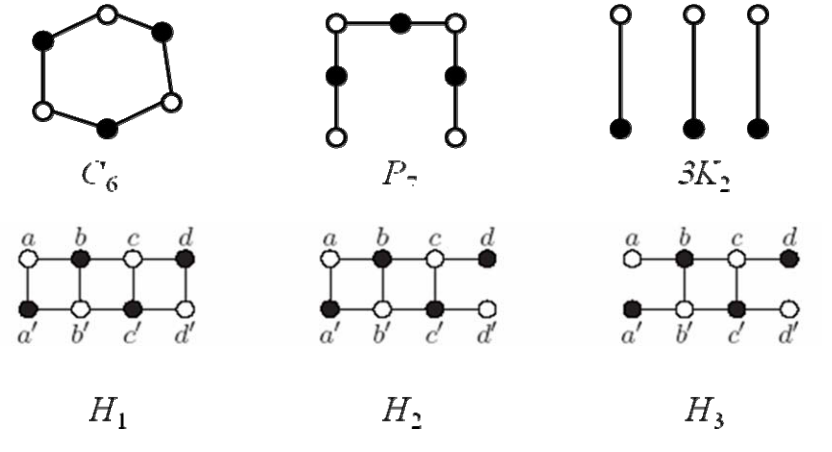
A bipartite graph $G = (X \cup Y, E)$ is a *chain probe graph* if there exists an independent set S of vertices (also known as a stable set) and a subset $F \subseteq S \times S$ of pairs of vertices from S such that when you fill in the graph with this subset F, you obtain a chain graph $G' = (X \cup Y, E \cup F)$. This is actually the *non-partitioned version* of the probe problem, that is, the set S is not given as part of the input to the problem.

As an example, consider the figure below. Remembering that chain graphs are the ones that are characterized by the forbidden $2K_2$, you have to "bust up" the two occurrences of $2K_2$ in the graph on the left. If the probes are indicated by the three vertices circled, you can destroy the copies of $2K_2$ by adding the two new edges shown in the graph on the right. Then you would have a chain graph. So in this case we have a chain probe graph by selecting these three vertices as S, filling in the two edges, giving a solution in this particular example.



Just like in the family tree problem, where I was looking for forbidden configurations, here too I am looking for forbidden configurations to characterize chain probe graphs. In a very recent paper by Frederic Maffray, a student of his, Gregory Morel and myself, we solved this problem—now you know what I do when I go to France, I don't just lie out on the Riviera. Our theorem characterizes this family of graphs.

Theorem. A bipartite graph is chain probe if and only if it contains none of the six forbidden graphs in the figure below.



Now, if I had the time to give you the proof, then I would first prove two lemmas that would be enough to show that those six graphs are not chain probes, showing the "only if" implication. The proof in the opposite direction is a little more mathematically complicated, and for that I would let you read the paper. Moreover, I would show you that algorithmically we have a solution to the complexity question and that recognizing chain probe graphs can be done in $O(n^2)$ time.

I hope you have all gotten the point. Mathematicians are looking for a complete set of forbidden subgraphs that fully characterizes this kind of graph. And then they can write a computer program to find them.

More graph theory problems

Blocking together the next two five-year periods, 1990-2000, I soon left IBM to return to academia full time at Bar-Ilan—working on temporal reasoning problems, introducing sandwich problems with Ron Shamir and Haim Kaplan, on induced matchings with Renu Laskar and Moshe Lewenstein, on clique width with Udi Rotics, on acyclic hypergraphs, and on factoring Boolean functions with Avi Mintz. All of this culminating with two more landmarks that sort of "started the new millennium".

The years 2000-2005. The first personal landmark was writing the book *Tolerance Graphs* with Ann Trenk. It was fantastic experience for me. We were a great working pair. I think there's an appropriate song, "We belong to a mutual admiration society." It was a lot of fun writing the book, and we both got a tremendous sense of professional satisfaction having produced it.

The second landmark was founding CRI: The Caesarea Rothschild Institute at the University of Haifa, with my cofounders, Libi Oded and Irith Hartman; Libi on the administration side and Irith on the scientific side. I call them my cofounders because it would have been impossible to do something on this scale without a really solid team, and I thank them.

I have other people at CRI to thank, the many great staff people that have followed along in those years: Miriam Daya, Meirav Resnick, Orna Nagar-Hillman, Ornit Bar-Or, Sara Kaufman, Avital Berkowich, George Karapetyan, Orly Ross, and Rona Perkis. And especially for this conference, thanks go to Rona and Orly who have done the bulk of the administrative work, to George making sure that technical things run smoothly and Hananel Hazan making sure the things that George is too busy to do, get done, to Elad Cohen who did the abstract book and Avital who did the backroom stuff and had to stay home in the office to make sure that no fires took place.

And there are another set of projects—the projects with Trento, Italy on Innovative Technology for Human Development—with my dear friend, colleague and partner in all kinds of crazy ideas, Oliviero Stock, who is also here for this meeting. Thank you, Oliviero.

But during this period of time, I didn't just sit around doing nothing research-wise: new results on tolerance graphs, on rank tolerance graphs with Robert Jamison, who unfortunately isn't here—but he sent me a copy of a paper he wrote for the Boca conference proceedings this year, which he actually dedicated to my birthday, and which may inspire us to do some new things coming up; the work on chordal probe graphs with Marina Lipshteyn and Ann Berry; and the work on k-EPT graphs

and (*h,s,t*)-hierarchies with Marina and Michal Stern. I would also like to thank them for their help in this conference, along with Irith who ran the program, selected the speakers and made this into a technically high-powered—and amazing seven days.

Finally, there was this other "*meshuga*" idea, writing the book *Fighting Terror Online*, that just appeared this year published by Springer. This goes back to the very first or second year in CRI, when we sponsored a workshop and a seminar with two professors from the Law School, Michael Birnhack and Niva Elkin-Koren, and a bunch of students who were majoring in computer science and law. They gathered lots of material, summarized many discussions with experts, and wrote up a white paper in Hebrew on fighting terror online. We then translated it, and I started editing and adding more material until it grew into a book. I am extremely grateful to Michael and Niva for working with me on this book. Although they decided not to be coauthors, they really are in every sense of the word, coauthors. I also owe a lot of debt to my editors Sara Kaufman and Diane Romm. This is really not a Marty book; this is a book that Marty helped edit, enlarge, add some of my personality too. I get some credit but the credit goes to a lot of people. Finally, I must also thank my wife Lynn because at an early stage, she looked at the manuscript and said, "You know, this is important stuff. People should know about it." It confirmed what I felt, and one way or another, I knew that it had to come out and it did.

Vivaldi's Allegro from Spring

We've talked about my spring. We've talked about my summer. And now starting tonight, on the Hebrew calendar, autumn.

So what will I do next?

My next talk to be written is for March 2009 in Warwick, England. I will be working during the coming months to put some meat into this title: *Conflict and Tolerance in Graph Theor*y. It will be a model, similar to rank tolerance, but broader, like the new paper by Jamison. The technical details, for mathematicians, are that it will be a graph where each vertex has a rank, indicating its tendency to have edges, just like tolerance is a weight or function that gives the tendency for not having edges. Join them together, you'll get an edge if and only if a ranking function exceeds the tolerance—the tendency to have edges exceeding the tendency to not have edges. All of our literature on intersection graphs and types of tolerance graphs fit into this framework and much more. I think that there are a lot of places where we can do good work.

So this is the beginning of the autumn.

4 But Now the Days Are Short…

Frank Sinatra… and wine…and a toast with Lynn.

Vivaldi's Allegro from Autumn

Thank you.

The Golumbic Family

A Higher-Order Graph Calculus for Autonomic Computing

Oana Andrei[1] and Hélène Kirchner[2]

[1] INRIA Nancy Grand-Est & LORIA, France
[2] INRIA Bordeaux Sud-Ouest, France
Oana.Andrei@loria.fr, Helene.Kirchner@inria.fr

Abstract. In this paper, we present a high-level formalism based on port graph rewriting, strategic rewriting, and rewriting calculus. We argue that this formalism is suitable for modeling autonomic systems and briefly illustrate its expressivity for modeling properties of such systems.

1 Introduction

Autonomic computing [1] refers to self-manageable systems initially provided with some high-level instructions from administrators. This is a concept introduced in 2001 with an intended biological connotation. The four most important aspects of self-management as presented in [1] are self-configuration, self-optimization, self-healing, and self-protection.

This idea of biologically inspired formalisms gained much interest with the recent development of large scale distributed systems such as service infrastructures and grids. For such systems, there is a crucial need for theories and formal frameworks to model computations, to define languages for programming and to establish foundations for verifying important properties of these systems. Several approaches contributed to this ambitious goal. Without exhaustivity, let us mention in particular the brane calculus [2, 3], membrane computing and P-systems [4], and the bigraphical reactive systems [5], but also several calculi inspired from biology such as [6–8].

Another connected approach is provided by chemical programming, which uses the chemical reaction metaphor to express the coordination of computations. This metaphor describes computation in terms of a chemical solution in which molecules (representing data) interact freely according to reaction rules. Chemical solutions are represented by multisets (a set data structure that allows several occurrences of the same element). Computation proceeds by rewritings, which consume and produce new elements according to conditions and transformation rules. The Gamma formalism was first proposed in [9] and later extended to the γ-calculus and HOCL (Higher-Order Chemical Language) in [10, 11] for modeling self-organizing and autonomic systems or grids in particular. MGS is another formalism based on the chemical model. It was designed to represent and manipulate local transformations of entities structured by abstract topologies [12].

Beyond the chemical programming idea, another approach presented in [13], called the Organic Grid, is similarly a radical departure from current approaches and is inspired by the self-organization property of complex biological systems.

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 15–26, 2009.

Our previous work on biochemical applications led us to consider the structure of port graph [14] (or multigraph with ports) to model interactions between molecules, in particular proteins [15]. The behavior of a protein is given by its functional domains which determine which other proteins it can bind to or interact with; these domains are usually abstracted as sites that can be bound or free, visible or hidden. Hence a protein is characterized by the collection of interaction sites on its surface and proteins can bind to each other forming molecular complexes. Based on such structures, we considered graphs with multiple edges and loops, with nodes having explicit connection points, called *ports*, and edges attaching, more specifically, to ports of nodes; we called them *port graphs*. Port graphs provide a modeling formalism for molecular complexes by restricting the connectivity of a port (called *site* in the biological model) to at most one other port. In [15], port graph rewriting and rewrite strategies are used to model molecular complexes and their interaction on a fragment of the epidermal growth factor receptor (EGFR) signaling pathway.

We extend the chemical model with high-level features by considering a graph structure for the molecules and permitting control on computations to combine rule applications. The result is a higher-order port graph rewriting calculus. By lifting port graph rewriting to a calculus, we are able to express rules and strategies as port graphs and so to rewrite them as well. The calculus also permits the design of rules that create new rules, providing a way of modeling emergence in a system. We borrow various concepts from graph theory, in particular from graph transformations [16], and we use different representations for graphs already intensively formalized.

In this paper, we propose port graphs as a formal model for distributed resources and grid infrastructures. Each resource is modeled by a node with ports. We model the lack of global information, the autonomous and distributed behavior of components by a multiset of port graphs and rewrite rules which are applied locally, concurrently, and non-deterministically. Hence the computations take place wherever it is possible and in parallel as soon as they do not interfere. This approach also provides a formal framework to reason about computations and to verify desirable properties.

The paper is structured as follows. Section 2 introduces the rewriting relation for port graphs and presents some rewrite strategies used in this paper. Having all ingredients in hand, in Sect. 3 we give the main ideas of a high-level calculus for port graphs, and in Sect. 4, we argue that this calculus is a suitable formalism for modeling autonomic systems. In Sect. 5 we give some suggestions on expressing properties of a modeled system as strategies.

2 Port Graph Rewriting

In order to illustrate our approach and the proposed concepts, we develop the example of a mail delivery system borrowed from [17]. It consists of a network of several mail servers each with its own address domain; the clients send messages for other clients first to their server domain, which in turn forwards them to the network and recovers the messages sent to its clients. Servers are distributed resources with connections between them, created when sending and receiving the messages.

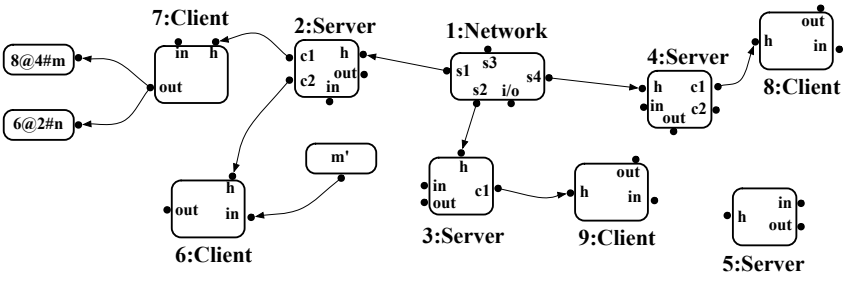


Fig. 1. A mail system configuration

In Fig. 1, we illustrate an initial configuration of the mail delivery system. The network is a node with several ports, each port being connected to at most one server. We represent graphically a node as a box with the unique identifier and the name placed outside the box. The ports are represented as small points on the surface of the box. A server node has a handler port for connecting to the network, and several ports for the clients. A client node has a handler port for connecting to a server. All client and server nodes have two ports for the incoming and outgoing messages respectively. The network node has one port for the messages. Messages are nodes with only one port and their names have the form (rec @ $domain$ # m) where rec is the identifier of the recipient client, $domain$ is the identifier of the server domain, and m the body of the message. If redundant, the domain and/or the client identifiers are removed: this is the case as soon as the message is arrived in the server domain or at the client. In the system described in Fig. 1, the server identified by **5** is disconnected from the network node, meaning that it is crashed.

Formally, given a finite set of node names and a finite set of port names, a *p-signature* is a function associating to each node name a set of port names.

A *port graph rewrite rule* $L \Rightarrow R$ is a port graph consisting of two port graphs L and R over the same p-signature and one special node $\Rightarrow$, called *arrow node* connecting them. L and R are called, as usual, the *left-* and *right-hand side* respectively. We assume here that all node identifiers are variables. The arrow node has the following characteristics:

1. for each port p in L, to which corresponds a non-empty set of ports $\{p_1, \ldots, p_n\}$ in R, the arrow node has a unique port r and the incident edges (p, r) and (r, p_i), for all $i = 1, \ldots, n$;
2. all ports from L that are deleted in R are connected to a *black hole* port, named bh.

The arrow node together with its adjacent edges embed the correspondence between elements of L and elements of R.

We illustrate some port graph rewrite rules in Fig. 2. We represent graphically the edges incident to the arrow node only if the correspondence is ambiguous. In consequence, port graphs represent a unifying structure for representing port graph rewrite rules as well.

A *port graph rewrite system* $\mathcal{R}$ is a finite set of port graph rewrite rules.

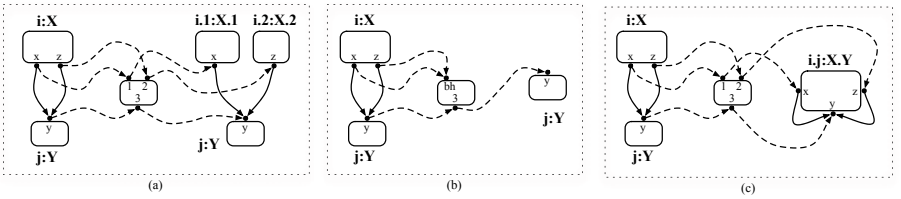


Fig. 2. Some port graph rewrite rules: (a) splitting node i in two; (b) deleting node i; (c) merging nodes i and j

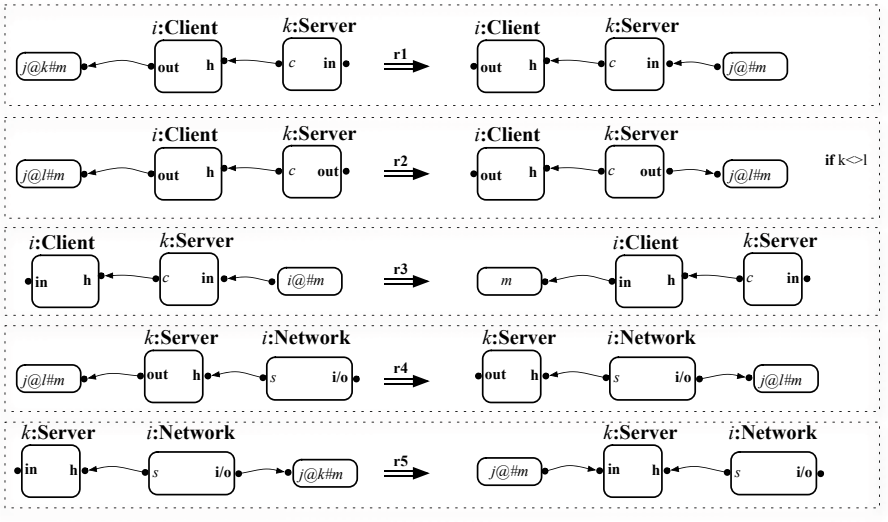


Fig. 3. Basic rules for the mail delivery system

Coming back to our example, the evolution of the mail delivery distributed system is modeled via port graph transformations, themselves expressed by port graph rewrite rules and the generated rewriting relation. We illustrate in Fig. 3 some basic rules for the mail system. A mail sent by a client goes to its server: if the mail is sent to a client in the same server domain then it goes to the input port by **r1**, otherwise to the outgoing port by **r2**. By rule **r3** a server forwards a mail to a client if he is the recipient. Rule **r4** specifies that a server forwards a mail to the network if its recipient is not in the domain, while rule **r5** specifies that the network forwards a mail to the appropriate server as specified in the mail.

Let us now formalize the graph transformations induced by port graph rewrite rules. Let $L \Rightarrow R$ be a port graph rewrite rule and G a port graph such that there is an injective graph morphism g from L to G; hence $g(L)$ is a subgraph of G. Replacing $g(L)$ by $g(R)$ and connecting it appropriately in the context, we obtain a port graph G' which represents a result of *one-step rewriting* of G using the rule $L \Rightarrow R$, written $G \rightarrow_{L \Rightarrow R} G'$. There can be different such injective morphisms g from L to G leading to

different results. They are built as solutions of a *matching* problem from L to a subgraph of G. If there is no such injective morphism, we say that G is *irreducible* with respect to $L \Rightarrow R$. Given a port graph rewrite system $\mathcal{R}$, a port graph G *rewrites* to a port graph G', denoted by $G \rightarrow_{\mathcal{R}} G'$, if there exists a port graph rewrite rule r in $\mathcal{R}$ such that $G \rightarrow_r G'$. The formal definition of port graph rewriting is given in [14].

The port graph rewrite system $\mathcal{R}$ generates an abstract reduction system, whose nodes are graphs and whose oriented edges are rewriting steps. Then a derivation in $\mathcal{R}$ is a path in the underlying graph of the associated abstract reduction system. The notions of *strategy* and *strategic rewriting* were introduced in the rewriting community in order to control rule applications, i.e. to select relevant derivations. Strategies are formalized as a subset of derivations in [18]. A strategy can be described by a *strategy language*. Various approaches have been followed, yielding different strategy languages such as ELAN [19], Stratego [20], TOM [21] or Maude [22]. All these languages share the concern to provide abstract ways to express control of rule applications. Following [18], we can distinguish two classes of constructs in the strategy language: the first class allows construction of derivations from the basic elements, namely the rewrite rules, identity (Id) and failure ($Fail$). The second class corresponds to constructs that express the control, like sequence ($Sequence$ or _; _) or left-biased choice ($First$). Moreover, the capability of expressing recursion in the language brings even more expressive power. The strategies can be composed to build other useful strategies. One composed strategy, for instance, is Try which applies a strategy if it can, and the identity strategy otherwise. Similarly, the $Repeat$ combinator is used in combination with a fixpoint operator to iterate the application of a strategy. We will give later a formal description of strategies in our port graph rewriting calculus.

3 The Port Graph Rewriting Calculus

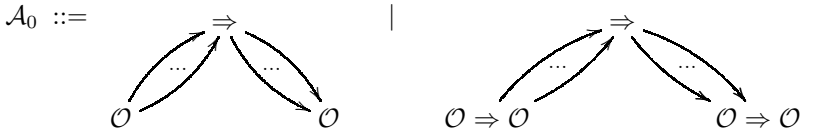
In this section, we define a rewriting calculus for port graphs, the ρ_{pg}-calculus, whose first-class citizens are object port graphs, port graph rewrite rules and rule application. This is an instance of an Abstract Biochemical Calculus, the $\rho_{\langle\Sigma\rangle}$-calculus, modeling interactions between abstract molecules over a structure described by the objects of a category Σ and presented in [23]. Here, we consider the port graph structure and the port graph rewriting relation defined in Section 2 for the abstract molecules and their interactions respectively.

The ρ_{pg}-calculus generalizes the ρ-calculus [24] and the ρ_g-calculus [25], since port graphs generalize the tree-like structure of terms and the graph-like structure of termgraphs respectively. It inherits from the ρ-calculus the fact that it also generalizes the λ-calculus through a more powerful abstraction power that considers for matching not only a variable, like in the λ-calculus, but a port graph with variables.

The ρ_{pg}-calculus provides a formal model for systems whose states correspond to a multiset of object port graphs and whose transitions are reductions obtained by applying port graph rewrite rules. Due to the intrinsic concurrent (or parallel) nature of rewriting on disjoint redexes, we model a kind of *Brownian motion*, a basic principle in the chemical paradigm consisting in *"the free distribution and unspecified reaction order among molecules"* [17], if we consider port graphs as molecules.

3.1 Syntax

Let $\mathcal{O}$ be the set of *object port graphs* modeling systems states. We denote by $\mathcal{A}$ the set of abstractions, which are port graph rewrite rules consisting of two port graphs for the left- and right-hand side, and the arrow node embedding the correspondence between the two sides. Graphically, the abstractions are first defined as follows:

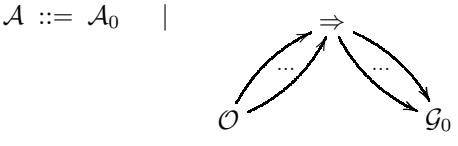


We construct in an iterative way the set of *port graph molecules*. In a first step, let us consider the set $\mathcal{G}_0$ of port graph molecules which are either an object port graph in $\mathcal{O}$, an abstraction in $\mathcal{A}_0$, a juxtaposition of molecules from $\mathcal{G}_0$, or the empty port graph ε:

$$\mathcal{G}_0 ::= \mathcal{O} \mid \mathcal{A}_0 \mid \mathcal{G}_0\ \mathcal{G}_0 \mid \varepsilon$$

The juxtaposition "_" is associative, commutative with ε the neutral element.

In a second step, the set of abstractions $\mathcal{A}_0$ is extended with port graph rewrite rules that transform an object from $\mathcal{O}$ into a port graph molecule from $\mathcal{G}_0$:



Then the set of port graph molecules $\mathcal{G}$ includes $\mathcal{G}_0$ and $\mathcal{A}$, as well as a set of variables $\mathcal{X}$:

$$\mathcal{G} ::= \mathcal{X} \mid \mathcal{G}_0 \mid A \mid \mathcal{G}\ \mathcal{G}$$

Finally, molecules are encapsulated into *worlds*. A world represents a state of the modeled system that contains all molecules present in the environment at a current step together with the connections between them. A world is again represented as a port graph with a node [ ] connected to all object port graphs and all abstractions in the environment. This node corresponds to a permutative variadic operator.

3.2 Reduction Semantics

In a world, an abstraction A and a port graph molecule G can interact non-deterministically. This interaction is modeled thanks to the **Heating** rule given in Fig. 4. This rule introduces an application node @ which connects the abstraction A and the port graph molecule G in the context of other molecules C in the world.

A successful application of an abstraction A on a port graph molecule G yields a port graph G' according to a port graph rewriting step $G \rightarrow_A G'$ with a port graph rewrite rule A of the form $L \Rightarrow R$. The successful application of an abstraction to a port graph molecule may produce different molecules, according to different matching

solutions. So we get in general a multiset of results if the application succeeds (see Rule **Application** in Fig. 4), while a matching failure returns the initial abstraction and the port graph molecule unchanged (see Rule **ApplicationFail** in Fig. 4).

After the application of the abstraction on the port graph molecule has taken place, a *cooling* rule, the counterpart of the heating rule, is in charge of rebuilding the state of the different produced systems. This is Rule **Cooling** in Fig. 4.

$$\textbf{(Heating)}\ [C\ A\ G] \longrightarrow [C\ A@G]$$
$$\textbf{(Application)}\ A@G \longrightarrow \{[G_1] \dots [G_n]\}\ \textbf{if}\ G \rightarrow_A G_i, 1 \leq i \leq n$$
$$\textbf{(ApplicationFail)}\ A@G \longrightarrow A\ G\ \textbf{if}\ G\ \textbf{is}\ A\textbf{-irreducible}$$
$$\textbf{(Cooling)}\ [C\ \{[G_1] \dots [G_n]\}] \longrightarrow \{[C\ G_1] \dots [C\ G_n]\}$$

Fig. 4. Semantic rules with explicit application

The full calculus is developed in [23]. All steps computing the application of an abstraction to a port graph molecule, including the matching and the replacement operations, are expressible using port graphs by considering more auxiliary nodes and extending the reduction relation with appropriate graph reduction rules. This illustrates well the expressivity of the port graph structure and transformation. Matching and replacement mechanisms are internalized in the calculus as port graph transformations, but since the rules are quite technical, we do not include them here. They can be found in [23].

3.3 Explicit Failure Handling

In the previous reduction rules for the semantics in Fig. 4, failure is implicit and the failure information is not exploited. In order to do that, we introduce the failure node stk in the **ApplicationFail** rule:

$$\textbf{(ApplicationFail')}\ A@G \longrightarrow \mathsf{stk} \quad \textbf{if}\ G\ \textbf{is}\ A\textbf{-irreducible}$$

Fig. 5. Semantic rule for explicit failure

In addition, when handling explicitly failure in this way, other rules are needed to clean up the worlds, such as :

$$[G]\ [\mathsf{stk}] \longrightarrow [G] \qquad [\mathsf{stk}]\ [\mathsf{stk}] \longrightarrow [\mathsf{stk}]$$

3.4 Strategies as Abstractions

Instead of having this highly non-deterministic and non-terminating behavior of port graph rewrite rule application, one may want to introduce some control to compose or

choose the rules to apply, possibly exploiting failure information. This is possible by defining strategies as extended abstractions.

In this section, we define strategies as objects of the calculus, using the basic constructs, as one can do in the λ-calculus or the γ-calculus. For such definitions, we use an approach similar to the one used in [26] where rewrite strategies are encoded by rewrite rules. Let us consider, for the rewrite strategies given in Sect. 2, the following objects: id, $\mathtt{fail}$, $\mathtt{seq}$, $\mathtt{first}$ and $\mathtt{try}$.

Let S, S_1, S_2 denote strategies. We encode the strategies Id, $Fail$, $\_;\_$, $First$ and Try as the following aliases for extended abstractions respectively:

$$\begin{aligned}
\mathsf{id} &\triangleq X \Rightarrow X \\
\mathtt{fail} &\triangleq X \Rightarrow \mathsf{stk} \\
\mathtt{seq}(S_1, S_2) &\triangleq X \Rightarrow S_2@(S_1@X) \\
\mathtt{first}(S_1, S_2) &\triangleq X \Rightarrow (S_1@X)\ \ (\mathsf{stk} \Rightarrow (S_2@X))@(S_1@X) \\
\mathtt{try}(S) &\triangleq \mathtt{first}(S, \mathsf{id})
\end{aligned}$$

Other useful strategies are provided for negation and test:

$$\begin{aligned}
\mathtt{not}(S) &\triangleq X \Rightarrow \mathtt{first}(\mathsf{stk} \Rightarrow X, X' \Rightarrow \mathsf{stk})@(S@X) \\
\mathtt{ifThenElse}(S_1, S_2, S_3) &\triangleq X \Rightarrow \mathtt{first}(\mathsf{stk} \Rightarrow S_3@X, X' \Rightarrow S_2@X)@(S_1@X)
\end{aligned}$$

The composed strategy $Repeat$ is defined with a recursion operator μ as follows:

$$\mathtt{repeat}(S) \triangleq \mu X.\mathtt{try}(\mathtt{seq}(S, X))$$

We can encode the μ abstraction using the fixed-point combinator of the λ-calculus as already done for encoding iterators in the ρ-calculus(See [24]).

Based on these strategy definitions, we can reformulate the heating rule using a failure catching mechanism: if $S@G$ reduces to failure, i.e., to the stk node, then the abstraction $\mathsf{stk} \Rightarrow S\ G$ restores the initial port graphs.

(Heating') $[C\ S\ G] \longrightarrow [C\ \mathtt{first}(S, \mathsf{stk} \Rightarrow S\ G)@G]$

Fig. 6. Semantic rule for failure catching

3.5 Persistent Strategies

At this level of definition of the calculus, strategies are consumed by a non-failing interaction with a port graph molecule. One advantage is that, since we work with multisets, a strategy can be given a multiplicity, and each interaction between the strategy and a port graph molecule consumes one occurrence of the strategy. This permits controlling the maximum number of times an interaction can take place.

Sometimes it may be suitable to have persistence of strategies. In this case, the strategies should not be consumed by the reduction. For that purpose, we define the *persistent* strategy combinator that applies a strategy given as argument and, if successful, replicates itself:

$$S! \triangleq \mu X.\mathtt{seq}(S, \mathtt{first}(\mathsf{stk} \Rightarrow \mathsf{stk}, Y \Rightarrow Y\ X))$$

Indeed if $S@G \longrightarrow \{[G_1] \dots [G_n]\}$, then $S!@G \longrightarrow \{[S!\ G_1] \dots [S!\ G_n]\}$.

4 Expressivity of the ρ_{pg}-calculus: Modeling Autonomic Systems

In autonomic computing, systems and their components reconfigure themselves automatically according to directives (rewrite rules and strategies) given initially by administrators. Based on these primary directives and their acquired knowledge along the execution, the systems and their components seek new ways of optimizing their performance and efficiency *via* new rewrite rules and strategies that they deduce and include in their own behavior. Since there is no ideal system, functioning problems and malicious attacks or failure cascades may occur, and the systems must be prepared to face them and solve them. Let us consider here and illustrate on the mail delivery system example, three aspects that an autonomic system must handle, namely self-configuration, self-healing, and self-protection.

The self-configuration is simply described by the concurrent application of the five rules given in Fig. 3 using the reduction semantics introduced in Sect. 3. An interesting problem may concern the operations of splitting and merging servers (their domains). Then biologically inspired port graph rules from Fig. 2 (a) and (c) could be applied as well for servers.

An autonomic system detects when a server crashes and the connection of the crashed server to the network is cut. It is expected to repair the problem of the clients connected to the crashed server and the problem of the mails that were about to be sent from that particular server. This self-healing behavior can be described by rules that detect the problems and by rules that repair them by modifying the configuration or introducing new rules in the system. The same method can be used as well for self-optimization. For finding the problem in the system, the calculus is powerful enough to find the right pattern and apply the appropriate rule.

We show in the following a concrete example for self-protection behavior of the mail system. Let us consider the rules in Fig. 7. When a spam arrives at a server node, the filtering rule **r6** deletes it, assuming that the server has a procedure for deciding when a mail is a spam. The rules **r7** and **r8** are analogous to **r6** but for a client node and a network node respectively, assuming as well that both entities have their own spam detection procedure. In order to limit spam sending, the rule **r8** should have a higher priority than **r5** in Fig. 1, and the rule **r6** a higher priority than **r3**. Then we replace **r3** and **r6** by $\mathtt{seq}(\mathtt{try}(\mathbf{r6}), \mathbf{r3})$, and **r5** and **r7** by $\mathtt{seq}(\mathtt{try}(\mathbf{r8}), \mathbf{r5})$.

When a client receives a mail and, based on a spam decision procedure, concludes that the mail is a spam, it deletes the mail and provides the server with a new rule specifying that from now on the server node should delete all mails of this kind. This behavior is specified by the rule **r9**.

A bigger development of this mail delivery system example with further rules can be found in [23].

5 Embedding Runtime Verification

We have shown in Sect. 4 how a particular autonomic system can be modeled using the ρ_{pg}-calculus. The model should also ensure formally that the intended self-managing

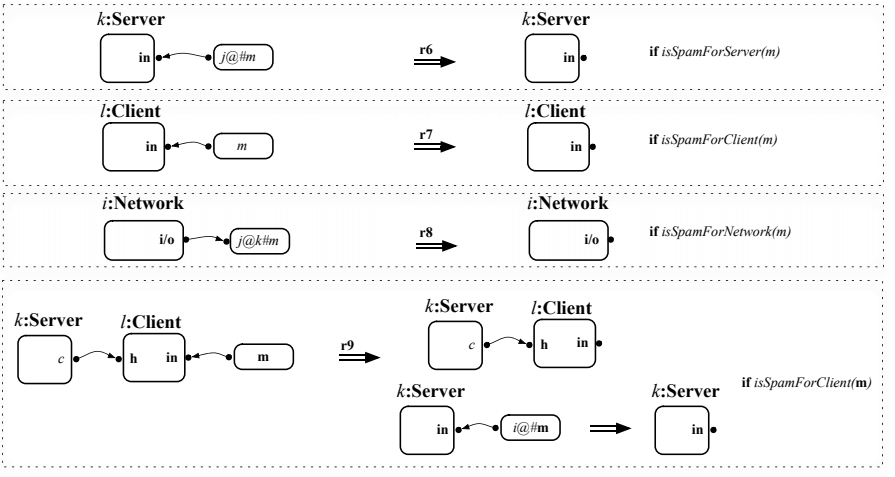


Fig. 7. Rules for self-protection in the mail delivery system

specification of the system helps indeed preserving the properties of the system. Some properties can be verified by checking the presence of particular port graphs. Such properties can be easily encoded as object port graphs, abstractions, or strategies, hence as entities of the calculus. Consequently, the properties can be placed at the same level as the specification of the modeled system and they can be tested at any time.

An invariant of the system can be expressed as a port graph rewrite rule with identical sides, $G \Rightarrow G$, testing the presence of a port graph G. The failure of the invariant is handled by a failure port graph Failure that does not allow the execution to continue. The strategy verifying such an invariant is then:

$$\texttt{first}(G \Rightarrow G, X \Rightarrow \mathsf{Failure})!$$

Such strategy is useful for instance to ensure, in our running example, the persistence of a given critical server of the network, or may be used also to check that there is always a minimal number of servers available in the network. From another perspective, we express the unwanted occurrence of a concrete port graph G in the system using the strategy:

$$(G \Rightarrow \mathsf{Failure})!$$

In practice, such strategies are employed in model checking applications to test unwanted situations.

In both cases above, instead of yielding the failure Failure signaling that a property of the system is not satisfied, the problem can be "repaired" by associating to each property the necessary rules or strategies to be inserted in the system in case of failure. Such ideas need to be further explored since they open a wide field of possibilities for combining runtime verification and self-healing in ρ_{pg}-calculus.

6 Conclusion

In this paper our main objective was to propose a formalism, the port graph calculus, for modeling autonomic systems.

From the computational point of view, we have shown that this calculus allows us to model concurrent interactions between port graph rewrite rules and object port graphs, as well as interactions between rewrite rules or interactions creating new rewrite rules. Thanks to strategies, some interactions may be designed with more control. The suitable balance between controlled and uncontrolled interactions is an interesting question to address for a given application. Here again, biological systems may provide us with valuable intuitions.

From the verification point of view, we can take advantage of the classical techniques used in rewriting for checking properties of autonomic systems. However, rules may also interfere giving rise to some conflicts. Detecting them can be done through confluence check and computation of critical pairs. Also some processes may be required to terminate when they are involved in computations. On the contrary, for known non-terminating processes, detecting periodicity of the processes may be of interest. Therefore, further work needs to address the verification of such properties for port graph rewriting. We have also outlined in this paper some ideas for runtime verification of properties in such systems, that need further exploration.

References

1. Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. IEEE Computer 36(1), 41–50 (2003)
2. Cardelli, L.: Brane Calculi. In: Danos, V., Schächter, V. (eds.) CMSB 2004. LNCS (LNBI), vol. 3082, pp. 257–278. Springer, Heidelberg (2005)
3. Danos, V., Pradalier, S.: Projective Brane Calculus. In: Danos, V., Schächter, V. (eds.) CMSB 2004. LNCS (LNBI), vol. 3082, pp. 134–148. Springer, Heidelberg (2005)
4. Paun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
5. Milner, R.: Pure bigraphs: Structure and dynamics. Inf. Comput. 204(1), 60–122 (2006)
6. Regev, A., Panina, E.M., Silverman, W., Cardelli, L., Shapiro, E.Y.: BioAmbients: an abstraction for biological compartments. TCS 325(1), 141–167 (2004)
7. Chabrier-Rivier, N., Fages, F., Soliman, S.: The Biochemical Abstract Machine BIOCHAM. In: Danos, V., Schachter, V. (eds.) CMSB 2004. LNCS (LNBI), vol. 3082, pp. 172–191. Springer, Heidelberg (2005)
8. Laneve, C., Tarissan, F.: A simple calculus for proteins and cells. ENTCS 171(2), 139–154 (2007)
9. Banatre, J.P., Metayer, D.L.: A new computational model and its discipline of programming. Technical Report RR-566, INRIA (1986)
10. Banâtre, J.P., Fradet, P., Radenac, Y.: A Generalized Higher-Order Chemical Computation Model. ENTCS 135(3), 3–13 (2006)
11. Banâtre, J.P., Fradet, P., Radenac, Y.: Programming Self-Organizing Systems with the Higher-Order Chemical Language. International Journal of Unconventional Computing 3(3), 161–177 (2007)
12. Giavitto, J.L., Michel, O.: MGS: a Rule-Based Programming Language for Complex Objects and Collections. Electronic Notes in Theoretical Computer Science 59(4) (2001)

13. Chakravarti, A.J., Baumgartner, G., Lauria, M.: Application-Specific Scheduling for the Organic Grid. In: Buyya, R. (ed.) GRID, pp. 146–155. IEEE Computer Society, Los Alamitos (2004)
14. Andrei, O., Kirchner, H.: A Rewriting Calculus for Multigraphs with Ports. In: Proceedings of RULE 2007, International Workshop on Rule-Based Programming. Electronic Notes in Theoretical Computer Science, vol. 219, pp. 67–82, November 20 (2008)
15. Andrei, O., Kirchner, H.: Graph Rewriting and Strategies for Modeling Biochemical Networks. In: SYNASC 2007, pp. 407–414. IEEE Computer Society, Los Alamitos (2007)
16. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Löwe, M.: Algebraic Approaches to Graph Transformation - Part I: Basic Concepts and Double Pushout Approach. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformations. Foundations, vol. 1, pp. 163–246. World Scientific, Singapore (1997)
17. Banâtre, J.P., Radenac, Y., Fradet, P.: Chemical Specification of Autonomic Systems. In: IASSE 2004, pp. 72–79. ISCA (2004)
18. Kirchner, C., Kirchner, F., Kirchner, H.: Strategic computations and deductions. In: Festchrift in honor of Peter Andrews. Studies in Logic and the Foundations of Mathematics. Elsevier, Amsterdam (2008)
19. Borovanský, P., Kirchner, C., Kirchner, H., Ringeissen, C.: Rewriting with strategies in ELAN: a functional semantics. Int. J. Found. Comput. Sci. 12(1), 69–98 (2001)
20. Visser, E.: Stratego: A Language for Program Transformation based on Rewriting Strategies. In: Middeldorp, A. (ed.) RTA 2001. LNCS, vol. 2051, pp. 357–361. Springer, Heidelberg (2001)
21. Balland, E., Brauner, P., Kopetz, R., Moreau, P.E., Reilles, A.: Tom: Piggybacking rewriting on Java. In: Baader, F. (ed.) RTA 2007. LNCS, vol. 4533, pp. 36–47. Springer, Heidelberg (2007)
22. Martí-Oliet, N., Meseguer, J., Verdejo, A.: A Rewriting Semantics for Maude Strategies. In: Proc. of WRLA 2008 (2008)
23. Andrei, O.: A Graph Rewriting Calculus: Applications to Biology and Autonomous Systems. Ph.D thesis, INPL, Nancy, France (2008)
24. Cirstea, H., Kirchner, C.: The rewriting calculus - Part I and II. Logic Journal of the IGPL 9(3), 427–498 (2001)
25. Bertolissi, C., Baldan, P., Cirstea, H., Kirchner, C.: A Rewriting Calculus for Cyclic Higher-order Term Graphs. ENTCS 127(5), 21–41 (2005)
26. Cirstea, H., Kirchner, C., Liquori, L., Wack, B.: Rewrite strategies in the rewriting calculus. ENTCS 86(4), 593–624 (2003)

Algorithms on Subtree Filament Graphs

Fanica Gavril

Computer Science Dept., Technion, Haifa 32000, Israel
gavril@cs.technion.ac.il

Abstract. We describe polynomial time algorithms to find in subtree filament graphs a minimum dominating hole, a clique intersecting all its maximum independent sets and a maximum induced split subgraph.

Keywords: subtree filament graph, dominating hole, perfect graph, induced split subgraph.

1 Introduction

We consider only finite graphs $G(V,E)$ with no parallel edges and no self-loops, where V is the set of vertices and E the set of edges; coG is the *complement* of G. For $U \subseteq V$, $G(U)$ is the *vertex subgraph* defined by U. Two vertices connected by an edge $u—v$ are called *adjacent*; a directed edge from u to v is denoted $u \rightarrow v$. We also denote $N(v,G)=\{u \mid u—v \in E\}$, $N[v,G]=N(v,G) \cup \{v\}$, $N^{in}(v,G)=\{u \mid u \rightarrow v \text{ in } G\}$, $N^{out}(v,G)=\{u \mid v \rightarrow u \text{ in } G\}$, $N^{in}[v,G]=N^{in}(v,G) \cup \{v\}$, $N^{out}[v,G]=N^{out}(v,G) \cup \{v\}$. By a path $p(v_1,v_k)=v_1—...—v_k$ we always mean a simple path; p is an *induced path* if it has no chords. A *hole* $h(v_1,v_k)=v_1—...—v_k—v_1$ is a chordless cycle with four or more vertices. A hole or an induced path h is *dominating in* G, if every vertex of G is adjacent to a vertex of h. A subset of V is a *clique* (an *independent set*) if every two of its vertices are adjacent (not adjacent, respectively). We denote by $\alpha(G)$ the size of a maximum independent set and by $\theta(G)$ the size of a minimum covering by cliques.

A graph G is an *intersection graph* of a family S of distinct subsets of a set if there is a one-to-one correspondence between the vertices of G and the subsets in S such that two subsets intersect iff their corresponding vertices are adjacent; S is a *representation* of G. Intersection graphs are of interest in various domains such as computer science, genetics and ecology [11,13,15]. A family of sets is called *Helly* if every subfamily of mutually intersecting sets has a non-empty intersection. Two distinct sets b,d *overlap* if $b \cap d \neq \phi$, $b \not\subset d$ and $d \not\subset b$. An oriented graph $G(V,E)$ is *transitive* if it is acyclic and for every three vertices $u,v,w \in V$, $u \rightarrow v, v \rightarrow w \in E$ implies $u \rightarrow w \in E$ [11]. A graph G is *perfect* if in every subgraph H of G $\alpha(H)=\theta(H)$. These graphs were introduced by Berge [1] with his famous Strong Perfect Graph Conjecture proved recently by Chudnovsky et al. [3]. Golumbic's book [11] is an excellent compendium of problems, algorithms and applications of perfect graphs and intersection graphs.

A graph $GI(V,F)$ is *chordal* if it has no holes; a chordal graph $GI(V,F)$ has $O(|V|)$ maximal cliques [6]. Every chordal graph $GI(V,F)$ is the intersection graph of a family $FI=\{a(v) \mid v \in V\}$ of subtrees of a tree T such that (by the Helly property) the

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 27–35, 2009.

intersection of every maximal subfamily of mutually intersecting subtrees contains a vertex $x \in T$ [6]. As proved in [4,9], we can assume that in *FI* no two subtrees have a common endpoint and the intersection of every maximal set of intersecting subtrees contains an edge of *T*. Assume that *T* is in a plane *PL* and let *PP* be a surface perpendicular to *PL* whose intersection with *PL* is exactly *T*. For $a(v) \in FI$, let *PP(a(v))* be the subsurface of *PP* whose intersection with *PL* is exactly *a(v)*. In *PP(a(v))*, above *T*, we connect all the endpoints of *a(v)* by a continuous function $f(v):a(v) \rightarrow R^+$ (i.e., $f(x)=0$ for every endpoint *x* of *v*) called a *subtree filament*, such that if *a(u),a(v)* overlap the two filaments intersect, if *a(u),a(v)* are disjoint, the two filaments do not intersect, and if $a(u) \subset a(v)$, the two filaments may or may not intersect. The intersection graph *G(V,E)* of a family *FT* of subtree filaments on *FI* is called a *subtree filament graph*. When *T* is a line and *FI* is a family of intervals, the filaments are called *interval filaments* and their intersection graphs are called *interval filament graphs*.

The subtree filament graphs and the interval filament graphs were introduced by Gavril [8] and he proved that they contain the families of polygon-circle, cocomparability, circular-arc and chordal graphs [6,7,10]. In [9] Gavril gave for them improved definitions and constructions and extended them to *3D-interval filament graphs*. Enright and Stewart [4] proved that the subtree filament graphs are exactly the overlap subtree graphs and this was extended in [9] to 3*D*-interval filament graphs.

Consider a family *AR* of arcs on a circle *CR*, no two arcs covering *CR*. Let *GI(V,F)* be the intersection graph of *AR*, $AR=\{a(v) \mid v \in V\}$. Assume that *CR* is in a plane *PL* and let *PP* be a surface perpendicular to *PL* whose intersection with *PL* is exactly *CR*. For $a(v) \in AR$, let *PP(a(v))* be the subsurface of *PP* whose intersection with *PL* is exactly *a(v)*. In *PP(a(v))*, above *CR*, we connect the endpoints of every $a(v) \in AR$ by a continuous function $f(v):a(v) \rightarrow R^+$ called a *circular-arc filament*, such that if *a(u),a(v)* overlap the two filaments intersect, if *a(u),a(v)* are disjoint, the two filaments do not intersect, and if $a(u) \subset a(v)$, the two filaments may or may not intersect. The intersection graph *G(V,E)* of a family of circular-arc filamets $FAR=\{v \mid a(v) \in AR\}$ is a *circular-arc filament graph* [8].

A cactus *CA* is a graph which can be decomposed by cut-vertices into chordless cycles and trees. We can define subtree filaments on a family of subtrees on a cactus *CA*, similarly to the circular-arc and subtree filaments; their intersection graphs are called *cactus-subtree graphs*.

The various families of filament graphs have polynomial time algorithms for many problems: Gavril [8,9] described polynomial time algorithms for maximum weight cliques, maximum independent sets and induced holes and antiholes of given parity. Cameron [2] described polynomial time algorithms for maximum weight induced matchings. On the other hand, Pergel [14] proved that the recognition problem for polygon circle graphs and interval filament graphs is NP-complete.

In the present paper, we describe algorithms to find in subtree filament graphs a clique intersecting all maximum independent sets (Section 2), a minimum dominating hole (Section 3) and a maximum induced split subgraph (Section 4). These algorithms are extended to circular-arc and cactus-subtree filament graphs. Note that Kratsch and Stewart [12] described a polynomial time algorithm to find minimum induced dominating paths in cocomparability graphs and Faigle et al. [5] described a polynomial time algorithm to find a maximum induced split subgraph in comparability graphs.

Given the intersection graph $G(V,E)$ of a family of filaments, we denote the filament corresponding to a vertex v of G also by v. The *base graph GI* of G is the intersection graph of its respective family of subtrees or arcs. We denote by $a(v)$ the element in GI corresponding to v. Our algorithms do not require an intersection representation of the filament graphs, but only a representation of their base graphs.

2 Algorithm for a Clique Intersecting All Maximum Independent Sets

Consider a family of graphs having a polynomial time algorithm for maximum independent set and fulfilling that every graph $G(V,E)$ in the family has a polynomial number of cocomparability subgraphs $G(V_d)$, indexed by d, which can be found in polynomial time, such that every clique of G is contained in one of them. Every $coG(V_d)$ being transitively orientable, we assume such an orientation. Our purpose is to describe an algorithm to find a clique CL which intersects all maximum independent sets of G. For a maximum independent set IND of G and any $coG(V_d)$, the vertices in $IND \cap V_d$ fulfill $v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_k$ in $coG(V_d)$; we call v_1 the *innermost vertex* of IND *in* V_d. Let $NN(v,G,d)=G(V-N(v,G)-N^{in}(v,coG(V_d)))$ denote the subgraph obtained from G by deleting the vertices adjacent to v in G and the vertices u having an edge $u \rightarrow v$ in $coG(V_d)$.

Lemma 1. A vertex v, $v \in V_d$, is the innermost vertex in V_d for some maximum independent set IND if and only if $\alpha(NN(v,G,d))=\alpha(G(V))$.

Proof. If $v \in V_d$, is the innermost vertex in V_d for some maximum independent set IND, then, clearly, $\alpha(NN(v,G,d))=\alpha(G(V))=|IND|$.

Conversely, assume that $\alpha(NN(v,G,d))=\alpha(G(V))$ for some $v \in V_d$. Since v has no adjacent vertices in $G(NN(v,G,d))$, every maximum independent set IND of $G(NN(v,G,d))$ contains v and contains no u having $u \rightarrow v$. Thus, v is the innermost vertex of IND in V_d. □

The algorithm to find in G a clique CL which intersects all its maximum independent sets, works as follows: We consider every cocomparability subgraph $G(V_d)$ of G. If $\alpha(G(V-V_d))=\alpha(G(V))$, $G(V_d)$ contains no such CL. Assume that $\alpha(G(V-V_d))<\alpha(G(V))$. We want to find in $G(V_d)$ a clique CL of G, such that $\alpha(G(V-CL))<\alpha(G(V))$. Let $Y=\{v \mid v \in V_d, \alpha(NN(v,G,d))=\alpha(G(V))\}$; Y is the set of innermost vertices in V_d of the maximum independent sets. If Y is a clique, then $CL=Y$. Assume that Y is not a clique. Consider $u,v \in Y$, u innermost for IND_u and v innermost for IND_v, such that $u \rightarrow v$ in $coG(V_d)$. For every $w \in IND_v \cap V_d$ we have $u \rightarrow v \rightarrow w$, hence $u \rightarrow w$ in $coG(V_d)$ implying that u is not adjacent in G to any vertex of $IND_v \cap V_d$. Thus $u \notin CL$, otherwise $CL \cap IND_v = \phi$; still $CL \cap IND_u$ can be a vertex different from u. We delete from V_d and Y every vertex $u \in Y$ having $u \rightarrow v$, $v \in Y$, in $coG(V_d)$. If the new V_d fulfills $\alpha(G(V-V_d))=\alpha(G(V))$, then it does not contain a clique CL intersecting all the maximum independent sets of G and we go to the next V_d. If the new V_d fulfils $\alpha(G(V-V_d))<\alpha(G(V))$, we look in $V_d - Y$ for the vertices which replace in Y the deleted vertices as innermost

in the new V_d, by adding to Y the vertices $v \in V_d - Y$ fulfilling $\alpha(NN(v,G,d)) = \alpha(G(V))$. We continue in this way, until for some V_d we have $\alpha(G(V-V_d)) < \alpha(G(V))$ and Y is a clique, or we declare that such a clique does not exists.

Let $G(V,E)$ be an intersection graph of a family FT of subtree filaments with base graph GI; GI is the intersection graph of a family $FI=\{a(v) \mid v \in V\}$ of subtrees on a tree T. Every maximal clique CL of G is contained in a maximal clique c of GI. The graph GI is chordal and has $O(|V|)$ maximal cliques which can be found by the algorithm described in [6]. By the Helly property, the intersection of the subtrees corresponding to any maximal clique c of G contains a point d_c of T. Consider now a point $d=d_c$ on T and let $V_d=\{v \mid v \in V,\ d \in a(v)\}$. For every two vertices $v,u \in V_d$, there exists $a(v) \cap a(u) \neq \phi$, hence V_d is a maximal clique c of the chordal graph GI. Every two vertices $v,u \in V_d$ not adjacent in G, have either $a(v) \subset a(u)$ or $a(u) \subset a(v)$, since $u \cap v = \phi$ and $d \in a(v) \cap a(u)$. If $a(v) \subset a(u)$, we orient in $coG(V_d)$ the edge $v \text{—} u$ as $v \rightarrow u$. We obtain a transitive orientation of $coG(V_d)$. Therefore, there are $O(|V|)$ points d on T and cocomparability subgraphs $G(V_d)$ of G, such that every clique CL of G is contained in some $G(V_d)$.

Let $G(V,E)$ be an intersection graph of circular-arc filaments with base graph GI; GI is the intersection graph of a family AR of arcs on a circle CR, no two arcs covering CR. Let c be a maximal clique of GI corresponding to a set of mutually intersecting arcs of AR; AR may not have the Helly property, but [8], there are two points $x,y \in CR$ such that each arc corresponding to a vertex in CL contains x or y and not both. Consider now two points x,y on CR and let $V_{x,y}$ be the subset of vertices v in V such that $a(v)$ contains either x or y but not both. For every two non-adjacent vertices $v,u \in V_{x,y}$, we orient in $coG(V_{x,y})$ the edge $v \text{—} u$ as $v \rightarrow u$ whenever filament v is between filament u and the point x of CR. We obtain a transitive orientation of $coG(V_{x,y})$; it is enough to consider $O(|V|^2)$ such pairs x,y defined by the pairs of arcs containing no arc endpoints. Therefore, there are $O(|V|^2)$ cocomparability subgraphs $G(V_d)$ of G, such that every clique CL of G is contained in some $G(V_d)$.

In a cactus-subtree filament graph G, a clique CL is contained either in some $V_{x,y}$, x,y points in a circle of the cactus CA or in some V_d for some point d of CA, as shown above for circular-arc and subtree filament graphs.

Therefore, the algorithm to find a clique intersecting all maximum independent sets, can be applied to subtree filament, circular-arc filament and cactus-subtree filament graphs, since they have algorithms [8] to evaluate $\alpha(G)$ in time $O(|V|^3)$. The algorithm works in time $O(|V|^5)$ for subtree filament graphs and in time $O(|V|^6)$ for circular-arc filament and cactus-subtree filament graphs.

In a perfect graph G and in any of its subgraphs H, $\alpha(H)=\theta(H)$. Thus, any clique CL in a minimum covering by cliques of G, intersects all maximum independent sets of G. In a perfect graph, the converse is also true: if a clique CL intersects all maximum independent sets, then it is a clique of a minimum covering by cliques, and $\alpha(G(V-CL)) = \alpha(G(V)) - 1 = \theta(G(V)) - 1$. Therefore, we can use the above algorithm to find a minimum covering by cliques in a perfect filament graph: we find a clique CL intersecting all maximum independent sets, we delete CL from G and continue on the graph $G(V-CL)$. The algorithm works in time $O(|V|^6)$ for subtree filament graphs and in time $O(|V|^7)$ for circular-arc filament and cactus-subtree filament graphs.

3 Minimum Dominating Holes

First we prove that the existence problem of a dominating hole in general graphs is NP-complete, using a reduction from the NP-complete problem 3SAT. Next, we give a characterization of the dominating holes in subtree filament graphs. The problems are:

The 3SAT problem:
Instance: A set U of Boolean variables and a collection C of clauses over U, where a clause is a disjunction of at most three literals and a literal is a variable $x \in U$ or its negation $-x$.
Question: Is there an assignment of *true* or *false* to the variables in U such that each clause in C is *true*? Such a truth assignment is called *satisfiable*.

The Dominating Hole problem:
Instance: A graph $G(V,E)$.
Question: Is there a dominating hole in G?

Lemma 2. The 3SAT problem is reducible to the Dominating Hole problem, hence the Dominating Hole problem is NP-complete.

Proof. Consider an instance $U=\{u_1,u_2,\ldots,u_r\}$, $C=\{c_1,c_2,\ldots,c_s\}$ of the 3SAT problem. We construct a graph G with vertex set $X \cup C$, where $X=\{u_1,-u_1,y_1,u_2,-u_2,y_2,\ldots,u_r,-u_r,y_r\}$, and edge set defined as follows: $\{u_i,-u_i,u_{i+1},-u_{i+1}\}$, $\{u_i,-u_i,y_i\}$ are cliques for every $1 \le i \le r-1$, $\{u_r,-u_r,u_1,-u_1\}$, $\{u_r,-u_r,y_r\}$ are cliques, and every c in C is adjacent to its literals.

If G has no dominating holes, then there is no satisfiable truth assignment for C, since such a truth assignment would define a hole of G which dominates $C \cup X$.

Conversely, consider a dominating hole h of G; h contains a vertex from every pair $u_i,-u_i$ to dominate y_i. Assume that h contains both $u_i,-u_i$ for some variable u_i. Since one of $u_{i+1},-u_{i+1}$, say u_{i+1}, is contained in h, it follows that h contains the clique $\{u_i,-u_i,u_{i+1}\}$, contradicting that h is a hole. Thus, $|h \cap \{u_i,-u_i\}|=1$ for every i, $h \subset X$ and h dominates C; by assigning *true* to the literals in h we obtain a satisfiable truth assignment of C. □

Let $G(V,E)$ be an intersection graph of a family of subtree filaments with base graph GI; GI is the intersection graph of a family $FI=\{a(v) \mid v \in V\}$ of subtrees on a tree T. Let us define $IN(v)=\{u \mid u \notin N[v,G]$ *and* $a(u) \subset a(v)\}$, $OUT(v)=\{u \mid u \notin N[v,G]$ *and* $a(v) \cap a(u)=\phi$ *or* $a(v) \subset a(u)\}$. Let $p(v_1,v_k)$ be an induced path in $G(IN(v))$. Then $a(v_1) \cup a(v_2) \cup \ldots \cup a(v_k)$ is a subtree of T, and the filament $w=v_1 \cup v_2 \cup \ldots \cup v_k$ is properly contained in the surface $s(v)$ bounded by $v \cup a(v)$, since for getting out of $s(v)$, w must intersect v. Using this observation, Gavril [9] proves the following Lemma:

Lemma 3. [9] Consider a hole $h(v_1,v_k)$ of G. Then, either h is a hole of its base graph GI, or h has a vertex v_j such that $h-\{v_j,v_{j-1},v_{j+1}\}$ is an induced path of $GI(IN(v_j))$.

Using Lemma 3, we characterize dominating holes:

Lemma 4. Consider a dominating hole h of a subtree filament graph G. Then, either h is a hole of GI which dominates G or h has a vertex v_j such that $\{v_{j-1},v_{j+1}\}$ dominates $G(OUT(v_j))$ and $h\text{-}\{v_j,v_{j-1},v_{j+1}\}$ is an induced path of $GI(IN(v_j))$ which dominates the vertex set $IN(v_j)\text{-}N[v_j,v_{j-1},v_{j+1},G]$.

Proof. Consider a dominating hole h of a filament graph G. By Lemma 3, h is a hole of GI, or h has a vertex v_j such that $h\text{-}\{v_j,v_{j-1},v_{j+1}\}$ is an induced path of $GI(IN(v_j))$.

In the first case, the hole h of GI dominates G. In the second case, the connected filament defined by $h\text{-}\{v_j,v_{j-1},v_{j+1}\}$ is properly contained in the surface bounded by $v_j \cup a(v_j)$ and cannot intersect the filaments corresponding to the vertices in $OUT(v_j)$; hence, the vertices in $OUT(v_j)$ are dominated by $\{v_{j-1},v_{j+1}\}$. The vertices which are not dominated by $\{v_j,v_{j-1},v_{j+1}\}$ are those in $IN(v_j)\text{-}N[v_j,v_{j-1},v_{j+1},G]$. Thus, $h\text{-}\{v_j,v_{j-1},v_{j+1}\}$ is an induced path of $GI(IN(v_j))$ which dominates the vertex set $IN(v_j)\text{-}N[v_j,v_{j-1},v_{j+1},G]$. □

The algorithm to find a dominating hole h in a subtree filament graph G works as follows: Chordal graphs have no holes. We consider every three vertices $\{v_j,v_{j-1},v_{j+1}\}$ which dominate $OUT(v_j)$, such that $v_{j-1},v_{j+1} \in N[v_j,G]$, and $v_{j-1} \notin N[v_{j+1},G]$, as $\{v_j,v_{j-1},v_{j+1}\}$ of the hole h we try to find. We consider every two non adjacent vertices v_{j-2},v_{j+2} such that $v_{j-2} \in N[v_{j-1},G] \cap IN(v_j)$, $v_{j+2} \in N[v_{j+1},G] \cap IN(v_j)$,

Let x,y be two farthest endpoints of $a(v_{j+2}),a(v_{j-2})$ in $a(v_j)$. Let $PT(x,y)$ be the unique path in $a(v_j)$ connecting x and y, and let GI' be the interval graph defined by the intervals $I=\{PT(x,y) \cap a(v_q) \mid v_q \in IN(v_j)\text{-}N[v_j,v_{j-1},v_{j+1},G]\}$; since $a(v_j)$ is a tree, if h exists, then $h\text{-}\{v_j,v_{j-1},v_{j+1}\}$ is an induced path of GI'.

Therefore, the problem reduces to finding in the interval graph GI', defined by I on $PT(x,y)$, a minimum induced path $p(u_1,u_k)$, between two given vertices u_1,u_k, $\{u_1,u_k\}=\{v_{j+2},v_{j-2}\}$, which dominates all vertices in $V_j=IN(v_j)\text{-}N[v_j,v_{j-1},v_{j+1},G]$. This is done as follows: We assume that $PT(x,y)$ is drawn horizontally from x at left to y at right on a line L with all other subtrees of T corresponding to the vertices in V_j attached below. For every interval $i \in I$ we denote by $l(i)$ its left endpoint in PT and by $r(i)$ its right endpoint in PT. On this representation I, PT we perform the following algorithm:

We consider every vertex u_2 in GI' whose interval overlaps $i(u_1)$ at the right, such that $\{u_1,u_2\}$ dominates the vertices $u \in V_j$ having $l(i(u))<r(i(u_1))$ or having subtrees attached below $i(u_1)$. Note that if $l(i(u))<r(i(u_1))<r(i(u))$ then u_1 dominates u. We continue on L from left to right on the left endpoints of the intervals, from $r(i(u_1))$ to $l(i(u_k))$. Assume that we are at $i(u_i)$ and for every $i(u_{i-1})$ overlapping $i(u_i)$ at the left we found already a minimum induced path $p(u_1,u_{i-1},u_i)$ ending in u_{i-1},u_i, which dominates all the vertices $u \in V_j$ having $l(i(u))<r(i(u_{i-1}))$ or having subtrees attached below $i(u_1) \cup \ldots \cup i(u_{i-1})$. If $u_i \notin N(u_k,G)$ we consider every $i(u_{i+1})$ overlapping $i(u_i)$ at the right (advancing on the left endpoints), and for every such u_{i+1} we consider every u_{i-1} fulfilling: u_{i-1} overlaps u_i at the left, $i(u_{i-1}) \cap i(u_{i+1})=\phi$ and u_i,u_{i+1} dominate the vertices $u \in V_j$ having $r(i(u_{i-1}))<l(i(u))<r(i(u_i))$ or having subtrees attached below. For a given

u_{i+1}, we choose among these paths $p(u_1,u_{i-1},u_i)$ the one with a minimum length, we add to it u_{i+1} and this path is assigned to the pair u_i,u_{i+1}. If $u_i \in N(u_k,G)$ and u_i,u_k dominate all the vertices $u \in V_j$ having $r(i(u_{i-1})) < l(i(u))$ or having subtrees attached below, then $p(u_1,u_{i-1},u_i) \cup \{u_k\}$ is a dominating induced path. Among all such pairs u_{i-1},u_i, we take the one with minimum length.

The algorithm to find a minimum induced dominating hole in a circular-arc filament graph G, with base graph GI on a circle CR, works as follows: Lemmas 3,4 are true also for circular-arc filament graphs, implying that a dominating hole $h(v_1,v_k)$ of G either is a hole of GI which dominates G or has a vertex v_j such that $\{v_{j-1},v_{j+1}\}$ dominates $OUT(v_j)$ and $h\text{-}\{v_j,v_{j-1},v_{j+1}\}$ is an induced path of $GI(IN(v_j))$ which dominates the vertex set $IN(v_j)\text{-}N[v_j,v_{j-1},v_{j+1},G]$. To find a minimum induced hole h of GI which dominates G, we consider every vertex as v_1 of h, we take every two non-adjacent vertices in $N(v_1,G)$, which dominate the vertices in $OUT(v_1)$ as v_2,v_k and on the arc $CR\text{-}a(v_1)$ we find a minimum induced path from v_2 to v_k which dominates in the vertex set $IN(v_1)\text{-}N[v_1,v_2,v_k,G]$, using the above algorithm for interval graphs. Among all choices of vertices, we take the one giving a minimum dominating hole. To find a minimum induced dominating hole h of G which may not be a hole of GI, we consider every vertex as v_j of h and the intersections of the other arcs with the arc $a(v_j)$ as a family of intervals on $a(v_j)$. We then apply the algorithm for subtree (in fact interval) filament graphs above. A similar algorithm exists for cactus-subtree filament graphs. The algorithms work in time $O(|V|^7)$.

4 Algorithm for a Maximum Induced Split Subgraph

Reference [5] describes a simple, nice algorithm to find a maximum induced split subgraph in a comparability graph and proves that the problem is NP-complete for general graphs. We extend this algorithm to any family of graphs having a polynomial time algorithm for maximum independent set and fulfilling that every graph $G(V,E)$ in the family has a polynomial number of comparability or cocomparability subgraphs $G_d=G(V_d)$, indexed by d, which can be found in polynomial time, such that every clique of G is contained in one of them. By the definition of the $G(V_d)$'s in Section 2, this family of graphs contains the circle graphs, the subtree filament graphs, the circular-arc filament graphs and the cactus subtree filament graphs. When G_d or its complement coG_d is transitively orientable, we assume such an orientation. Clearly, we can find in G_d a maximum clique. A maximum induced split subgraph of G is composed of a maximum independent set and a maximum clique, which are disjoint or have exactly one vertex in common. Thus, the algorithm for a maximum induced split subgraph must find a pair of disjoint maximum independent set and maximum clique, if such a pair exists, otherwise it can take any pair of maximum independent set and maximum clique.

First, we consider maximum cliques CL of $G(V,E)$ contained in subgraphs G_d of G such that coG_d is transitively oriented. For every maximum independent set IND of G, the set $IND(d)=IND \cap V_d$ is a clique (not necessarily maximal) of coG_d and $IND(d)=\{v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_{|IND(d)|}\}$ in the orientation of coG_d.

Lemma 5. For every pair of disjoint maximum clique CL, in a cocomparability graph G_d, and maximum independent set IND, where $IND(d)=\{v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_{|IND(d)|}\}$ in coG_d, either $CL \subseteq V_d - N^{in}[v_{|IND(d)|}, coG_d]$ or $CL \subseteq V_d - N^{out}[v_1, coG_d]$ or there is a vertex v_i in $IND(d)$ such that $CL \subseteq V_d - (N^{out}[v_i, coG_d] \cup N^{in}[v_{i-1}, coG_d])$.

Proof. Consider a maximum clique $CL \subseteq V_d$ and a maximum independent set IND s.t. $IND \cap CL = \phi$, $IND(d)=\{v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_{|IND(d)|}\}$ in coG_d and $CL \cap N^{in}[v_{|IND(d)|}, coG_d] \neq \phi$.

Let i be the minimal index such that $CL \cap N^{in}[v_i, coG_d] \neq \phi$. Then $CL \cap N^{out}[v_i, coG_d] = \phi$ since $u,w \in CL$, $u \in N^{in}[v_i, coG_d]$ and $w \in N^{out}[v_i, coG_d]$ would imply $u \rightarrow v_i \rightarrow w$ in coG_d, which by the transitivity of coG_d would imply that u,w are adjacent in coG_d and thus are not adjacent in the clique CL. If $i>1$ then $CL \cap N^{in}[v_{i-1}, coG_d] = \phi$, by the minimality of i, thus $CL \subseteq V_d - (N^{out}[v_i, coG_d] \cup N^{in}[v_{i-1}, coG_d])$. If $i=1$ then $CL \subseteq V_d - N^{out}[v_1, coG_d]$. □

Next, we consider maximum cliques CL of $G(V,E)$ contained in transitively oriented subgraphs G_d of G, where $CL=\{v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_{|CL|}\}$ in the orientation of G_d.

Lemma 6. For every pair of disjoint maximum independent set IND, and maximum clique $CL=\{v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_{|CL|}\}$ contained in a comparability graph G_d, either $IND \subseteq V - N^{in}[v_{|CL|}, G_d]$ or $IND \subseteq V - N^{out}[v_1, G_d]$ or there is a vertex v_i in CL such that $IND \subseteq V - (N^{out}[v_i, G_d] \cup N^{in}[v_{i-1}, G_d])$.

Proof. Consider a maximum independent set IND and a maximum clique $CL=\{v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_{|CL|}\}$ such that $IND \cap CL = \phi$ and $IND \cap N^{in}[v_{\setminus CL}, G_d] \neq \phi$.

Let i be the minimal index such that $IND \cap N^{in}[v_i, G_d] \neq \phi$. Then, $IND \cap N^{out}[v_i, G_d] = \phi$ since $u,w \in IND$, $u \in N^{in}[v_i, G_d]$ and $w \in N^{out}[v_i, G_d]$ would imply $u \rightarrow v_i \rightarrow w$, which by the transitivity of G_d would imply that u,w are adjacent in the independent set IND. If $i>1$ then $IND \cap N^{in}[v_{i-1}, G_d] = \phi$, by the minimality of i, thus, $IND \subseteq V - (N^{out}[v_i, G_d] \cup N^{in}[v_{i-1}, G_d])$. If $i=1$ then $IND \subseteq V - N^{out}[v_1, G_d]$. □

By Lemmas 5,6, we can find in G a pair of disjoint maximum independent set and maximum clique, as follows: We consider every G_d.

If G_d is a cocomparability graph, we check if G_d has a vertex v such that $V_d - N^{in}[v, coG_d]$ contains a maximum clique of G and $(V - V_d) \cup N^{in}[v, coG_d]$ contains a maximum independent set of G, or if G_d has a vertex v such that $V_d - N^{out}[v, coG_d]$ contains a maximum clique of G and $(V - V_d) \cup N^{out}[v, coG_d]$ contains a maximum independent set of G, or if G_d has two adjacent vertices $u \rightarrow v$ in coG_d such that $V_d - (N^{out}[v, coG_d] \cup N^{in}[u, coG_d])$ contains a maximum clique of G, and $(V - V_d) \cup N^{out}[v, coG_d] \cup N^{in}[u, coG_d]$ contains a maximum independent set of G.

If G_d is a comparability graph, we check if G_d, has a vertex v such that $N^{in}[v, G_d]$ contains a maximum clique of G and $V - N^{in}[v, G_d]$ contains a maximum independent set of G, or if G_d has a vertex v such that $N^{out}[v, G_d]$ contains a maximum clique of G and $V - N^{out}[v, G_d]$ contains a maximum independent set of G, or if G_d, has two adjacent vertices $u \rightarrow v$ in G_d such that $N^{out}[v, G_d] \cup N^{in}[u, G_d]$ contains a maximum clique of G, and $V - (N^{out}[v, G_d] \cup N^{in}[u, G_d])$ contains a maximum independent set of G. We can

use this algorithm to find a maximum induced split subgraph in subtree filament graphs in $O(|V|^5)$ time and in circular-arc or cactus-subtree filament graphs in $O(|V|^6)$ time.

Lemmas 5,6 hint to a new interesting subfamily of subtree filament graphs in which the subraphs $G_d=G(V_d)$, V_d as defined in Section 2, are both comparability and cocomparability graphs, that is they are permutation graphs.

Acknowledgements. I am grateful to the two referees whose suggestions greatly improved the paper.

References

1. Berge, C.: Graphs and Hypergraphs. North-Holland Publ. Co., Amsterdam (1976)
2. Cameron, K.: Induced Matchings in Intersection Graphs. Discrete Math. 278, 1–9 (2004)
3. Chudnovsky, M., Robertson, N., Seymour, P.D., Thomas, R.: The Strong Perfect Graph Theorem. Ann. Math. 164, 51–229 (2006)
4. Enright, J., Stewart, L.: Subtree Filament Graphs are Subtree Overlap Graphs. Inform. Proc. Lett. 104, 228–232 (2007)
5. Faigle, U., Fuchs, B., Peis, B.: Note on Maximal Split-Stable Subgraphs. Discrete Appl. Math. 155, 2031–2038 (2007)
6. Gavril, F.: The Intersection Graphs of Subtrees in Trees are Exactly the Chordal Graphs. J. Comb. Theory Ser. B 16, 47–56 (1974)
7. Gavril, F.: Algorithms on Circular-Arc Graphs. Networks 4, 357–369 (1974)
8. Gavril, F.: Maximum Weight Independent Sets and Cliques in Intersection Graphs of Filaments. Inform. Proc. Lett. 73, 181–188 (2000)
9. Gavril, F.: 3-D-Interval-Filament Graphs. Discrete Appl. Math. 155, 2625–2636 (2007)
10. Golumbic, M.C., Rotem, D., Urrutia, J.: Comparability Graphs and Intersection Graphs. Discrete Math. 43, 37–46 (1983)
11. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, New York (1980)
12. Kratsch, D., Stewart, L.: Domination in Cocomparability Graphs. SIAM J. Disc. Math. 6, 400–417 (1993)
13. Pevsner, P.A.: Computational Molecular Biology: An Algorithmic Approach. The MIT Press, Cambridge (2000)
14. Pergel, M.: Recognition of Polygon-Circle Graphs and Graphs of Interval Filaments is NP-complete. In: Brandstädt, A., Kratsch, D., Müller, H. (eds.) WG 2007. LNCS, vol. 4769, pp. 238–247. Springer, Heidelberg (2007)
15. Roberts, F.S.: Discrete Mathematical Models with Applications to Social, Biological and Environmental Problems. Prentice-Hall, Englewood Cliffs (1976)

A Note on the Recognition of Nested Graphs

Mark Korenblit[1] and Vadim E. Levit[2]

[1] Holon Institute of Technology, Israel
korenblit@hit.ac.il
[2] Ariel University Center of Samaria, Israel
levitv@ariel.ac.il

Abstract. We consider a two-terminal directed acyclic graph (*st-dag*) characterized by a special structure of its mincuts and call it a *nested graph*. This graph is of interest as an st-dag with a minimum possible number of mincuts.We present a linear-time algorithm for recognizing nested graphs.

1 Basic Concepts

A two-terminal directed acyclic graph (*st-dag*) has only one source s and only one sink t [2]. In an st-dag, every vertex lies on some path from s to t.

A *multigraph* may contain multiple edges between any pair of vertices. A *simple graph* is a graph without multiple edges.

A cut is a set of edges whose removal disconnects t from s. A *mincut* is a minimal cut [1], [3], [8], [10].

For example, the st-dag in Figure 1 has four mincuts: $\{(1,2),(1,3),(1,4)\}$, $\{(2,4),(1,3),(1,4)\}$,$\{(1,2),(3,4),(1,4)\}$, $\{(2,4),(3,4),(1,4)\}$.

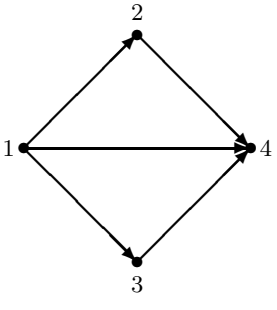


Fig. 1. An st-dag

A *series-parallel graph* is an st-dag defined recursively as follows:

(i) A single edge (u,v) is a series-parallel graph with source u and sink v.

(ii) If G_1 and G_2 are series-parallel graphs, so is the graph obtained by either of the following operations:

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 36–40, 2009.

(a) Parallel composition: identify the source of G_1 with the source of G_2 and the sink of G_1 with the sink of G_2.

(b) Series composition: identify the sink of G_1 with the source of G_2.

For instance, the st-dag in Figure 1 is a series-parallel graph.

The construction of a series-parallel graph in accordance with its recursive definition may be represented by a binary tree which is called a *decomposition tree*. The edges of the graph are represented by the leaves of the tree. The inner nodes of the tree are labeled S, indicating a series composition, or P, indicating a parallel composition. Each subtree in the decomposition tree corresponds to a series-parallel subgraph. Figure 2 shows an example of a series-parallel graph (a) together with its decomposition tree (b).

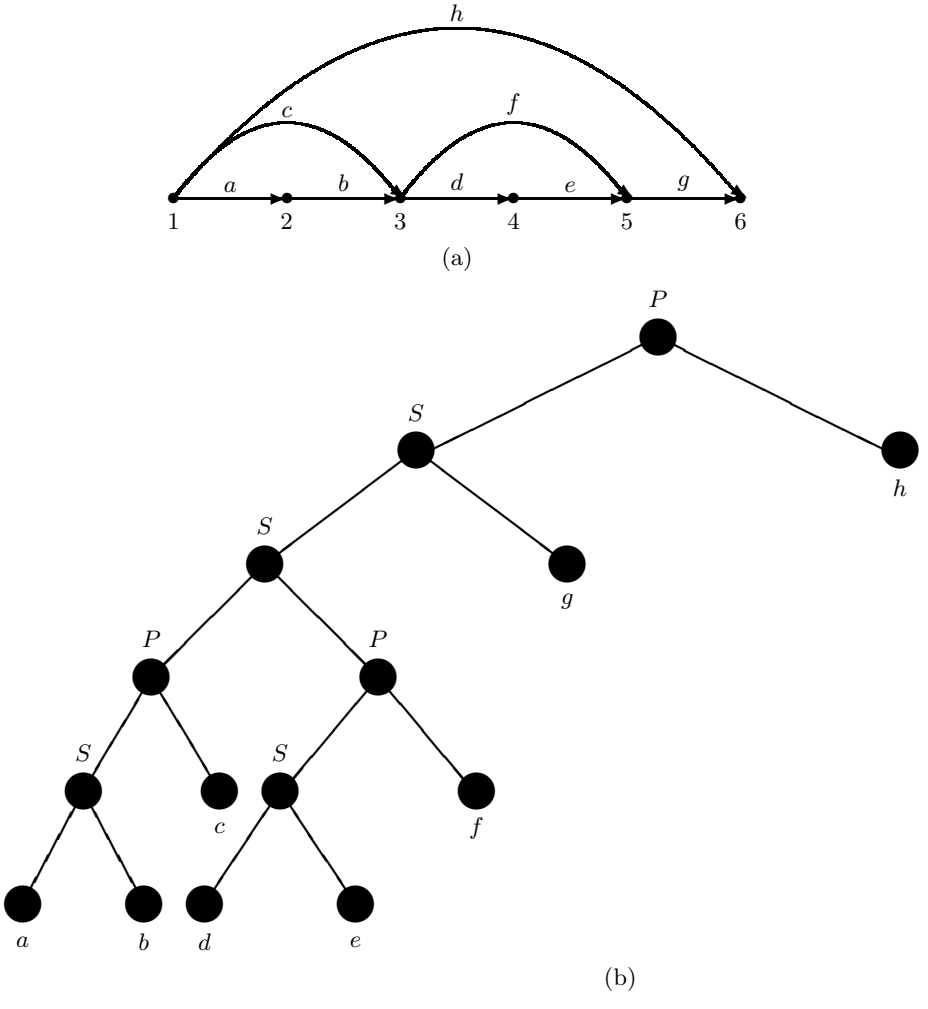


Fig. 2. A series-parallel graph (a) and its decomposition tree (b)

2 Nested Graphs

For any mincut C in the st-dag $G = (V, E)$, Provan and Ball [8] identify the two disjoint sets: $SN(C) = \{u \in V$: there exists a path from s to u containing no edges of $C\}$ and $TN(C) = \{v \in V$: there exists a path from v to t containing no edges of $C\}$. The mincut C consists exactly of those edges with exactly one endpoint in $SN(C)$ and one endpoint in $TN(C)$.

Consider an st-dag G that has μ mincuts denoted C_i, $i = 1, 2 \ldots, \mu$. We define G as a *nested graph* if its mincuts can be ordered in such a way that

$$SN(C_1) \subset SN(C_2) \ldots \subset SN(C_\mu).$$

The series-parallel graph G in Figure 2(a) has five mincuts: $\{a, c, h\}$, $\{b, c, h\}$, $\{d, f, h\}$, $\{e, f, h\}$, and $\{g, h\}$. Their sets $SN(C_i)$ $(i = 1, 2, 3, 4, 5)$ are $\{1\}$, $\{1, 2\}$, $\{1, 2, 3\}$, $\{1, 2, 3, 4\}$, and $\{1, 2, 3, 4, 5\}$, respectively. Thus G is nested. The series-parallel graph in Figure 1 is not nested. The sets $SN(C_i)$ $(i = 1, 2, 3, 4)$ of its mincuts are $\{1\}$, $\{1, 2\}$, $\{1, 3\}$, and $\{1, 2, 3\}$.

The following general findings on structural features of nested graphs were presented in [4] and [6].

Proposition 1. *A nested graph is series-parallel.*

Proposition 2. *If G is an n-vertex st-dag, the following are equivalent.*

1. If G is not a single-edge graph, it can be obtained by a parallel composition of a nested graph and a single edge or by a series composition of nested graphs.
2. G is a nested graph.
3. G has exactly $n - 1$ mincuts.

Proposition 3. *Suppose an st-dag G_1 is a nested graph and G_2 is a 2-vertex st-dag that presents a set of parallel edges with a common source and a common sink. Then the st-dag G obtained by a parallel composition of G_1 and G_2 is nested.*

As follows from Proposition 2, a non-single-edge st-dag is nested if and only if it can be obtained by a parallel composition of a nested graph and a single edge or by a series composition of nested graphs. So, the graph that cannot be obtained in this way, is not nested.

As shown in [4] and [7], the minimum possible number of mincuts in an n-vertex st-dag is $n - 1$. Hence, by Proposition 2, a nested graph is of interest as an st-dag with a minimum possible number of mincuts.

3 The Nested Graph Recognition Algorithm

The proposed algorithm for the nested graph recognition uses Propositions 1, 2, and 3. It is based on revealing the specific structure, inherent in a nested graph exclusively, in a series-parallel graph.

It is clear that a single edge is a nested graph. As follows from Proposition 2, a non-single-edge simple graph is nested if and only if it is obtained by a parallel composition of a nested graph and a single edge or by a series composition of nested graphs. In the general case, when a considered st-dag may be a multigraph, it is nested if and only if it is obtained by a parallel composition of a nested graph and a set of parallel edges (see Proposition 3) or by a series composition of nested graphs. As noted in Section 2, the graph that cannot be obtained in this way, is not nested.

An algorithm for the nested graph recognition is organized as follows. Initially, a series-parallel graph recognition algorithm is executed. If the graph is non-series-parallel, then, by Proposition 1, it is not nested. There exist several algorithms recognizing series-parallel graphs, which are linear in the number of edges in a graph (see [9], [11], [12]). Specifically, as noted in [11], the algorithm presented there can be modified so that it computes a binary decomposition tree of a graph under consideration, on condition that the graph is series-parallel. Thus, we run the algorithm from [11] accompanied by computing the binary decomposition tree. Further we apply an additional routine which is based on the postorder tree walk procedure that traverses the tree and checks whether it respects the structure of the nested graph.

Therefore, the structure of the recursive algorithm for the nested graph recognition reads as follows.

1. Replace in the given graph G the sets of parallel edges with single edges; let G' denote the obtained simple graph.
2. If G' is a single edge, then G is nested, and the procedure stops.
3. Run the algorithm from [11] to check that G' is a series-parallel graph, while constructing the decomposition tree T. If G' is not series-parallel, then G is not nested, and the procedure stops.
4. If G' is a series-parallel graph, traverse T in postorder and check that (a) for every parallel composition, one subgraph is a single edge and another subgraph is nested; (b) for every series composition, both subgraphs are nested. If this condition is true, then G is nested. Otherwise, G is not nested.

The complexity of the algorithm is determined as follows. As noted, the algorithm from [11] requires $O(m)$ time for an m-edge graph. The decomposition tree T of a series-parallel graph is a *full binary tree* (each node has exactly zero or two children). A k-leaf full binary tree has $2k - 1$ nodes. Hence, since T has m leaves, the number of nodes in T is equal to $2m - 1$. On the other hand, the time taken by the postorder tree walk procedure depends linearly on the number of nodes in the tree. Thus, the running time of the algorithm is $O(m)$, where m is a number of edges in the simple graph obtained by the replacement in the given graph G the sets of parallel edges with single edges. The number of edges in an n-vertex simple series-parallel graph does not exceed $2n - 3$ (see [9], [5]). Therefore, if G is a series-parallel graph, then the algorithm is executed in $O(n)$ time.

4 Conclusions

We have considered a special kind of series-parallel graphs called nested graphs. Using the recursive structure of nested graphs, we have presented the algorithm for their recognition, which can be used for both simple graphs and multigraphs. The algorithm is linear-time in the size of a graph.

References

1. Ball, M.O., Colbourn, C.J., Provan, J.S.: Network Reliability. In: Handbooks in Operations Research and Management Science. Network Models, vol. 7, pp. 673–762. Elsevier Science B.V., North-Holland, Amsterdam (1995)
2. Bein, W.W., Kamburowski, J., Stallmann, M.F.M.: Optimal Reduction of Two-Terminal Directed Acyclic Graphs. SIAM Journal of Computing 21(6), 1112–1129 (1992)
3. Colbourn, C.J.: The Combinatorics of Network Reliability. Oxford University Press, Oxford (1987)
4. Korenblit, M.: Efficient Computations on Networks, Ph.D. Thesis, Bar-Ilan University, Israel (2004)
5. Korenblit, M., Levit, V.E.: On Algebraic Expressions of Series-Parallel and Fibonacci Graphs. In: Calude, C.S., Dinneen, M.J., Vajnovszki, V. (eds.) DMTCS 2003. LNCS, vol. 2731, pp. 215–224. Springer, Heidelberg (2003)
6. Korenblit, M., Levit, V.E.: Nested Graphs. In: 5th Cracow Conference on Graph Theory USTRON 2006, Ustron, Poland, September 2006. Electronic Notes in Discrete Mathematics, vol. 24, pp. 93–99. Elsevier B.V., Amsterdam (2006)
7. Korenblit, M., Levit, V.E.: Minimal Cuts in Two-Terminal Directed Acyclic Graphs. In: 39th Southeastern International Conference on Combinatorics, Graph Theory, and Computing, Florida Atlantic University, Boca Raton, Florida (March 2008)
8. Provan, J.S., Ball, M.O.: Computing Network Reliability in Time Polynomial in the Number of Cuts. Operations Research 32(3), 516–526 (1984)
9. Schoenmakers, L.A.M.: A New Algorithm for the Recognition of Series Parallel Graphs, Report CS-R9504, ISSN 0169-118X, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands (1995)
10. Shier, D.R.: Network Reliability and Algebraic Structures. Oxford University Press, Oxford (1991)
11. Valdes, J., Tarjan, R.E., Lawler, E.L.: The Recognition of Series Parallel Digraphs. SIAM Journal of Computing 11(2), 298–313 (1982)
12. Wagner, D.K.: Forbidden Subgraphs and Graph Decomposition. Networks 17, 105–110 (1987)

Asynchronous Congestion Games*

Michal Penn[1], Maria Polukarov[2,**], and Moshe Tennenholtz[1,3]

[1] Faculty of Industrial Engineering and Management, Technion, Haifa 32000, Israel
{mpenn,moshet}@ie.technion.ac.il
[2] School of Electronics and Computer Science, University of Southampton, SO17 1BJ, Southampton, UK
mp3@ecs.soton.ac.uk
[3] Microsoft Israel R&D Center
moshet@microsoft.com

Abstract. We introduce a new class of games, *asynchronous congestion games* (ACGs). In an ACG, each player has a task that can be carried out by any element of a set of resources, and each resource executes its assigned tasks in a random order. Each player's aim is to minimize his expected cost which is the sum of two terms – the sum of the fixed costs over the set of his utilized resources and the expected cost of his task execution. The cost of a player's task execution is determined by the earliest time his task is completed, and thus it might be beneficial for him to assign his task to several resources. We show the existence of pure strategy Nash equilibria in ACGs. Moreover, we present a polynomial time algorithm for finding such an equilibrium in a given ACG.

1 Introduction

Congestion games received a lot of attention in the recent game theory and computer science literature [2,3,4,5,6]. In a classic congestion game [7], each player chooses a subset of a set of available resources in order to perform his task. The cost of using a particular resource is determined by its congestion. The important property of congestion games is that they possess pure strategy Nash equilibria. Monderer and Shapley [6] introduced the notions of *potential function* and *potential game* and proved that the existence of a potential function implies the existence of a pure strategy Nash equilibrium. They also showed that the classes of finite potential games and congestion games coincide.

Classic congestion games can be viewed as synchronous: the cost suffered by a player when selecting a particular resource is determined only by the number of users who have chosen that resource, and does not take into account the actual *order* in which the assigned tasks are executed. In this extended abstract we present a new class of games – *asynchronous congestion games* (ACGs) – that model noncooperative congestion settings in which resources execute their

* Based on a short paper (4 pages) in Proceedings of AAMAS-08 [1].
** The work was done when the author was a Ph.D. student at the Technion, Haifa 32000, Israel.

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 41–53, 2009.

assigned tasks in a randomly chosen order. The random order of task execution reflects, for instance, a situation where players and resources are the elements of an asynchronous distributed system, in which each process has its own independent clock[1].

In ACGs, we consider a finite set of players, each having a unit length task that can be carried out by any element of a finite set of independent resources (machines). Each resource executes its assigned tasks in a randomly chosen order. As a result, a player may selfishly assign his task to several resources, hoping that his task will be completed in a short time by at least one of the resources. It is assumed that resource usage is costly; that is, every player has to pay for utilizing each of his chosen resources. More specifically, a player's aim is to minimize his expected total cost which is composed of the sum of the fixed costs over the set of his chosen resources and the cost of his task execution which is determined by the minimum completion time of his task by any of his chosen resources.

By considering the order of task execution, the study of ACGs is related to the literature on selfish scheduling. There are two types of selfish scheduling: scheduling involving *selfish machines* [9,10,11] in which resources attempt to optimize their own objectives, and scheduling involving *selfish tasks* [12,13,14] in which each participant's objective is to minimize the completion time of his task. The latter type is closely related to congestion games.

Introducing a new class of games raises the important question of the existence of pure strategy equilibria as well as the computation of such equilibria. There are only few known classes of games which possess pure strategy equilibria, and there seems to be relatively little work providing efficient and exact algorithms for computing such equilibria. In this extended abstract we introduce the class of ACGs and show that these games possess a Nash equilibrium in pure strategies, despite the non-existence of a potential function. In addition, we present a polynomial time algorithm for finding such an equilibrium in a given ACG. Proofs are not presented due to space limitations.

2 The Model

Let $N = \{1, \ldots, n\}$ be a set of n players and let $M = \{e_1, \ldots, e_m\}$ be a set of m resources. Player $i \in N$ chooses a strategy $\sigma_i \in \Sigma_i$ which is a *nonempty* subset of the resources: $\Sigma_i = P(M) \setminus \{\varnothing\}$. Given a subset $S \subseteq N$ of the players, the set of strategy combinations of the members of S is denoted by $\Sigma_S = \times_{i \in S} \Sigma_i$, and the set of strategy combinations of the complement subset of players is denoted by Σ_{-S} ($\Sigma_{-S} = \Sigma_{N \setminus S} = \times_{i \in N \setminus S} \Sigma_i$). The set of pure strategy profiles of all the players is denoted by Σ ($\Sigma = \Sigma_N$). Let $\sigma = (\sigma_1, \ldots, \sigma_n) \in \Sigma$ be a strategy profile. The (m-dimensional) *congestion vector* that corresponds to σ is $h(\sigma) = (h_e(\sigma))_{e \in M}$, where $h_e(\sigma) = \left|\{i \in N : e \in \sigma_i\}\right|$.

[1] The idea of using random ordering in order to reflect the asynchronous nature of processes in distributed systems is discussed, for example, in Monderer and Tennenholtz [8].

The *outcome* for player $i \in N$ from σ is the vector $x^i(\sigma) = \left(x_e^i(\sigma)\right)_{e \in M} \in \{1, \ldots, n, \infty\}^m$ of the ordering numbers of player i's task on all the resources, where $x_e^i(\sigma) \in \{1, \ldots, n\}$ for $e \in \sigma_i$ and $x_e^i(\sigma) = \infty$ for $e \notin \sigma_i$. The player's objective is to minimize his total cost that consists of the sum of the fixed costs over the set of resources he uses and the cost of the player's task execution. The fixed cost for utilizing each of the resources equals t units of money. The cost of task execution is a nonnegative, nondecreasing function of its completion time; thus, the longer it takes to complete the task execution, the greater is the cost incurred by the player. We assume that each player pays a fixed price, say c, for a unit of time his task is in the system before completed by at least one of the resources and, w.l.o.g., that this cost is one unit of money per unit of time. That is, the cost of a player's task execution is determined by the minimum among the completion times of his task by his chosen resources. Hence, the *cost* to player i from a strategy profile σ and his outcome $x^i(\sigma)$, $c_i\left(\sigma, x^i(\sigma)\right)$, is defined as follows:

$$c_i\left(\sigma, x^i(\sigma)\right) = \min_{e \in \sigma_i} x_e^i(\sigma) + |\sigma_i| t.$$

Given a strategy profile σ, for any player $i \in N$ and resource $e \in \sigma_i$, let $X_e^i(\sigma)$ denote a random variable representing the ordering number of player i's task on resource e. Since it is assumed that each task requires a unit of time to be processed and each unit of time costs one unit of money, $X_e^i(\sigma)$ represents the cost to player i for his task execution by resource e. We assume that $X_e^i(\sigma)$ is uniformly distributed over $\{1, \ldots, h_e(\sigma)\}$. The *expected cost* of player i from strategy profile σ, $C_i(\sigma)$, is therefore:

$$\begin{aligned} C_i(\sigma) &= E\left(\min_{e \in \sigma_i} X_e^i(\sigma)\right) + |\sigma_i| t \\ &= \sum_{q=1}^{\min_{e \in \sigma_i} h_e(\sigma)} Pr\left(\min_{e \in \sigma_i} X_e^i(\sigma) \geq q\right) + |\sigma_i| t \\ &= \sum_{q=1}^{\min_{e \in \sigma_i} h_e(\sigma)} \prod_{e \in \sigma_i} \frac{h_e(\sigma) - q + 1}{h_e(\sigma)} + |\sigma_i| t. \end{aligned}$$

The aim of each player is to minimize his own expected cost.

Note that if $t = 0$ then the dominant strategy of each player is to assign his task to all of the resources. As a result, the system is overloaded and less efficient.

3 The (Non)-Existence of a Potential Function

Monderer and Shapley [6] introduced the notion of *potential function* (or, *potential*) as follows. Let G be a game in strategic form with a finite set of players, N. The set of strategies of player $i \in N$ is Σ_i, and the payoff function of player

i is $C_i : \Sigma \to \mathbb{R}$, where $\Sigma = \times_{i \in N} \Sigma_i$ is the set of strategy profiles. A function $P : \Sigma \to \mathbb{R}$ is a *potential* function of G if for every $i \in N$ and for every $\sigma_{-i} \in \Sigma_{-i}$,

$$C_i(\sigma_{-i}, x) - C_i(\sigma_{-i}, y) = P(\sigma_{-i}, x) - P(\sigma_{-i}, y),$$

for any $x, y \in \Sigma_i$. G is called a *potential game* if it admits a potential function. The authors [6] showed that the classes of finite potential games and congestion games coincide.

In this section, we study the existence of a potential function in ACGs. We show that a 2×2 ACG is a potential game but any ACG with $n > 2$ players or $m > 2$ resources does not possess a potential function. Hence, ACGs are not congestion games.

3.1 ACGs with 2 Players and 2 Resources

Here we present a potential function for an ACG with 2 players and 2 resources. Let two players $N = \{1, 2\}$ share a set of two resources $M = \{e_1, e_2\}$. In Figure 1 we present the payoff matrix of the game. A potential function of the game is presented in Figure 2.

	$\{e_1\}$	$\{e_2\}$	$\{e_1, e_2\}$
$\{e_1\}$	$C_1 = \frac{3}{2} + t$ $C_2 = \frac{3}{2} + t$	$C_1 = 1 + t$ $C_2 = 1 + t$	$C_1 = \frac{3}{2} + t$ $C_2 = 1 + 2t$
$\{e_2\}$	$C_1 = 1 + t$ $C_2 = 1 + t$	$C_1 = \frac{3}{2} + t$ $C_2 = \frac{3}{2} + t$	$C_1 = \frac{3}{2} + t$ $C_2 = 1 + 2t$
$\{e_1, e_2\}$	$C_1 = 1 + 2t$ $C_2 = \frac{3}{2} + t$	$C_1 = 1 + 2t$ $C_2 = \frac{3}{2} + t$	$C_1 = \frac{5}{4} + 2t$ $C_2 = \frac{5}{4} + 2t$

Fig. 1. Players' payoffs in the 2×2 ACG

	$\{e_1\}$	$\{e_2\}$	$\{e_1, e_2\}$
$\{e_1\}$	$\frac{3}{4}$	$\frac{1}{4}$	$\frac{1}{4} + t$
$\{e_2\}$	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{4} + t$
$\{e_1, e_2\}$	$\frac{1}{4} + t$	$\frac{1}{4} + t$	$2t$

Fig. 2. A potential function of the 2×2 ACG

By exploring Figures 1 and 2, one can verify that for any two strategy profiles differing by the choice of a single player, the difference in the payoff of that player between the two profiles equals the corresponding increment in the function presented in Figure 2. Therefore, this function is a potential.

3.2 ACGs with $n > 2$ Players or $m > 2$ Resources

Here we show that any ACG with $n > 2$ players or $m > 2$ resources does not admit a potential function. To prove this statement we use the following technical characterization of potential games.

Let G be a game in strategic form with a set $N = \{1, \ldots, n\}$ of players, a set $\Sigma = \times_{i \in N} \Sigma_i$ of strategy profiles, and a vector $C = (C_1, \ldots, C_n)$ of payoff functions. A 4-*cycle*, i.e. a cycle of length 4, in Σ is a sequence $\tau = (\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \delta \rightarrow \alpha)$ of strategy profiles, such that $\alpha = (x_i, x_j, z)$, $\beta = (y_i, x_j, z)$, $\gamma = (y_i, y_j, z)$, $\delta = (x_i, y_j, z)$, where $i, j \in N$, $x_i, y_i \in \Sigma_i$, $x_j, y_j \in \Sigma_j$, and $z \in \Sigma_{-\{i,j\}}$. A 4-cycle τ is *zero-sum* if

$$C(\tau) = C_i(\alpha) - C_i(\beta) + C_j(\beta) - C_j(\gamma) + C_i(\gamma) - C_i(\delta) + C_j(\delta) - C_j(\alpha) = 0,$$

and *non-zero-sum* otherwise. Monderer and Shapley [6] showed that G is a potential game if and only if it does not possess non-zero-sum 4-cycles.

Based on the above characterization, we show that every ACG with $n > 2$ players or $m > 2$ resources does not admit a potential function. Let G_1 be any ACG with $n > 2$ players and $m \geq 2$ resources, and consider the 4-cycle τ_1 which is formed by $\alpha = (\{e_1\}, \{e_2\}, z)$, $\beta = (\{e_2\}, \{e_2\}, z)$, $\gamma = (\{e_2\}, \{e_1, e_2\}, z)$, $\delta = (\{e_1\}, \{e_1, e_2\}, z)$, where $z \in \Sigma_{-\{1,2\}}$ satisfies $h_{e_1}(z) < h_{e_2}(z)$ (see Figure 3). By exploring Figure 3, one can verify that $C(\tau_1)$ is positive, which implies that τ_1 is a non-zero-sum 4-cycle. Hence, G_1 is not a potential game.

	$\{e_2\}$	$\{e_1, e_2\}$
$\{e_1\}$	$C_1 = \frac{h_{e_1}(z)+2}{2} + t$ $C_2 = \frac{h_{e_2}(z)+2}{2} + t$	$C_1 = \frac{h_{e_1}(z)+3}{2} + t$ $C_2 = \frac{h_{e_1}(z)+3}{2}$ $+\frac{1-(h_{e_1}(z)+2)^2}{6(h_{e_2}(z)+1)} + 2t$
$\{e_2\}$	$C_1 = \frac{h_{e_2}(z)+3}{2} + t$ $C_2 = \frac{h_{e_2}(z)+3}{2} + t$	$C_1 = \frac{h_{e_2}(z)+3}{2} + t$ $C_2 = \frac{h_{e_1}(z)+2}{2}$ $+\frac{1-(h_{e_1}(z)+1)^2}{6(h_{e_2}(z)+2)} + 2t$

Fig. 3. Non-existence of potentials in ACGs with $n > 2$ players

Now, let G_2 be any ACG with $n = 2$ players and $m > 2$ resources. Consider the 4-cycle τ_2 which is formed by $\alpha = (\{e_1\}, \{e_3\})$, $\beta = (\{e_1, e_2\}, \{e_3\})$, $\gamma = (\{e_1, e_2\}, \{e_1, e_2, e_3\})$, $\delta = (\{e_1\}, \{e_1, e_2, e_3\})$ (see Figure 4).

Direct calculation shows that $C(\tau_2) = -\frac{1}{4}$, implying that τ_2 is a non-zero-sum 4-cycle and G_2 is not a potential game.

	$\{e_3\}$	$\{e_1, e_2, e_3\}$
$\{e_1\}$	$C_1 = 1 + t$ $C_2 = 1 + t$	$C_1 = \frac{3}{2} + t$ $C_2 = 1 + 3t$
$\{e_1, e_2\}$	$C_1 = 1 + 2t$ $C_2 = 1 + t$	$C_1 = \frac{5}{4} + 2t$ $C_2 = 1 + 3t$

Fig. 4. Non-existence of potentials in ACGs with $m > 2$ resources

4 The Existence of a Pure Strategy Nash Equilibrium

Despite the fact that ACGs, in general, are not potential games, in this section we show that every ACG possesses a Nash equilibrium in pure strategies. If the number of resources is greater than or equal to the number of players ($m \geq n$) then the profile $\sigma = (e_i)_{i \in N}$ is a Nash equilibrium as well as an optimal strategy (one that minimizes the sum of the players' expected costs). If $m < n$ then proving the existence of such an equilibrium is not trivial, as is demonstrated below.

Our proof uses the notion of stability under single moves, previously presented in [15], and proceeds as follows. Below, in 4.1 we define three types of single moves (A-, D- and S-moves) and show that a profile which is stable under all these moves is a Nash equilibrium (see Lemma 1). In 4.2 we observe that the DS-stable[2] profile is easy to find, but the existence of a profile which is stable under all three types of single moves is not obvious (see Lemma 2 and the discussion following it). We look for such a profile using two types of addition operations, which are defined in 4.3. Lemma 3 in this subsection describes how these additions affect DS-stable profiles. Based on this lemma, in 4.4 we show that for some DS-stable profiles the above additions do not ruin the DS-stability (see Lemma 4). We complete our proof by showing that applying a finite series of addition operations to such a profile results in an equilibrium (see Lemma 5 and Corollary 1).

4.1 The Single Profitable Move Property

As pointed out in [15], in a congestion setting, we are mainly interested in three types of *single moves*, where each type is a deviation involving a single resource, as follows.

Definition 1. [15] *For any strategy profile $\sigma \in \Sigma$ and for any player $i \in N$, the operation of adding precisely one resource to his strategy, σ_i, is called an **A-move** of i from σ. Similarly, the operation of dropping a single resource is called a **D-move**, and the operation of switching one resource with another is called an **S-move**.*

The following observation provides technical characterizations of single moves and is heavily utilized in the presentation and the proofs of our results.

Observation 1. *Given a profile σ, let h denote its corresponding congestion vector ($h = h(\sigma)$), and assume there exist $a, b \in M$ and $i \in N$ such that $a \in \sigma_i$ and $b \notin \sigma_i$. Then,*

(1) *If a D-move with a is profitable for i then*

$$t > \sum_{q=1}^{\min_{e \in \sigma_i - a} h_e} \prod_{e \in \sigma_i - a} \frac{h_e - q + 1}{h_e} \cdot \frac{q-1}{h_a}.$$

[2] A strategy profile which is stable under D- and S-moves (see Definition 2).

If $\min_{e\in\sigma_i-a} h_e = \min_{e\in\sigma_i} h_e$ *and the D-move with* a *is non-profitable for* i *then*

$$t \leq \sum_{q=1}^{\min_{e\in\sigma_i-a} h_e} \prod_{e\in\sigma_i-a} \frac{h_e-q+1}{h_e}\cdot\frac{q-1}{h_a}.$$

(2) *If an A-move with* b *is non-profitable for* i *then*

$$t \geq \sum_{q=1}^{\min_{e\in\sigma_i} h_e} \prod_{e\in\sigma_i} \frac{h_e-q+1}{h_e}\cdot\frac{q-1}{h_b+1}.$$

If $\exists e\in\sigma_i$ *such that* $h_e \leq h_b+1$ *and the A-move with* b *is profitable for* i *then*

$$t < \sum_{q=1}^{\min_{e\in\sigma_i} h_e} \prod_{e\in\sigma_i} \frac{h_e-q+1}{h_e}\cdot\frac{q-1}{h_b+1}.$$

(3) *An S-move from* a *to* b *is profitable for* i *if and only if* $h_b+1<h_a$.

Lemma 1 below implies that any strategy profile in which no player wishes unilaterally to apply a single A-, D- or S-move, is a Nash equilibrium. This property is called the *single profitable move property* and it allows us to consider only single moves rather than considering all possible deviations.

Lemma 1. (The single profitable move property) *Given an ACG, let* $\sigma\in\Sigma$ *be a strategy profile which is not in equilibrium, and let* $i\in N$ *be a player for which a profitable deviation from* σ *is available. Then,* i *has a profitable A-, D- or S-move from* σ.

4.2 Stability Under Single Moves

By Lemma 1, in order to prove the existence of a pure strategy Nash equilibrium in games possessing the single profitable move property, it suffices to present a strategy profile for which no player wishes to unilaterally apply an A-, D- or S-move. This observation motivates the following definition.

Definition 2. [15] *A strategy profile* σ *is said to be* ***A-stable*** *(resp.,* ***D-stable****,* ***S-stable****) if there are no players with a profitable A- (resp., D-, S-) move from* σ. *An A- and D-stable profile (resp., A- and S-stable, D- and S-stable) will be termed* ***AD-stable*** *(resp.,* ***AS-stable****,* ***DS-stable****).*

In order to investigate stability under single moves in ACGs we use the notions of light and heavy resources as well as of even and nearly-even strategy profiles.

Definition 3. [15] *Given a strategy profile* σ*, resource* e' *is called* σ-***light*** *if* $e'\in\arg\min_{e\in M} h_e(\sigma)$ *and* σ-***heavy*** *otherwise. A strategy profile* σ *with no heavy resources will be termed* ***even****. An even strategy profile with a common congestion of* k *on the resources will be termed* ***k-even****. A strategy profile* σ *satisfying* $|h_e(\sigma)-h_{e'}(\sigma)|\leq 1$ *for all* $e,e'\in M$ *will be termed* ***nearly-even****.*

Obviously, every even strategy profile is nearly-even. In addition, in a nearly-even strategy profile all heavy resources (if such exist) have the same congestion. Moreover, as is shown in the following lemma, the notions of nearly-eveness and S-stability are strongly connected.

Lemma 2. *In an ACG, a strategy profile is S-stable if and only if it is nearly-even.*

Note that the pairwise intersections of the set of S-stable strategy profiles with the set of A-stable profiles or the set of D-stable profiles are not empty. In particular, the strategy profile $\sigma^M = (M, \ldots, M)$ is AS-stable, while the profile $\sigma^0 = (e_{i \bmod m})_{i \in N}$ is DS-stable. However, at first glance, it is not clear whether there exists a profile which is stable under all three types of single moves, or even if there is an AD-stable profile.

Intuitively, one can try to achieve a Nash equilibrium by selecting a profile which is stable under two types of single moves and applying on it a series of single moves of the third type. For instance, one can pick a DS-stable strategy profile and try to transform it into a Nash equilibrium by applying on it a series of profitable A-moves. However, simple examples show that such moves may destroy the D- or the S-stability of the selected profile; moreover, an A-move from the selected profile may initiate a long chain of D- and S-moves. Therefore, the chosen actions have to be picked out in a careful and subtle way. In this context, we first restrict the set of available A-moves to the subset of one- and two-step addition operations, as defined in the sequel.

4.3 One- and Two-Step Additions

Let $\sigma \in \Sigma$ be a strategy profile and let h denote its corresponding congestion vector ($h = h(\sigma)$). For any $X \subseteq M$ such that $h_e < n$ for all $e \in X$, we denote by h^X the congestion vector with the congestion of each resource in X being increased by 1, while the congestion of all other resources remains unchanged. That is, $h^X_e = h_e + 1$ for all $e \in X$ and $h^X_e = h_e$ for all $e \in M \setminus X$. For each player $i \in N$, let $e^i \in \arg\min_{e \in M \setminus \sigma_i} h_e$. That is, e^i is a lightest resource not previously chosen by i. Then, one can make the following (straightforward) observation.

Observation 2. *If there exists a profitable A-move for player i, then an A-move with e^i, a lightest resource not chosen previously by i, is profitable for i as well.*

If no player wishes to change his strategy in this manner, i.e. $C_i(\sigma) \leq C_i(\sigma_{-i}, \sigma_i + e^i)$ for all $i \in N$, then by Observation 2, $C_i(\sigma) \leq C_i(\sigma_{-i}, \sigma_i + a)$ for all $i \in N$ and $a \in M \setminus \sigma_i$. Hence, σ is A-stable. Otherwise, let $N(\sigma)$ denote the subset of all players for which there exists e^i such that a unilateral addition of e^i is profitable. Let $a \in \arg\min_{e^i : i \in N(\sigma)} h_{e^i}$. Let also $i \in N(\sigma)$ be the player for which $e^i = a$. If a is σ-light, then let $\sigma' = (\sigma_{-i}, \sigma_i + a)$. In this case we say that σ' is obtained from σ by a *one-step addition* of resource a, and a is called an *added* resource. If a is σ-heavy then there exists a σ-light resource b and a player j such that

$a \in \sigma_j$ and $b \notin \sigma_j$. Then let $\sigma' = \big(\sigma_{-\{i,j\}}, \sigma_i + a, \sigma_j - a + b\big)$. In this case we say that σ' is obtained from σ by a *two-step addition* of resource b, and b is called an *added* resource.

We notice that, in both cases, the congestion of each resource in σ' is the same as in σ, except for the added resource, with the congestion in σ' increased by 1. Thus, if σ is nearly-even then σ' is also nearly-even (since the added resource is σ-light). Then, Lemma 2 implies the S-stability of σ'. Lemma 3 below shows that if, in addition, σ is D-stable then the only potential cause for the non-D-stability of σ' is the existence of player $i \in N$ with $\sigma'_i = \sigma_i$ who wishes to drop the added resource a.

Lemma 3. *Let σ be a nearly-even and D-stable strategy profile of a given ACG, and let σ' be obtained from σ by a one- or two-step addition of resource a. Then, there are no profitable D-moves for any player $i \in N$ with $\sigma'_i \neq \sigma_i$. For $i \in N$ with $\sigma'_i = \sigma_i$, the only possible profitable D-move (if such exists) is to drop the added resource a.*

Note that although we did not succeed in keeping the D-stability, we have significantly reduced the set of possible post-addition D-moves. This motivates us to present the term of post-addition D-stability which plays a central role in our method, as follows.

4.4 Post-Addition D-Stability

Let $\sigma \in \Sigma$ be a strategy profile and let σ' be obtained from σ by applying a one- or two-step addition operation. Then, based on Lemma 3, σ is said to be *post-addition D-stable* if σ' does not admit profitable D-moves with the added resource. Formally, the post-addition D-stability is defined as follows.

Definition 4. *A strategy profile σ of a given ACG is called **post-addition D-stable** if*

$$t \leq \sum_{q=1}^{\min_{e \in \sigma_i - a} h_e(\sigma)} \left(\prod_{e \in \sigma_i - a} \frac{h_e(\sigma) - q + 1}{h_e(\sigma)} \right) \frac{q-1}{h_a(\sigma) + 1}, \tag{1}$$

for every $i \in N$ with $|\sigma_i| > 1$ and for every σ-light resource $a \in \sigma_i$.

We note that by Observation 1, inequality (1) implies the non-profitability of a D-move with the added resource.

Let $\Sigma^0 \subseteq \Sigma$ denote the subset of all D-stable strategy profiles, and let $\Sigma^1 \subseteq \Sigma^0$ be the subset of all even and D-stable strategy profiles. By Lemma 2, every profile in Σ^1 (if such exists) is S-stable. For any $\sigma \in \Sigma^1$, let h^σ denote the common congestion on the resources, and let $\Sigma^2 \subseteq \Sigma^1$ be the subset of Σ^1 consisting of all those profiles with maximum congestion on the resources: $\Sigma^2 = \arg\max_{\sigma \in \Sigma^1} h^\sigma$. Then,

Lemma 4. *Given an ACG, there exists a strategy profile $\sigma \in \Sigma^2$ that is either a pure strategy Nash equilibrium or post-addition D-stable.*

It is not clear, by first look, if the existence of a post-addition D-stable strategy profile implies the existence of a pure strategy Nash equilibrium. To show such an implication, post-addition D-stability should be preserved while applying a series of addition operations. In addition, such a series of addition operations should converge to a pure strategy Nash equilibrium in a finite number of steps. In this context, Lemma 5 and Corollary 1 below provide the needed steps for completing the proof of existence of a pure strategy equilibrium.

Lemma 5. *Given an ACG, let σ be a nearly-even and post-addition D-stable strategy profile, and let σ' be obtained from σ by applying on it a one- or two-step addition operation. If $\min_{e\in M} h_e(\sigma') = \min_{e\in M} h_e(\sigma)$ then σ' is also nearly-even and post-addition D-stable.*

Corollary 1. *By Lemma 5, if we can find a post-addition D-stable strategy profile σ' that lies in Σ^2, then a pure strategy Nash equilibrium can be achieved by applying on σ', in a sequential manner, less than m one-/two-step addition operations. This is because if we perform m addition operations then an even D-stable strategy profile σ'' with $h^{\sigma''} > h^{\sigma'}$ is obtained, contradicting $\sigma' \in \Sigma^2$.*

Theorem 1 below follows directly from Lemmas 4 and 5, and Corollary 1.

Theorem 1. *Every ACG possesses a Nash equilibrium in pure strategies.*

5 Computation of a Pure Strategy Nash Equilibrium

We are now ready to present our Asynchronous Nash equilibrium (ANE)-algorithm that constructs a pure strategy Nash equilibrium in any given ACG. Let us start with a brief description of the algorithm:

• Based on Lemma 5, the goal of the algorithm is to find a strategy profile in Σ^2 which is either a pure strategy Nash equilibrium or post-addition D-stable. In the latter case, the strategy profile can be turned into a Nash equilibrium by applying on it at most $m-1$ one-/two-step addition operations. For that, the algorithm has to determine a value $k^* = \max_{\sigma\in\Sigma^1} h^\sigma$ that represents the common congestion on the resources for any strategy profile in Σ^2.

• To find k^* as above, the algorithm uses a variable k initiated with the value $k = n$ and gradually decreases until k^* is found (Steps [0] – [1]).

• For $k = n$, the only even strategy profile with n being its common congestion is $\sigma = (M, \ldots, M)$, which is obviously A- and S-stable. If σ is also D-stable then $k^* = n$, and the algorithm outputs σ and halts (Step [0]). Otherwise, $k^* < n$ and the algorithm proceeds with $k = n-1$ (Step [1]).

• Given $\lfloor \frac{n}{m} \rfloor < k < n$, the algorithm checks whether a k-even D-stable strategy profile exists. If there is no such profile then $k^* < k$ and the algorithm proceeds with the next value of k (repeating Step [1]). Otherwise, $k^* = k$.

• If $k^* = \lfloor \frac{n}{m} \rfloor$ then the algorithm constructs a strategy profile $\sigma = (e_{i \bmod m})_{i\in N}$ (Step [2]). As we show in the proof of Theorem 2, σ is a Nash equilibrium.

• Otherwise, $k^* > \lfloor \frac{n}{m} \rfloor$. In this case, the algorithm constructs a k^*-even strategy profile σ with $n^* = n\left(\lfloor \frac{k^*m}{n} \rfloor + 1\right) - k^*m$ players using $\lfloor \frac{k^*m}{n} \rfloor$ resources and $n - n^* = k^*m - n\lfloor \frac{k^*m}{n} \rfloor$ players using $\lfloor \frac{k^*m}{n} \rfloor + 1$ resources (Step [3]). As we show in the proof of Theorem 2, the obtained σ is D- and S-stable. If σ is also A-stable then the algorithm outputs σ and halts (Step [4]). Otherwise, we show that $\sigma \in \Sigma^2$ and is post-addition D-stable. Then, based on Lemma 5 and Corollary 1, a pure strategy Nash equilibrium is achieved by applying at most $m - 1$ one- or two-step additions on σ (Steps [5] – [9]).

The ANE-algorithm is presented below.

ANE-Algorithm

Step [0] If $t \leq \sum_{q=1}^{n} \left(\frac{n-q+1}{n}\right)^{m-1} \frac{q-1}{n}$ then set
$\sigma := (M, \ldots, M)$ and QUIT;
Otherwise, set $k := n - 1$ and go to Step [1];

Step [1] If $t > \sum_{q=1}^{k} \left(\frac{k-q+1}{k}\right)^{\lceil \frac{km}{n} \rceil - 1} \frac{q-1}{k}$ then set
$k := k - 1$; Otherwise go to Step [3];

Step [2] If $k = \lfloor \frac{n}{m} \rfloor$ then set $\sigma := (e_{i \bmod m})_{i \in N}$
and QUIT; Otherwise go to Step [1];

Step [3] Set $n^* := n\left(\lfloor \frac{km}{n} \rfloor + 1\right) - km$;
For $i = 1$ to n^*:
Set $\sigma_i = \{e_r \in M : 1 \leq r \leq \lfloor \frac{km}{n} \rfloor\}$
and reorder the resources:
for all $e_r \in M$ set $e_r := e_{\left(r + \lfloor \frac{km}{n} \rfloor\right) \bmod m}$;
If $n^* = n$ then go to Step [4];
Otherwise, for $i = n^* + 1$ to n:
Set $\sigma_i = \{e_r \in M : 1 \leq r \leq \lfloor \frac{km}{n} \rfloor + 1\}$
and reorder the resources:
for all $e_r \in M$ set $e_r := e_{\left(r + \lfloor \frac{km}{n} \rfloor + 1\right) \bmod m}$;

Step [4] If $t \geq \sum_{q=1}^{k} \left(\frac{k-q+1}{k}\right)^{\lfloor \frac{km}{n} \rfloor} \frac{q-1}{k+1}$ then QUIT;

Step [5] For all $i \in N$, select $e^i \in \arg\min_{e \in M \setminus \sigma_i} h_e(\sigma)$;

Step [6] Set $N(\sigma) := \{i \in N : C_i(\sigma_{-i}, \sigma_i + e^i) < C_i(\sigma)\}$;
If $N(\sigma) = \varnothing$ then QUIT;

Step [7] Set $M(\sigma) := \{e \in M : \exists i \in N(\sigma), e = e^i\}$;

Step [8] Select $a^* \in \arg\min_{e \in M(\sigma)} h_e(\sigma)$
and $i^* \in \{i \in N(\sigma) : e^i = a^*\}$;

Step [9] If a^* is σ-light set $\sigma_{i^*} := \sigma_{i^*} + a^*$
and go to Step [5];
Otherwise select a σ-light resource b^*
and $j^* \in \{i \in N : a^* \in \sigma_i, b^* \notin \sigma_i\}$,
set $\sigma_{i^*} := \sigma_{i^*} + a^*$, $\sigma_{j^*} := \sigma_{j^*} - a^* + b^*$,
and go to Step [5].

Theorem 2. *The ANE-algorithm finds a pure strategy Nash equilibrium in any given ACG, and its time complexity is* $O(nm^2)$.

6 Summary and Future Work

In this extended abstract, we introduced and investigated the class of asynchronous congestion games – ACGs – which extends the models of congestion games to allow for a random ordering of task execution. In an ACG, each player aims to minimize his own cost which is determined by the sum of two terms: the execution cost of his task which is assumed to be proportional to its completion time, and the sum of the fixed costs over the resources he uses. The completion time of the player's task is determined by the minimum among its completion times by all of his chosen resources.

We studied the existence of a pure strategy Nash equilibrium and a potential function in ACGs. We showed that only ACGs with 2 players and 2 resources are potential games, and any other ACG is not a potential game. Nevertheless, we showed that any ACG possesses a pure strategy Nash equilibrium. We presented a polynomial time algorithm for constructing a pure strategy Nash equilibrium in a given ACG.

The model of ACGs can be extended in various ways. One can consider other probability distributions over the set of permutations (orders) of the tasks assigned to a particular resource. In addition, it will be a challenge to consider different processing times rather than these of single units, different subsets of resources available to each of the players, players with multiple tasks etc. We believe such extensions will be significantly more difficult to analyze. It is also of interest to study the stability under deviations of coalitions and the social (in)efficiency of equilibria in ACGs.

References

1. Penn, M., Polukarov, M., Tennenholtz, M.: Asynchronous congestion games. In: Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008), pp. 1605–1608 (May 2008)
2. Christodoulou, G., Koutsoupias, E.: The price of anarchy of finite congestion games. In: Proceedings of the 37th Annual ACM Symposium on Theory and Computing (STOC 2005), pp. 67–73 (2005)
3. Fabrikant, A., Papadimitriou, C., Talwar, K.: The complexity of pure Nash equilibria. In: STOC 2004, pp. 604–612 (2004)
4. Milchtaich, I.: Congestion games with player-specific payoff functions. Games and Economic Behavior 13, 111–124 (1996)
5. Monderer, D.: Solution-based congestion games. Advances in Mathematical Economics 8, 397–407 (2006)
6. Monderer, D., Shapley, L.: Potential games. Games and Economic Behavior 14, 124–143 (1996)
7. Rosenthal, R.: A class of games possessing pure-strategy Nash equilibria. International Journal of Game Theory 2, 65–67 (1973)

8. Monderer, D., Tennenholtz, M.: Distributed games. Games and Economic Behavior 28, 181–188 (1999)
9. Auletta, V., Prisco, R., Penna, P., Persiano, G.: Determinisitc truthful approximation mechanisms for scheduling related machines. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 608–619. Springer, Heidelberg (2004)
10. Grosu, D., Carroll, T.: A strategyproof mechanism for scheduling divisible loads in distributed systems. In: ISPDC 2005, pp. 83–90. IEEE Computer Society, Los Alamitos (2005)
11. Nisan, N., Ronen, A.: Algorithmic mechanism design. Games and Economic Behavior 35(1/2), 166–196 (2001)
12. Angel, E., Bampis, E., Pascual, F.: Truthful algorithms for scheduling selfish tasks on parallel machines. In: Deng, X., Ye, Y. (eds.) WINE 2005. LNCS, vol. 3828, pp. 698–707. Springer, Heidelberg (2005)
13. Carroll, T., Grosu, D.: Selfish multi-user task scheduling. In: ISPDC 2006, pp. 99–106. IEEE Computer Society, Los Alamitos (2006)
14. Koutsoupias, E.: Selfish task allocation. Bulletin of EATCS 81, 79–88 (2003)
15. Penn, M., Polukarov, M., Tennenholtz, M.: Congestion games with load-dependent failures: identical resources. In: Proceedings of the 8th ACM Conference on Electronic Commerce (EC 2007), pp. 210–217 (2007)

Combinatorial Problems for Horn Clauses*

Marina Langlois[1], Dhruv Mubayi[1], Robert H. Sloan[1], and György Turán[1,2]

[1] University of Illinois at Chicago, Chicago, IL, USA
mirodo1@uic.edu, mubayi@math.uic.edu
sloan@uic.edu
[2] Hungarian Academy of Sciences & University of Szeged, Hungary
gyt@uic.edu

Abstract. Given a family of Horn clauses, what is the minimal number of Horn clauses implying all other clauses in the family? What is the maximal number of Horn clauses from the family without having resolvents of a certain kind? We consider various problems of this type, and give some sharp bounds. We also consider the probability that a random family of a given size implies all other clauses in the family, and we prove the existence of a sharp threshold.

1 Introduction

Horn formulas form a basic framework for knowledge representation, being an expressive and tractable fragment of logic. They have been studied from many different aspects, such as reasoning and learning [1,2].

In our recent work [3,4,5], we studied related problems on Horn formulas in the context of Horn approximation and belief revision. Motivated by applications such as the Open Mind Common Sense [6] project for the acquisition of commonsense knowledge bases, we formulated the KnowBLe (Knowledge Base Learning) problem on synthesizing learning and belief revision. The objective is to learn a Horn formula in the model of learning with entailment, using a learning algorithm which updates its hypotheses in a rational manner in the spirit of the AGM paradigm [7].

In order to analyze various approaches to this problem, it would be useful to have a good understanding of the combinatorial and probabilistic properties of Horn formulas, such as how many additional clauses are implied and how many resolution steps can be formed. In this paper we consider some combinatorial problems of this sort.

Given a family of Horn clauses, what is the minimal number of Horn clauses needed to imply all Horn clauses in the family? What is the maximal number of Horn clauses in the family such that no resolution steps of a certain kind can be performed? These questions also appear to be of independent interest in combinatorics, as related questions about hypergraphs are much studied in extremal hypergraph theory.

As an interesting basic case, we discuss definite Horn clauses of size 3 in most of the paper (we briefly discuss the simple case of definite Horn clauses of size 2 as well, as it provides some interesting analogies). In the intended commonsense knowledge base

* This material is based upon work supported by the National Science Foundation under Grants No. CCF-0431059 and DMS-0653946.

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 54–65, 2009.

application it seems reasonable to assume that the knowledge base contains definite clauses of size at least 2 (in contrast to other applications where the knowledge base is used to derive facts from other facts).

It is noted that if a set of definite, size-3 Horn formulas implies a definite, size-3 Horn clause, then that clause has a resolution derivation using intermediate clauses of size at most 4 (and size-4 intermediate clauses may be necessary). The minimal number of definite, size-3 Horn clauses implying all definite Horn clauses of size 3 over n variables is determined exactly. At the other end, asymptotically sharp bounds are given for the maximal number of definite, size-3 Horn clauses over n variables, such that no resolution (resp., no resolution giving a resolvent of size 3, and no resolution giving a resolvent of size 4) can be performed among those clauses.

We also consider the probability that a given number of random definite, size-3 Horn clauses imply all other definite, size-3 Horn clauses. It is shown that this probability has a sharp threshold.

The paper is organized as follows. Section 2 gives some preliminaries, Section 3 discusses the case of definite Horn clauses of size 2 and Section 4 gives the bound on the size of intermediate clauses. Section 5 is on the minimal number of definite, size-3 Horn clauses implying all definite, size-3 Horn clauses. The bounds for the maximal number of definite, size-3 Horn clauses without different kinds of resolvents are contained in Section 6. Random formulas are considered in Section 7. We make a few final remarks in Section 8.

2 Preliminaries

We use standard terms from propositional logic such as literal and clause. Formulas are over n variables, and the variables are $X_n = \{x_1, \ldots, x_n\}$. A clause is *Horn* (resp., *definite Horn*) if it contains at most one (resp. exactly one) unnegated literal. We will generally write the Horn clause $(\bar{x} \vee \bar{y} \vee z)$ in the form $x, y \to z$. For a definite Horn clause C, let Body(C) be the set of variables corresponding to the negated literals in C and let Head(C) be the unnegated variable of C.

The *size* of a clause is the number of literals it contains. We use $\mathcal{D}_k^n$ to denote the collection of all size-k definite Horn clauses on n variables. Its size is

$$|\mathcal{D}_k^n| = k \cdot \binom{n}{k} \ , \tag{1}$$

which is $\Theta(n^k)$ for constant k.

A *(definite) Horn formula* is a conjunction—or a set, whichever view is more convenient—of (definite) Horn clauses.

A clause C is an *implicate* of a Boolean formula φ if every assignment satisfying φ also satisfies C; clause C is a *prime implicate* if it is an implicate but none of C's sub-clauses is an implicate.

We say that two clauses have an *opposing literal* when there is a variable that appears negated in one clause and unnegated in the other. A pair of Horn clauses can have either zero, one, or two opposing literals. We define the familiar operation of *resolution* for the case of Horn clauses to apply to a pair of Horn clauses that have exactly one opposing

literal. Let C_1 and C_2 be such Horn clauses, and assume w.l.o.g. that C_1 is definite with its head being the opposing literal: $\text{Head}(C_1) \in \text{Body}(C_2)$. Resolution returns the clause $(\text{Body}(C_1) \cup \text{Body}(C_2) \setminus \{\text{Head}(C_1)\}) \rightarrow \text{Head}(C_2)$, which is called the *resolvent of* C_1 *and* C_2. Thus the resolvent of the two clauses $(a, b \rightarrow c)$ and $(c, d \rightarrow e)$ is $(a, b, d \rightarrow e)$. The resolvent of two definite, size-3 Horn clauses has size 3 or 4, referred to as a *3-resolvent*, resp., a *4-resolvent*. The resolvent of two definite size-2 Horn clauses always has size 2. A set of clauses F is called *resolvent-free* if no two clauses in F can be resolved.

We will use standard facts about Horn resolution, such as that every prime implicate of a Horn formula is a Horn clause, and we will also refer to the standard procedure of forward chaining, which can also be viewed as a unit resolution proof procedure (e.g., [2,8]).

3 Definite Horn Formulas with Size-2 Clauses

In this section we consider some extremal problems for definite Horn formulas with size-2 clauses. A definite, size-2 Horn clause $a \rightarrow b$ can be thought of as a directed edge (a, b), so definite Horn formulas with size-2 clauses can be viewed as directed graphs.

Proposition 1. *There is a subset of $\mathcal{D}_2^n$ of size n that has every clause in $\mathcal{D}_2^n$ as an implicate, and no smaller size subset has this property.*

Proof. The formula

$$(x_1 \rightarrow x_2) \wedge (x_2 \rightarrow x_3) \wedge \cdots \wedge (x_n \rightarrow x_1)$$

of clauses forming a cycle implies every size-2 definite Horn clause: For $i < j$, the clause $x_i \rightarrow x_j$ can be obtained by resolving $(x_i \rightarrow x_{i+1}), (x_{i+1} \rightarrow x_{i+2}), \ldots, (x_{j-1} \rightarrow x_j)$, two at a time in order. If instead $i > j$, then resolve $(x_i \rightarrow x_{i+1}), (x_{i+1} \rightarrow x_{i+2}), \ldots, (x_{n-1} \rightarrow x_n), (x_n \rightarrow x_1), (x_1 \rightarrow x_2), \ldots, (x_{j-1} \rightarrow x_j)$. At least n clauses are needed, as otherwise there is a variable that never appears as a head, and there is no way to obtain implicates having that head.

Proposition 2. *If $F \subseteq \mathcal{D}_2^n$ is resolvent-free, then $|F| \leq \left\lfloor \frac{n^2}{4} \right\rfloor$ for $n \geq 3$, and the bound is sharp.*

Proof. Partition the set X_n of variables into sets A and B, and consider all clauses of the form $a \rightarrow b$ with $a \in A$ and $b \in B$. Clearly, this is a resolvent-free family. The number of clauses is maximized if $|A| = \lfloor \frac{n}{2} \rfloor$ and $|B| = \lceil \frac{n}{2} \rceil$, giving a family of the claimed size.

Now we show that the bound is the largest possible for $n \geq 3$. The cases $n = 3, 4$ are trivial. In digraph terms, we want the directed graph on n vertices with a maximum number of edges, having no simple path of length 2. If there is cycle of length two then its vertices cannot be incident to any other edge and the statement follows by induction. Otherwise, every vertex has either in-degree 0 or out-degree 0, and so the graph is a subgraph of a complete directed bipartite graph described above.

4 Size of Intermediate Clauses in Resolution

A resolution derivation of a short clause from a formula consisting of short clauses may contain large intermediate clauses. It is a basic observation with far-reaching implications that in some cases large intermediate clauses are unavoidable [9,10]. A trivial example for a class of clauses where such a phenomenon cannot occur is size-2 clauses, as every resolvent of such clauses has size at most 2. We note that there is also no blow-up of intermediate clauses for Horn formulas. (See also [11,12].)

Theorem 1. *Let φ be a definite Horn formula with clauses of size at most 3. Then any size-3 prime implicate of φ has a resolution derivation where all clauses occurring in the proof have size at most 4.*

Proof. (Sketch) Since φ is definite, all its resolvents and hence all its prime implicates must be definite Horn clauses. Assume that $C = a, b \to c$ is the implied clause. Then there is a forward chaining derivation of c from $\varphi \wedge a \wedge b$. In this derivation, each resolvent is shorter than its non-unit parent by one. Thus intermediate clauses all have size at most 2. Now omit any resolutions that used a or b. This new resolution derivation contains the same clauses as the original one, except that some clauses have a and/or b added to their body. Intermediate clause sizes could therefore be as large as 4. The final clause of this derivation, which is an implicate of φ, could in general have any subset of $\{a, b\}$ as its body and c as its head. However, since C is a prime implicate, the final clause must be C.

The bound of 4 cannot be improved, as there may be size-3 prime implicates where we must use some intermediate clause of size 4. For example, we must use a size-4 intermediate resolvent to derive the prime implicate $a, b \to e$ of the Horn formula

$$(a, b \to c) \wedge (a, b \to d) \wedge (c, d \to e) \ .$$

Theorem 1 can be generalized, for example, to the following statement, using the same argument.

Corollary 1. *Let φ be a definite Horn formula with clauses of size at most s. Then any prime implicate of φ with t variables in its body has a resolution derivation where all intermediate clauses occurring in the proof have size at most $s - 1 + t$.*

5 Small Formulas with All Implicates

In this section we consider the problem of finding the smallest family of definite size-3 Horn clauses implying every clause in $\mathcal{D}_3^n$, and as in Proposition 1, we find the exact minimum.

Theorem 2. *There is a subset of $\mathcal{D}_3^n$ of size $\binom{n}{2}$ that has every clause in $\mathcal{D}_3^n$ as an implicate, and no smaller size subset has this property.*

Proof. To show that $\binom{n}{2}$ clauses are sufficient, we exhibit a set $S_n \subseteq \mathcal{D}_3^n$ of this size and demonstrate that S_n implies all definite size-3 clauses. Each clause of S_n is in one of three categories:

I. $x_i, x_j \rightarrow x_{i+1}$, for $i \leq n-2$ and $i+1 < j$,
II. $x_i, x_{i+1} \rightarrow x_{i+2}$, for $i \leq n-2$,
III. $x_{n-1}, x_n \rightarrow x_1$.

Note that S_n can be viewed as the size-3 analog of the directed cycle considered in Proposition 1.

All definite Horn clauses of size 3 are satisfied by the all 1's vector, the all 0's vector and all unit vectors. Call these vectors *standard*.

Assume for contradiction that $C \in \mathcal{D}_3^n$ is not implied by S_n. Then there is a truth assignment that satisfies S_n and falsifies C. This truth assignment has at least two 1's (corresponding to variables in Body(C)) and at least one 0 (namely, Head(C)), thus it is non-standard. Therefore, it is sufficient to show that every non-standard vector falsifies at least one clause of S_n. A non-standard vector can have the following forms (using regular expressions):

1. (0+1)*10(0+1)*1(0+1)*,
2. (0+1)*110(0+1)*,
3. 0(0+1)*11.

Vectors of form 1 falsify clauses of class I, vectors of form 2 falsify clauses of class II, and vectors of form 3 falsify clauses of class III.

For the lower bound, note that resolution of two definite clauses of size at least 3 does not produce clauses with any new bodies of size 2. Therefore a set of clauses implying all other clauses must contain all possible bodies.

Incidentally, an examination of the resolutions needed shows that in order to derive all the clauses of $\mathcal{D}_3^n$ from S_n one does not need any size-4 intermediate clauses, unlike the general case given by Theorem 1.

Theorem 3. *Every clause $C \in \mathcal{D}_3^n$ can be derived from S_n with every intermediate clause being in $\mathcal{D}_3^n$.*

Table 1. All definite size-3 Horn clauses with head x_h

n	n-1	...	h+1	h-1	...	2
$x_1, x_n \rightarrow x_h$ 1	$x_1, x_{n-1} \rightarrow x_h$	...	$x_1, x_{h+1} \rightarrow x_h$	$x_1, x_{h-1} \rightarrow x_h$	...	$x_1, x_2 \rightarrow x_h$
$x_2, x_n \rightarrow x_h$	$x_2, x_{n-1} \rightarrow x_h$	...	$x_2, x_{h+1} \rightarrow x_h$	$x_2, x_{h-1} \rightarrow x_h$	...	
$x_3, x_n \rightarrow x_h$	$x_3, x_{n-1} \rightarrow x_h$	...	$x_3, x_{h+1} \rightarrow x_h$	$x_3, x_{h-1} \rightarrow x_h$	...	
...	...	...	...	...	...	
$x_{h-1}, x_n \rightarrow x_h$	$x_{h-1}, x_{n-1} \rightarrow x_h$	...	$x_{h-2}, x_{h+1} \rightarrow x_h$	$x_{h-2}, x_{h-1} \rightarrow x_h$	...	
$x_{h+1}, x_n \rightarrow x_h$	$x_{h+1}, x_{n-1} \rightarrow x_h$	...	$x_{h-1}, x_{h+1} \rightarrow x_h$		...	
...	...					
$x_{n-2}, x_n \rightarrow x_h$	$x_{n-2}, x_{n-1} \rightarrow x_h$	...			...	
$x_{n-1}, x_n \rightarrow x_h$					...	

Proof. (Sketch) A careful series of inductions shows that eventually all clauses can be derived. Table 1 gives a hint of the idea for each of the $\binom{n-1}{2}$ clauses with head x_h, for some arbitrary h. Starting at the leftmost column, with the boxed clause, we can derive all the clauses above the boxes, up through column $h+1$. Next these can be used to obtain the rest of the clauses in the leftmost columns (going "down" these same columns.) Two similar steps then handle the right columns.

6 Large Formulas without Resolvents

A set of clauses $F \subseteq \mathcal{D}_3^n$ is *3-resolvent-free* (resp. *4-resolvent-free*) if no two of its clauses can be resolved to produce a 3-resolvent (resp., 4-resolvent). In this section we prove upper bounds on the size of resolvent-free, 3-resolvent-free, and 4-resolvent-free clause sets. First, we formulate a few technical lemmas.

6.1 Duplicates and Escher Configurations

Definite, size-3 clauses C_1, C_2 form a *duplicate* if they contain the same set of variables. Thus duplicate clauses are of the form $a, b \to c$ and $a, c \to b$.

Lemma 1. *Let $F \subseteq \mathcal{D}_3^n$ be 3-resolvent-free. Then there is an $S \subseteq F$ of size at most $\binom{n}{2}$ such that $F \setminus S$ contains no duplicates.*

Proof. It is sufficient to show that if clause $a, b \to c$ occurs in a duplicate in F, then F cannot contain another clause with the same body. Indeed, such a clause $a, b \to d$ gives a size-3 resolvent with both $a, c \to b$ and $b, c \to a$, and one of these clauses occurs in F.

Lemma 2. *Let $F \subseteq \mathcal{D}_3^n$ be 4-resolvent-free. Then there is an $S \subseteq F$ of size at most $(3/2)n^2$ such that $F \setminus S$ contains no duplicates.*

Proof. Consider the weighted undirected graph G on the vertex set X_n with an edge (b, c) for every pair of duplicates $a, b \to c$ and $a, c \to b$. The weight of such an edge (b, c) is the number of such vertices a. If we delete a clause from each such pair then no duplicates remain. Therefore, it is sufficient to prove the claimed upper bound for the sum of the edge weights in G.

We claim that the edges of G having weight at least 3 are independent. Assume that (b, c) and (c, d) both have weight at least 3. Then there is a vertex e such that $b, e \to c$ is in F, and there is a vertex f such that $c, f \to d$ is in F. For the last assertion we use the fact that (c, d) has weight at least 3, as f then can be chosen to be different from b and e. But the two clauses can be resolved to produce a 4-resolvent.

Hence the number of edges in G with weight at least 3 is at most $n/2$. Every weight is at most n, so the total weight of edges in G is at most $(n^2/2) + 2\binom{n}{2} \leq (3/2)n^2$.

Definite, size-3 clauses C_1, C_2 form an *Escher configuration* if $\text{Head}(C_1) \in \text{Body}(C_2)$ and $\text{Head}(C_2) \in \text{Body}(C_1)$. (The name is inspired by Escher's *Drawing Hands* , though here it is heads rather than hands that are on one another.) Note that such a pair of clauses cannot be resolved.

Lemma 3. *Let $F \subseteq \mathcal{D}_3^n$ be resolvent-free. Then there is an $S \subseteq F$ of size at most $2n^2$ such that $F \setminus S$ contains no Escher configurations.*

Proof. Consider the undirected graph G on the vertex set X_n with an edge $(\text{Head}(C_1), \text{Head}(C_2))$ for every pair of clauses $C_1, C_2 \in F$ forming an Escher configuration. We claim that every vertex of this graph has degree at most 2. Assume that d has neighbors a, b, c in G. Then F contains clauses

$$(a. \to d), (d. \to a), (b. \to d), (d. \to b), (c. \to d), (d. \to c),$$

where the dots correspond to one additional literal in each body. One can use a "sudoku" argument to derive a contradiction. If the second and third clauses cannot be resolved then the third clause must be $b, a \rightarrow d$. Similarly, if the fourth and fifth (resp., sixth and first) clauses cannot be resolved then the fifth (resp., first) clause must be $c, b \rightarrow d$ (resp., $a, c \rightarrow d$). But then the first and fourth clauses can be resolved.

Thus G has at most n edges. Every edge corresponds to at most $2(n-2)$ clauses (those obtained by adding a second literal to the bodies), and so the bound of the lemma follows.

6.2 No Resolvents

Let us partition the set of variables X_n into set A and B, and consider the set of clauses of the form $a, b \rightarrow c$ with $a, b \in A$, $c \in B$. Clearly, this is a resolvent-free family of definite, size-3 Horn clauses, which can be viewed as the size-3 analog of the complete directed bipartite graph of Section 3. The number of clauses in the family is $\binom{m}{2}(n-m)$, where $|A| = m$. This quantity is maximized for m with $|m - 2n/3| \leq 1$, and the maximum is

$$p(n) = \frac{4}{9}\binom{n}{3} + O(n^2).$$

The family constructed for the optimal value of m thus has size $p(n)$. We now show that this size is asymptotically optimal.

Theorem 4. *There is a positive c such that if $F \subseteq \mathcal{D}_3^n$ is resolvent-free, then*

$$|F| \leq p(n) + cn^2 \ .$$

Proof. Let $F \subseteq \mathcal{D}_3^n$ be resolvent-free. Applying Lemma 3, we can delete $O(n^2)$ clauses such that no Escher configuration remains. In the remaining set F' of clauses, no variable can occur in a body of a clause and in the head of another clause, as those two clauses would either be resolvable or form an Escher configuration. Thus every variable is either head only, or body only, or neither. Thus F' is a subfamily of some family obtained by the above construction, and so its size is at most $p(n)$.

6.3 No Resolvents of Size 3

Let us again partition the set of variables X_n into sets A and B, and this time consider the set of clauses of the form $a, b \rightarrow c$ with $a, b \in A$, $c \in B$, or $a, b \in B$, $c \in A$. This is a 3-resolvent-free family of definite, size-3 Horn clauses. (On the other hand, there are many resolvents of size 4.) The number of clauses in the family is $\binom{m}{2}(n-m) + \binom{n-m}{2}m$, where $|A| = m$. This quantity is maximized for m with $m = \lfloor n/2 \rfloor$, and the maximum is

$$q(n) = \frac{3}{4}\binom{n}{3} + O(n^2).$$

The family constructed for the optimal value of m thus has size $q(n)$. We now show that this size is asymptotically optimal.

Theorem 5. *There is a positive c such that if $F \subseteq \mathcal{D}_3^n$ is 3-resolvent-free, then*

$$|F| \leq q(n) + cn^2 \ .$$

Proof. Let $F \subseteq \mathcal{D}_3^n$ be 3-resolvent-free. Applying Lemma 1, we can delete $O(n^2)$ clauses such that no duplicates remain. Let $a, b \rightarrow c$ be a clause in the remaining family F'. Then no clause in F' can have body $a, c \rightarrow$ or $b, c \rightarrow$, as any such clause would either produce a 3-resolvent with $a, b \rightarrow c$, or form a duplicate with it.

Consider the undirected graph G with vertices X_n and edges corresponding to the bodies of clauses in F', and let t be the number of vertex triples containing precisely one edge of G. The remark above implies that $|F'| \leq t$.

Therefore we get the required upper bound on $|F|$ by noting that the number of triples containing precisely one edge of G is at most

$$\frac{1}{2} \sum d(v)(n-1-d(v)) \leq \frac{n}{2} \left(\frac{n-1}{2} \right)^2 .$$

6.4 No Resolvents of Size 4

The families constructed in Section 6.2 have no resolvents, thus the same construction trivially shows that there are families of size $p(n)$ without 4-resolvents. Our final result shows that this is asymptotically optimal even for 4-resolvent-free families. This theorem thus supersedes Theorem 4 (ignoring lower order terms). On the other hand, Theorem 4 has a very simple proof, while the proof of Theorem 6 uses difficult recent results from extremal hypergraph theory. A *3-uniform hypergraph* is specified by a set of vertices and a set of 3-element subsets of the set of vertices.

Theorem 6. *For every $\epsilon > 0$ and sufficiently large n, if $F \subseteq \mathcal{D}_3^n$ is 4-resolvent-free then $|F| \leq p(n) + \epsilon n^3$.*

Proof. Let $F \subseteq \mathcal{D}_3^n$ be a 4-resolvent-free set of clauses. Applying Lemma 2, we can delete $O(n^2)$ clauses from F such that no duplicates remain. Let the remaining set of clauses be F', and consider the 3-uniform hypergraph H obtained from F' by replacing every clause $a, b \rightarrow c$ with the triple $\{a, b, c\}$. From now on we omit curly braces for triples and write abc for simplicity.

Let T be the 3-uniform hypergraph $\{abc, abd, abe, cde\}$ and T' be the 3-uniform hypergraph obtained from T by duplicating vertices a and b. Thus T' consists of the 13 triples $abc, ab'c, a'bc, a'b'c, abd, ab'd, a'bd, a'b'd, abe,\ ab'e, a'be, a'b'e, cde$. By an *orientation* of this family we mean a family of 13 definite, size-3 Horn clauses, each containing the 3 variables of a different triple from T'.

Lemma 4. *Any orientation of T' contains two clauses with a resolvent of size 4.*

Proof. Consider an orientation of T'. We may assume by symmetry that cde is oriented as $c, d \rightarrow e$. Then the clauses $a, b \rightarrow e$ and $a', b' \rightarrow e$ must be present, otherwise we get a 4-resolvent with $c, d \rightarrow e$. Now considering the triple $ab'c$, we find that every orientation leads to a 4-resolvent.

It follows from Lemma 4 that H contains no copy of T'. From this, in the next lemma, we conclude that H contains only "few" copies of T.

Lemma 5. *For sufficiently large n, every 3-uniform, n-vertex, T'-free hypergraph has at most $n^{4.5}$ copies of T.*

Proof. Assume that G has more than $n^{4.5}$ copies of T. For every triple cde, let us consider the set of pairs ab which form a copy of T in G. Triples that have fewer than $n^{3/2}$ such pairs contribute at most $\binom{n}{3}n^{3/2} < n^{4.5}$ copies of T. Thus there is a triple cde with at least $n^{3/2}$ such pairs. The pairs corresponding to such a triple form a cycle $aba'b'$ of length 4 [13]. But then $\{a, b, a', b', c, d, e\}$ forms a T' in G.

Now we can apply a special case of a deep result, the hypergraph removal lemma [14,15] to show that one can delete a "few" edges from H such that *no* copies of T remain.

Theorem 7 ([14,15]). *For every $\epsilon > 0$ and sufficiently large n, if H is a 3-uniform, n-vertex hypergraph containing at most $n^{4.5}$ copies of T, then one can delete ϵn^3 edges of H such that no copies of T remain.*

The maximal number of edges in a 3-uniform hypergraph without a copy of T has been determined exactly by [16].

Theorem 8 ([16]). *For sufficiently large n, every 3-uniform, n-vertex, T-free hypergraph has at most $p(n)$ edges.*

Now, putting things together, we get that the original set of clauses F contains at most $p(n) + O(n^2) + \epsilon n^3$ clauses, proving the theorem.

7 Random Formulas

In this section we consider a probabilistic version of the problem studied in Section 5. Let $p(n, s)$ be the probability that the conjunction of s random clauses from $\mathcal{D}_3^n$ implies every clause from $\mathcal{D}_3^n$. (Each clause is drawn from the uniform distribution over $\mathcal{D}_3^n$.) Informally, the property of implying every clause has a *sharp threshold* if around a certain number of clauses its probability jumps from low to high over a short interval (see, e.g., [17]). The following result shows that $n^2 \ln n$ is a sharp threshold for this property. See [18] for a similar result in a related probability model. Note that for definite, size-2 Horn clauses the analogous property is the strong connectivity of random digraphs, which has been studied for a long time (e.g., [19,20,21]). Phase transitions for Horn formulas in other probability models have been considered in [22].

Theorem 9. *For every $\epsilon > 0$ there exists some $c > 0$ such that if n is sufficiently large then*

a) $p(n, n^2 \ln n - cn^2) < \epsilon$,
b) $p(n, n^2 \ln n + cn^2) > 1 - \epsilon$.

Proof. (Sketch) We use the following facts about the coupon collector problem: the expected number of trials needed to collect all of m coupons is $m \ln m + \Theta(m)$, its variance is $\Theta(m^2)$, and hence its standard deviation is $\Theta(m)$. (See, e.g., [23].) Part *a)* follows directly from these facts, the Chebyshev inequality and the observation used in Theorem 2 that having all $\binom{n}{2}$ possible bodies in the formula is a necessary condition for generating every clause in $\mathcal{D}_3^n$.

In order to prove part *b)* we use another observation of Theorem 2: in order to show that a random formula of a given size implies every clause with high probability, it is sufficient to show that with high probability it is falsified by every non-standard vector.

Let $\mathbf{F}_s$ be the conjunction of s random clauses from $\mathcal{D}_3^n$. For $2 \leq k \leq n-1$ let

$$q(n,k,s) = \Pr(\text{some weight } k \text{ vector satisfies } \mathbf{F}_s),$$

where the weight of a vector is the number of its 1's. We would like to prove upper bounds for $q(n,k,s)$.

A vector of weight k falsifies $\binom{k}{2}(n-k)$ clauses in $\mathcal{D}_3^n$, as the body of such a clause must contain variables set to 1, and the head of such a clause must be a variable set to 0. So

$$q(n,k,s) \leq \left(1 - \frac{\binom{k}{2} \cdot (n-k)}{3 \cdot \binom{n}{3}}\right)^s \cdot \binom{n}{k}.$$

For $k = 2$, a direct computation shows that for $s = n^2 \ln n + (\ln(1/\epsilon))\, n^2$ it holds that

$$q(n,2,s) < e^{-\frac{2s}{n(n-1)}} \binom{n}{2} < \frac{\epsilon}{2}.$$

If $3 \leq k \leq n-2$ then for $s = n^2 \ln n$ it holds that

$$\begin{aligned} q(n,k,s) &< e^{-\frac{\binom{k}{2}\cdot(n-k)}{3\cdot\binom{n}{3}} \cdot s + k \ln\left(\frac{e\cdot n}{k}\right)} \\ &< e^{k\left(-\frac{(k-1)(n-k)}{n} \cdot \ln n + 1 + \ln n\right)} < n^{-2}. \end{aligned}$$

If $k = n-1$ then

$$q(n,n-1,s) \leq \left(1 - \frac{1}{n}\right)^s \cdot n$$

and so for $s = n^2 \ln n$ it holds that $q(n,n-1,s) = o(1)$. Thus

$$\sum_{k=3}^{n-1} q(n,k,n^2 \ln n) = o(1),$$

hence part *b)* of the theorem follows.

8 Further Comments

The proof of Theorem 6 can be strengthened to show that $p(n)$ is actually the exact maximum (and thus Theorem 4 is also sharp). This result will be contained

in a future paper. Along the same lines, it would be interesting to show that $q(n)$ is the exact maximum in Theorem 5.

There are many other open problems related to the ones discussed here. Extending the results to definite Horn clauses of size greater than 3 seems to be interesting. For size-3 clauses the problems could be reduced to questions about graphs in several cases. For larger sizes this may not be the case anymore. Instead, one may get questions about hypergraphs, which tend to be more difficult.

From the point of view of the intended knowledge base learning application it would be interesting to extend Theorem 9 in several different ways. What is the expected number of clauses implied by a random family of s clauses for s below $n^2 \ln n$? Other questions involve the length of resolution proofs of implied clauses. For s in the range $n^2 \ln n$ or higher, a random family of s clauses implies every other clause with high probability. What is the expected length of the shortest resolution derivation of clauses?

References

1. Angluin, D., Frazier, M., Pitt, L.: Learning conjunctions of Horn clauses. Machine Learning 9, 147–164 (1992)
2. Kleine Büning, H., Lettmann, T.: Propositional Logic: Deduction and Algorithms. Cambridge University Press, Cambridge (1999)
3. Langlois, M., Sloan, R.H., Turán, G.: Horn upper bounds of random 3-CNF: A computational study. In: Ninth Int. Symp. Artificial Intelligence and Mathematics (2006), http://anytime.cs.umass.edu/aimath06/
4. Langlois, M., Sloan, R.H., Turán, G.: Horn upper bounds and renaming. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 80–93. Springer, Heidelberg (2007)
5. Langlois, M., Sloan, R.H., Szörényi, B., Turán, G.: Horn complements: Towards Horn-to-Horn belief revision. In: AAAI 2008, pp. 466–471 (2008)
6. Singh, P.: The public acquisition of commonsense knowledge. In: Proc. AAAI Spring Symposium on Acquiring (and Using) Linguistic (and World) Knowledge for Information Access (2002)
7. Alchourrón, C.E., Gärdenfors, P., Makinson, D.: On the logic of theory change: partial meet functions for contraction and revision. J. Symb. Logic 50, 510–530 (1985)
8. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice Hall, Englewood Cliffs (2003)
9. Ben-Sasson, E., Wigderson, A.: Short proofs are narrow–resolution made simple. J. ACM 48(2), 149–169 (2001)
10. Haken, A.: The intractability of resolution. Theoret. Comput. Sci. 39, 297–308 (1985)
11. Čepek, O.: Restricted consensus method and quadratic implicates of pure Horn functions. RUTCOR Research Report RRR 31-94, Rutgers University, New Brunswick, NJ (1994)
12. Čepek, O.: Structural Properties and Minimization of Horn Boolean functions. Ph.D thesis, Rutgers University (1995)
13. Kővári, T., Sós, V.T., Turán, P.: On a problem of K. Zarankiewicz. Colloquium Math. 3, 50–57 (1954)
14. Gowers, T.: Quasirandomness, counting and regularity for 3-uniform hypergraphs. Combin. Probab. Comput. 15, 143–184 (2006)
15. Nagle, B., Rödl, V.: Regularity properties for triple systems. Random Struct. Algorithms 23(3), 264–332 (2003)

16. Füredi, Z., Pikhurko, O., Simonovits, M.: On triple systems with independent neighbourhoods. Combinatorics, Probability and Computing 14, 795–813 (2005)
17. Friedgut, E.: Necessary and sufficient conditions for sharp threshold of graph properties and the k-SAT problem. J. Amer. Math. Soc. 12, 1017–1054 (1999)
18. Dunne, P.E., Bench-Capon, T.J.M.: A sharp threshold for the phase transition of a restricted satisfiability problem for Horn clauses. J. Log. Algebr. Program. 47(1), 1–14 (2001)
19. Karp, R.M.: The transitive closure of a random digraph. Random Struct. Algorithms 1, 73–94 (1990)
20. Palásti, I.: On the strong connectedness of directed random graphs. Studia Sci. Math. Hungar. 1, 205–214 (1966)
21. Uno, Y., Ibaraki, T.: Reachability problems of random digraphs. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. E81-A, 2694–2702 (1998)
22. Moore, C., Istrate, G., Demopoulos, D., Vardi, M.Y.: A continuous–discontinuous second-order transition in the satisfiability of random Horn-SAT formulas. Random Struct. Algorithms 31(2), 173–185 (2007)
23. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge Univ. Press, Cambridge (1995)

Covering a Tree by a Forest

Fanica Gavril and Alon Itai

Department of Computer Science, Technion - IIT, Haifa, Israel
gavril@cs.technion.ac.il, itai@cs.technion.ac.il

Abstract. Consider a tree T and a forest F. The paper discusses the following new problems: The *Forest vertex-cover problem (FVC)*: cover the vertices of T by a minimum number of copies of trees of F, such that every vertex of T is covered exactly once. The *Forest edge-cover problem (FEC)*: cover the edges of T by a minimum number of copies of trees of F, such that every edge of T is covered exactly once. For a solution to always exist, we assume that F contains a one vertex (one edge) tree.

Two versions of Problem FVC are considered: ordered covers (OFVC), and unordered covers (UFVC). Three versions of Problem FEC are considered: ordered covers (OFEC), unordered covers (UFEC) and consecutive covers (CFEC). We describe polynomial time algorithms for Problems OFVC, UFVC and CFEC, and prove that Problems OFEC and UFEC are NP-complete.

Keywords: vertex-cover of a tree by a forest, edge-cover of a tree by a forest, graph algorithms.

1 Introduction

In the present paper we consider only rooted trees. The root of a tree t is denoted by *root(t)*. For a vertex v of t, we denote by $p_t(v)$ the father of v, by $Ch(v)$ its set of children and by $deg(v)=|Ch(v)|$ their number. For a subset $X \subseteq V(t)$, we denote $Ch[X]=\cup_{v \in X} Ch(v)$. We denote by t_v the subtree of t rooted at v and containing v and all its descendants. The number of edges in the unique path between two vertices x,y of t is called *distance between x and y* and is denoted by $dist(x,y)$; *height(t)* is the distance from *root(t)* to the farthest leaf. *Isomorphism* between two rooted trees t, f is denoted by $t \approx f$. The connected components $f_1,...,f_q$ of a forest F are rooted trees; for simplicity, we denote by F also the family $\{f_1,...,f_q\}$. For a forest F, we denote by $V(F)$, $E(F)$, $L(F)$ its set of vertices, edges and leaves, respectively.

A *forest vertex-cover* (*forest edge-cover*) of a tree T by a forest F, is a partition of T into vertex (edge) disjoint subtrees $t_1,...,t_k$ such that each t_i is isomorphic to some $f_j \in F$. A *minimum forest vertex-cover* is one which uses a minimum number of copies of trees of F.

We define the following two new problems:

FVC: Find a minimum *forest vertex-cover* of a tree T by a forest F.
FEC: Find a minimum *forest edge-cover* of a tree T by a forest F.

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 66–76, 2009.

To ensure that a cover exists, we assume throughout the paper that F contains a unique tree f_1 consisting of a single vertex, for vertex covers, and of a single edge, for edge covers.

A rooted tree is *ordered* if there exists an order between the children of every vertex. A cover is *ordered* if the trees t_i and f_j are isomorphic as ordered trees. A cover without this restriction is *unordered*. We discuss two versions of Problem FVC: *Problem OFVC* – ordered forest vertex-cover (see Fig. 1), and *Problem UFVC* – unordered forest vertex-cover. Likewise, for edges we have *Problem OFEC* – ordered forest edge-cover, and *Problem UFEC* – unordered forest edge-cover.

In an ordered tree T, let $\{u_1,\ldots,u_{deg(u)}\}$ be the children of a vertex u . We say that the children of u are *covered consecutively* by the children of a vertex x of F if for some $1 \le i \le deg(u)-deg(x)$, the children $u_i,\ldots,u_{i+deg(x)-1}$ of u are all covered by the children of x. A *cover* of T by F is *consecutive* if for every vertex u in T which is covered by a vertex x of F, u's children are covered consecutively by x's children. *Problem CFEC* is to find a minimum consecutive edge-cover of an ordered tree T by a forest F. Note that *Problem CFVC* – to find a minimum consecutive vertex-cover – is a restricted case of OFVC.

For a vertex-cover $t_1,\ldots,t_k$ of a tree T by a forest F, let FC be the multiset of non-trivial subtrees t_j in the forest vertex-cover fulfilling $|V(t_j)|>1$. Let $r=|FC|$: FVC is equivalent to finding a forest vertex-cover which minimizes $r+(|V(T)|-\sum|V(t_j)|)$, that is, it maximizes $\sum|V(t_j)|-r$. The problem is new, having a flavor of both max and min: for a constant $\sum|V(t_j)|$, it minimizes r, while for a constant r, it maximizes $\sum|V(t_j)|$. Therefore, FVC is equivalent to packing into T a set of copies of trees in $F-\{f_1\}$, where $|V(f_1)|=1$, such that $\sum|V(t_j)|-r$ is maximized. When the trees in $F-\{f_1\}$ are of equal cardinality, the problem is of maximizing r, becoming equivalent to the maximum packing problem. When more than one set of trees covers the same number of vertices in T, the problem becomes one of minimizing the number r of trees. Similarly, Problem FEC is equivalent to finding a forest edge-cover which minimizes $r+(|E(T)|-\sum|E(t_j)|)$, that is, maximizes $\sum|E(t_j)|-r$.

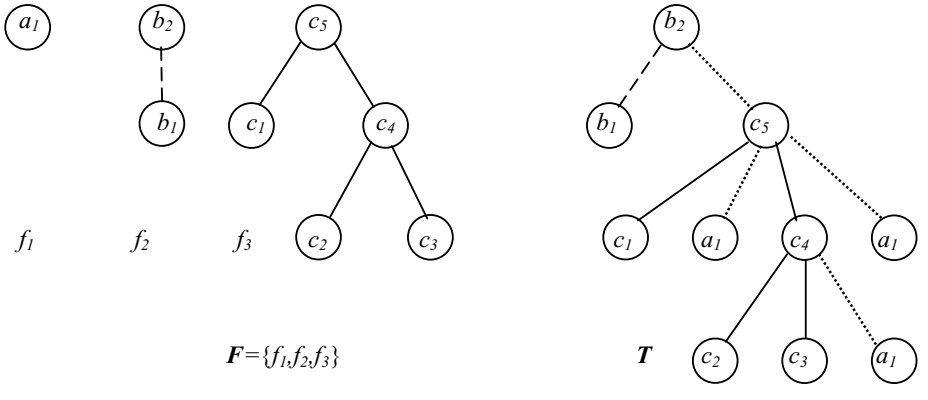


Fig. 1. An ordered forest vertex-cover of size 5: three copies of f_1 and one of f_2 and f_3

The above problems are related to the *subtree isomorphism problem*. Algorithms for the subtree isomorphism problem were given in [6,7,8], while the problem of subgraph isomorphism of a forest F into a tree T is NP-complete [3]. In the present paper we describe two polynomial time algorithms to solve Problems OFVC, UFVC and CFEC, and prove that Problems OFEC and UFEC are NP-complete. One algorithm called MAP-CHILDREN is similar to the algorithm for graph isomorphism and its complexity depends on $|V(F)|$. The other algorithm called MAP-LEAVES seems to be new, and works by replacing in F every maximal directed path in which the internal vertices have only one child, by a single edge; its complexity depends on $|L(F)|$, being very efficient when $|L(F)|$ is much smaller than $|V(F)|$, for example when the trees in F are paths. The algorithms can be extended to unrooted trees by considering copies of T rooted at each one of its vertices and extending the family $F=\{f_1,...,f_q\}$ to contain copies of every f_i rooted at each one of its vertices.

Problem FVC in a restricted form was discussed by Golumbic [4] for the factorization of a tree Boolean function as a read-once (fan-out) function, and by Levin and Pinter [5] for the realization of a tree Boolean function using a minimum number of logic circuits. An additional application is in translation: we wish to cover a syntax tree of a source language sentence by a minimum number of phrases, each of which has an optimal translation to the target language.

In Sections 2,3 we describe polynomial time algorithms to solve Problems OFVC and UFVC: in Section 2 the complexity depends on $|V(F)|$, while in Section 3, the complexity depends on $|L(F)|$. In Section 4 we describe similar algorithms to solve maximum packing problems of copies of trees of F into T. In Section 5 we prove that Problems OFEC and UFEC are NP-complete. In Section 6 we describe a polynomial time algorithm to solve CFEC.

2 Algorithm MAP-CHILDREN for Forest Vertex-Cover

Consider a tree T and a forest F. We shall describe how to extend covers of a subtree T_u of T, consisting of u and all its descendants, to a complete cover of T. Let $u \in V(T)$, $f \in F$ and $x \in V(f)$. An $[T_u, f_x]$ *forest vertex-cover of* T_u is an $F \cup \{f_x\}$ forest vertex-cover of T, such that $u=root(T_u)$ is covered by $x=root(f_x)$ and when $x \neq root(f)$, f_x is used only once in the cover. Note that if a vertex u is covered by a vertex x of $f \in F$, $x \neq root(f)$, then the parent $p_f(x)$ of x must cover the parent $p_T(u)$ of u. Let $W(u,x)$ be the number of trees in a minimum $[T_u, f_x]$ forest vertex-cover of T_u. Let $W(u)$ denote the number of trees in a minimum forest vertex-cover of T_u. Clearly $W(u)=min_{f \in F}\{W(u,root(f))\}$.

Consider first the case where x is a leaf of f, i.e., $V(f_x)=\{x\}$. Then in any $[T_u, f_x]$ forest vertex-cover of T_u, u is covered by x and each of its children is covered by the root of a tree of F. Hence, $W(u,x) = 1 + \sum_{v \in Ch(u)} W(v)$.

In the general case, consider an $[T_u, f_x]$ forest vertex-cover of T_u; let $X \subseteq V(T)$ be the set of vertices of the subtree rooted at u of T_u covered by f_x. $T_u - X$ is a set of disjoint subtrees of T_u, each subtree rooted at a vertex in $Ch[X] - X$ and covered by a forest vertex-cover of F. Let $T[X]$ denote the vertex subgraph of T induced by X. Hence,

$$W(u,x)=1+min_X\{\sum_{z \in Ch[X-\{u\}]-X} W(z)+\sum_{v \in Ch(u)-X} W(v) : X \subseteq V(T_u),\ T[X] \approx f_x,\ root(T[X])=u\}. \quad (1)$$

The above equation requires us to find all isomorphic copies of f_x rooted at u and hence might lead to an exponential time algorithm. To get a polynomial time algorithm we will show how to compute $W(u,x)$ from the values of $W(v),W(y)$, $v \in Ch(u)$, $y \in Ch(x)$.

An $[T_u, f_x]$ forest vertex-cover of T_u exists only if $deg(x) \leq deg(u)$. We construct the cover of T_u from optimal covers of the children of u: $deg(x)$ of u's children are covered by the $deg(x)$ trees in $\{f_z : z \in Ch(x)\}$ and the remaining children of u are covered by trees of F. A *matching* is an injection $\mu : Ch(x) \rightarrow Ch(u)$; let $M[Ch(x),Ch(u)]$ denote the set of possible matchings of $Ch(x)$ into $Ch(u)$. Therefore,

$$W(u,x) = 1 - deg(x) + min_{\mu \in M[Ch(x),Ch(u)]}\{\sum_{1 \leq j \leq deg(x)} W(\mu(x_j),x_j) + \sum_{v \in Ch(u)-\mu(Ch(x))} W(v)\} \quad (2)$$

Finally, $W(u)=min_{f \in F} W(u,root(f))$, and the size of a minimum cover is $W(root(T))$.

We describe a generic dynamic programming algorithm which is conducted by a postorder traversal of T and F, that is, it considers the children of u and x before considering u and x. This ensures that when evaluating the left side of equation (2), the values of the right side have been evaluated.

The dynamic programming algorithm traverses T and F in postorder and for every $u \in V(T)$, $f \in F$ and $x \in V(f)$, it finds the size of a minimum $[T_u, f_x]$ forest vertex-cover of T_u. To obtain $W(u)$ the algorithm finds the $f \in F$ which minimizes $W(u) = min_{f \in F}\{W(u,root(f))\}$. Since the trees of F are disjoint, once a vertex x is chosen, the tree $f \in F$ to which it belongs and the vertex of T covered by $root(f)$ are uniquely determined. When x is a leaf of f, hence $f_x=\{x\}$, the algorithm already evaluated for T_u the minimum cover of every tree T_v, $v \in Ch(u)$, thus $W(u,x) = 1 + \sum_{1 \leq i \leq deg(u)} W(u_i)$.

When x is not a leaf of f, the algorithm simulates a minimum weight isomorphism algorithm of f_x into T_u, by assuming that the corresponding ancestor of u, will be covered by $root(f)$. Thus, by optimally covering the children of u by the children of x, it carries to u (and to $root(f)$) the sizes of the minimum covers. This is done as follows:

A matching μ between a sequence of vertices $(u_1,\ldots,u_m)$ and $(x_1,\ldots,x_r)$ *is non-crossing* when every pair x_i,x_{i+1} is matched to a pair u_j,u_k fulfilling $j<k$. For each subproblem P we will restrict the permitted matchings to a subset $MP[Ch(x),Ch(u)]$ of all possible matchings. For OFVC, MP is the set of non-crossing matchings and for UFVC, MP is the set of all matchings. To every pair $[u_i,x_j]$ we assign a cost equal to $W(u_i,x_j)$. Hence, a minimum cost matching μ^* of $Ch(x)$ into $Ch(u)$ in equation (2), will give us the optimal way of covering the children of u by the children of x. This, together with the postorder traversal ensures by induction that the dynamic programming algorithm is correct.

Equation (2) is rewritten below as a minimum cost matching $\mu^* \in MP[Ch(x),Ch(u)]$ of a complete bipartite graph $[Ch(x),Ch(u)]$, in which the cost of every edge (u_i,x_j) is $W(u_i,x_j)-W(u_i)$:

$$W(u,x) = 1 - deg(x) + \sum_{1 \leq i \leq deg(u)} W(u_i) + \sum_{1 \leq j \leq deg(x)} [W(\mu^*(x_j),x_j) - W(\mu^*(x_j))] . \quad (3)$$

We denote $WS(u) = \sum_{1 \leq i \leq deg(u)} W(u_i)$ and $cost(\mu^*)=\sum_{1 \leq j \leq deg(x)} [W(\mu^*(x_j),x_j) - W(\mu^*(x_j))]$.

The algorithm keeps pointers along the minimum cost matchings to retrace the minimum forest vertex-cover when $u=root(T)$, and does not have to remember the intermediate forest vertex-covers.

Algorithm MAP-CHILDREN
for every $u \in V(T)$ in postorder $W(u)=\infty$;
  if u is a leaf **then** $W(u)=1$, $WS(u)=0$;
  if u is not a leaf **then** $WS(u) = \sum_{v \in Ch(u)} W(v)$;
  for every $x \in V(F)$ in postorder
    if u is a leaf
      if x is a leaf **then** $W(u,x)=1$ **else** $W(u,x)=\infty$;
    if u is not a leaf
      if x is a leaf **then** $W(u,x) = 1+WS(u)$;
      if x is not a leaf
        Let μ^* be a minimum cost matching in $MP[Ch(u),Ch(x)]$;
        $W(u,x) = 1 - deg(x) + WS(u) + cost(\mu^*)$ // equation (3)
    if $x=root(f)$, for some $f \in F$ **then** $W(u)=min\{W(u,x),W(u)\}$;
  if $u=root(T)$ **then** $W(u)$ is the size of the minimum cover;
end

In order to use the generic algorithm MAP-CHILDREN we need to specify how to calculate equation (3). For OFVC, MP is the set of non-crossing matchings. Assuming that $W(u_i)$ and $W(u_i,x_j)$ ($u_i \in Ch(u)$ and $x_j \in Ch(x)$) have been computed, we need to find a minimum cost non-crossing matching μ^*. We use dynamic programming again:

Let $S_{Ch(u)Ch(x)}[i,j]$ be the value of the minimum cost non-crossing matching between $u_1,\ldots,u_i$ and $x_1,\ldots,x_j$ where $S_{Ch(u)Ch(x)}[1,1]=W(u_1,x_1)$ and $S_{Ch(u)Ch(x)}[i,j]=\infty$ if $i<j$. Then

$$S_{Ch(u)Ch(x)}[i,j]=min\{S_{Ch(u)Ch(x)}[i-1,j-1]+W(u_i,x_j),S_{Ch(u)Ch(x)}[i-1,j]+W(u_j)\} \text{ for } i>1,\ j \le i\ . \quad (4)$$

To compute all $S_{Ch(u)Ch(x)}[i,j]$'s for given vertices u,x it takes $O(deg(u)deg(x))$ time. For all vertices, the required time is

$$\sum_{u \in V(T)} \sum_{x \in V(F)} deg(u)deg(x) = \sum_{u \in V(T)} deg(u) \sum_{x \in V(F)} deg(x) < |V(T)||V(F)|\ . \quad (5)$$

To find a minimum unordered cover UFVC, when the tree T and the forest F are unordered, we also apply MAP-CHILDREN. The only difference is that in equation (3) we drop the constraint that the matching be non-crossing, i.e., MP is the set of all matchings between $Ch(x)$ and $Ch(u)$.

To find the matching μ^* in the complete bipartite graph $MP[Ch(x),Ch(u)]$ we follow [1,7,8] and employ the maximal flow minimum cost algorithm of [2] to yield an algorithm that requires $O(\sum_{f \in F}|V(f)|^{1.5}\ |V(T)|\ log|V(T)|)$ time.

3 Algorithm MAP-LEAVES for Forest Vertex-Cover

In this section we construct an algorithm for OFVC and UFVC whose complexity depends on the number $|L(F)|$ of leaves of F, which may be much smaller than the number of vertices of F. The key step is to cover the vertices of T by the leaves

of F: covering a vertex $u \in V(T)$ by a leaf x of f determines how the path from x to *root(f)* covers vertices of T. However, in order not to scan each vertex of f separately, we shall replace the tree f by its *skeleton* – the tree *skel(f)* – resulting by replacing every maximal directed path in which the internal vertices have only one child, by a single edge. Now, every internal vertex of the tree *skel(f)* has at least two children. Thus, $|V(skel(f))| \le 2|L(f)|$. Let $SKEL = \{skel(f) : f \in F\}$ and let $SKEL^+$ be the set containing the vertices of $SKEL$ and their children in F. Since every edge of $SKEL$ gives rise to one child of F, $|SKEL^+| \le 2V(SKEL)| = O(|L(F)|)$.

Consider covering the vertex $u \in V(T)$ by a vertex $x \in V(skel(f))$, $skel(f) \in SKEL$. If u is a leaf then it can be covered only by leaves of $SKEL$. If u is not a leaf and x is a leaf, then u is covered by x, each of its children $v \in Ch(u)$ is covered by the root of a tree of F, and T_v is covered by a minimum forest vertex-cover with trees of F. If neither u nor x is a leaf, then each child of x in f must cover a child of u. The edge in *skel(f)* connecting x to a child y_x, corresponds in f to a path $(x,x_j,...,y_x)$. Assume that x_j covers u_i. Then u_i must have a descendant at distance $dist(x_j,y_x) = dist(x,y_x)-1$ which is covered by y_x.

Let $W(u)$, $WS(u)$ and $W(u,x)$ be as defined in Section 2. If u' is a descendant of u, let $Path_T(u,u')$ denote the path in T from u to u'. Let $CPath_T(u,u')$ be the set of children of vertices in $Path_T(u,u')-\{u'\}$, children which are not in $Path_T(u,u')$, and let $WP(u,u') = \sum_{v \in CPath(u,u')} W(v)$. We compute $WP(p_T(u),u')$ from $WP(u,u')$ by

$$WP(p_T(u),u') = WP(u,u') + \sum \{W(v) : v \in Ch(p_T(u))\} - W(u) = WP(u,u') + WS(u) - W(u). \quad (6)$$

To compute $W(u,x)$ for $u \in V(T)$, $x \in V(f)$, $f \in F$, we need to decide which children of u should be covered by the children of x in $f(x)$. Let (x,y_x) be an edge in *skel(f)*, let x_j be the child of x on the path in f from x to y_x and let $d = dist(x_j,y_x)$; to every child x_j of x corresponds exactly one y_x and one d. Let $\mathcal{D} = \{dist(x_j,y_x) : x_j \in SKEL^+, p_f(x_j) = x\}$. Since to every child x_j of x corresponds exactly one y_x it follows that $|\mathcal{D}| \le |SKEL^+| = O(|L(F)|)$. Let $D[u_i,d]$ be the list of descendants of $u_i \in V(T)$ at distance d from u_i. Thus, for children u_i of u and x_j of x we have

$$W(u_i,x_j) = min_{u'} \{W(u',y_x) + WP(u_i,u') : d = dist(x_j,y_x),\ u' \in D[u_i,d]\} . \quad (7)$$

Now, according to equation (3), rewritten below as (8), we need to find a minimum cost matching $\mu^* \in MP[Ch(u),Ch(x)]$ of a complete bipartite graph $(Ch(u),Ch(x))$, where the cost of every edge (u_i,x_j) is $W(u_i,x_j) - W(u_i)$ and evaluate:

$$W(u,x) = 1 - deg(x) + \sum_{v \in Ch(u)} W(v) + \sum_{z \in Ch(x)} [W(\mu^*(z),z) - W(\mu^*(z))] . \quad (8)$$

By equations (5-7) we do not have to compute $W(u,x)$ for all vertices $x \in V(F)$, but only for vertices x in $SKEL^+$. Thus we need to compute only $O(|V(T)||L(F)|)$ such values.

In the preprocessing stage we discard from F the trees whose height exceeds $height(T)$ and prepare $SKEL$, $SKEL^+$ and $\mathcal{D}$; this requires $O(|V(F)|)$ time. Also, we prepare $D[u,d]$ for all $u \in V(T)$ and $d \in \mathcal{D}$. Let $d_{max} = max\ \{d : d \in \mathcal{D}\}$; clearly

$d_{max} \leq height(T)$. In the worst case, each vertex appears in the list of all its ancestors, and so this requires $O(|V(T)|\ d_{max})$ time. The algorithm traverses T in postorder, and at vertex u it examines the cost of covering u by every $x \in SKEL^+$.

Algorithm MAP-LEAVES
for every $u \in V(T)$ in postorder $W(u)=\infty$;
  if u is a leaf **then** $W(u)=1,\ WS(u)=0$;
  if u is not a leaf **then** $WS(u) = \sum_{v \in Ch(u)} W(v)$;
    for every $v \in Ch(u)$ // *compute WP*
      for all descendants w of v at distance at most d_{max}
        $WP(u,w) = WS(u) - W(v) + WP(v,w)$;
  for every $x \in SKEL^+$ in postorder
    if u is a leaf
      if x is a leaf **then** $W(u,x)=1$ **else** $W(u,x)=\infty$;
    if u is not a leaf
      if x is a leaf **then** $W(u,x) = 1+WS(u)$;
      else if $x \in V(SKEL)$
        Let μ^* be a minimum cost matching in $MP[Ch(u),Ch(x)]$
        $W(u,x) = 1 - deg(x) + WS(u) + cost(\mu^*)$ // equation (7)
        if $x \in SKEL^+ - V(SKEL)$
          let y_x be the closest descendant of x that belongs to a tree in $SKEL$;
            $d = dist(x,y_x)$;
            $W(u,x) = min_{w \in D[u,d]} \{W(w,y_x) + WP(u,w)\}$; // y_x is unique for x (9)
  if $x=root(f), f \in F$ **then** $W(u)=min\ \{W(u,x), W(u)\}$;
if $u=root(T)$ **then** $W(u)$ is the size of the minimum cover;
end

The computation of $WP(u,w)$ for every u,w requires $O(|V(T)|\ d_{max})$ time, since each vertex w appears only in the list of its ancestors. The vertex $w \in V(T)$ in equation (9) is considered for its ancestor u at distance $d = dist(x,y_x)$. Thus for each $x \in SKEL^+$, w appears in $O(|SKEL^+|)$ computations of the minimum in (9). Hence, over all $w \in V(T)$, the number of vertices considered in the computation of all the minima in (9) is $O(|V(T)||SKEL^+|)=O(|V(T)||L(F)|)$. Since the matching can be found as in Section 2, we obtain: For OFVC, Algorithm MAP-LEAVES requires $O(|V(T)|d_{max}+|V(T)||L(F)|) \leq O(|V(T)|height(T) +|V(T)||L(F)|)$ time. For UFVC, Algorithm MAP-LEAVES requires $O(|V(T)|d_{max}+\sum_{f \in F}|L(f)|^{1.5}|V(T)|\ \log|V(T)|) \leq O(|V(T)|height(T)+\sum_{f \in F}|L(f)|^{1.5}|V(T)|\log|V(T)|)$ time.

4 Algorithms for Maximum Packing of a Forest in a Tree

Algorithms MAP-CHILDREN and MAP-LEAVES can be used for many other optimization problems on T and F. For example, finding a maximum packing of vertex disjoint copies of trees of F into T, can be solved in polynomial time; here we assume that F contains no single vertex tree, otherwise the problem is trivial. This problem has

two versions, one to maximize the number of packed trees and another to maximize the number of covered vertices of T. Denote by $W(u)$ the number of trees in a maximum forest packing of T_u. An $[T_u, f_x]$ *forest packing of* T_u is an $F \cup \{f_x\}$ forest packing of T_u, such that $u=root(T_u)$ is covered by $x=root(f_x)$ and when $x \neq root(f)$, f_x is used only once in the packing. Let $W(u,x)$ be the number of trees in a maximum $[T_u, f_x]$ forest packing of T_u. Then, similarly to equation (1),

$$W(u,x)=1+max_X\{\sum_{z\in Ch[X-\{u\}]-X} W(z)+\sum_{v\in Ch(u)-X} W(v)\ X\subseteq V(T_u),\ T[X]\approx f_x,\ root(T[X])=u\}. \quad (10)$$

Note that if a vertex u is covered by a vertex x of $f\in F$, $x\neq root(f)$, then $p_T(u)$ must be covered by $p_f(x)$. $W(u,x)$ can be evaluated as a maximum weight matching $\mu^* \in MP[Ch(x),Ch(u)]$ of a complete bipartite graph $[Ch(x),Ch(u)]$, in which the weight of every edge (u_i,x_j) is $W(u_i,x_j)-W(u_i)$:

$$W(u,x) = 1 - deg(x) + \sum_{1\leq i\leq deg(u)} W(u_i) + \sum_{1\leq j\leq deg(x)} [W(\mu^*(x_j),x_j) - W(\mu^*(x_j))]. \quad (11)$$

Clearly $W(u)=max\ \{\sum_{1\leq i\leq deg(u)} W(u_i)\ ,\ max_{f\in F}\{W(u,root(f))\}\}$ and the size of a maximum packing is $W(root(T))$; when u is a leaf, $W(u)=0$.

For a packing covering a maximum number of vertices of T, equation (11) is replaced by: $W(u,x) = 1+ \sum_{1\leq i\leq deg(u)} W(u_i) + \sum_{1\leq j\leq deg(x)} [W(\mu(x_j),x_j) - W(\mu(x_j))]$.

The complexity of the algorithms is similar to the complexity of the algorithms in Sections 2, 3.

5 Covering the Edges by a Minimum Ordered or Unordered Forest Edge-Cover

Consider a tree T and a forest F, $F=\{f_i : i=1,..,k\}$. For $u\in V(T)$ and $x\in V(F)$, since we are looking for an edge-disjoint cover, the edge from the parent $p_T(u)$ of u to u must be covered by exactly one edge of the forest F.

We prove that the Problems OFEC and UFEC are NP-complete, by reducing to them the NP-complete problem of Exact Cover by 3-Sets (X3C) [3].

Problem: Exact Cover by 3-Sets (X3C)
Instance: A set $X=\{v_1,...,v_n\}$ and a family of 3-subsets $S=\{s_1,...,s_k\}$ of X.
Question: Is there a subfamily $S'\subseteq S$ s.t. every $v_i\in X$ is contained in exactly one set in S' ?

Theorem 1: The problems of exact covering of the edges of a tree T by a minimum ordered or unordered forest edge-cover are NP-complete.

Proof. We show that the problem X3C is reducible to the Problems OFEC and UFEC, i.e., for each instance of X3C we show an instance of the edge-cover problem that has a cover of size $n/3$ if and only if X3C has a solution.

Consider a set $X=\{v_1,...,v_n\}$ and a family of 3-subsets $S=\{s_1,...,s_k\}$ of X. We construct a tree T (Fig. 2a) with root v whose children are $v_1,...,v_n$ and at every v_i we attach a subtree T_i defined as follows (Fig. 2b): T_i's root is v_i, v_i has i children which are leaves and v_i has attached a path with $n-i+1$ vertices. Clearly, every T_i has exactly $n+2$ vertices and no two T_i's are isomorphic. For every $s_j=\{v_a,v_b,v_c\}\in S$, $a<b<c$, we

define a tree f_j (Fig. 2c) with root s_j, children v_a, v_b, v_c from left to right, and copies of T_a, T_b, T_c attached as subtrees. Let $F=\{f_1,f_2,...,f_k\}\cup\{e\}$, where e is the tree consisting of two vertices and an edge between them. Consider an ordered or unordered forest edge-cover of size $n/3$ of T; such a covering is minimum since all vertices s_j's are mapped on v. Then, for every child v_i of v in T, there exists some f_j with child v_i of s_j, covering the child v_i of v. Thus, a forest edge-cover of size $n/3$ of T by trees in F gives an exact covering of X by subsets in S.

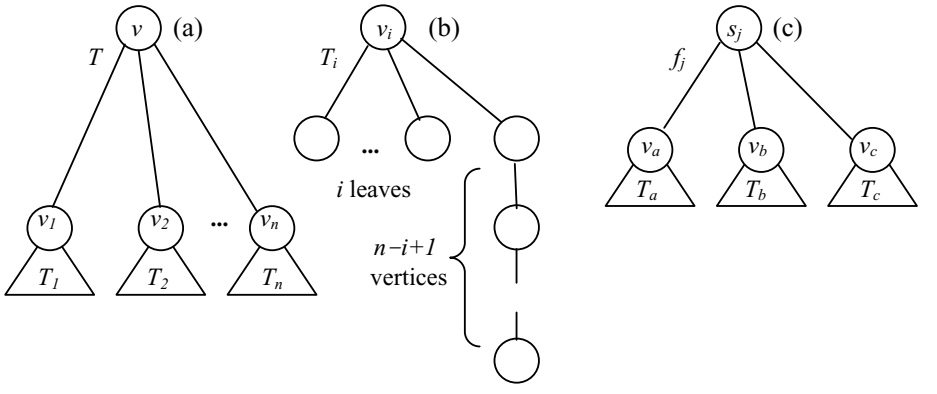


Fig. 2. An instance of forest edge-cover problem corresponding to an instance of X3C

Conversely, an exact covering of X by subsets $s_{i,1},...,s_{i,n/3}\in S$ will give a cover $f_{i,1},...,f_{i,n/3}\in F$ of the edges of T. Note that the order of the edges in the f_i's is compatible to that of T. Thus any exact cover is an ordered cover. □

By the same reduction, the maximum packing problems of edge disjoint copies of trees of F into a tree T, are NP-complete.

6 An algorithm for a Minimum Consecutive Forest Edge-Cover

Problem CFEC, finding a minimum consecutive edge-cover, assumes that T is an ordered tree and if $u\in V(T)$ is covered by $x\in V(F)$, then the children of u covered by the children of x, are consecutive in the order of T. Let the edge from the parent of a vertex v to v in a tree t be denoted by $p_t(v)\rightarrow v$. For non-roots $u\in T$ and $x\in V(F)$, since we are looking for an edge-disjoint cover, the edge $p_T(u)\rightarrow u$ should be covered by exactly one edge of F, the edge $p_f(x)\rightarrow x$. Let us denote $f_x^+ = f_x\cup\{p_f(x)\rightarrow x\}$, $T_u^+=T_u\cup\{p_T(u)\rightarrow u\}$. For non-roots u, x, let $[T_u^+, f_x^+, j]$ denote a *consecutive edge-cover* of the subtree T_u^+ by the forest $F\cup\{f_x^+\}$ such that the edge $p_f(x)\rightarrow x$ covers the edge $p_T(u)\rightarrow u$, the children $\{x_1,...,x_{deg(x)}\}$ of x cover the children $u_{j-deg(x)+1},...,u_j$ of u, and the tree f_x^+ is used only once in the cover.

Let $T_u(i,j)$ denote the subtree of T_u containing u (as root), the children $u_i,...,u_j$ of u and all the children's descendants. The algorithm is based on the observation that in a

$[T_u^+, f_x^+, j]$ consecutive edge-cover, the subtrees $T_u(1,j-deg(x))$ and $T_u(j+1,deg(u))$ have consecutive edge-covers by F, that is, u is covered only by roots of trees in F, while $T_u(j-deg(x)+1,\ j) \cup \{p_T(u) \to u\}$ has a consecutive edge-cover by the forest $F \cup \{f_x^+\}$, using the tree f_x^+ only once.

Let $\mu_{u,x,j}$ be a minimum cost matching of the complete bipartite graph $[Ch(x), \{u_{j-deg(x)+1}, \ldots, u_j\}]$ in which the cost of every edge (v,z), $v \in \{u_{j-deg(x)+1}, \ldots, u_j\}$, $z \in Ch(x)$, is the size of a minimum $[T_v^+, f_z^+, j]$ consecutive edge-cover. Let $W(u,i,j)$ be the cardinality of a minimum consecutive edge-cover of $T_u(i,j)$ by F. For every u and j we will evaluate

$$W(u,1,j) = min_{f \in F} \{ W(u,1,\ j - deg(root(f)) + cost(\mu_{u,root(f),j}) \} \tag{12}$$

$$W(u,j,deg(u)) = min_{f \in F} \{ W(u,\ j + deg(root(f),deg(u)) + cost(\mu_{u,root(f),j+deg(root(f)-1)}) \} \ . \tag{13}$$

For non-roots u, x, let $WR(p_T(u) \to u, p_f(x) \to x)$ be the size of a minimum among all j, $1 \le j \le deg(u)$, of a $[T_u^+, f_x^+, j]$ consecutive edge-cover, Therefore

$$WR(p_T(u) \to u, p_f(x) \to x) = min_{deg(x) \le j \le deg(u)} \{ W(u,1,j-deg(x)) + W(u,j+1,deg(u)) + cost(\mu_{u,x,j}) \}. \tag{14}$$

Algorithm MAP-EDGES

for every $u \in V(T)$ in postorder
  for every $f \in F$
   for $j=1,\ldots,deg(u)$ set $W(u,1,j)=W(u,j,deg(u))=\infty$;
  if u is not a leaf **then**
    for every $j=deg(u)-deg(root(f)+1,\ldots,deg(u)$
      find a minimum cost matching $\mu_{u,root(f),j}$;
      $W(u,1,j)=min_{f \in F}\{ W(u,1,\ j-deg(root(f)) + cost(\mu_{u,root(f),j}) \}$;
    for every $j=deg(u),deg(u)-1,\ldots,\ deg(u)-deg(root(f)+1$
     $W(u,j,deg(u))=min_{f \in F}\{ W(u,\ j+deg(root(f),deg(u)) + cost(\mu_{u,root(f),j+deg(root(f)-1)}) \}$;
 if $u \ne root(T)$ **then**
   for every $x \in f \in V(F)$, $x \ne root(f)$, in postorder
     if u is a leaf **then**
       if x is a leaf **then** $WR(p_T(u) \to u, p_f(x) \to x)=1$ **else** $WR(p_T(u) \to u, p_f(x) \to x)=\infty$;
       if u is not a leaf **then**
       if x is a leaf **then** $WR(p_T(u) \to u, p_f(x) \to x)=1+W(u,1,deg(u))$;
       if x is not a leaf **then**
         Let $\mu_{u,x,j}$ be a minimum cost matching of the complete bipartite graph
                                                $MP[Ch(x), \{u_{j-deg(x)+1}, \ldots, u_j\}]$;
        $WR(p_T(u) \to u, p_f(x) \to x)=min_{deg(x) \le j \le deg(u)}\{ W(u,1,j-deg(x)) + W(u,j+1,deg(u))$
                                                $+cost(\mu_{u,x,j}) \}$;
    if $u=root(T)$ **then** $W(u,1,deg(u))$ is the size of the minimum edge-cover;
end

The algorithm works in $O(\sum_{f \in F} |V(f)|^{1.5}\ |V(T)|\ log|V(T)|)$ time by the matching algorithms in [1,2].

The maximum packing problems of edge disjoint copies of trees of F into T, where the children of u covered by the children of x, are consecutive in the order of T, can also be solved in polynomial time; here we assume that F contains no single vertex and no single edge tree otherwise the problem is trivial. This is done by an algorithm similar to the above, by changing the equations (12), (13) to:

$$W(u,1,j) = min\ \{W(u,1,j-1)\ ,\ min_{f\in F}\{W(u,1,\ j - deg(root(f)) + cost(\mu_{u,root(f),j})\}\}. \quad (15)$$

$$W(u,j,deg(u)) = min\ \{W(u,1,j+1),\ min_{f\in F}\{W(u,j+deg(root(f),deg(u)) + cost(\mu_{u,root(f),j+deg(root(f)1)})\ \}\}\ . \quad (16)$$

References

1. Feder, T., Botwani, R.: Clique Partitions, Graph Compression and Speeding-up Algorithms, J. Comput. Syst. Sci. 51, 261–272 (1995)
2. Gabow, H.N., Tarjan, R.E.: Faster Scaling Algorithms for Network Problems. SIAM J. Comput. 18, 1013–1036 (1989)
3. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory NP-Completeness. W. H. Freeman and Co., San Francisco (1979)
4. Golumbic, M.C., Mintz, A., Rotics, U.: Factoring and Recognition of Read-once Functions using Cographs and Normality. In: DAC 2001, Las Vegas, pp. 109–114 (2001)
5. Levin, I., Pinter, R.Y.: Realizing Expression Graphs using Table-Lookup FPGAs. In: Proceedings of EuroDAC 1993, pp. 306–311 (1993)
6. Lingas, A.: An Application of Maximum Bipartite c-Matching to Subtree Isomorphism. In: CAAP 1983. LNCS, vol. 159, pp. 284–299. Springer, Heidelberg (1983)
7. Pinter, R.Y., Rokhlenko, O., Tsur, D., Ziv-Ukelson, M.: Approximate Labelled Subtree Homeomorphism. In: Sahinalp, S.C., Muthukrishnan, S.M., Dogrusoz, U. (eds.) CPM 2004. LNCS, vol. 3109, pp. 59–73. Springer, Heidelberg (2004)
8. Shamir, R., Tsur, D.: Faster Subtree Isomorphism. J. Algorithms 33, 267–280 (1999)

Dominating Induced Matchings

Domingos M. Cardoso[1,⋆] and Vadim V. Lozin[2,⋆⋆]

[1] Departamento de Matemática da Universidade de Aveiro,
3810-193 Aveiro, Portugal
dcardoso@ua.pt
[2] DIMAP and Mathematics Institute, University of Warwick,
Coventry, CV4 7AL, UK
V.Lozin@warwick.ac.uk

Abstract. We study the problem of determining whether or not a graph G has an induced matching that dominates every edge of the graph, which is also known as EFFICIENT EDGE DOMINATION. This problem is known to be NP-complete in general as well as in some restricted domains, such as bipartite graphs or regular graphs. In this paper, we identify a graph parameter to which the complexity of the problem is sensible and produce results of both negative (intractable) and positive (solvable in polynomial time) type.

Keywords: Dominating induced matching; Efficient edge dominating set; Polynomial-time algorithm.

1 Introduction

Let G be a simple graph, i.e, an undirected graph without loops and multiple edges. Given an edge e in G, we say that e dominates itself and every edge sharing a vertex with e. An *induced matching* in G is a subset of edges such that each edge of G is dominated by at most one edge of the subset. In this paper we study the problem of determining whether a graph has a dominating induced matching, i.e., an induced matching that dominates *every* edge of the graph. This problem is also known in the literature as EFFICIENT EDGE DOMINATION. It can also be viewed as a restricted version of VERTEX 3-COLORABILITY, i.e., the problem of determining whether the vertices of a graph can be partitioned into three independent sets. In the DOMINATING INDUCED MATCHING problem we are looking for a partition of a graph into three independent sets such that two of them induce a 1-regular graph.

Our concern in this paper is the computational complexity of the DOMINATING INDUCED MATCHING problem in special classes of graphs. Recently, it was shown

⋆ Research of Domingos M. Cardoso supported by the Centre of Research on Optimization and Control from the Fundação para a Ciência e Tecnologia cofinanced by the European Community Funf FEDER/POCI/2010.

⋆⋆ Research of Vadim Lozin supported by DIMAP – Center for Discrete Mathematics and its Applications at the University of Warwick.

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 77–86, 2009.

in [4] that an induced matching in a graph is dominating only if it is of maximum size. Finding a maximum induced matching is a well studied problem (see e.g. [1,2,6]). It is NP-hard in general graphs and in many particular classes such as bipartite graphs of degree at most three [10] or line graphs [8]. This, however, does not necessarily imply the NP-completeness of the DOMINATING INDUCED MATCHING problem in the same class, as we shall see later.

The NP-completeness of the DOMINATING INDUCED MATCHING problem was first shown in [7]. In [9], the author proved the NP-completeness of this problem in cubic graphs, while [4] extended this result to d-regular graphs for an arbitrary $d \geq 3$. The NP-completeness was also shown for bipartite graphs [13] and planar bipartite graphs [12]. On the other hand, polynomial-time solutions have been developed for the problem in the class of bipartite permutation [13] and chordal graphs [12].

Following this line of research, we present results of two types. First, we generalize many of the NP-completeness results mentioned above by identifying a graph parameter to which the complexity of the problem is sensible. Then we present a new polynomially solvable case which deals with the class of claw-free graphs. Observe that the MAXIMUM INDUCED MATCHING problem is NP-hard in this class [8], which reveals the computational difference between the two problems. It is interesting to note that in the abstract of paper [7] the authors mistakenly claimed the NP-completeness of the DOMINATING INDUCED MATCHING problem in the class of line graphs (a proper subclass of claw-free graphs). Our solution to the problem in claw-free graphs corrects this wrong statement.

The organization of the paper is as follows. In the rest of this section, we introduce general notations and definitions. In Section 2, we prove the NP-completeness result, while in Section 3 develop a polynomial-time algorithm for the problem in the class of claw-free graphs. Section 4 concludes the paper with some further observations on the complexity of the problem in special graph classes.

For a graph G, we denote by $V(G)$ and $E(G)$ the vertex set and the edge set of G, respectively. If $v \in V(G)$, then $N(v)$ is the neighborhood of v, i.e., the set of vertices adjacent to v. The degree of v is $|N(v)|$. A graph is k-regular of the degree of each vertex is k. An independent set in G is a subset of pairwise nonadjacent verteices. For a subset, $U \subseteq V(G)$, we denote by $G[U]$ the subgraph of G induced by vertices of U. If G does not contain induced subgraphs isomorphic to a graph H, we say that G is H-free and call H a forbidden induced subgraph for G. The class of graphs containing no induced subgraphs isomorphic to graphs in a set M will be denoted $Free(M)$.

As usual, K_n is the complete graph on n vertices, $K_{n,m}$ is the complete bipartite graph with parts of size n and m, and P_n (C_n) is the chordless path (cycle) on n vertices. By $G + H$ we denote the disjoint union of two graphs G and H. In particular, $mG = G + \ldots + G$ is the disjoint union of m copies of G. Also, $S_{i,j,k}$ and H_i are two graphs represented in Figure 1.

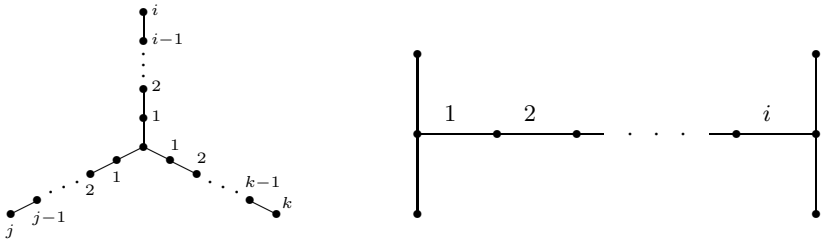


Fig. 1. Graphs $S_{i,j,k}$ (left) and H_i (right)

2 An NP-Completeness Result

From [7] we know that determining if G has a dominating induced matching is an NP-complete problem. Moreover, it is NP-complete even for bipartite graphs [13]. In this section, we prove a somewhat stronger result. To this end, let us first present the following technical lemma.

Lemma 1. *Let G be a graph and e an edge in G. If G' is the graph obtained from G by subdividing the edge e exactly three times, then G has a dominating induced matching if and only if G' has.*

Proof. Denote the endpoints of e by a and b, and the three vertices subdividing the edge e by x, y, z. Assume first that G has a dominating induced matching M. If $e = ab \in M$, then the set $M' = M \cup \{ax, zb\}$ is a dominating induced matching in G'. If $e = ab \notin M$ and e is dominated by a certain edge of M incident to a, then $M' = M \cup \{yz\}$ is a dominating induced matching in G'.

Conversely, suppose G' has a dominating induced matching M'. If neither xy nor yz belong to M', then $ax, zb \in M'$ and hence $M = (M' - \{ax, zb\}) \cup \{ab\}$ is a dominating induced matching in G. Assume now without loss of generality that $yz \in M'$. Then the set $M = M' - \{yz\}$ is a dominating induced matching in G. □

A direct consequence of this lemma is the following result.

Lemma 2. *For any k, the* DOMINATING INDUCED MATCHING *problem is NP-complete in the class of bipartite $(C_3, \ldots, C_k, H_1, \ldots, H_k)$-free graphs of vertex degree at most three.*

Proof. We prove the lemma by reducing the problem from graphs of vertex degree at most three, where the problem is known to be NP-complete.

Let G be a graph of vertex degree at most 3 and G' a graph obtained from G by a triple subdivision of an edge of G. Then G' is also of degree at most three and it has a dominating induced matching if and only if G has. If we subdivide each edge of G three times, then we obtain a bipartite graph, since the length

of each edge of G increases 4 times. Applying this operation repeatedly, we can get rid of small induced cycles and small graphs of the form H_i. The resulting graph is bipartite, of maximum degree three and it has a dominating induced matching if and only if G has. This proves the lemma. □

We now generalize this lemma in the following way. Let $\mathcal{S}_k$ denote the class of $(C_3, \ldots, C_k, H_1, \ldots, H_k)$-free bipartite graphs of vertex degree at most 3. To every graph G we associate the parameter $\kappa(G)$, which is the maximum k such that $G \in \mathcal{S}_k$. If G belongs to no class $\mathcal{S}_k$, we define $\kappa(G)$ to be 0, and if G belongs to all classes $\mathcal{S}_k$, then $\kappa(G)$ is defined to be ∞. Finally, for a set of graphs M, we define $\kappa(M) = \sup\{\kappa(G) \ : \ G \in M\}$.

Theorem 1. *Let M be a set of graphs and X the class of M-free bipartite graphs of vertex degree at most 3. If $\kappa(M) < \infty$, then the* DOMINATING INDUCED MATCHING *problem is NP-complete in the class X.*

Proof. To prove the theorem, we will show that there is a k such that $\mathcal{S}_k \subseteq X$. Denote $k := \kappa(M) + 1$ and let G belong to $\mathcal{S}_k$. Assume that G does not belong to X. Then G contains a graph $A \in M$ as an induced subgraph. From the choice of G we know that A belongs to $\mathcal{S}_k$, but then $k \leq \kappa(A) \leq \kappa(M) < k$, a contradiction. Therefore, $G \in X$ and hence, $\mathcal{S}_k \subseteq X$. By Lemma 2, this implies the NP-completeness of the problem in the class X. □

3 Dominating Induced Matchings in Claw-Free Graphs

The results of the previous section suggest that, unless $P = NP$, the problem is solvable in polynomial time in the class $Free(M)$ only if $\kappa(M) = \infty$. One of the ways to push $\kappa(M)$ to infinity is to include in M a graph G with $\kappa(G) = \infty$. By definition, $\kappa(G) = \infty$ if and only if G belongs to all classes $\mathcal{S}_k$, which means G has no induced cycles, no induced graphs of the form H_i and no vertices of degree more than three. In other words, $\kappa(G) = \infty$ if and only if every connected component of G is of the form $S_{i,j,k}$ represented in Figure 1. In this section, we analyze the first non-trivial cases when $\kappa(G) = \infty$, namely, $G = S_{1,1,1}$. The class of graphs excluding $S_{1,1,1}$ as an induced subgraph is known in the literature as the claw-free graphs. The main result of the section is that in the class of claw-free graphs the DOMINATING INDUCED MATCHING problem is solvable in polynomial-time. In the proof, we use an alternative definition of the problem which asks to determine if the vertex set of a graph G admits a partition into two subsets W and B such that W is an independent set and B induces a 1-regular graph. Throughout the section we will call the vertices of W *white* and the vertices of B *black*, and the partition $V(G) = B \cup W$ *black-white partition* of G. In other words, a graph G has a dominating induced matching if and only if G admits a black-white partition. We will use these two notions interchangeably.

Assigning one of the two possible colors to the vertices of G will be called *coloring* of G. A coloring is *partial* if only part of the vertices of G have been assigned colors, otherwise it is *total*. A partial coloring is *valid* if no two white

vertices are adjacent and no black vertex has more than one black neighbor. A total coloring is *valid* if no two white vertices are adjacent and every black vertex has exactly one black neighbor.

We start with preliminary results valid for all graphs, not necessarily claw-free. First, we observe that

Observation 1. *If G admits a black-white partition, then in any triangle of G two vertices are black and one is white.*

Second, we prove the following helpful lemma.

Lemma 3. *If a graph G has a dominating induced matching, the neighborhood of each vertex of G induces a subgraph each connected component of which is a star $K_{1,s}$ for some s.*

Proof. Let v be a vertex in a graph G with a dominating induced matching. Then $G[N(v)]$ is K_3-free, since otherwise G is not 3-colorable (and hence has no dominating induced matching).

Assume $G[N(v)]$ contains an induced $P_4 = (a, b, c, d)$. Then v is not white, since otherwise the vertices a, b, c, d are all black, which is not possible in a valid black-white partition. If v is black, then at most one of the vertices a, b, c, d is black and then at least three are white with two of these three connected by an edge, which is a contradiction.

Similarly, we can show that $G[N(v)]$ is C_4-free. Therefore, $G[N(v)]$ is a forest. Since $G[N(v)]$ is P_4-free, each connected component of this forest is a star $K_{1,s}$ for some s. □

From now on, G is a claw-free graph. Without loss of generality we will assume that G is connected and the maximum vertex degree in G is at least 3 (for graphs of degree at most 2 the problem is trivial). The next lemma shows that we also may assume that the maximum vertex degree in G is at most 4.

Lemma 4. *If a claw-free graph G has a vertex of degree more than 4, then G has no dominating induced matching.*

Proof. Let v be a vertex of degree more than 4 and assume by contradiction that G has a dominating induced matching. From Lemma 3 we know that $G[N(v)]$ is a forest each connected component of which is a star $K_{1,s}$ for some s. Since G is claw-free, the number of components is at most 2 and for each component we have $s \leq 2$. Moreover, to avoid a claw, we conclude that if $G[N(v)]$ has a component $K_{1,2}$, then there are no other components, i.e., the degree of v is 3. If each component has at most 2 vertices, then the degree of v is at most 4. This contradiction completes the proof of the lemma. □

From Lemmas 3 and 4, we conclude that a claw-free graph G has an efficient edge dominating set only if each vertex v of G is one of the following six types:

(1) degree 1,
(2) degree 2 with two adjacent neighbors,

(3) degree 2 with two non-adjacent neighbors,
(4) degree 3 with $G[N(v)]$ inducing a $K_1 + K_2$,
(5) degree 3 with $G[N(v)]$ inducing a $K_{1,2}$,
(6) degree 4 with $G[N(v)]$ inducing a $2K_2$.

Before we proceed to an algorithm, let us simplify the input graph as follows. First, we replace any three consecutive vertices of type 3 by an edge. According to Lemma 1, the modified graph has an efficient edge dominating set if and only if the original one has. In other words, we will assume without loss of generality that

(a) if v is a vertex of type 3, then at least one of its neighbors is not of type 3.

This implies in particular that any vertex of degree 1 is connected to the nearest vertex of degree 3 by a chordless path of length at most 3. Moreover, it is not difficult to see that if the length of the path is 3, we can delete this path and the new graph has an efficient edge dominating set if and only if the original one has. Therefore, in what follows we assume that

(b) any vertex of degree 1 is connected to the nearest vertex of degree 3 by a chordless path of length at most 2.

For the reader's convenience, we also recall that

(c) G is connected and has at least one vertex of degree more than 2.

With the assumptions (a), (b), (c) in mind we can derive the following conclusion the proof of which is simple and hence is omitted.

Lemma 5. *Let G be a graph with a black-white partition and v a vertex of G. If*

- *v is of type 1 with a neighbor of type 4, then v is white.*
- *v is of type 1 with a neighbor of type 3, then v is black.*
- *v is of type 3 with a neighbor of type 3, then v is black.*
- *v is of type 3 with a neighbor of type 1, then v is black.*
- *v is of type 3 with both neighbors of type 4, then v is white.*
- *v is of type 5, then v is black.*
- *v is of type 6, then v is white.*

According to Lemma 5, the vertices of types 1,3,5 and 6 can be colored before the main step of the algorithm starts. If this initial coloring produces a conflict (two white adjacent vertices or a black vertex with more than one black neighbor), then the algorithm terminates and reports that the input graph has no black-white partition. Otherwise, the initial coloring can be further propagated according to the following obvious rules:

R1 : each neighbor of a white vertex must be black;
R2 : all neighbors of two black adjacent vertices must be white;

R3 : if v is a vertex of type 4 and w its neighbor which is isolated in $G[N(v)]$, then v and w must be colored differently.

A partial coloring will be called *maximal* if it cannot be extended by application of rules $R1 - R3$. Assuming that the initial coloring is valid and maximal and G still has some uncolored vertices, we divide the rest of the algorithm into two steps. First, in the subgraph of G induced by uncolored vertices, we color the cycles of length more than three.

Lemma 6. *If the subgraph of G induced by uncolored vertices contains a chordless cycle C with at least 4 vertices, then C is of even length. Moreover, if G admits a black-white partition, then the vertices of C are colored alternately black and white, and furthermore, by switching the colors along the cycle we again obtain a valid black-white partition of G.*

Proof. Clearly, no vertex of C can be of type 2. Therefore, each vertex of C is of type 4. For any vertex v of type 4, exactly two edges incident to v belong to a triangle and we will call them *heavy* edges, and the remaining edge belong to no triangle, and we will call it a *light* edge.

Since each vertex of C is of type 4, light edges in C alternate with heavy edges. Therefore, C is of even length. Moreover, since the endpoints of light edges must be colored differently (rule $R3$), the colors of vertices of C must alternate. Each vertex u of G that has a neighbor on C must be adjacent to two consecutive vertices of the cycle (otherwise a claw arises). Since one of these neighbors is white, u must be colored black. Therefore, switching the colors along the cycle does not produce any conflicts, and hence leads to another black-white partition of G. □

Lemma 6 reduces the problem to the induced subgraph of G which is a chordal graph, i.e., a graph containing no chordless cycles of length more than 3. It turns out this subgraph can be always colored without producing any conflicts (if the current coloring is valid and maximal).

Lemma 7. *Let ϕ is a maximal partial valid coloring of G such that the subgraph of G induced by uncolored vertices has no chordless cycles of length more than three. Then ϕ can be extended to a total valid coloring of G.*

Proof. The connected components of the subgraph of G induced by uncolored vertices can be colored separately and independently of each other. Therefore, we may assume without loss of generality that the uncolored vertices induce a connected graph. Denote by H the subgraph of G obtained by deleting those colored vertices that have no neighbors among uncolored ones (clearly, the deleted vertices are of no importance for finding an extension of ϕ).

By Lemma 5 and maximality of ϕ, every vertex of H belongs to exactly one triangle and any two triangles of H are disjoint. Let $T_1, T_2, \ldots, T_k$ be the list of all these triangles. Also, from maximality of ϕ we know that

- each triangle T_i has either two or three uncolored vertices;
- if a triangle T_i has two uncolored vertices, then the only colored vertex of T_i is black;
- no two black vertices of different triangles T_i and T_j are adjacent.

The last item implies in particular that H has no chordless cycles of length more than three. In other words, by contracting each triangle T_i into a single vertex we obtain a tree. A triangle of H that becomes a leaf in this tree will be called a leaf triangle of H.

We will prove the lemma by induction on k, i.e., on the number of triangles in H. If $k = 1$, then G contains exactly two uncolored vertices. By coloring one of them white and the other black we obtain a total valid coloring of G.

Assume the lemma is true for any number of triangles less then k and let T_i be a leaf triangle of H. By deleting T_i we obtain a subgraph of H which, by the induction hypothesis, admits a total valid coloring ϕ'. This subgraph contains a unique vertex x that has a neighbor y in T_i. Observe that y is necessarily uncolored in ϕ, as every colored vertex of H has degree 2 in this graph. If x is black in ϕ', then we color y white, and vice versa. The rest of T_i is colored arbitrarily according to the rule that any triangle must contain strictly two black vertices. □

We summarize the above discussion in Algorithm α below. This algorithm is robust in the sense that it does not require the input graph G to be claw-free. The algorithm either finds a black-white partition of G or reports that G has no such partition or G is not claw-free. As before, we assume without loss of generality that G satisfies (a), (b) and (c).

Algorithm α

Input: a graph G

Output: a black-white partition of G or report "G has no black-white partition or G is not claw-free"

1. If at least one vertex of G is not of type $1, 2, 3, 4, 5$ or 6, then STOP and output "G has no black-white partition or G is not claw-free".
2. If G has no vertices of type 1,3,5,6, then $A := \emptyset$, otherwise, color the vertices of type 1,3,5,6 according to Lemma 5, extend this coloring to a maximal one according to rules $R1 - R3$, and denote the set of colored vertices by A.
3. If the coloring of vertices of A is not valid, then STOP and output "G has no black-white partition or G is not claw-free".
4. If $A = V(G)$, then STOP and output the coloring of $V(G)$, otherwise, $U := V(G) - A$.
5. As long as $G[U]$ has a cycle C of length more than 3, do

   5.1. color the vertices of C alternately black and white (starting with an arbitrary vertex), extend the coloring to a maximal one, add the newly colored vertices to A and delete them from U;

5.2. if the coloring of vertices of A is not valid, then STOP and output "G has no black-white partition or G is not claw-free";
5.3. if $A = V(G)$, then STOP and output the coloring of $V(G)$.
6. Extend the coloring of A to a total coloring according to Lemma 7 and output the black-white partition of G.

Theorem 2. *Algorithm α correctly solves the* DOMINATING INDUCED MATCHING *problem for any claw-free graph G with n vertices in time $O(n^2)$.*

Proof. Correctness of the algorithm follows from Lemmas 3-7. The most time consuming steps of the algorithm are 5 and 6. In the analysis of step 5, it is helpful to consider the subgraph H of G obtained by deleting those colored vertices that have no neighbors among uncolored ones. Similarly as in Lemma 7, every vertex of H belongs to exactly one triangle and any two triangles of H are disjoint. By contracting each triangle of H into a single vertex, we obtain an auxiliary (multi)graph H' that has a cycle if and only if H has a cycle of length more than 3. Finding a cycle in H' is a linearly solvable problem, hence step 5 can be implemented in quadratic time. Obviously, this time is also sufficient to execute step 6. □

4 Concluding Remarks

In this paper, we studied the computational complexity of the DOMINATING INDUCED MATCHING problem (also known as EFFICIENT EDGE DOMINATION) on special graph classes. We tightened the gap between classes where the problem is NP-complete (by strengthening some of the NP-completeness results) and those where the problem is solvable in polynomial time (by revealing a new polynomially solvable case). But still the gap contains a vast variety of unexplored classes. Further tightening can be obtained, for instance, on the basis of the following observation.

Observation 2. *The* DOMINATING INDUCED MATCHING *problem can be solved in polynomial time in any class of graphs of bounded clique-width.*

Recently, many classes of graphs have been shown to be of bounded clique-width (see e.g. [5,11,14]), which implies by Observation 2 and results in [3] linear-time solvability of the problem in such classes. For some other classes, efficient algorithms can be derived by combining Observation 2 with other results. We complete the paper with an example of this type. In particular, by combining Observation 2 with Lemma 3 we obtain an alternative proof of the fact that the problem is solvable in polynomial time in the class of chordal graphs, i.e., graphs without chordless cycles of length at least 4. Observe that in general the clique-width of chordal graphs is unbounded.

Proposition 1. *The* DOMINATING INDUCED MATCHING *problem can be solved in the class of chordal graphs in linear time.*

Proof. A *gem* is a graph on 5 vertices of which 4 induce a P_4 and one is adjacent to each vertex of the P_4. By Lemma 3, if a graph contains a gem as an induced subgraph then it has no dominating induced matching. Chordal graphs that are gem-free are known as *ptolemaic* graphs and they from a subclass of distance-hereditary graphs. Therefore, the clique-width of gem-free chordal graphs is at most 3 [5]. This suggests the following algorithm to solve the problem for a chordal graph G: if G contains a gem (which can be determined in linear time), then G has no dominating induced matching. Otherwise, apply the results from [3] to solve the problem for G in linear time. □

References

1. Brandstädt, A., Eschen, E.M., Sritharan, R.: The induced matching and chain subgraph cover problems for convex bipartite graphs. Theoret. Comput. Sci. 381, 260–265 (2007)
2. Cameron, K.: Induced matchings. Discrete Appl. Math. 24, 97–102 (1989)
3. 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)
4. Cardoso, D.M., Cerdeira, J.O., Delorme, C., Silva, P.C.: Efficient edge domination in regular graphs. Discrete App. Math. 156, 3060–3065 (2008)
5. Golumbic, M.C., Rotics, U.: On the clique-width of some perfect graph classes. International J. Foundations of Computer Sci. 11, 423–443 (2000)
6. Golumbic, M.C., Lewenstein, M.: New results on induced matchings. Discrete Appl. Math. 101, 157–165 (2000)
7. Grinstead, D.L., Slater, P.J., Sherwani, N.A., Holmes, N.D.: Efficient edge domination problems in graphs. Inform. Process. Lett. 48, 221–228 (1993)
8. Kobler, D., Rotics, U.: Finding maximum induced matchings in subclasses of claw-free and P_5-free graphs, and in graphs with matching and induced matching of equal maximum size. Algorithmica 37, 327–346 (2003)
9. Kratochvíl, J.: Regular codes in regular graphs are difficult. Discrete Math 133, 191–205 (1994)
10. Lozin, V.V.: On maximum induced matchings in bipartite graphs. Inform. Process. Lett. 81, 7–11 (2002)
11. Lozin, V.V., Rautenbach, R.: The tree- and clique-width of bipartite graphs in special classes. Australasian J. Combinatorics 34, 57–67 (2006)
12. 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)
13. Lu, C.L., Tang, C.Y.: Solving the weighted efficient edge domination problem on bipartite permutation graphs. Discrete Appl. Math. 87, 203–211 (1998)
14. Makowsky, J.A., Rotics, U.: On the clique-width of graphs with few P_4's. International J. Foundations of Computer Sci. 10, 329–348 (1999)

HyperConsistency Width for Constraint Satisfaction: Algorithms and Complexity Results

Georg Gottlob[1], Gianluigi Greco[2], and Bruno Marnette[1]

[1] Oxford University, OX1 3QD Oxford, UK
{georg.gottlob,bruno.marnette}@comlab.ox.ac.uk
[2] University of Calabria, Via P.Bucci 30B, 87036, Rende, Italy
ggreco@mat.unical.it

Abstract. Generalized hypertree width (short: ghw) is a concept that leads to a large class of efficiently solvable CSP instances, whose associated recognition problem (of checking whether the ghw of a CSP is bounded by a constant k) is however known to be NP-hard. An elegant way to circumvent this intractability has recently been proposed in the literature, by means of a "no-promise" approach solving CSPs of bounded ghw without the need of actually computing a generalized hypertree decomposition. In fact, despite the conceptual relevance of this approach, its computational issues have not yet been investigated and, indeed, precise bounds on the running time of the no-promise algorithm are missing.

The first contribution of this paper is precisely to fill this gap. Indeed, the computational complexity of the no-promise approach is analyzed, by exploiting an intuitive characterization relying on the notion of *hyperconsistency width*. It turns out that, in the basic formulation, the approach is hardly suited for practical applications mainly because of its bad scaling in the size of the constraint database. Motivated by these news and based on a variant of hyperconsistency width, a different and more efficient method to decide whether CSPs of bounded ghw admit solutions is then provided. Importantly, the improved method exhibits the same scaling as current evaluation algorithms for instances of bounded hypertree width, nonetheless allowing to isolate a larger class of queries. Finally, to give a complete picture of the complexity issues of the no-promise approach, the problems of computing one solution and of enumerating all the solutions are also studied.

1 Introduction

The Constraint Satisfaction Problem (CSP) is a well-known framework to model and solve search problems in several application domains. Formally, a CSP instance (e.g., [5]) is a triple $(Var, U, \mathcal{C})$, where Var is a finite set of variables, U is a finite domain of values, and $\mathcal{C} = \{C_1, C_2, \ldots, C_q\}$ is a finite set of constraints, where each C_i is a pair (S_i, r_i), in which $S_i \subseteq Var$ is called the *constraint scope*, and $r_i \subseteq U^{|S_i|}$ is called the *constraint relation*. Then, a solution for a CSP instance is simply a substitution $\theta : Var \mapsto U$ that satisfies all constraints.

Following [13], in this paper we shall exploit the logic-based characterization of a CSP instance as a pair $(Q, \mathcal{D})$, where $\mathcal{D}$ is the *constraint database*, i.e., the set of all the constraint relations r_i, for each constraint $C_i = (S_i, r_i)$, and Q is a *conjunctive query*, i.e., an existentially quantified first-order formula with no negations or disjunctions,

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 87–99, 2009.

over the relational vocabulary consisting of the atoms $r_i(S_i)$. Then, a solution θ is a substitution such that $\theta(Q)$ evaluates true over $\mathcal{D}$. The set of all the solutions for $(Q, \mathcal{D})$ will be denoted by $Q[\mathcal{D}]$.

Since solving CSPs in their general formulation is an NP-hard problem, much research has been spent to identify restricted classes of CSPs over which solutions can efficiently be computed. In this paper, we shall focus on classes of CSPs defined by *structural* restrictions on the queries (see, e.g., [15,5,11,7]). In this context, we recall that a class $\mathcal{C}$ is generally considered an "island of tractability" for CSPs if both the *recognition* problem of deciding whether Q is in $\mathcal{C}$, and the following *promise* problem

$$\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}] : \begin{cases} \text{Input} & : \text{a CSP instance } (Q, \mathcal{D}) \text{ s.t. } Q \in \mathcal{C} \\ \text{Output} & : \text{Yes iff } Q[\mathcal{D}] \neq \emptyset, \\ & \quad \text{No iff } Q[\mathcal{D}] = \emptyset \end{cases}$$

are feasible in polynomial time.

When the recognition problem is hard, the *promise* problem $\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}]$ might still be of interest in some applications, since one may have a guarantee that Q belongs to the class $\mathcal{C}$. But, this is in general not the case and, hence, one is more likely interested in solving the *no-promise* problem $\mathsf{SAT}^{\mathsf{np}}_{\emptyset}[\mathcal{C}]$, where an arbitrary CSP instance $(Q, \mathcal{D})$ is given in input, and where an algorithm may possibly decline to answer (e.g., answering $\mathsf{IDon't Know}$) when $Q \notin \mathcal{C}$:

$$\mathsf{SAT}^{\mathsf{np}}_{\emptyset}[\mathcal{C}] : \begin{cases} \text{Input} & : \text{a CSP instance } (Q, \mathcal{D}) \\ \text{Output} & : \text{Yes only if } Q[\mathcal{D}] \neq \emptyset, \\ & \quad \text{No only if } Q[\mathcal{D}] = \emptyset, \\ & \quad \mathsf{IDon't Know} \text{ only if } Q \notin \mathcal{C} \end{cases}$$

The need of dealing with no-promise problems has recently been argued in [3], where the class of queries whose *generalized hypertree width* [8] is bounded by a constant k (short: class $\mathcal{C}(\mathsf{ghw}, k)$) has been considered, and where it is shown that $\mathsf{SAT}^{\mathsf{np}}_{\emptyset}[\mathcal{C}(\mathsf{ghw}, k)]$ can be solved in polynomial time, by means of a *projective k-consistency* algorithm. Even though this result is conceptually relevant because checking whether the ghw of a CSP is bounded by a constant k is NP-hard [9], its practical applicability is still unclear because a thorough analysis of the complexity issues of the projective k-consistency algorithm has not been conducted and, in fact, precise bounds on its running time were missing.

In this paper we continue along this line of research, and we face the above research questions by shedding light on the computational aspects of the projective k-consistency algorithm, which have not been discussed in [3]. Indeed:

(1) We introduce the notion of *hyperconsistency width* as a measure for characterizing the intricacy of constraints, and we study the computational properties of the class $\mathcal{C}(\mathsf{hcw}, k)$ of those queries whose hyperconsistency width is bounded by k.
(2) We show that this notion is a different, yet equivalent reformulation of the *projective width*, implicit in [3]. In fact, our reformulation leads to a conceptually simpler method whose correctness is proven without the need of any game-theoretic characterization for it.

	$\mathsf{hcw} \leq \mathsf{ghw}$	$\mathsf{ahcw} \leq \mathsf{ghw}$
$\mathsf{SAT}^{\mathsf{p}}_{\emptyset}$	$O(\mathbf{q}^{3k}\mathbf{r}^{2k})$	$O(\mathbf{q}^{2k+1}\mathbf{r}^{k})$
$\mathsf{SAT}^{\mathsf{p}}_{\exists}$	$O(\mathbf{q}^{3k+1}\mathbf{r}^{2k+1})$	$O(\mathbf{q}^{2k+2}\mathbf{r}^{k+1})$
$\mathsf{SAT}^{\mathsf{p}}_{\forall}$	$O(\mathbf{q}^{3k+1}\mathbf{r}^{2k+1}\mathbf{o})$	$O(\mathbf{q}^{2k+2}\mathbf{r}^{k+1}\mathbf{o})$
$\mathsf{SAT}^{\mathsf{np}}_{\emptyset}$	$O(\mathbf{q}^{3k+1}\mathbf{r}^{2k+1})$	$O(\mathbf{q}^{2k+2}\mathbf{r}^{k+1})$
$\mathsf{SAT}^{\mathsf{np}}_{\exists}$	$O(\mathbf{q}^{3k+1}\mathbf{r}^{2k+1})$	$O(\mathbf{q}^{2k+2}\mathbf{r}^{k+1})$
$\mathsf{SAT}^{\mathsf{np}}_{\forall}$	$O(\mathbf{q}^{3k+2}\mathbf{r}^{2k+2}\mathbf{o})$	$O(\mathbf{q}^{2k+3}\mathbf{r}^{k+2}\mathbf{o})$

Fig. 1. Results for $\mathsf{SAT}[C(\mathsf{ghw}, k)]$. On the left: bounds for the algorithm in [3]; on the right: improved bounds.

(3) We introduce and investigate a different method to solve problems in $\mathcal{C}(\mathsf{ghw}, k)$, whose complexity improves on the bounds derived in **(1)**. In particular, the basic version of the projective k-consistency algorithm scales as $O(\mathbf{q}^{3k}\mathbf{r}^{2k})$, where $\mathbf{q}$ is the size of Q (i.e. the number of atoms plus the number of variables in the formula), and $\mathbf{r}$ is the size of the largest relation in $\mathcal{D}$ (i.e. the number of its entries). To the contrary, our novel method scales as $O(\mathbf{q}^{2k+1}\mathbf{r}^{k})$, thereby significantly enlarging the class of instances that can practically be managed. Indeed, this scaling is basically the same that one may achieve when solving CSPs on the class of queries having bounded *hypertree width* [8] (short: $\mathcal{C}(\mathsf{hw}, k)$), even though this class is properly contained in $\mathcal{C}(\mathsf{ghw}, k)$ and is, in fact, one of the largest classes of tractable CSPs that is known to be also efficiently recognizable (see, e.g., [2,7]).

(4) Technically, the result in **(3)** is achieved by exploiting the fact that the hyperconsistency width is defined in a way that is parametric w.r.t. the underlying decompositions. This is the main conceptual difference with the projective width. In fact, our improved method is based on restricting the consistency algorithm on the class of *acyclic* decompositions whose width is bounded by k (short: the subclass $\mathcal{C}(\mathsf{ahcw}, k)$ of $\mathcal{C}(\mathsf{hcw}, k)$). The relationships among these two notions and the notion of generalized hypertree width are also clarified in the paper.

(5) Finally, to complement the results on the decision problems related to the class $\mathcal{C}(\mathsf{ghw}, k)$, we investigate the complexity issues arising for the problems of computing *one* solution (short: $\mathsf{SAT}_{\exists}$) and of computing *all* the solutions (short: $\mathsf{SAT}_{\forall}$).

Our results are summarized in Figure 1, where $\mathbf{o}$ denotes the size of the output $Q[\mathcal{D}]$. Notice that the complexity of the problems SAT, $\mathsf{SAT}_{\exists}$, and $\mathsf{SAT}_{\forall}$ has completely been characterized for both the cases of promise and no-promise problems, and for both the approach investigated in [3] and for the improved methods based on the notion of acyclic hyperconsistency width.

The rest of the paper is organized as follows. In Section 2, the basic notion of hyperconsistency width is presented and its links with the notion of projective width are discussed. Then, in Section 3, the subclass $\mathcal{C}(\mathsf{ahcw}, k)$ of $\mathcal{C}(\mathsf{hcw}, k)$ is defined and its computational properties are studied, by also comparing them with the bounds derived

for the approach in [3]. The problems of computing one solution and of computing all the solutions are discussed in Section 4. Eventually, a few final remarks and some directions for further work are reported in Section 5.

2 HyperConsistency Width

In this section, we present the notion of hyperconsistency width, which can be viewed as a natural generalization of the various consistency concepts that have been studied in the context of binary representations of constraints (e.g., [4,14]). Then, we compare it with the notion of *projective width* that is implicit in [3].

The first step is to introduce the concept of *hypergraph decomposition*. In fact, the reader might already be familiar with the notion of *(generalized) hypertree decomposition* [8], which is basically a tree whose vertices are associated with a set of atoms and with a subset of the variables occurring in them, and where for each variable X, the subgraph induced over the vertices containing X is connected (see Section 3.1). Then, hypergraph decompositions can be viewed as an extension of hypertree decompositions where the underlying shape is an arbitrary graph rather than a tree, and where the connectedness condition is omitted. In the following, let us denote by $\mathsf{var}(Q)$ (resp., $\mathsf{atoms}(Q)$) the set of variables (resp., atoms) in a query Q.

Definition 1. A *hypergraph decomposition* J of a query Q, is a tuple $(\mathcal{G}, \chi, \lambda)$ such that:

- $\mathcal{G} = (V(J), E(J))$ is a directed graph;
- $\lambda : V(J) \rightarrow 2^{\mathsf{atoms}(Q)}$ associates to each vertex $v \in V(J)$ a set of atoms $\lambda(v) \subseteq \mathsf{atoms}(Q)$;
- $\chi : V(J) \rightarrow 2^{\mathsf{var}(Q)}$ associates to each vertex $v \in V(J)$ a set of variables $\chi(v) \neq \emptyset$ satisfying $\chi(v) \subseteq \bigcup_{A \in \lambda(v)} \mathsf{var}(A)$.

The *width* of J is defined as $\max_{v \in V(J)} |\lambda(v)|$. □

Next, we shall make use of some standard *relational operators* to manipulate constraint relations (see, e.g., [1]). Thus, if r_i is a relation over the scope S_i, and X is a subset of S_i, we denote by $\Pi_X(r_i)$ the relation obtained by projecting r_i over X. Also, given two relations r_i and r_j, the *join* of r_i and r_j is denoted by $r_i \bowtie r_j$; and, the *semi-join* of r_i and r_j, denoted by $r_i \ltimes r_j$, is just a shortcut for $r_i \bowtie (\Pi_{S_i \cap S_j} r_i)$.

Let $\mathcal{G} = (V, E)$ be a directed graph. A $\mathcal{G}$-set of relations $\mathcal{R}$ is a set of relations in one-to-one correspondence with the nodes V; thus, for each $v \in V$, $r_v \in \mathcal{R}$ is the associated relation in $\mathcal{R}$. The *$\ltimes_{\mathcal{G}}$-fixed-point* of $\mathcal{R}$, denoted by $\gamma_{\mathcal{G}}(\mathcal{R})$, is the $\mathcal{G}$-set of relations obtained as the unique fixed-point of $\mathcal{R}$ for the set of rules $\{r_v := r_v \ltimes r_{v'} \mid (v, v') \in E\}$.

Based on these notions, we can define an extremely simple and clear algorithm $\mathcal{A}_J$ (for a decomposition J) deciding whether $Q[\mathcal{D}] = \emptyset$. This algorithm, reported in Figure 2, firstly computes for each node v, the joins and the projections corresponding to $\lambda(v)$ and $\chi(v)$, respectively. Then, it computes the $\ltimes_{\mathcal{G}}$-fixed-point of the resulting $\mathcal{G}$-set of relations. Finally, it reports No (resp., IDon'tKnow) if some relation in this $\ltimes_{\mathcal{G}}$-fixed-point is (resp., is not) empty.

Input: a CSP instance $(Q, \mathcal{D})$, and a decomposition J;
Output: No or IDon'tKnow;

begin
  for each vertex $v \in V(\mathcal{G})$ **do**
    let $r_v = \Pi_{\chi(v)}(\ltimes_{A \in \lambda(v)} (A[D]))$;
  compute $\{r'_v \mid v \in V(\mathcal{G})\} = \gamma_{\mathcal{G}}(\{r_v \mid v \in V(\mathcal{G})\})$;
  if there is $v \in V(\mathcal{G})$ such that $r'_v = \emptyset$, **then return** No;
  else return IDon'tKnow;
end.

Fig. 2. Algorithm $\mathcal{A}_J$

Actually, $\mathcal{A}_J$ may be incomplete, but one may easily see that it is sound. In addition, we next show that it terminates in polynomial time and that, for some special classes of decomposition, it leads to a very efficient procedure.

Proposition 1. *For a k-width hypergraph decomposition J, algorithm $\mathcal{A}_J$ returns* No *on input $(Q, \mathcal{D})$ only if $Q[\mathcal{D}] = \emptyset$. Moreover, the $\ltimes_{\mathcal{G}}$-fixed-point can be computed so that $\mathcal{A}_J$ terminates in time: $O(|V| \times |E| \times \mathbf{r}^{2k})$, in the general case, and $O((|V|+|E|) \times \mathbf{r}^k)$ when $\mathcal{G}$ is acyclic.*

Proof. If some relation r'_v in the $\ltimes_{\mathcal{G}}$-fixed-point is empty, then any substitution θ for the variables in $\chi(v)$ can not be extended to a solution, i.e., to a substitution for all the variables in the query satisfying the constraints.

As for the running time, on a RAM machine, the join of k relations can be computed in time $O(\mathbf{r}^k)$, the projection of one relation in linear time, and the semijoin of two relations in linear time [6]. Therefore, the set $\{r_v \mid v \in V(\mathcal{G})\}$ can be computed in time $O(|V| \times \mathbf{r}^k)$, and each semijoin required for computing $\gamma_{\mathcal{G}}(\{r_v \mid v \in V(\mathcal{G})\})$ takes $O(\mathbf{r}^k)$. When $\mathcal{G}$ is acyclic, its edges can be sorted according to a topological order, so that only $|E|$ steps are needed to reach the fixed point. In the general case, instead, the result follows since $|V| \times \mathbf{r}^k$ steps are required at most to reach the fixed point, because this is an upper bound on the number of tuples in $\{r_v \mid v \in V(\mathcal{G})\}$, and since each step involves $|E|$ semijoin computations. □

Depending on the structure of Q and J, $\mathcal{A}_J$ might be a complete decision procedure. Formally, we say that a hypergraph decomposition J of a query Q is *consistent* if, for each database $\mathcal{D}$,

$$\mathcal{A}_J \text{ returns } \begin{cases} \mathsf{No} & \text{iff } Q[\mathcal{D}] = \emptyset \\ \mathsf{IDon't Know} & \text{iff } Q[\mathcal{D}] \neq \emptyset \end{cases}$$

Then, the *hyperconsistency width* $\mathsf{hcw}(Q)$ of Q is the minimum width over all its consistent decompositions. The class of those queries whose hyperconsistency width is bounded by k is denoted by $\mathcal{C}(\mathsf{hcw}, k)$. Focusing on this class provides a guarantee for the tractability of CSPs. This is shown below, by providing a polynomial bound for solving the $\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}(\mathsf{hcw}, k)]$ problem.

Theorem 1. *Let $(Q, \mathcal{D})$ be a CSP instance, let* $\mathbf{q}$ *be the size of* Q *and* $\mathbf{r}$ *the size of the largest relation in* $\mathcal{D}$*. Then,* $\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}(\mathsf{hcw}, k)]$ *is feasible in* $O(\mathbf{q}^{3k}\mathbf{r}^{2k})$.

Proof. Let $J = (\mathcal{G}, \chi, \lambda)$ and $J' = (\mathcal{G}', \chi', \lambda')$ be two hypergraph decompositions for the same query Q. We say that J is contained in J', denoted by $J \sqsubseteq J'$, if there is a morphism f from the nodes of $\mathcal{G}$ to the nodes of $\mathcal{G}'$ such that $f(\mathcal{G}')$ is a subgraph of $\mathcal{G}$ and, for each node v of $\mathcal{G}$: $\chi'(f(v)) \supseteq \chi(v)$, and $\lambda'(f(v))) \supseteq \lambda(f(v)$.

Armed with this notion, one may note that $J \sqsubseteq J' \wedge J$ is consistent $\Rightarrow$ J' is consistent. In addition, we observe that for any fixed natural number k, the hypergraph decomposition J_k where:

- $V(J_k) = \{v \subseteq \mathsf{atoms}(Q) \mid |\mathsf{atoms}(v)| \leq k)\}$;
- $E(J_k) = V(J_k) \times V(J_k)$;
- $\forall v \in V(J_k)$, $\lambda(v) = v$ and $\chi(v) = (\cup_{A \in v} \mathsf{var}(A))$.

is $\sqsubseteq$-maximal among the set of all the k-width hypergraph decompositions.

When $\mathsf{hcw}(Q) \leq k$, we know that J_k is consistent and we can decide $\mathsf{SAT}^{p}_{\emptyset}[\mathcal{C}(\mathsf{hcw}, k)]$ by using the algorithm $\mathcal{A}_{J_k}$. Then, we can apply Proposition 1, and the running time follows by observing that $|V(J_k)| \times |E(J_K)| = |V(J_k)|^3$ and that $|V(J_k)| = O(\mathbf{q}^k)$, by construction. □

2.1 Links with Projective Width

We are now in the position of showing that the notion of hyperconsistency width is a different formulation of the notion of projective width of [3]. The result is next proven by observing that this notion provides a simple and natural characterization of the k-cover game, introduced in [3] to define classes of queries with bounded projective width.

For a query Q and a database $\mathcal{D}$, the game $\mathsf{k\text{-}cover}(Q, \mathcal{D})$ can be described as follows. Two players, *spoiler* and *duplicator*, play one after the other. At each step i, the spoiler chooses a pair (X_i, L_i) such that $L_i \subseteq \mathsf{atoms}(Q)$, $|L_i| \leq k$ and $X_i \subseteq \bigcup_{A \in L_i} \mathsf{var}(A)$. Then, the duplicator chooses a homomorphism h_i from variables in X_i to constants in $\mathcal{D}$ such that $h_i(X_i) \in \Pi_{X_i}(\bowtie_{A \in L_i} A[\mathcal{D}])$. Moreover, from the step $i > 1$, the duplicator is also asked to choose h_i such that, for each variable $V \in X_{i-1} \cap X_i$, $h_i(V) = h_{i-1}(V)$. The spoiler wins if the duplicator cannot find the required homomorphism, and the duplicator wins if the play is infinite.

Theorem 2. *Let Q be a query. Then, the following statements are equivalent:*

- *for each database $\mathcal{D}$, $Q[\mathcal{D}] = \emptyset$ implies that the spoiler has a winning strategy in the* $\mathsf{k\text{-}cover}(Q, \mathcal{D})$ *game;*
- $\mathsf{hcw}(Q) \leq k$.

Proof. (Sketch) For each k-width hypergraph decomposition $J = (\mathcal{G}, \chi, \lambda)$ of Q, and for each database $\mathcal{D}$, we can naturally define the game $\mathsf{k\text{-}cover}_J(Q, D)$ as a restriction of $\mathsf{k\text{-}cover}(Q, D)$, where the spoiler is asked to choose tuples of the form $(X_i, L_i) = (\chi(v_i), \lambda(v_i))$ for some $v_i \in V(J)$, and to follow the edges of E, i.e., for each $i > 1$, $(X_{i-1}, L_{i-1}) = (\chi(v_{i-1}), \lambda(v_{i-1}))$ is such that $(v_{i-1}, v_i) \in E(J)$. Then, we can show that for each node $v \in V(J)$, the relation r'_v computed by the

algorithm $\mathcal{A}_J$ in Figure 2 exactly corresponds to a mapping witnessing a winning strategy for the duplicator in the game $\mathsf{k\text{-}cover}_J(Q, \mathcal{D})$. Therefore, a hypergraph decomposition J is consistent if and only if for each database $\mathcal{D}$ such that $Q[\mathcal{D}] = \emptyset$, the spoiler has a winning strategy in $\mathsf{k\text{-}cover}_J(Q, D)$. To conclude the proof, we claim that $\mathsf{k\text{-}cover}(Q, \mathcal{D})$ coincides with the game $\mathsf{k\text{-}cover}_{J_k}(Q, D)$, where J_k is the $\sqsubseteq$-maximal hypergraph decomposition constructed as in the proof of Theorem 1. □

Note that a similar link was established in [14] for binary representations of CSPs.

3 Acyclic HyperConsistency Width

According to Theorem 1, the promise problem for the class $\mathcal{C}(\mathsf{hcw}, k)$ can be solved in polynomial time in the size of the query and the database. However, despite this theoretical guarantee on the tractability, the running time $O(\mathbf{q}^{3k}\mathbf{r}^{2k})$ seems not suited for practical applications, mainly because of the bad scaling in the size of the database. In particular, we point out that solving CSPs in the class $\mathcal{C}(\mathsf{hw}, k)$ (or, in $\mathcal{C}(\mathsf{ghw}, k)$, when a decomposition is given to hand) can be done by means of algorithms scaling as $O(\mathbf{r}^k)$ in the size of the database (see, e.g., [8]).

Hence, improving on the scaling we have derived for the notion of hyperconsistency width is a major requirement for making this approach practical and competitive w.r.t. structural methods already used in the literature. To achieve this goal, the basic idea we shall exploit is to introduce a refinement of the notion of hyperconsistency width, for which a good scaling is obtained by suitably restricting the shape of the underlying hypergraph decompositions to acyclic graphs (as suggested by the better scaling which can be achieved in this case according to Proposition 1).

Definition 2. The *acyclic hyperconsistency width* $\mathsf{hcw}(Q)$ of a query Q is the minimum width over all its consistent decompositions $J = (\mathcal{G}, \chi, \lambda)$ such that:

(1) J is acyclic; and,
(2) the *depth* of J (the length of the longest path in $\mathcal{G}$) is bounded by the size of Q. □

Clearly enough, this novel notion comes as a refinement of the hyperconsistency width and, therefore, it holds that $\mathcal{C}(\mathsf{ahcw}, k) \subseteq \mathcal{C}(\mathsf{hcw}, k)$. On the other hand, it allows us to improve the complexity result of Theorem 1, as stated below.

Theorem 3. $\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}(\mathsf{ahcw}, k)]$ *is feasible in* $O(\mathbf{q}^{2k+1}\mathbf{r}^k)$.

Proof. We observe that for any fixed natural number k, the following hypergraph decomposition $J_{k,n}$ where:

- $V(J_{k,n}){=}V \times \{0, 1, ..., n\}$, where $V = \{v \subseteq \mathsf{atoms}(Q) \mid \ |\mathsf{atoms}(v)| \leq k)\}$;
- $E(J_{k,n}){=}\{((v, i), (v', i')) \mid (v, v'){\in}V{\times}V \wedge \ i'{=}i + 1\}$
- $\forall v \in V(J_k)$, $\lambda(v) = v$ and $\chi(v) = (\cup_{A \in v}\mathsf{var}(A))$.

is $\sqsubseteq$-maximal among all the k-width decompositions of depth bounded by n. Therefore, when $\mathsf{ahcw}(Q) \leq k$, and $n = |Q|$ we know that $J_{k,n}$ is consistent and we can decide $\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}(\mathsf{hcw}, k)]$ by using the algorithm $\mathcal{A}_{J_{k,n}}$. The running time, then, follows from Proposition 1. □

3.1 Links with Generalized Hypertree Width

We next investigate on how the classes $\mathcal{C}(\mathsf{ahcw}, k)$ and $\mathcal{C}(\mathsf{hcw}, k)$ compare with $\mathcal{C}(\mathsf{ghw}, k)$. Let us start by preliminary recalling the definition of generalized hypertree width [8], by stating it as a specialization of the notion of hypergraph decomposition.

In fact, the focus is on acyclic decompositions (similarly to the case of ahcw), satisfying some additional requirements.

Definition 3. A *generalized hypertree decomposition* J of a query Q is a hypergraph decomposition $(\mathcal{G}, \chi, \lambda)$ such that:

- $\mathcal{G}$ is a rooted tree;
- $\forall A \in \mathsf{atoms}(Q)$, $\exists v \in V(G)$ such that $A \in \lambda(v)$;
- $\forall X \in \mathsf{var}(Q)$, the subgraph of $\mathcal{G}$ induced over the nodes in $\{v \in V(G) \mid X \in \chi(v)\}$ is a tree.

The *generalized hypertree width* of Q, denoted by $\mathsf{ghw}(Q)$, is the minimum width over all its generalized hypertree decompositions. □

Proposition 2. *Let J be a generalized hypertree decomposition. Then, J is consistent.*

Proof. (Sketch) It is sufficient to observe that when J is a generalized hypertree decomposition, the algorithms $\mathcal{A}_J$ in Figure 2 basically coincides with the evaluation algorithm in [16], proposed for the class of acyclic queries. □

Therefore, $\mathcal{C}(\mathsf{ghw}, k) \subseteq \mathcal{C}(\mathsf{hcw}, k)$ (cf. [3]) and, hence, when $\mathsf{ghw}(Q) \leq k$ and when a generalized hypertree decomposition J is given to hand, we can decide whether $Q[\mathcal{D}] = \emptyset$, for each database $\mathcal{D}$, by using Theorem 1.

Unfortunately, this approach to solve instances of bounded generalized hypertree width has two important drawbacks. First, its scaling is $O(\mathbf{q}^{3k}\mathbf{r}^{2k})$ as a direct consequence of the result on hyperconsistency width. And, second, no polynomial-time algorithm may exist to recognize instances in the class $\mathcal{C}(\mathsf{ghw}, k)$, unless $\mathrm{P} = \mathrm{NP}$ (cf. [9]). Dealing with the intractability of ghw will be addressed in Section 4. Here, we focus instead on the former drawback, by relating the three notions studied in the paper and by observing that the nice scaling results of the ahcw can be extended to the generalized hypertree width.

Theorem 4. *For every natural number k, it holds that: $\mathcal{C}(\mathsf{ghw}, k) \subseteq \mathcal{C}(\mathsf{ahcw}, k) \subseteq \mathcal{C}(\mathsf{hcw}, k)$.*

Proof. (Sketch) We know that $\mathcal{C}(\mathsf{ahcw}, k) \subseteq \mathcal{C}(\mathsf{hcw}, k)$; hence, only $\mathcal{C}(\mathsf{ghw}, k) \subseteq \mathcal{C}(\mathsf{ahcw}, k)$ remains to be proven. Actually, since generalized hypertree decompositions are always acyclic, it suffices to show that when $\mathsf{ghw}(Q) \leq k$ for a query Q, a generalized hypertree decomposition of Q exists whose depth is bounded by size of Q. In fact, this follows from the game-theoretic characterization of generalized hypertree width in [10], where monotone winning strategies can be mapped into generalized hypertree decompositions; indeed, because of the monotonicity of the game, the depth of the generalized hypertree decomposition is bounded by $|\mathsf{var}(Q)|$. □

A consequence of the theorem above is that the algorithm for $\mathsf{SAT}^p_\emptyset[\mathcal{C}(\mathsf{ahcw}, k)]$ discussed in the proof of Theorem 3 is also an algorithm for $\mathsf{SAT}^{\mathsf{p}}_\emptyset[\mathcal{C}(\mathsf{ghw}, k)]$.

Corollary 1. $\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}(\mathsf{ghw}, k)]$ *is feasible in* $O(\mathbf{q}^{2k+1}\mathbf{r}^{k})$.

To conclude the analysis on the relationships among the various notions, we observe that from [3] and Theorem 2 it follows that there are classes of queries where the generalized hypertree width is unbounded, while the hyperconsistency width remains bounded.

Indeed, this is because the hyperconsistency width is preserved under taking *cores* [12], while the generalized is not. Recall, here, that the *core* of a query Q, denoted by $\mathsf{CORE}(Q)$, is a query Q' such that: *(1)* $\mathsf{atoms}(Q') \subseteq \mathsf{atoms}(Q)$; *(2)* there is a mapping $f : \mathsf{var}(Q) \mapsto \mathsf{var}(Q')$ such that for each atom $r_i(X_1, ..., X_n) \in \mathsf{atoms}(Q)$, $r_i(f(X_1), ..., f(X_n))$ is in $\mathsf{atoms}(Q')$; and, *(3)* there is no query Q'' satisfying *(1)* and *(2)* such that $\mathsf{atoms}(Q'') \subset \mathsf{atoms}(Q')$.

In fact, we can show that a similar result holds when comparing the notion of acyclic hyperconsistency width and the generalized hypertree width.

Proposition 3. *There is a class* $\{Q_n \mid n > 0\}$ *of queries with* $\frac{\mathsf{ghw}(Q_n)}{\mathsf{ahcw}(Q_n)} \to +\infty$.

Proof. Let $(Q_n)_{n\in\mathbb{N}}$ be a class of queries such that $\mathsf{ghw}(Q_n) \to +\infty$. For each query Q_n, we select a variable $X \in \mathsf{var}(Q_n)$, and we define the novel query $Q'_n = Q_n \wedge \bigwedge\{r_n(X, \ldots, X) \mid r_n(S_n) \in \mathsf{atoms}(Q_n)\}$. Then, we can check that $\mathsf{ghw}(Q'_n) = \mathsf{ghw}(Q_n)$, $\mathsf{ghw}(\mathsf{CORE}(Q'_n)) = 1$, and $\mathsf{ahcw}(Q_n) = 1$. □

4 Complexity Results on No-Promise Problems

In this section, we complete our investigation by focusing on the description of incomplete algorithms for solving CSPs. The main aim is to extend the tractability result in Corollary 1 to the case of no-promise problems, i.e., when a generalized hypetree decomposition is not given to hand. Actually, besides the decision problem of checking whether $Q[\mathcal{D}]$ is empty for an instance $(Q, \mathcal{D})$, we will also consider the related computation problem $\mathsf{SAT}_{\exists}$ of computing an element in $Q[\mathcal{D}]$, and the problem $\mathsf{SAT}_{\forall}$ of computing the whole set $Q[\mathcal{D}]$.

Moreover, rather than focusing on some specific structural decomposition method, the algorithms we shall present correctly behave on any class of queries which is *stable*. Formally, a class $\mathcal{C}$ of queries is stable if, for each $Q \in \mathcal{C}$, for each unary constraint r not in Q, and for each variable X, we have that $Q \wedge r(X)$ is in $\mathcal{C}$.

Theorem 5. *Let* $\mathcal{C}$ *be a class of queries which is stable. Let* $\mathcal{A}$ *be an algorithm solving in* $O(\mathbf{q}^{a}\mathbf{r}^{b})$ *the problem:*

> **Input**: a CSP instance $(Q, \mathcal{D})$;
> **Output**: No only if $Q[\mathcal{D}] = \emptyset$, and
> IDon′tKnow only if $Q[\mathcal{D}] \neq \emptyset$ or $Q \notin \mathcal{C}$

Then, we can decide:

$\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}]$ *in* $O(\mathbf{q}^{a}\mathbf{r}^{b})$;	$\mathsf{SAT}^{\mathsf{np}}_{\emptyset}[\mathcal{C}]$ *in* $O(\mathbf{q}^{a+1}\mathbf{r}^{b+1})$;
$\mathsf{SAT}^{\mathsf{p}}_{\exists}[\mathcal{C}]$ *in* $O(\mathbf{q}^{a+1}\mathbf{r}^{b+1})$;	$\mathsf{SAT}^{\mathsf{np}}_{\exists}[\mathcal{C}]$ *in* $O(\mathbf{q}^{a+1}\mathbf{r}^{b+1})$;
$\mathsf{SAT}^{\mathsf{p}}_{\forall}[\mathcal{C}]$ *in* $O(\mathbf{q}^{a+1}\mathbf{r}^{b+1}\mathbf{o})$;	$\mathsf{SAT}^{\mathsf{np}}_{\forall}[\mathcal{C}]$ *in* $O(\mathbf{q}^{a+2}\mathbf{r}^{b+2}\mathbf{o})$.

Input: a CSP instance $(Q, \mathcal{D})$;
Output: a solution $\theta \in Q[\mathcal{D}]$ only if $Q[\mathcal{D}] \neq \emptyset$, and
No only if $Q[\mathcal{D}] = \emptyset$, and
IDon'tKnow only if $Q \notin \mathcal{C}$

Given: An incomplete algorithm $\mathcal{A}$ solving the following :
Input: a CSP instance $(Q', \mathcal{D}')$;
Output: No only if $Q'[\mathcal{D}'] = \emptyset$, and
IDon'tKnow only if $Q'[\mathcal{D}'] \neq \emptyset$ or $Q' \notin \mathcal{C}$

begin
if $\mathcal{A}(Q, \mathcal{D})$ returns No **then return** No;
let $\theta = \emptyset$;
for each variable $X \in \mathsf{var}(Q)$ **do**
let r_X be a fresh predicate symbol of arity 1;
let $A \in \mathsf{atoms}(Q)$ be such that $x \in \mathsf{var}(A)$;
let $Q_X = Q \wedge r_X(X)$;
let $l_X = \Pi_{\{X\}}(A[\mathcal{D}])$;
for each $e \in l_X$ **do**
let $\mathcal{D}_{X,e} = \mathcal{D} \cup \{r_X(e)\}$;
if $\forall e \in l_X \;\; \mathcal{A}(Q_X, \mathcal{D}_{X,e})$ returns No **then**
stop and **return** IDon'tKnow:
else
let e be s.t. $\mathcal{A}(Q_X, \mathcal{D}_{X,e})$ returns IDon'tKnow;
$(Q, \mathcal{D}) := (Q_X, \mathcal{D}_{X,e})$;
let $\theta(X) = e$;
if $\theta \notin Q[\mathcal{D}]$ **then return** IDon'tKnow;
else return θ;
end.

Fig. 3. Algorithm $\mathcal{A}^{\mathsf{np}}_{\exists}$ for $\mathsf{SAT}^{\mathsf{np}}_{\exists}[\mathcal{C}]$ (cf. [3])

Proof. ($\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}]$). We can solve $\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}]$ by means of an algorithm $\mathcal{A}^{\mathsf{p}}_{\emptyset}$ that returns No iff $\mathcal{A}(Q, \mathcal{D})$ returns No, and returns Yes iff $\mathcal{A}(Q, \mathcal{D})$ returns IDon'tKnow.

($\mathsf{SAT}^{\mathsf{np}}_{\exists}[\mathcal{C}]$). Consider the algorithm $\mathcal{A}^{\mathsf{np}}_{\exists}$ reported in Figure 3, that is basically the algorithm discussed in [3] for the class of queries having bounded generalized hypertree width. This algorithm only returns No when $\mathcal{A}(Q, \mathcal{D})$ returns No (in which case, $Q[\mathcal{D}] = \emptyset$). Otherwise, i.e., when $Q[D] \neq \emptyset$, the algorithm starts computing a solution θ by iteratively fixing a value $e \in l_X$, for each variable X such that $Q_X[D_{X,e}] \neq \emptyset$. Note that during the computation, the query Q is updated by adding a unary constraint precisely fixing the value for X. Hence, provided that the class $\mathcal{C}$ is stable, any such modification preserves the membership of Q in $\mathcal{C}$. Finally, the algorithm returns IDon'tKnow iff θ is not a solution. Note that when $Q[D] \neq \emptyset$, there always exists at least one value $e \in l_X$, for each variable X such that $Q_X[D_{X,e}] \neq \emptyset$. For the complexity of $\mathcal{A}^{\mathsf{np}}_{\exists}$, note that the size of Q (resp. $\mathcal{D}$) always remains smaller than $2\mathbf{q}$ (resp. $2\mathbf{r}$) and therefore

Input: a CSP instance $(Q, \mathcal{D})$, a set $\mathcal{O} \subseteq \mathsf{var}(Q)$;
Output: $\Pi_{\mathcal{O}}(Q[\mathcal{D}])$, and
IDon'tKnow only if $Q \notin \mathcal{C}$;

Given: An algorithm $\mathcal{A}^{\mathsf{np}}_{\emptyset}$ for $\mathsf{SAT}^{\mathsf{np}}_{\emptyset}[\mathcal{C}]$

begin
  if $\mathcal{A}^{\mathsf{np}}_{\emptyset}(Q, \mathcal{D})$ returns IDon'tKnow **then**
    stop and **return** IDon'tKnow;
  if $\mathcal{A}^{\mathsf{np}}_{\emptyset}(Q, \mathcal{D})$ returns No **then**
    stop and **return** $\emptyset$;
  if $\mathcal{O} = \emptyset$ **then**
    stop and **return** $\emptyset$;
  else
    choose a variable X in $\mathcal{O}$ and **let** $\mathcal{O}' = \mathcal{O} - \{X\}$;
    let r_X be a fresh predicate symbol of arity 1;
    let $A \in \mathsf{atoms}(Q)$ be such that $X \in \mathsf{var}(A)$;
    let $Q_X = Q \wedge r_X(X)$, $l_X = \Pi_{\{X\}}A[\mathcal{D}]$, and $\mathcal{R} = \emptyset$;
    for each $e \in l_X$
      let $\mathcal{D}_{X,e} := \mathcal{D} \cup \{r_X(e)\}$
      if $\mathcal{A}^{\mathsf{np}}_{\emptyset}(Q_X, \mathcal{D}_{X,e})$ returns IDon'tKnow **then**
        stop and **return** IDon'tKnow
      if $\mathcal{A}^{\mathsf{np}}_{\emptyset}(Q_X, \mathcal{D}_{X,e})$ returns Yes **then**
        for each $h' \in \mathcal{A}^{\mathsf{np}}_{\forall}(Q_X, \mathcal{D}_{X,e}, \mathcal{O}')$ **do**
          let $h'(X) := e$;
          $\mathcal{R} := \mathcal{R} \cup \{h'\}$;
    return $\mathcal{R}$;
end.

Fig. 4. Algorithm $\mathcal{A}^{\mathsf{np}}_{\forall}$ for $\mathsf{SAT}^{\mathsf{np}}_{\forall}[\mathcal{C}]$

each call to the algorithm $\mathcal{A}$ costs $O(\mathbf{q}^a\mathbf{r}^b)$. The number of such calls is bounded by $\Sigma\{|l_X|, X \in \mathsf{var}(Q)\} = O(\mathbf{qr})$. Therefore, $\mathcal{A}^{\mathsf{np}}_{\exists}$ terminates in time $O(\mathbf{q}^{a+1}\mathbf{r}^{a+1})$.

($\mathsf{SAT}^{\mathsf{np}}_{\emptyset}[\mathcal{C}]$ and $\mathsf{SAT}^{\mathsf{p}}_{\exists}[\mathcal{C}]$). We can use the algorithm $\mathcal{A}^{\mathsf{np}}_{\emptyset}$ which simply returns Yes iff $\mathcal{A}^{\mathsf{np}}_{\exists}$ returns one solution $\theta \in Q[D]$, and the same output of $\mathcal{A}^{\mathsf{np}}_{\exists}$, if no solution is computed. Clearly enough, $\mathcal{A}^{\mathsf{np}}_{\emptyset}$ is also an algorithm for $\mathsf{SAT}^{\mathsf{p}}_{\exists}[\mathcal{C}]$.

($\mathsf{SAT}^{\mathsf{np}}_{\forall}[\mathcal{C}]$). Armed with $\mathcal{A}^{\mathsf{np}}_{\emptyset}$, we present in Figure 4 the recursive algorithm $\mathcal{A}^{\mathsf{np}}_{\forall}$ for $\mathsf{SAT}^{\mathsf{np}}_{\forall}[\mathcal{C}]$, taking as extra-argument a set of variables $\mathcal{O}$ (for $\mathcal{O} = \mathsf{var}(Q)$, the whole relation $Q[\mathcal{D}]$ is computed). The algorithm $\mathcal{A}^{\mathsf{np}}_{\forall}$ only returns a relation $\mathcal{R} = \Pi_{\mathcal{O}}(Q[\mathcal{D}])$ when every call to $\mathcal{A}^{\mathsf{np}}_{\emptyset}(Q_X, \mathcal{D}_{X,e})$ have returned either Yes or No, in which case we know by correctness of $\mathcal{A}^{\mathsf{np}}_{\emptyset}$ that we didn't forget any (or add any extra) tuple in $\Pi_{\mathcal{O}}(Q[\mathcal{D}])$. Therefore, $\mathcal{A}^{\mathsf{np}}_{\forall}$ only returns IDon'tKnow when $Q \notin \mathcal{C}$. Each call to $\mathcal{A}^{\mathsf{np}}_{\emptyset}$ costs $O(\mathbf{q}^{a+1}\mathbf{r}^{b+1})$. Indeed, $\mathcal{A}^{\mathsf{np}}_{\forall}$ first executes $\mathbf{r}$ calls to the algorithm $\mathcal{A}^{\mathsf{np}}_{\emptyset}$ for the first variable $X \in \mathcal{O}$, and then applies $\mathbf{r}$ more calls each time that the case "$\mathcal{A}^{\mathsf{np}}_{\emptyset}(Q_x, \mathcal{D}_{x,e})$ returns Yes" appears. This may happen $O(|\mathcal{O}| \times \mathbf{o}) = O(\mathbf{qo})$ times at most. Therefore, $\mathcal{A}^{\mathsf{np}}_{\forall}$ terminates in time $O(\mathbf{qro}) \times O(\mathbf{q}^{a+1}\mathbf{r}^{b+1}) = O(\mathbf{q}^{a+2}\mathbf{r}^{b+2}\mathbf{o})$.

($\mathsf{SAT}^{\mathsf{p}}_{\forall}[\mathcal{C}]$). The problem can be solved in time $O(\mathbf{q}^{a+2}\mathbf{r}^{b+2}\mathbf{o})$, by a slight variation of the algorithm $\mathcal{A}^{\mathsf{np}}_{\forall}$, where $\mathcal{A}^{\mathsf{p}}_{\emptyset}$ is used instead of $\mathcal{A}^{\mathsf{np}}_{\emptyset}$. The complexity is then $O(\mathbf{q}^{a+1}\mathbf{r}^{b+1}\mathbf{o})$. □

As an example application of the above theorem, we next show that the class of queries having bounded generalized hypertree width is stable, so that bounds in Figure 1 are correct.

Proposition 4. *For every k, the class $\mathcal{C}(\mathsf{ghw}, k)$ is stable.*

Proof. Let $J = (\mathcal{G}, \chi, \lambda)$ be a k-width generalized hypergraph decomposition of Q, let r be a unary constraint not in Q, let X be a variable, and let $Q' = Q \wedge r(X)$. We build a hypergraph decomposition J' by adding a node v' to $V(J)$ such that $\chi(v') = \{X\}$ and $\lambda(v') = \{r(X)\}$, and an edge (v, v') to $E(J)$, where v is an arbitrarily chosen node such that $X \in \chi(v)$. Then, J' is a k-width generalized hypertree decomposition. □

5 Discussion and Conclusion

We have characterized the computational complexity of solving the promise and no-promise problems on the class of queries having bounded generalized hypertree width, by means of consistency-based algorithms. On the one hand, we have provided precise bounds for the technique of [3], which firstly suggested the idea of no-promise algorithms for $\mathcal{C}(\mathsf{ghw}, k)$. On the other hand, the notion of hyperconsistency width being parameterized by the underlying decomposition allowed us to obtain improved complexity bounds. These bounds have been established for both the case of promise and no-promise problems, and for the problems SAT, $\mathsf{SAT}_{\exists}$, and $\mathsf{SAT}_{\forall}$.

It turned out that instances of bounded generalized hypertree width can be solved with the enhanced no-promise approach by means of an algorithm whose scaling is the same as current evaluation algorithms for instances of bounded hypertree width (see [8], for the description of these algorithms), but with the advantage of isolating a larger class of instances (recall that $\mathcal{C}(\mathsf{ghw}, k) \supset \mathcal{C}(\mathsf{hw}, k)$). This is relevant in the light that $\mathcal{C}(\mathsf{hw}, k)$ is one of the largest classes of tractable CSPs that is known to be also efficiently recognizable (see, e.g., [2,7]).

Several research questions naturally arise from this contribution. First, assessing the *exact* relationship among the classes $\mathcal{C}(\mathsf{ghw}, k)$, $\mathcal{C}(\mathsf{hcw}, k)$, and $\mathcal{C}(\mathsf{ahcw}, k)$ is still an open problem. In particular, given that one may easily observe that for each query Q, it is the case that $\mathsf{hcw}(Q)$=$\mathsf{hcw}(\mathsf{CORE}(Q))$, it is natural to ask whether $\mathsf{ghw}(\mathsf{CORE}(Q))$ coincides with $\mathsf{hcw}(Q)$ (and $\mathsf{ahcw}(Q)$).

Second, the (improved) no-promise algorithm scales as $O(\mathbf{q}^{2k})$ in the size of the query. Interestingly, this is also the best upper bound known for computing a hypertree decomposition. Whether it is possible to improve on this exponent appears therefore a crucial question for a deeper and unifying understating of these notions.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley, Reading (1995)
2. Adler, I., Gottlob, G., Grohe, M.: Hypertree-width and related hypergraph invariants. In: Proc. of EUROCOMB 2005, pp. 5–10 (2005)
3. Chen, H., Dalmau, V.: Beyond hypertree width: Decomposition methods without decompositions. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 167–181. Springer, Heidelberg (2005)
4. Dechter, R.: From local to global consistency. Artificial Intelligence 55(1), 87–108 (1992)
5. Dechter, R.: Constraint Processing. Morgan Kaufmann, San Francisco (2003)
6. Flum, J., Frick, M., Grohe, M.: Query evaluation via tree-decompositions. Journal of the ACM 49(6), 716–752 (2002)
7. Gottlob, G., Leone, N., Scarcello, F.: A comparison of structural csp decomposition methods. Artificial Intelligence 124(2), 243–282 (2000)
8. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decompositions and tractable queries. Journal of Computer and System Sciences 64(3), 579–627 (2002)
9. Gottlob, G., Miklos, Z., Schwentick, T.: Generalized hypertree decompositions: Np-hardness and tractable variants. In: Proc. of PODS 2007 (2007)
10. Greco, G., Scarcello, F.: Tree projections: Hypergraph games and minimality. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 736–747. Springer, Heidelberg (2008)
11. Gyssens, M., Jeavons, P., Cohen, D.A.: Decomposing constraint satisfaction problems using database techniques. Artificial Intelligence 66(1), 57–89 (1994)
12. Hell, P., Nesetril, J.: The core of a graph. Discrete Math. 109(1-3), 117–126 (1992)
13. Kolaitis, P.G., Vardi, M.Y.: Conjunctive-query containment and constraint satisfaction. Journal of Computer and System Sciences 61(2), 302–332 (1998)
14. Kolaitis, P.G., Vardi, M.Y.: A game-theoretic approach to constraint satisfaction. In: Proc. of AAAI 2000, pp. 175–181 (2000)
15. Pearson, J., Jeavons, P.: A survey of tractable constraint satisfaction problems. Technical Report CSD-TR-97-15, Royal Holloway, University of London (July 1997)
16. Yannakakis, M.: Algorithms for acyclic database schemes. In: Proc. of VLDB 1981, pp. 82–94 (1981)

Local Search Heuristics for the Multidimensional Assignment Problem

G. Gutin and D. Karapetyan

Department of Computer Science,
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK
G.Gutin@rhul.ac.uk
Daniel.Karapetyan@gmail.com

Abstract. The Multidimensional Assignment Problem (MAP) (abbreviated s-AP in the case of s dimensions) is an extension of the well-known assignment problem. The most studied case of MAP is 3-AP, though the problems with larger values of s have also a number of applications. We consider several known and new MAP local search heuristics for MAP as well as their combinations. Computational experiments with three instance families are provided and discussed. As a result, we select dominating local search heuristics. One of the most interesting conclusions is that combination of two heuristics may yield a superior heuristic with respect to both solution quality and the running time.

1 Introduction

The Multidimensional Assignment Problem (MAP) (abbreviated s-AP in the case of s dimensions) is a well-known optimization problem with a host of applications (see, e.g., [3,5,6] for 'classic' applications and [4,15] for recent applications in solving systems of polynomial equations and centralized multisensor multitarget tracking). In fact, several applications described in [4,5,15] naturally require the use of s-AP for values of s larger than 3.

MAP is an extension of a well-known Assignment Problem (AP) which is exactly two dimensional case of MAP. While AP can be solved in a polynomial time [12], s-AP for every $s \geq 3$ is NP-hard [7].

For a fixed $s \geq 2$, the s-AP is stated as follows. Let $X_1 = X_2 = \ldots = X_s = \{1, 2, \ldots, n\}$. We will consider only vectors that belong to the Cartesian product $X = X_1 \times X_2 \times \ldots \times X_s$. Each vector $e \in X$ is assigned a non-negative weight $w(e)$. For a vector $e \in X$, the component e_j denotes its jth coordinate, i.e., $e_j \in X_j$. A collection A of $t \leq n$ vectors $e^1, e^2, \ldots, e^t$ is a *(feasible) partial assignment* if $e^i_j \neq e^k_j$ holds for each $i \neq k$ and $j \in \{1, 2, \ldots, s\}$. The *weight* of a partial assignment A is $w(A) = \sum_{i=1}^{t} w(e^i)$. An *assignment* (or *full assignment*) is a partial assignment with n vectors. The objective of s-AP is to find an assignment of minimum weight.

We will sometimes use the term *x-assignment* for an assignment of weight x.

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 100–115, 2009.

2 Heuristics

In this section we discuss and compare several known and new MAP local search heuristics as well as their combinations.

2.1 Dimensionwise Variation

Dimensionwise Variation (DV) for 3-AP is introduced by Huang and Lim as a local search procedure for their Genetic Algorithm [10]. We propose an extension of this heuristic for s-AP. Fixing all but one dimensions, it constructs a 2-AP, solves it, and modifies the unfixed dimension according to the solution of the 2-AP.

More formally, consider the current assignment $A = \{e^1, e^2, \ldots, e^n\}$. On each step of the heuristic all the dimensions but some kth are fixed and an $n \times n$ matrix $M_{i,j} = w(v^{i,j})$ is generated, where

$$v_d^{i,j} = \begin{cases} e_d^i & \text{if } d \neq k \\ e_d^j & \text{if } d = k \end{cases} \qquad \text{for } d = 1, 2, \ldots, s.$$

Let permutation ρ be a solution of the corresponding 2-AP. If ρ is not an identity permutation, the heuristic changes the s-AP assignment in the following way:

$$e_d^i = \begin{cases} e_d^i & \text{if } d \neq k \\ e_d^{\rho(i)} & \text{if } d = k \end{cases} \qquad \text{for } i = 1, 2, \ldots, n \text{ and } d = 1, 2, \ldots, s.$$

DV performs such steps sequentially for $k = 1, 2, \ldots, s$. If any improvement was made by any of these steps, the whole procedure is repeated. The maximum number of iterations can be very large theoretically, however, our experiments show that the number of iterations does not correlate with the instance size and is actually between 3 and 7 in the most cases. For some limitation, we allow at most 10 iterations. This note is also true for all the heuristics considered in the paper.

The time complexity of one step of DV is $O(n^2 + n^3) = O(n^3)$ since the matrix M generation requires n^2 operations, and the 2-AP solver time complexity is $O(n^3)$. The time complexity of the whole heuristic is $O(n^3 \cdot s)$.

2.2 Multi-dimensionwise Variation

The *Multi-dimensionwise Variation* (MDV) heuristic is introduced in this paper. It is a modification of DV (see Subsection 2.1). While DV fixes all but one dimensions, our heuristic tries every combination of the fixed and unfixed dimensions subsets. A nonempty set of distinct dimensions $F \subsetneq \{1, 2, \ldots, s\}$ is selected on each step. The corresponding dimensions are fixed, while the others are varied. An $n \times n$ matrix $M_{i,j} = w(v^{i,j})$ is generated, where

$$v_d^{i,j} = \begin{cases} e_d^i & \text{if } d \in F \\ e_d^j & \text{if } d \notin F \end{cases} \qquad \text{for } d = 1, 2, \ldots, s.$$

Let permutation ρ be a solution of the corresponding 2-AP. If ρ is not an identity permutation, the heuristic changes the s-AP assignment in the following way:

$$e_d^i = \begin{cases} e_d^i & \text{if } d \in F \\ e_d^{\rho(i)} & \text{if } d \notin F \end{cases} \qquad \text{for } i = 1, 2, \ldots, n \text{ and } d = 1, 2, \ldots, s.$$

There are $2^s - 2$ distinct sets F of the fixed dimensions, but a half of them may be omitted since there is no difference whether to fix the selected dimensions and to vary the others, or to vary the selected dimensions and to fix the others. So, every iteration of the heuristic tries $2^{s-1} - 1$ distinct sets F in the following order: on the ith step, where $i = 1, 2, \ldots, 2^{s-1} - 1$, the set F contains a value j if the jth bit of the s-digit binary presentation of the number i is zero (for example, if $s = 5$ and $i = 5_{10} = 00101_2$, then $F = \{1, 3\}$). The algorithm repeats until no improvement is obtained during an iteration, but at most 10 times (see Subsection 2.1). The time complexity of MDV is $O(n^3 \cdot 2^s)$.

2.3 2-Opt

The *2-Opt* local search heuristic was first introduced by Balas and Saltzman [3] for 3-AP as a *pairwise interchange heuristic*. We propose an extension of this heuristic for s-AP.

2-opt proceeds as follows. Remove two vectors p, q from the current assignment and replace them with the best recombination p', q', i.e., $p'_1 = p_1$, $p'_d \in \{p_d, q_d\}$ for every $1 < d \leq s$, and $q'_d = \{p_d, q_d\} \setminus \{p'_d\}$ for every $1 \leq d \leq s$, to minimize $w(p') + w(q')$.

There are 2^{s-1} recombinations p', q' for every vector pair p and q. Thus, the complexity of one 2-opt iteration is $O(n^2 \cdot 2^s)$. In fact, the heuristic works repeatedly until no improvement takes place during an iteration, but at most 10 times (see Subsection 2.1). The time complexity of 2-opt is $O(n^2 \cdot 2^s)$.

As an improvement, we do not consider a pair of vectors p and q if either $w(p) = w(q) = \min_{e \in X} w(e)$, or both p and q have remained unchanged during the previous iteration.

2.4 3-Opt

3-Opt local search is like 2-opt but we take three vectors each time and try every possible recombination of them, i.e., $3!^{s-1} - 1$ recombinations. The time complexity of 3-opt is $O(n^3 \cdot 6^s)$.

As we did for 2-opt, we do not consider the vector triples (p, q, r) if either all the vectors p, q, and r have remained unchanged during the previous iteration, or $w(p) = w(q) = w(r) = \min_{e \in X} w(e)$.

2.5 Variable Opt

The *Variable Opt* (v-opt) was first introduced by Balas and Saltzman as a *Variable Depth Interchange* (VDI) [3] for 3-AP. We provide here a natural extension of the VDI heuristic for the s-dimensional case, $s \geq 3$ and, then, we improve this extension

like MDV improves DV (see Subsections 2.1 and 2.2). Our computational experiments showed that the improved version of v-opt is superior to the natural extension of VDI in the solution quality at a reasonable increase in running time. Thus, v-opt refers to the improved version of the heuristic in what follows unless otherwise specified.

We explain the heuristic in a different way to the description provided in [3], however, both versions of our algorithm are equal to the VDI in case of $s = 3$. This fact was also checked by reproduction of the computational evaluation results reported in [3].

For a set $F \subset \{1, 2, \ldots, s\}$, let the *$F$-interchange between vectors v and w* be a vector $u(v, w, F)$ equal to v in all the coordinates that are not listed in F and equal to w in all the coordinates listed in F:

$$u(v, w, F)_j = \begin{cases} v_j & \text{if } j \notin F \\ w_j & \text{if } j \in F \end{cases} \qquad \text{for } 1 \le j \le s.$$

The difference between the two versions of v-opt is only in the $U(v, w)$ definition. For the natural extension of VDI, let $U(v, w)$ be a set of all the possible F-interchanges between the vectors v and w such that $|F| = 1$:

$$U(v, w) = \bigcup_{|F|=1} \big\{u(v, w, F)\big\} = \bigcup_{1 \le k \le s} \big\{u(v, w, \{k\})\big\}.$$

For the improved version of v-opt, let $U(v, w)$ be a set of all the possible F-interchanges between the vectors v and w such that $|F| \le s/2$:

$$U(v, w) = \bigcup_{|F| \le s/2} \big\{u(v, w, F)\big\}.$$

The constraint $|F| \le s/2$ guarantees that at least half of the coordinates of an F-interchange are equal to the first vector coordinates. The computational experiments show that removing this constraint increases the running time and decreases the average solution quality.

Let vector $u_{\min}(v, w)$ be the minimum weight interchange between v and w:

$$u_{\min}(v, w) = \operatorname*{argmin}_{u \in U(v,w)} w(u).$$

Let $A = e^1, e^2, \ldots, e^n$ be an initial assignment. The heuristic repeats the following algorithm at most 10 times until no improvement is found (the VDI algorithm has no limitation for the iteration number, however this limitation does not affects the heuristic quality in our experiments).

1. For every $c = 1, 2, \ldots, n$ do the rest of the algorithm.
2. Initialize the *total gain* $G = 0$, the *best assignment* $A_{\text{best}} = A$, and a set of available vector indices $I = \{1, 2, \ldots, n\} \setminus \{c\}$.
3. Set $m = \operatorname{argmin}_{i \in I} w\big(u_{\min}(e^c, e^i)\big)$, $v = u_{\min}(e^c, e^m)$, and $\overline{v}_j = \{e^c_j, e^m_j\} \setminus \{v_j\}$ for every $1 \le j \le s$. Now $v \in U(e^c, e^m)$ is the minimum weight interchange of e^c with some other vector in the assignment, and $\overline{v}$ is an addition to v.

4. Set $G = G + w(e^c) - w(v)$. If now $G \leq 0$, set $A = A_{\text{best}}$ and stop.
5. Replace e^m with v and e^c with $\overline{v}$. Mark e^m as an unavailable for the further interchanges vector: $I = I \setminus \{m\}$.
6. If $w(A) < w(A_{\text{best}})$, record the new assignment as the best one: $A_{\text{best}} = A$.
7. Repeat from Step 3 while the total gain is positive (see Step 4) and $I \neq \emptyset$. Note that e^c now refers to $\overline{v}$.

The time complexity of the v-opt algorithm is $O(n^3 \cdot 2^s)$.

The time complexity of the natural extension of VDI is $O(n^3 \cdot s)$, and the computation experiments also show a significant difference between the running times of the improved and the natural extensions. However, the solution quality of the natural extension for $s \geq 7$ is quite poor, while for the smaller values of s it produces solutions similar to or even worse than MDV solutions at the cost of much larger running times.

2.6 Combined Heuristics

All the heuristics considered above can be split in two groups: the *vectorwise heuristics* (2-opt, 3-opt, and v-opt), which can modify all the coordinates of a subset of the assignment vectors on each step, and *dimensionwise heuristics* (DV and MDV), which can modify all the vectors of the assignment but just in a subset of the coordinates on every step. It is likely that a combination of the heuristics from different groups can lead to a superior heuristic. Next we consider all the six possible combinations of the vectorwise and dimensionwise heuristics and report the computational results for the most successful of them in Section 4.

All the combined heuristics start with a trivial assignment $e^i = (i, i, \ldots, i)$. Then a dimensionwise heuristic and a vectorwise heuristic are applied sequentially. The optimization procedure repeats until one of these heuristics fails to improve the assignment but at most 10 times (see Subsection 2.1). Note that the dimensionwise heuristic is applied first in all the combinations since the dimensionwise heuristics are more efficient, i.e., they usually produce a better solution in a smaller time and, thus, they suit better the first step that is the hardest step for the combined heuristic.

After evaluation and analysis of the computational experiment results, we have excluded three of the six combinations due to different reasons.

- The combination MDV + 2-opt is very similar to just MDV in the solution quality but the running time is significantly larger.
- The combination DV + 3-opt is mostly worse than MDV + 3-opt with respect to both solution quality and the running time.
- The combination DV + v-opt is mostly worse than MDV + v-opt with respect to both solution quality and the running time.

In what follows, we consider DV + 2-opt (DV2), MDV + 3-opt (MDV3), and MDV + v-opt (MDVV) only.

2.7 Other Algorithms

Here we observe the most successful MAP algorithms presented in literature.

– A Memetic Algorithm by Huang and Lim [10] (Local Search Genetic Algorithm, LSGA) can be considered as a state-of-the art heuristic for 3-AP. However, LSGA is intended for quality-critical applications and, thus, it cannot be compared with the local search, which are much faster. Observe that LSGA takes more than a second on our evaluation platform to solve a Random instance (see Subsection 3.1) of size 26 while the slowest heuristic considered in the paper takes less than a second to solve a Random instance of size 100.
– A Greedy Randomized Adaptive Search Procedure (GRASP) for 3-AP by Aiex et al. is presented in [1]. According to [10], LSGA outperforms GRASP with respect to both the solution quality and the running time.
– Crama and Spieksma's approximation heuristic [6] is designed for a special case of the Composite instance set (see Subsection 3.2), called $T\triangle$ in [6]. We have conducted the computational experiments for $T\triangle$ instances and concluded that even the fast heuristic DV clearly outperforms Crama and Spieksma's heuristic with respect to the solution quality (the running time is not provided in [6]).

3 Test Bed

In this section we discuss instance families used for experimental evaluation of the local search heuristics.

3.1 Random Instance Family

In *Random Instance Family* the weight assigned to a vector is a random uniformly distributed integer value in the interval $[a, b-1]$. Random instances were used in [1,3,14] and some others.

Since the instances are random and quite large, it is possible to estimate the average solution value for the Random Instance Family. In fact, we prove that it is very likely that every Random instance we consider in our experiments has an an-assignment, i.e., a minimal possible assignment (note that a minimal assignment includes n vectors of weight a).

Let α be the number of assignments of weight an and let $c = b - a$. We would like to have an upper bound on the probability $\Pr(\alpha = 0)$. Such an upper bound is given in the following theorem whose proof is based on the Extended Jansen Inequality (see Theorem 8.1.2 of [2]).

Theorem 1. *For values of n such that $n \geq 3$ and*

$$\left(\frac{n-1}{e}\right)^{s-1} \geq c \cdot 2^{\frac{1}{n-1}}, \tag{1}$$

we have $\Pr(\alpha = 0) \leq e^{-\frac{1}{2\sigma}}$, where $\sigma = \sum_{k=1}^{n-2} \frac{\binom{n}{k} \cdot c^k}{[n \cdot (n-1) \cdots (n-k+1)]^{s-1}}$.

Proof. Let $[n] = \{1, 2, \ldots, n\}$. Let t be the number of all feasible assignments and let A be an arbitrary assignment consisting of vectors $e^1, e^2, \ldots, e^n$ such that $e^i_1 = i$ for each $i \in [n]$. There are $n!$ possibilities to choose the jth coordinate of all vectors in A for each $j = 2, 3, \ldots, n$ and, thus, $t = (n!)^{s-1}$.

Let R be the set of vectors in X of weight a and let $\{A_1, A_2, \ldots, A_t\}$ be the set of all assignments. Let B_i be the event $\{A_i \subset R\}$ for each $i \in [n]$. Let $\mu = \sum_{i=1}^{t} \Pr(B_i)$ and $\Delta = \sum_{i \sim j} \Pr(B_i \cap B_j)$, where $i \sim j$ if $i \neq j$ and $A_i \cap A_j \neq \emptyset$ and the sum for Δ is taken over all ordered pairs (B_i, B_j) with $i \sim j$.

By the Extended Jansen Inequality,

$$\Pr(\alpha = 0) \leq e^{-\frac{\mu^2}{2\Delta}} \tag{2}$$

provided $\Delta \geq \mu$. We will compute μ and estimate Δ to apply (2) and to show $\Delta \geq \mu$. It is easy to see that $\mu = \frac{t}{c^n}$.

Now we will estimate Δ. Let $A_i \cap A_j = K$, $k = |K|$ and $i \neq j$. Thus, we have

$$\Pr(B_i \cap B_j) = \Pr(K \subset R) \cdot \Pr(A_i \setminus K \subset R) \cdot \Pr(A_j \setminus K \subset R) = \frac{1}{c^k} \left(\frac{1}{c^{n-k}}\right)^2 = \frac{1}{c^{2n-k}}.$$

Let $(f^1, f^2, \ldots, f^n)$ be an assignment with $f^i_1 = i$ for every $i \in [n]$ and consider the following two sets of assignments. Let

$$P(k) = \{(e^1, e^2, \ldots, e^n) : \forall i \in [n]\ (e^i_1 = i) \text{ and } \forall j \in [k]\ (e^j = f^j)\}$$

and let $Q(n-k) = \{(e^1, e^2, \ldots, e^n) : \forall i \in [n]\ (e^i_1 = i) \text{ and } \forall j \in [n-k]\ (e^{k+j} \neq f^{k+j})\}$. Let $h(n,k) = |P(k) \cap Q(n-k)|$. Clearly, $h(n,k) \leq |P(k)| = \big((n-k)!\big)^{s-1}$. Observe that

$$h(n,k) \geq |P(k)| - (n-k)|P(k+1)| = L(n,k,s),$$

where $L(n,k,s) = \big((n-k)!\big)^{s-1} - (n-k) \cdot \big((n-k-1)!\big)^{s-1}$.

Let $g(n,k)$ be the number of ordered pairs (A_i, A_j) such that $|A_i \cap A_j| = k$. Observe that $g(n,k) = t \cdot \binom{n}{k} \cdot h(n,k)$ and, thus, $t \cdot \binom{n}{k} \cdot L(n,k,s) \leq g(n,k) \leq t \cdot \binom{n}{k} \cdot \big((n-k)!\big)^{s-1}$.

Observe that $\Delta = \sum_{k=1}^{n-2} \sum_{|A_i \cap A_j| = k} \Pr(B_i \cap B_j) = \sum_{k=1}^{n-2} g(n,k) \cdot c^{k-2n}$. Thus,

$$\frac{(n!)^{s-1}}{c^{2n}} \cdot \sum_{k=1}^{n-2} \binom{n}{k} \cdot c^k \cdot L(n,k,s) \leq \Delta \leq \frac{(n!)^{s-1}}{c^{2n}} \sum_{k=1}^{n-2} \binom{n}{k} \cdot c^k \cdot \big((n-k)!\big)^{s-1} \tag{3}$$

Now $\Pr(\alpha = 0) \leq e^{-\frac{1}{2\sigma}}$ follows from (2) by substituting μ with $\frac{(n!)^{s-1}}{c^n}$ and Δ with its upper bound in (3). It remains to prove that $\Delta \geq \mu$. Since $n \geq 3$, $L(n,1,s) \geq \frac{1}{2}\big((n-1)!\big)^{s-1}$. By the lower bound for Δ in (3), we have $\Delta \geq \frac{(n!)^{s-1}}{c^{2n-1}} \cdot L(n,1,k)$. Therefore, $\frac{\Delta}{\mu} \geq \frac{0.5((n-1)!)^{s-1}}{c^{n-1}}$. Now using the inequality $(n-1)! > (\frac{n-1}{e})^{n-1}$, we conclude that $\frac{\Delta}{\mu} \geq 1$ provided (1) holds. □

The lower bounds of $\Pr(\alpha > 0)$ for different values of s and n and for $b - a = 100$, are reported below.

$s = 4$		$s = 5$		$s = 6$		$s = 7$	
n	$\Pr(\alpha > 0)$	n	$\Pr(\alpha > 0)$	n	$\Pr(\alpha > 0)$	n	$\Pr(\alpha > 0)$
15	0.575	10	0.991	8	1.000	7	1.000
20	0.823	11	0.998				
25	0.943	12	1.000				
30	0.986						
35	0.997						
40	1.000						

One can see that a 4-AP Random instance has an (an)-assignment with probability very close to 1 if $n \geq 40$; a 5-AP instance has an (an)-assignment with probability very close to 1 for $n \geq 12$, etc.; so, the optimal solutions for all the Random instances used in our experiments (see Section 4) are very likely to be of weight an. For $s = 3$ Theorem 1 does not provide a good upper bound, however we are able to use the results from Table II in [3] instead. Balas and Saltzman report that in their experiments the average optimal solution of 3-AP for Random instances reduces very fast and has a small value even for $n = 26$. Since the smallest Random instance we use in our experiments has size $n = 100$, we assume that all 3-AP Random instances considered in our experiment is very likely to have an an-assignment.

Useful results can also be obtained from (11) in [8] that is an upper bound for the average optimal solution. Grundel, Oliveira and Pardalos [8] consider the same instance family except the weights of the vectors are real numbers uniformly distributed in the interval $[a, b]$. However the results from [8] can be extended to our discrete case. Let $w'(e)$ be a real weight of the vector e in a continuous instance. Consider a discrete instance with $w(e) = \lfloor w'(e) \rfloor$ (if $w'(e) = b$, set $w(e) = b - 1$). Note that the weight $w(e)$ is a uniformly distributed integer in the interval $[a, b-1]$. The optimal assignment weight of this instance is not larger than the optimal assignment weight of the continuous instance, thus, the upper bound for the average optimal solution for the discrete case is correct.

In fact, the upper bound $\bar{z}_u^*$ (see [8]) for the average optimal solution is not accurate enough. For example, $\bar{z}_u^* \approx an + 6.9$ for $s = 3$, $n = 100$ and $b - a = 100$, and $\bar{z}_u^* \approx an + 3.6$ for $s = 3$, $n = 200$ and $b - a = 100$, so it cannot be used for $s = 3$ in our experiments. The upper bound $\bar{z}_u^*$ gives a better approximation for larger values of s, e.g., $\bar{z}_u^* \approx an + 1.0$ for $s = 4$, $n = 40$ and $b - a = 100$, but Theorem 1 provides stronger results ($\Pr(\alpha > 0) \approx 1.000$ for this case).

3.2 Composite Instance Family

Composite Instance Family is a family of semi-random instances. They were introduced by Crama and Spieksma for 3-AP as a problem T [6]. We extend this family for s-AP.

Let d^1, d^2, …, d^s be $n \times n$ matrices of non-negative uniformly distributed random integers in the interval $[a, b-1]$. Let us consider a graph $G(X_1 \cup X_2 \cup \ldots \cup X_s,$

$(X_1 \times X_2) \cup (X_2 \times X_3) \cup \ldots \cup (X_{s-1} \times X_s) \cup (X_1 \times X_s)\big)$, where the weight of an edge $(i,j) \in X_k \times X_{k+1}$ is $d^k_{i,j}$ for $1 \leq k < s$ and the weight of an edge $(i,j) \in X_1 \times X_s$ is $d^s_{i,j}$. In this interpretation of s-AP, the objective is to find a set of n vertex-disjoint s-cycles $C \subset X_1 \times X_2 \times \ldots \times X_s$ such that the total weight of all edges covered by the cycles C is minimized.

In other words, $w(e) = d^1_{e_1,e_2} + d^2_{e_2,e_3} + \ldots + d^{s-1}_{e_{s-1},e_s} + d^s_{e_1,e_s}$.

3.3 GP Instance Family

GP Instance Family contains pseudo-random instances with predefined optimal solutions. GP instances are generated by an algorithm given by Grundel and Pardalos [9]. The generator is naturally designed for s-AP for arbitrary large values of s and n. However, the GP generator is relatively slow and, thus, it was impossible to experiment with large GP instances.

4 Experiment Results

The experiments were conducted for the following instances:

- Random instances with $a = 1$ and $b = 101$. According to Subsection 3.1, the optimal solutions of all the considered Random instances are very likely to be $an = n$.
- Composite instances with $a = 1$ and $b = 101$. Instead of the optimal solution value we use the best known solution value.
- GP instances with predefined optimal solutions.

The instance name consists of three parts: the number of dimensions s, the type of the instance ('gp' for GP, 'r' for Random, and 'c' for Composite), and the size n of the instance. We use 10 different instances of every combination of type and size in our experiments.

For Random and Composite instances we use standard Miscrosoft .NET random generator [13] which is based on the Donald E. Knuth's subtractive random number generator algorithm [11]. The GP generator is implemented in C++ programming language, and, thus, the standard Visual C++ random number generator is used instead. As a seed of the random number sequences for all the instance types we use the following number: $seed = s + n + i$, where i is an index of the instance of this type and size, $i \in \{1, 2, \ldots, 10\}$.

All the heuristics are implemented in Visual C++. The evaluation platform is based on an AMD Athlon 64 X2 3.0 GHz processor.

The results of the experiments of two different types are provided and discussed in Subsections 4.1 and 4.2. In the single run experiments, a heuristic starts from a trivial assignment and runs exactly once. In repetitive search experiments, a heuristic runs several times starting from a random assignment every time until the allowed time is elapsed.

4.1 Single Run Experiments

In the first type of the experiments we run every heuristic for every instance exactly once. The results reported in Tables 1 and 2 are averages for 10 experiments since every row of these tables corresponds to 10 instances of some fixed type and size but of different seed values. The average values for different instance families and numbers of dimensions are provided at the bottom of each table. The winner is underlined in every row.

The value of the solution error is calculated as $\frac{v-v_{\text{best}}}{v_{\text{best}}} \cdot 100\%$, where v is the obtained solution value and v_{best} is the optimal solution value (or the best one in case of Composite instances, see above).

At first, we will compare only the single heuristics results (see Table 1). As regards the solution quality, 3-opt, v-opt, and MDV achieve the best results. Perhaps since 3-opt tries more different vectors than the other single heuristics, it achieves the best results for the Random and GP instances. The Composite instances have a special dimensionwise structure and, thus, suit better a dimensionwise heuristic MDV. For 3-AP, DV (or MDV which is the same for $s = 3$) produces the best results. v-opt is the best for 4-AP, but 3-opt and MDV are still good. For the larger numbers of dimensions (for $s > 4$) 3-opt produces the best results.

As regards the running time of the heuristics (see Table 2), 2-opt and DV are the winners, but MDV is also quite fast. v-opt and especially 3-opt are significantly slower.

Now we will discuss the combined heuristics. DV2 is the fastest among them but its solution quality is usually worse that the MDV solution quality, while the running times of these heuristics are similar. MDVV is approximately ten times slower than DV2 but it is superior to DV2 and is even the best heuristic for 3- and 4-AP with respect to the solution quality. MDV3 is the slowest combined algorithm we consider but its solution quality is the best or close to the best for every instance type and every number of dimensions.

Note that every combined heuristic is superior to the corresponding vectorwise heuristics with respect to both the solution quality and the running time. DV2 is the only exception as it is slightly slower than 2-opt. In all other cases both the solution error and the running time of a combined heuristic are smaller or very similar to the results of the corresponding vectorwise heuristic in every experiment. And, obviously, the solution quality of the combined heuristic is never worse than the solution quality of the corresponding dimensionwise heuristic.

As a conclusion, we may select MDV3, MDVV, and MDV as the most efficient heuristics. MDV3 produces very good solutions in just several seconds or even less for quite large instances (up to 64 millions of vectors). The MDVV heuristic's solution quality decreases with s growth but it is still good even for $s = 8$ while for 3- and 4-AP MDVV produces the best results among all the heuristics. At the same time, MDVV is significantly faster than MDV3. The single heuristic MDV is the best among the other fast heuristics and it produces good solutions for large instances.

Another interesting result is that a combination of two different heuristics may yield a superior heuristic with respect to both the solution quality and the running time. This happens if the first heuristic is significantly faster than the second one: the initial stage of the optimization is performed by the fast heuristic, while the more accurate local search is performed by the slower heuristic.

Table 1. Solution quality comparison

		Solution error, %							
Inst.	Best	2-opt	3-opt	v-opt	DV	MDV	DV2	MDV3	MDVV
3gp50	251.5	21.0	12.8	20.2	10.1	10.0	10.1	9.1	9.6
3gp100	504.4	19.6	10.0	19.8	4.6	5.2	4.6	5.0	5.2
3c100	1551.8	85.0	29.0	11.7	10.0	9.8	10.0	9.1	7.3
3r100	100.0	181.2	31.8	4.1	12.8	15.0	12.8	13.0	3.3
3c200	2231.6	113.6	41.7	13.1	9.1	8.3	9.1	7.3	7.0
3r200	200.0	113.5	10.8	0.8	0.3	0.2	0.3	0.2	0.1
3c250	2537.3	123.1	46.9	11.4	7.8	7.2	7.8	7.1	7.1
3r250	250.0	95.5	5.1	0.2	0.1	0.0	0.1	0.0	0.0
4gp30	145.2	17.4	4.2	13.4	9.9	8.1	9.8	3.9	7.7
4c50	1282.8	66.0	20.1	13.5	23.0	9.1	20.5	8.4	7.7
4r50	50.0	166.2	18.8	4.8	60.8	37.6	54.6	14.6	5.6
4c80	1610.8	82.4	31.4	18.3	24.6	13.3	23.7	11.5	11.8
4r80	80.0	115.0	7.3	2.0	22.5	10.1	19.6	3.9	2.1
5gp12	66.2	10.6	2.1	8.5	12.2	5.6	7.9	1.8	5.3
5c30	1192.6	43.0	16.1	19.5	30.6	14.7	25.4	12.1	12.0
5r30	30.0	118.7	8.3	9.0	109.7	52.3	98.0	8.3	7.7
5c40	1320.4	57.6	20.5	20.6	31.6	15.2	28.4	13.5	14.5
5r40	40.0	104.5	4.3	3.8	68.0	30.8	58.8	3.5	4.0
6gp8	41.8	6.7	2.4	5.3	15.8	6.5	7.2	2.4	4.8
6c20	1159.0	33.4	11.6	27.1	34.2	12.9	23.2	9.1	11.7
6r20	20.0	118.5	2.5	18.0	158.0	51.5	89.0	3.0	12.0
7gp5	25.6	6.3	3.9	10.2	16.4	5.9	5.9	3.9	5.5
7c12	1162.9	22.5	8.0	26.9	30.1	9.7	16.8	6.7	9.7
7r12	12.0	92.5	0.0	40.0	244.2	70.8	100.8	0.0	21.7
8gp4	19.2	6.8	5.2	10.9	19.3	5.2	6.2	5.2	5.2
8c8	1148.3	12.0	2.0	18.6	19.4	4.3	7.4	3.9	4.3
8r8	8.0	90.0	0.0	41.3	370.0	78.8	97.5	0.0	31.3
All avg.		71.2	13.2	14.5	50.2	18.4	28.0	6.2	8.3
Composite avg.		63.9	22.7	18.1	22.0	10.4	17.2	8.9	9.3
Random avg.		119.6	8.9	12.4	104.6	34.7	53.2	4.6	8.8
GP avg.		12.6	5.8	12.6	12.6	6.6	7.4	4.5	6.2
3-AP avg.		94.1	23.5	10.2	6.8	7.0	6.8	6.3	5.0
4-AP avg.		89.4	16.4	10.4	28.2	15.6	25.6	8.4	7.0
5-AP avg.		66.9	10.3	12.3	50.4	23.7	43.7	7.8	8.7
6-AP avg.		52.9	5.5	16.8	69.3	23.6	39.8	4.8	9.5
7-AP avg.		40.4	4.0	25.7	96.9	28.8	41.2	3.5	12.3
8-AP avg.		36.3	2.4	23.6	136.2	29.4	37.1	3.0	13.6

4.2 Repetitive Search Experiments

As we saw in the previous section, different heuristics have different running times. To equate the running times of different heuristics, in the second type of our experiments we used a simple strategy that we call Repetitive Search Metaheuristic (RSM). RSM

Table 2. Running times comparison

	Running time, ms							
Inst.	2-opt	3-opt	v-opt	DV	MDV	DV2	MDV3	MDVV
3gp50	2.56	82.68	17.08	1.41	1.59	3.43	46.80	10.58
3gp100	9.64	917.29	243.36	14.91	16.11	15.73	385.32	57.72
3c100	11.54	1170.01	321.36	14.09	16.89	17.43	499.20	173.16
3r100	13.93	678.60	34.32	10.19	9.88	10.81	196.56	35.88
3c200	42.95	10374.07	3614.54	48.36	53.04	49.92	4378.95	1077.97
3r200	29.64	3706.58	126.36	25.66	27.48	23.60	138.84	40.56
3c250	68.64	21400.22	8018.45	95.16	102.96	121.68	6920.20	1422.73
3r250	48.36	5099.67	184.08	37.44	37.44	35.88	249.60	53.04
4gp30	1.06	127.92	45.24	3.73	1.46	1.17	115.44	21.00
4c50	9.03	923.53	341.64	4.71	13.11	8.24	322.92	121.68
4r50	7.13	413.40	40.56	3.19	9.16	5.56	263.64	35.88
4c80	17.40	4505.31	1733.17	16.68	27.34	17.62	2187.13	737.88
4r80	17.21	1168.45	62.40	12.74	21.34	14.15	357.24	76.44
5gp12	0.30	31.20	6.98	0.25	1.95	0.49	24.96	5.97
5c30	7.57	1113.85	156.00	10.25	8.37	7.64	469.56	76.44
5r30	4.82	315.12	18.37	4.03	5.65	3.29	235.56	20.28
5c40	14.55	2892.26	439.92	7.49	18.27	11.79	1241.77	120.12
5r40	7.81	503.88	37.44	3.62	11.88	10.15	318.24	40.56
6gp8	1.71	49.92	5.00	0.22	0.43	0.32	43.68	3.21
6c20	8.21	1865.77	102.96	0.99	10.13	8.01	1054.57	37.67
6r20	3.06	277.68	16.04	0.70	2.63	3.47	251.16	17.25
7gp5	0.08	53.04	2.46	1.71	0.44	0.21	35.88	2.40
7c12	3.02	1748.77	34.32	0.49	4.27	4.28	1190.29	15.25
7r12	0.97	170.04	11.44	1.89	6.50	1.17	180.96	12.77
8gp4	0.08	126.36	3.81	0.13	0.60	0.25	56.16	3.92
8c8	1.13	2857.94	27.23	0.32	5.83	2.68	918.85	9.58
8r8	0.70	201.24	10.93	0.23	7.42	0.81	184.08	12.08
All avg.	12.34	2324.99	579.83	11.87	15.64	14.07	824.73	157.11
Comp. avg.	18.40	4885.17	1478.96	19.85	26.02	24.93	1918.34	379.25
Rand. avg.	13.36	1253.47	54.19	9.97	13.94	10.89	237.59	34.47
GP avg.	2.20	198.34	46.28	3.19	3.22	3.09	101.18	14.97
3-AP avg.	28.41	5428.64	1569.95	30.90	33.17	34.81	1601.94	358.96
4-AP avg.	10.37	1427.72	444.60	8.21	14.48	9.35	649.28	198.58
5-AP avg.	7.01	971.26	131.74	5.13	9.22	6.67	458.02	52.67
6-AP avg.	4.33	731.12	41.33	0.64	4.39	3.93	449.80	19.38
7-AP avg.	1.35	657.28	16.07	1.36	3.73	1.89	469.04	10.14
8-AP avg.	0.64	1061.85	13.99	0.23	4.62	1.25	386.36	8.53

runs the given heuristic several times starting from a random assignment each time. RSM stops when the time allowed is elapsed. The best assignment obtained in the conducted repeats is selected as an RSM solution. In our experiments RSM is given time enough for the slowest heuristic to run once.

Table 3. Repetitive Search Metaheuristic experimental results

Inst.	Time, sec	Solution error, % 2-opt	3-opt	v-opt	DV	MDV	DV2	MDV3	MDVV
3gp50	0.1	17.6	12.3	17.7	6.7	6.4	6.7	7.0	6.9
3gp100	0.9	17.7	10.2	18.8	3.2	3.1	3.2	4.0	3.4
3c100	1.1	68.3	31.5	7.3	1.5	2.6	1.5	7.5	3.3
3r100	0.7	131.3	30.2	1.4	6.9	7.1	6.6	10.3	0.7
3c200	10.0	97.3	41.0	9.4	1.0	1.1	1.2	5.0	3.7
3r200	3.4	86.8	9.2	0.0	0.0	0.0	0.0	0.0	0.0
3c250	21.9	106.8	43.7	8.9	1.0	0.8	1.0	4.9	2.3
3r250	5.0	78.0	5.3	0.0	0.0	0.0	0.0	0.0	0.0
4gp30	0.1	12.7	3.9	10.3	5.5	3.5	5.6	3.2	5.1
4c50	0.9	46.0	18.5	13.3	9.3	1.9	7.4	8.1	3.9
4r50	0.4	103.6	20.0	0.8	29.2	17.6	27.0	14.8	0.6
4c80	5.1	63.2	26.1	13.9	10.5	2.0	10.0	7.2	4.5
4r80	1.1	80.6	8.1	0.0	8.8	4.1	8.4	2.5	0.0
5gp12	0.0	3.6	1.8	6.2	5.1	2.0	3.3	1.8	2.7
5c30	1.1	28.9	17.8	14.1	13.7	1.4	10.6	5.8	1.6
5r30	0.3	73.7	9.3	0.0	46.7	27.7	37.3	10.7	1.3
5c40	2.7	39.4	21.2	13.6	16.3	2.8	13.7	7.4	4.2
5r40	0.6	65.0	4.0	0.0	33.8	16.5	26.5	4.0	0.5
6gp8	0.1	2.4	2.4	2.4	2.6	2.4	2.4	2.4	2.4
6c20	1.7	16.1	11.6	17.2	15.0	0.9	7.8	7.2	3.2
6r20	0.4	49.5	3.5	0.0	54.5	24.0	35.0	2.5	0.5
7gp5	0.1	3.9	3.9	3.9	3.9	3.9	3.9	3.9	3.9
7c12	2.0	6.0	4.9	10.1	9.4	0.7	2.8	3.4	1.0
7r12	0.2	25.8	1.7	0.8	79.2	17.5	24.2	1.7	1.7
8gp4	0.1	5.2	4.2	5.2	5.2	5.2	5.2	5.2	5.2
8c8	2.6	0.4	3.2	6.0	3.5	0.1	0.1	2.7	0.1
8r8	0.2	10.0	0.0	0.0	97.5	15.0	11.3	0.0	0.0
All avg.		45.9	12.9	6.7	17.4	6.3	9.7	4.9	2.3
Composite avg.		47.2	22.0	11.4	8.1	1.4	5.6	5.9	2.8
Random avg.		70.4	9.1	0.3	35.6	12.9	17.6	4.6	0.5
GP avg.		9.0	5.5	9.2	4.6	3.8	4.3	3.9	4.2
3-AP avg.		75.5	22.9	7.9	2.5	2.6	2.5	4.8	2.5
4-AP avg.		61.2	15.3	7.7	12.6	5.8	11.7	7.2	2.8
5-AP avg.		42.1	10.8	6.8	23.1	10.1	18.3	5.9	2.1
6-AP avg.		22.7	5.8	6.5	24.0	9.1	15.1	4.0	2.0
7-AP avg.		11.9	3.5	5.0	30.8	7.4	10.3	3.0	2.2
8-AP avg.		5.2	2.5	3.7	35.4	6.8	5.5	2.6	1.8
Avg. no. repeats		214.7	1.0	19.2	585.6	124.6	195.1	3.6	34.6

To provide equal opportunities to all the heuristics, our RSM algorithm stops if $T + t/2 \geq A$, where T is the total time spent by all previous repetitions, t is the time of the last heuristic's run, and A is the time allowed to RSM. In other words, in the second type

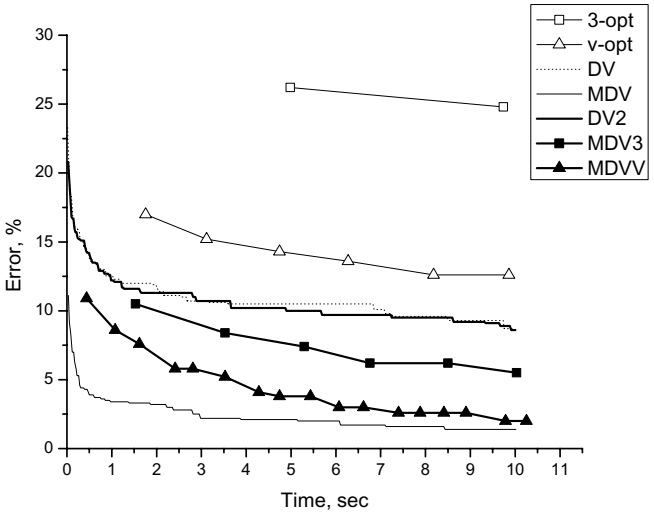


Fig. 1. Repetitive search for `4c80` instance

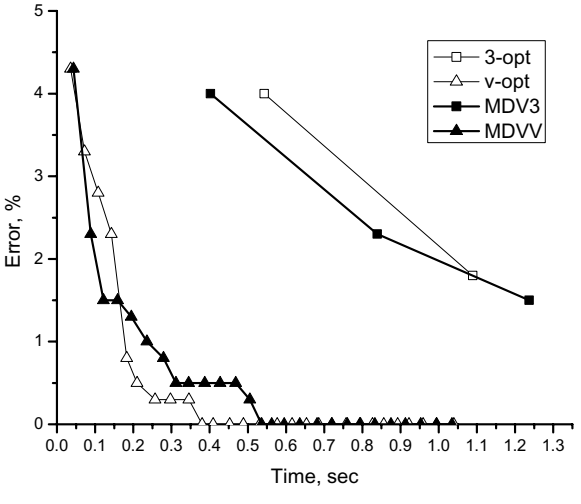


Fig. 2. Repetitive search for `5r40` instance

of the experiments we allow approximately equal time for every heuristic. The results are provided in Table 3. The time allowed to RSM is reported in the column *Time*. As with the other values in the table, this column contains the average for 10 instances of the given type and size. The average number of repeats for every heuristic is provided at the bottom of the table. For the optimal (best) solution values refer to Table 1. In other aspects Table 3 is similar to Table 1.

3-opt was the slowest heuristic in our experiments. In the most cases it was allowed just one run. MDV3 was repeated 5 times approximately on average, other heuristics had

more than 25 repeats on average. Unlike in the previous experiment, MDVV outperforms MDV3 almost always and, thus, shows the best results. Another interesting observation is that MDV outperforms all other heuristics for Composite and GP instances on average. Random instances are processed by v-opt (and MDVV that includes v-opt as a part) better than by other heuristics.

Dependence of the RSM solution quality from the running time is shown on Figures 1 and 2 for instances `4c80` and `5r40` respectively. Every graph is plotted for 10 instances of the fixed type and size. The solution quality and the running time of every repetition is average. The time allowed to RSM in both cases is enough for the slowest heuristic (3-opt) to repeat twice. Most of heuristics are able to improve the solution quality significantly during this time. However, some heuristics were clearly unsuccessful in these experiments (2-opt for `4c80` and 2-opt, DV, MDV, and DV2 for `5r40`) and, thus, are excluded from the graphs.

One can see that for `4c80` instance (and this tendency is common for the most of Composite and GP instances), MDV reaches a near optimal solution in a short time. MDVV works well for both instances and its performance is close to v-opt performance for `5r40` (as well as for other Random instances). Other heuristics are clearly outperformed by MDV, v-opt or MDVV and, thus, are not discussed here.

5 Conclusion

Several local search heuristics and their combinations are proposed and discussed in this paper. An efficient approach to construction of a heuristic by combination of two single heuristics is successfully applied; the yielded heuristics showed that they combine strengths of the both single heuristics. The experimental evaluation for a set of instances of different types show that MDV and MDVV heuristics, proposed in this paper, dominate all other heuristics. MDV is a fast heuristic, and it outperforms all other fast heuristics with respect to the solution quality. MDVV is significantly faster than the most powerful heuristics but it achieves the best or close to the best results in all the experiments. If the Repetitive Search Metaheuristic is applied, i.e., all the heuristics are allowed the same running time, MDV and MDVV clearly outperform all other heuristics.

References

1. Aiex, R.M., Resende, M.G.C., Pardalos, P.M., Toraldo, G.: Grasp with path relinking for three-index assignment. INFORMS J. on Computing 17(2), 224–247 (2005)
2. Alon, N., Spencer, J.: The Probabilistic Method, 2nd edn. John Wiley, Chichester (2000)
3. Balas, E., Saltzman, M.J.: An algorithm for the three-index assignment problem. Oper. Res. 39(1), 150–161 (1991)
4. Bekker, H., Braad, E.P., Goldengorin, B.: Using bipartite and multidimensional matching to select the roots of a system of polynomial equations. In: Gervasi, O., Gavrilova, M.L., Kumar, V., Laganá, A., Lee, H.P., Mun, Y., Taniar, D., Tan, C.J.K. (eds.) ICCSA 2005. LNCS, vol. 3483, pp. 397–406. Springer, Heidelberg (2005)
5. Burkard, R.E., Çela, E.: Linear assignment problems and extensions. In: Du, Z., Pardalos, P. (eds.) Handbook of Combinatorial Optimization, Dordrecht, pp. 75–149 (1999)

6. Crama, Y., Spieksma, F.C.R.: Approximation algorithms for three-dimensional assignment problems with triangle inequalities. European Journal of Operational Research 60(3), 273–279 (1992)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Series of Books in the Mathematical Sciences. W. H. Freeman, New York (1979)
8. Grundel, D., Oliveira, C., Pardalos, P.: Asymptotic properties of random multidimensional assignment problems. Journal of Optimization Theory and Applications 122(3), 33–46 (2004)
9. Grundel, D.A., Pardalos, P.M.: Test problem generator for the multidimensional assignment problem. Comput. Optim. Appl. 30(2), 133–146 (2005)
10. Huang, G., Lim, A.: A hybrid genetic algorithm for the three-index assignment problem. European Journal of Operational Research 127(1), 249–257 (2006)
11. Knuth, D.E.: Seminumerical Algorithms, 2nd edn. The Art of Computer Programming, vol. 2. Addison-Wesley, Reading (1981)
12. Kuhn, H.W.: The hungarian method for the assignment problem. Naval Research Logistic Quarterly 2, 83–97 (1955)
13. Microsoft. MSDN, chapter Random Class. Microsoft (2008), `http://msdn2.microsoft.com/en-us/library/system.random.aspx`
14. Pierskalla, W.P.: The multidimensional assignment problem. Operations Research 16, 422–431 (1968)
15. Robertson, A.J.: A set of greedy randomized adaptive local search procedure (grasp) implementations for the multidimensional assignment problem. Comput. Optim. Appl. 19(2), 145–164 (2001)

On Distance-3 Matchings and Induced Matchings

Andreas Brandstädt[1] and Raffaele Mosca[2]

[1] Institut für Informatik, Universität Rostock, D-18051 Rostock, Germany
ab@informatik.uni-rostock.de
[2] Dipartimento di Scienze, Universitá degli Studi "G. D'Annunzio", Pescara 65121, Italy
r.mosca@unich.it

Abstract. For a finite undirected graph $G = (V, E)$ and positive integer $k \geq 1$, an edge set $M \subseteq E$ is a *distance-k matching* if the mutual distance of edges in M is at least k in G. For $k = 1$, this gives the usual notion of matching in graphs, and for general $k \geq 1$, distance-k matchings were called *k-separated matchings* by Stockmeyer and Vazirani. The special case $k = 2$ has been studied under the names *induced matching* (i.e., a matching which forms an induced subgraph in G) by Cameron and *strong matching* by Golumbic and Laskar in various papers.

Finding a maximum induced matching is $\mathbb{NP}$-complete even on very restricted bipartite graphs but for $k = 2$, it can be done efficiently on various classes of graphs such as chordal graphs, based on the fact that an induced matching in G corresponds to an independent vertex set in the square $L(G)^2$ of the line graph $L(G)$ of G which, by a result of Cameron, is chordal for any chordal graph G.

We show that, unlike for $k = 2$, for a chordal graph G, $L(G)^3$ is not necessarily chordal, and finding a maximum distance-3 matching remains $\mathbb{NP}$-complete on chordal graphs. For strongly chordal graphs and interval graphs, however, the maximum distance-3 matching problem can be solved in polynomial time. Moreover, we obtain various new results for induced matchings.

Keywords: Distance-k matching; chordal graphs; Maximum Distance-k Matching Problem; Maximum Induced Matching Problem.

1 Introduction

A *distance-k matching* for positive integer $k \geq 1$ in an undirected graph G is a set of edges whose mutual distance is at least k in G. (For $k = 1$, this gives the usual notion of matching in graphs.) Stockmeyer and Vazirani [40] called those matchings *δ-separated matchings*, and the special case $k = 2$ has also been studied under the name *induced matching* (i.e., a matching which forms an induced subgraph in G) by Cameron in [10] and *strong matching* by Golumbic and Laskar in [25]. An induced matching can also be interpreted as risk-free marriage, and in various papers, δ-separated matchings are motivated by modeling communication network testing [40] or modeling concurrent transmission of messages in

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 116–126, 2009.

wireless ad hoc networks [1]. Induced matchings are closely related to irredundancy in graphs [25] as well as to secure communication channels as described by Golumbic and Lewenstein [26]; see [26] for other applications as well. Finding large induced matchings is also a subtask of finding a so-called *strong edge coloring in a graph* [19,20,28,39] where the edge set is partitioned into a minimum number of induced matchings.

Let $L(G)$ denote the *line graph* of $G = (V, E)$, i.e., the edges of G are the vertices of $L(G)$, and two edges of G are adjacent in $L(G)$ if they intersect each other.

For two vertices $x, y \in V$, let $dist_G(x, y)$ denote the *distance between x and y in G*, i.e., the length of a shortest path between x and y in G. For integer $k \geq 1$, the *k-th power G^k* of G is the graph with the same vertex set as G, and two vertices are adjacent in G^k if their distance in G is at most k. The *distance of two edges $e, e' \in E$* is the length of a shortest path between e and e', i.e., $dist_G(e, e') = \min\{dist_G(u, v) \mid u \in e, v \in e'\}$. In particular, this means that $dist_G(e, e') = 0$ if and only if $e \cap e' \neq \emptyset$.

A vertex set $U \subseteq V$ is *independent* (or *stable*) if the vertices in U are mutually nonadjacent in G. Obviously, for edges e and e', $dist_G(e, e') \geq k$ if and only if e and e' are nonadjacent in the k-th power $(L(G))^k$ (subsequently denoted by $L(G)^k$ for short) of the line graph of G for all $k \geq 1$. Thus, the following proposition holds:

Proposition 1. *For $k \geq 1$ and graph G, the edge set M is a distance-k matching in G if and only if M is an independent vertex set in $L(G)^k$.*

Let $\mu_k(G)$ denote the maximum size of a distance-k matching in G, and if w is a weight function on E, let $\mu_{k,w}(G)$ denote the maximum weight of a distance-k matching in G. The *Maximum Distance-k Matching* (*MDkM*) Problem asks for a distance-k matching of maximum size (and of maximum weight in the weighted case). By Proposition 1, the MDkM problem on graph G corresponds to the Maximum Independent Vertex Set problem on graph $L(G)^k$, and the first problem is efficiently solvable on graph G whenever the last problem is efficiently solvable on $L(G)^k$. For $k = 2$, the MDkM problem is also called *Maximum Induced Matching* (*MIM*) Problem and *MWIM* in the weighted case.

Recently, these problems attracted much attention because of their theoretical interest and practical motivation - see the papers mentioned above and also [5,10,11,12,13,22,30,31,33,43]. Stockmeyer and Vazirani [40] have shown that for every $k \geq 2$, the MDkM Problem is $\mathbb{NP}$-complete even for bipartite graphs of degree 4. In particular, the MIM Problem is $\mathbb{NP}$-complete on bipartite graphs which was shown independently by Cameron [10]. It is $\mathbb{NP}$-complete even on planar bipartite graphs of maximum degree 4 as shown in [30]. The MIM problem remains $\mathbb{NP}$-complete for line graphs [31] and thus also for claw-free graphs. It is $\mathbb{NP}$-complete even if the input graph is restricted to hamiltonian line graphs of so-called well-matched graphs as shown by Orlovich and Zverovich [37] and for line graphs of bipartite graphs (which are exactly the (claw,diamond,odd-hole)-free graphs) [38]. Various papers such as [16,36] deal with the hardness of approximating the MIM problem.

On the other hand, the MIM Problem is efficiently solvable on various classes $\mathcal{C}$ of graphs such as chordal, weakly chordal, circular-arc, interval-filament graphs and AT-free graphs (see e.g. [11] for a list of references) since for these classes, $L(G)^2$ is in the same class $\mathcal{C}$ as well and the Maximum Independent Set Problem can be solved efficiently on these classes. Most of the papers on distance-k matchings deal with the case $k = 2$, i.e., with the MIM problem, and do not study larger $k \geq 3$. For AT-free graphs, however, Chang [13] has shown that for every $k \geq 2$, $L(G)^k$ is AT-free.

We deal with the Maximum Distance-3 Matching (MD3M) Problem on various graph classes; in particular we show that for a chordal graph G (strongly chordal graph, respectively), $L(G)^3$ is not necessarily chordal (strongly chordal, respectively), and the MD3M Problem remains $\mathbb{NP}$-complete on chordal graphs. We also investigate strongly chordal graphs and interval graphs as subclasses of chordal graphs with respect to $L(G)^3$ and show that for strongly chordal G, $L(G)^3$ is chordal. Moreover, we extend some previous results for induced matchings. In particular, we use a combination of clique separators and modular decomposition in order to find induced matchings efficiently on various graph classes.

Due to space limitations, all proofs are omitted.

2 Basic Notions

Let $G = (V, E)$ be a finite undirected graph with vertex set V and edge set E. An edge e with vertices x and y is denoted as $e = \{x, y\}$ or $e = xy$ for short. We say that x and y *see each other* if $xy \in E$. As already mentioned, a vertex set is independent if its elements are mutually nonadjacent. A vertex set is a *clique* if its elements mutually see each other.

For $U \subseteq V$, let $G[U]$ denote the induced subgraph of G with vertex set U. Let P_k denote the induced path with k vertices, say $v_1, \ldots, v_k$ and edges $v_i v_{i+1}$, $1 \leq i \leq k-1$, and let C_k denote the induced cycle with the same vertices and edges $v_i v_{i+1}$, $1 \leq i \leq k$ (index arithmetic modulo k). A *hole* is a C_k for $k \geq 5$. For a set $\mathcal{F}$ of graphs, a graph G is called *$\mathcal{F}$-free* if G contains no induced subgraph from $\mathcal{F}$. A graph is *chordal* if it is C_k-free for all $k \geq 4$. For the many properties of chordal graphs see e.g. [7,23]. A subset $U \subseteq V$ is a *cutset* (or *separator*) in G if $G[V \setminus U]$ has more connected components than G. A *clique cutset* is a cutset which is a clique. An *atom* in G is an induced subgraph of G without clique cutset. Chordal graphs are known to be those graphs whose atoms are cliques (e.g., see [7]).

The *Maximum Weight Independent* (or *Stable*) *Set* problem (*MWS* problem) asks for an independent vertex set of maximum weight in the given graph G with vertex weight function w. Let $\alpha_w(G)$ ($\alpha(G)$, i.e., the independence number of G, respectively) denote the maximum weight (maximum cardinality, respectively) of an independent vertex set in G. The *MS problem* is the MWS problem where all vertices v have the same weight $w(v) = 1$. Frank [21] gave a linear time algorithm for the MWS problem on chordal graphs.

Let Π denote a hereditary graph property. A graph is *nearly* Π if for each of its vertices, the subgraph induced by the set of its nonneighbors has property Π (for example, $\Pi =$ chordal or $\Pi =$ perfect). Note that the concept of nearly Π graphs is explicitely or implicitely mentioned in various papers; thus, e.g., complements of nearly bipartite graphs appear in the literature as *quasi-line graphs* which are an important part of the structural characterization of claw-free graphs by Chudnovsky and Seymour - see [15].

Obviously, the MWS problem on a graph G with vertex weight function w can be reduced to the same problem on antineighborhoods of vertices in the following way:

$$\alpha_w(G) = \max\{w(v) + \alpha_w(G[\overline{N}(v)]) \mid v \in V\}. \tag{1}$$

Thus, whenever MWS is solvable in time T on a class with property Π then it is solvable on nearly Π graphs in time $|V| \cdot T$. Thus, the MWS problem can be solved in time $\mathcal{O}(|V| \cdot |E|)$ for nearly chordal graphs. Note that circular-arc graphs are nearly interval graphs and thus nearly chordal. This implies a straightforward way of solving MWS on circular-arc graphs in time $\mathcal{O}(|V| \cdot |E|)$ (which is conceptually simpler than that in [24]) and which is also applicable to the MIM problem on circular-arc graphs [25] in the following way: A graph $G = (V, E)$ is *edge-nearly* Π if for each of its edges $e = xy \in E$, the subgraph induced by the set of its nonneighbors, i.e., $G[\overline{N}(e)] = G[\overline{N}(x) \cap \overline{N}(y)]$ has property Π. Obviously, the following holds:

$$\mu_{2,w}(G) = \max\{w(e) + \mu_{2,w}(G[\overline{N}(e)]) \mid e \in E\}. \tag{2}$$

Thus, whenever the MIM problem is solvable in time T on a class with property Π then it is solvable on nearly Π graphs in time $|E| \cdot T$. This can be easily generalized to distance-k matchings for $k > 2$. We call (2) and its generalization for $k > 2$ the *antineighborhood approach* for MIM (for MDkM, respectively). Obviously, nearly Π implies edge-nearly Π. Whenever MIM is solvable in time T on a class with property Π, it is solvable on edge-nearly Π graphs in time $|E| \cdot T$. For example, MIM is efficiently solvable on edge-nearly chordal and on edge-nearly weakly chordal graphs.

A subset of vertices $M \subseteq V$ in graph $G = (V, E)$ is a *module* in G if every vertex $u \notin M$ sees either all vertices of M or none of them. A module is *trivial* if it is empty, a singleton or the set V. Nontrivial modules are called *homogeneous sets*. If an edge $xy \in E$ is a module then x and y are *true twins*. A graph is *prime* if all its modules are trivial. Note that prime graphs are connected. See [35] for the importance of modular decomposition and [34] for a linear time algorithm for obtaining the modular decomposition tree of a graph.

The *Minimum Distance-k Edge Coloring Problem* (*MDkEC Problem* for short) is the problem of finding a partition of the edge set E of a given graph $G = (V, E)$ into a minimum number of distance-k matchings. The *Minimum Distance-k Edge ℓ-Coloring Problem* (*MDkEℓC Problem* for short) is the problem of finding a partition of the edge set E of a given graph $G = (V, E)$ into ℓ distance-k matchings. For $k = 2$, this problem is called the *Strong Edge Coloring* problem.

For graph classes not defined here such as perfect graphs, circular-arc graphs and distance-hereditary graphs see e.g. [7].

3 The Maximum Distance-3 Matching Problem for Chordal Graphs Is $\mathbb{NP}$-Complete

As shown by Cameron, the following holds for $k = 2$:

Theorem 1 ([10]). *If graph G is chordal then $L(G)^2$ is chordal.*

Thus, by Proposition 1, and by using an efficient algorithm for MWS on chordal graphs, e.g. [21], the MIM problem can be solved efficiently on chordal graphs (see also [5] for a linear time algorithm). Since for chordal graph G^k, also G^{k+2} is chordal [17], it follows:

Corollary 1. *For every chordal graph G and every integer $k \geq 1$, $L(G)^{2k}$ is chordal.*

Now let $k = 3$ and G be a chordal graph. Then $L(G)^3$ is not necessarily chordal (and not even perfect) as the following example shows: As in Figure 1, let G have 15 vertices $a_1, \ldots, a_5, b_1, \ldots, b_5, c_1, \ldots, c_5$ such that $a_1, \ldots, a_5$ is a clique, and for $i \in \{1, \ldots, 5\}$, b_i, c_i are true twins such that b_i and c_i see exactly a_i and a_{i+1} (index arithmetic modulo 5). Obviously, the five edges b_ic_i, $i \in \{1, \ldots, 5\}$, induce a C_5 in $L(G)^3$.

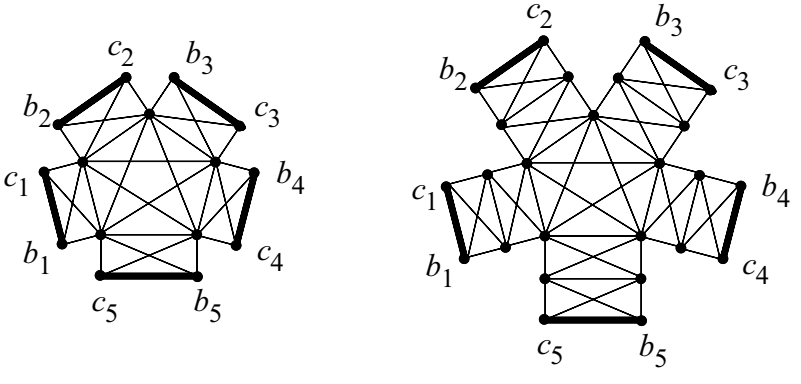


Fig. 1. Chordal graphs G_1 with a C_5 in $L(G)^3$ (indicated by boldface edges) and G_2 with C_5 in $L(G)^5$, respectively

It is easy to see that the example in Figure 1 can be generalized such that the edges b_ic_i, $i \in \{1, \ldots, 5\}$, have distance $k \geq 1$ and thus, $L(G)^{2k+1}$ contains a C_5, i.e., for all $k \geq 1$, $L(G)^{2k+1}$ is not perfect.

Theorem 2. *The Maximum Distance-3 Matching Problem is $\mathbb{NP}$-complete for chordal graphs.*

Theorem 3. *For each $k \geq 3$, the Minimum Distance-3 Edge k-Coloring Problem is $\mathbb{NP}$-complete for chordal graphs.*

4 Strongly Chordal Graphs, Ptolemaic Graphs and Interval Graphs

Graph G is a *block graph* if G is a (connected) graph whose 2-connected components are cliques. The *gem* consists of a P_4 and a fifth vertex seeing all vertices of the P_4. G is a *ptolemaic graph* if it is gem-free and chordal.

For $k \geq 3$, let S_k denote the (*complete*) *sun* with $2k$ vertices $u_1, \ldots, u_k$ and $w_1, \ldots, w_k$ such that $u_1, \ldots, u_k$ is an independent vertex set, $w_1, \ldots, w_k$ is a clique and, for $i \in \{1, \ldots, k\}$, u_i is adjacent to exactly w_i and w_{i+1} (index arithmetic modulo k).

A graph is *strongly chordal* if it is chordal and sun-free, i.e., S_k-free for all $k \geq 3$ - see e.g. [7] for various characterizations of strongly chordal graphs. Obviously, every block graph is ptolemaic, and every ptolemaic graph is strongly chordal.

In [10], it is mentioned that for strongly chordal graphs G, $L(G)^2$ is not necessarily strongly chordal. The example in [10] can be easily extended to an example of a strongly chordal graph (which is even a block graph) G such that $L(G)^3$ is not strongly chordal - see Figure 2. Obviously, the example can be extended to arbitrarily large $k \geq 2$, i.e., for every $k \geq 2$, there is a block graph G such that $L(G)^k$ is not strongly chordal.

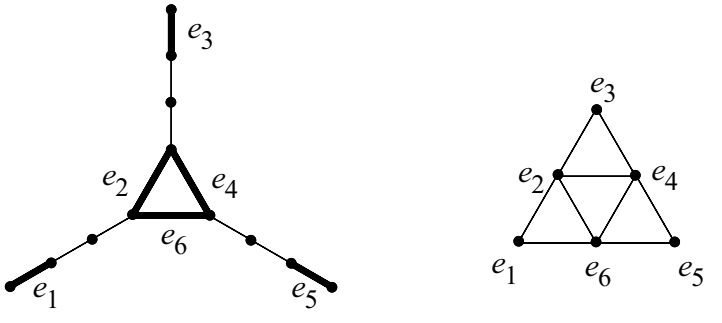


Fig. 2. A block graph G with a 3-sun in $L(G)^3$ (indicated by boldface edges)

The following is our main result in this section.

Theorem 4. *If graph G is strongly chordal then $L(G)^3$ is chordal.*

Corollary 2. *The Maximum Distance-3 Matching Problem is solvable in polynomial time for strongly chordal graphs.*

A graph is an *interval graph* if it is the intersection graph of a collection of intervals on the real line. It is known that interval graphs are strongly chordal. For interval graphs, we have the following result:

Theorem 5. *If G is an interval graph then $L(G)^3$ is an interval graph.*

Note that circular-arc graphs are nearly interval graphs and thus, the MD3M problem on circular-arc graphs can be solved efficiently by using the antineighborhood approach (2) and by solving the MD3M problem on interval graphs. This extends the corresponding result of Golumbic and Laskar [25] for the MIM problem on circular-arc graphs.

By a result of Golumbic and Rotics [27], the clique-width of ptolemaic graphs is at most 3. Since the MD3M problem can be formulated in Monadic Second Order Logic without edge predicates, it is efficiently solvable on ptolemaic graphs. It would be nice to give a direct way of solving the problem on ptolemaic graphs. The same remark holds for distance-hereditary graphs instead of ptolemaic graphs [27].

5 Combining Clique Separators, Modular Decomposition and the Antineighborhood Approach

In [41], Tarjan describes how clique separator decomposition of graphs can be obtained efficiently and how various problems on graphs such as MWS can be solved efficiently (see also [42]). One can extend this approach to the MIM problem as well and obtains the subsequent Theorem 6, details of which will be described in the forthcoming paper [8] and which is based on the following lemma.

Lemma 1. *If G is a graph G with clique separator $Q = V_1 \cap V_2$ such that Q separates G into $G_1 = G[V_1]$ and $G_2 = G[V_2]$ and there are edges $e_1 \in E(G_1)$ and $e_2 \in E(G_2)$ which are nonadjacent in $L(G)^2$ then $Q^* := \{e \in E \mid e \cap Q \neq \emptyset\}$ is a clique separator in $L(G)^2$.*

Then such clique separators in $L(G)^2$ can be used to find a maximum independent set in $L(G)^2$, i.e., by Proposition 1, a maximum induced matching in G.

Theorem 6. *Given a graph G in a hereditary graph class $\mathcal{G}$, if MWIM can be solved for every atom of G in time T, then MWIM can be solved for G in time $O(n^2 \cdot T)$.*

The following result allows to combine clique separator decomposition and modular decomposition for solving the MWS problem:

Theorem 7 ([4]). *Given a graph G in a hereditary graph class $\mathcal{G}$, if MWS can be solved for every prime atom of G in time T, then MWS can be solved for G in time $O(n^2 \cdot T)$.*

In [8], this is extended to the MIM problem (by a slight modification of prime to E-prime) as follows:

Theorem 8. *Given a graph G in a hereditary graph class $\mathcal{G}$, if MWIM can be solved for every E-prime atom of G in time T, then MWIM can be solved for G in time $O(n^2 \cdot T)$.*

In [31], modular decomposition has been used for solving the MIM problem on various graph classes. Theorem 8 together with the antineighborhood approach enables us to solve the MIM problem efficiently for various subclasses of claw-free graphs and of P_5-free graphs which extends some results of [31] as well as to the classes described in [6] which generalize chordal graphs.

6 The Maximum Distance-k Matching Problem for AT-Free Graphs, (Claw,Net)-Free Graphs and Some Other Subclasses of Claw-Free Graphs

An *asteroidal triple* (AT) in a graph G is a triple of pairwise nonadjacent vertices v_1, v_2, v_3 such that there is a path between v_i and v_{i+1} avoiding the neighborhood of v_{i+2}, $i \in \{1, 2, 3\}$ (index arithmetic modulo 3). A graph is *AT-free* if it contains no asteroidal triple.

The *claw* is a graph consisting of four vertices with a central vertex of degree three and three vertices being adjacent to the central vertex. Let $T_{i,j,k}$ denote the graph consisting of three vertex-disjoint induced paths Q_1, Q_2, Q_3, of lengths i, j, k, respectively, such that the only edges between the paths are a triangle between a choice a_1, a_2, a_3 of end vertices, i.e., a_i is one of the two end vertices of Q_i, $i \in \{1, 2, 3\}$. Then the *net* is the special case $T_{1,1,1}$, i.e., a graph consisting of six vertices such that three of them induce a triangle and each of the other three vertices sees exactly one of the triangle vertices.

Chang in [13] defines LAT-free graphs which are a generalization of AT-free graphs, and shows that for LAT-free graphs, the Maximum Distance-k Matching Problem is solvable in polynomial time for every fixed $k \geq 2$. As corollaries, he obtains:

Corollary 3. *If G is an AT-free graph then $L(G)^k$ is AT-free for every $k \geq 2$.*

Corollary 4. *The Maximum Distance-k Matching Problem is solvable in polynomial time for AT-free graphs for every $k \geq 3$.*

In [14], it is shown that for AT-free graphs G and every $k \geq 2$, G^k is a cocomparability graph. Since the Minimum Distance-k Edge Coloring Problem in G corresponds to the Chromatic Number Problem in $L(G)^k$, for AT-free graph G also $L(G)^k$, $k \geq 2$, is AT-free and for cocomparability graphs, the Chromatic Number Problem is solvable in polynomial time, it follows:

Corollary 5. *For even $k \geq 4$, the Minimum Distance-k Edge Coloring Problem is solvable in polynomial time on AT-free graphs.*

The problem seems to be open for $k = 2$ and $k = 3$ and for odd $k \geq 5$.

In [31], it is shown that the Maximum Distance-2 Matching Problem is NP-complete on line graphs (which implies NP-completeness on claw-free graphs)

and is also NP-complete on hamiltonian graphs. In this section, we deal with (claw,net)-free graphs (note that 2-connected (claw,net)-free graphs are hamiltonian [18] - see also [3,29]). To this purpose we use a result in [2]:

Lemma 2. *Every (claw,net)-free graph is nearly (claw,AT)-free.*

This leads to Theorem 9 by using Proposition 1 and a polynomial time algorithm for MWS on AT-free graphs [9,32].

Theorem 9. *The MIM problem is solvable in polynomial time on (claw,net)-free graphs.*

In [8], we extend Theorem 9 in the following way by using the approach in section 5.

Theorem 10. *For (claw,$T_{1,1,p}$)-free graphs, the MWIM problem can be solved in time $O(n^2m^{2p+3})$.*

Theorem 11. *For (claw,$T_{0,2,p}$)-free graphs, the MWIM problem can be solved in time $\mathcal{O}(n^2m^{2p+5})$.*

7 Conclusion

In this note, we have shown that the MD3M problem is $\mathbb{NP}$-complete on chordal graphs but efficiently solvable on strongly chordal graphs and thus on interval graphs. We mention that the antineighborhood approach, combined with clique separators and modular decomposition, is fruitful for various examples. Moreover we announce some new results on the MIM problem which extend some results by Kobler and Rotics [31].

References

1. Balakrishnan, H., Barrett, C.L., Anil Kumar, V.S., Marathe, M.V., Thite, S.: The Distance-2 Matching Problem and Its Relationship to the MAC-Layer Capacity of Ad Hoc Wireless Networks. IEEE J. on Selected Areas in Communications 22(6), 1069–1079 (2004)
2. Brandstädt, A., Dragan, F.F.: On the linear and circular structure of (claw,net)-free graphs. Discrete Applied Mathematics 129, 285–303 (2003)
3. Brandstädt, A., Dragan, F.F., Köhler, E.: Linear time algorithms for Hamiltonian problems on (claw,net)-free graphs. SIAM J. Computing 30, 1662–1677 (2000)
4. Brandstädt, A., Hoàng, C.T.: On clique separators, nearly chordal graphs and the Maximum Weight Stable Set problem. In: Jünger, M., Kaibel, V. (eds.) IPCO 2005. LNCS, vol. 3509, pp. 265–275. Springer, Heidelberg (2005); Theoretical Computer Science, vol. 389, pp. 295–306 (2007)
5. Brandstädt, A., Hoàng, C.T.: Maximum Induced Matchings for Chordal Graphs in Linear Time, appeared online. In: Algorithmica (2008)
6. Brandstädt, A., Klembt, T., Lozin, V.V., Mosca, R.: Maximum stable sets in subclasses of apple-free graphs, appeared online. In: Algorithmica (2008)

7. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. SIAM Monographs on Discrete Math. Appl, vol. 3. SIAM, Philadelphia (1999)
8. Brandstädt, A., Mosca, R.: Clique Separator Decomposition and Modular Decomposition for the Maximum Induced Matching Problem (2008) (manuscript)
9. Broersma, A., Kloks, T., Kratsch, D., Müller, H.: Independent sets in asteroidal-triple-free graphs. SIAM J. Discrete Math. 12, 276–287 (1999)
10. Cameron, K.: Induced matchings. Discrete Applied Math. 24, 97–102 (1989)
11. Cameron, K.: Induced matchings in intersection graphs. Discrete Mathematics 278, 1–9 (2004)
12. Cameron, K., Sritharan, R., Tang, Y.: Finding a maximum induced matching in weakly chordal graphs. Discrete Math. 266, 133–142 (2003)
13. Chang, J.-M.: Induced matchings in asteroidal-triple-free graphs. Discrete Applied Math. 132, 67–78 (2003)
14. Chang, J.-M., Ho, C.W., Ko, M.T.: Powers of asteroidal-triple-free graphs with applications. Ars Combinatoria 67, 161–173 (2003)
15. Chudnovsky, M., Seymour, P.: The Structure of Clawfree Graphs, Surveys in Combinatorics 2005. London Math. Soc. Lecture Note Series, vol. 327 (2005)
16. Duckworth, W., Manlove, D., Zito, M.: On the Approximability of the Maximum Induced Matching Problem. J. of Discrete Algorithms 3, 79–91 (2005)
17. Duchet, P.: Classical perfect graphs. Annals of Discrete Math. 21, 67–96 (1984)
18. Duffus, D., Jacobson, M.S., Gould, R.J.: Forbidden subgraphs and the Hamiltonian theme. In: The theory and applications of graphs (Kalamazoo, Mich. 1980), pp. 297–316. Wiley, New York (1981)
19. Erdös, P.: Problems and results in combinatorial analysis and graph theory. Discrete Math. 72, 81–92 (1988)
20. Faudree, R.J., Gyárfas, A., Schelp, R.H., Zuza, Z.: Induced matchings in bipartite graphs. Discrete Mathematics 78, 83–87 (1989)
21. Frank, A.: Some polynomial algorithms for certain graphs and hypergraphs. In: Proceedings of the Fifth British Combinatorial Conference, pp. 211–226. Univ. Aberdeen, Aberdeen (1975); Congressus Numerantium No. XV, Utilitas Math., Winnipeg, Man (1976)
22. Fricke, G., Laskar, R.: Strong matchings on trees. Congressus Numerantium 89, 239–243 (1992)
23. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, London (1980)
24. Golumbic, M.C., Hammer, P.L.: Stability in circular-arc graphs. J. Algorithms 9, 314–320 (1988)
25. Golumbic, M.C., Laskar, R.C.: Irredundancy in circular-arc graphs. Discrete Applied Math. 44, 79–89 (1993)
26. Golumbic, M.C., Lewenstein, M.: New results on induced matchings. Discrete Applied Math. 101, 157–165 (2000)
27. Golumbic, M.C., Rotics, U.: On the clique-width of some perfect graph classes. Internat. J. of Foundations of Computer Science 11, 423–443 (2000)
28. Horák, P., Qing, H., Trotter, W.T.: Induced matchings in cubic graphs. J. Graph Theory 17(2), 151–160 (1993)
29. Kelmans, A.: On Hamiltonicity of (claw,net)-free graphs. Discrete Math. 306, 2755–2761 (2006)
30. Ko, C.W., Shepherd, F.B.: Bipartite domination and simultaneous matroid covers. SIAM J. Discrete Math. 16, 517–523 (2003)

31. Kobler, D., Rotics, U.: Finding maximum induced matchings in subclasses of claw-free and P_5-free graphs and in graphs with matching and induced matching of equal maximum size. Algorithmica 37, 327–346 (2003)
32. Köhler, E.: Graphs without asteroidal triples, Ph.D. Thesis, FB Mathematik, TU Berlin (1999)
33. Lozin, V.V.: On maximum induced matchings in bipartite graphs. Information Processing Letters 81, 7–11 (2002)
34. McConnell, R.M., Spinrad, J.: Modular decomposition and transitive orientation. Discrete Mathematics 201, 189–241 (1999)
35. Möhring, R.H., Radermacher, F.J.: Substitution decomposition for discrete structures and connections with combinatorial optimization. Annals of Discrete Mathematics 19, 257–356 (1984)
36. Orlovich, Y., Finke, G., Gordon, V., Zverovich, I.E.: Approximability for the minimum and maximum induced matching problems, TR 130, Laboratoire Leibiz-IMAG, Grenoble (2005)
37. Orlovich, Y., Zverovich, I.E.: Well-matched graphs, RRR 12-2004 (2004)
38. Orlovich, Y., Zverovich, I.E.: Maximal induced matchings of minimum/maximum size, DIMACS TR 2004-26 (2004)
39. Steger, A., Yu, M.: On induced matchings. Discrete Math. 120, 291–295 (2004)
40. Stockmeyer, L.J., Vazirani, V.V.: NP-completeness of some generalizations of the maximum matching problem. Information Processing Letters 15, 14–19 (1982)
41. Tarjan, R.E.: Decomposition by clique separators. Discrete Math. 55, 221–232 (1985)
42. Whitesides, S.H.: A method for solving certain graph recognition and optimization problems, with applications to perfect graphs. In: Berge, C., Chvátal, V. (eds.) Topics on perfect graphs. North-Holland, Amsterdam (1984)
43. Zito, M.: Linear time maximum induced matching algorithm for trees. Nordic J. Comput. 7, 58–63 (2000)

On Duality between Local Maximum Stable Sets of a Graph and Its Line-Graph

Vadim E. Levit[1] and Eugen Mandrescu[2]

[1] Ariel University Center of Samaria, Israel
levitv@ariel.ac.il
[2] Holon Institute of Technology, Israel
eugen_m@hit.ac.il

Abstract. G is a *König-Egerváry graph* provided $\alpha(G)+\mu(G) = |V(G)|$, where $\mu(G)$ is the size of a maximum matching and $\alpha(G)$ is the cardinality of a maximum stable set, [2], [21].

S is a *local maximum stable set* of G, and we write $S \in \Psi(G)$, if S is a maximum stable set of the subgraph induced by $S \cup N(S)$, where $N(S)$ is the neighborhood of S, [11]. Nemhauser and Trotter Jr. proved that any $S \in \Psi(G)$ is a subset of a maximum stable set of G, [19].

In this paper we demonstrate that if $S \in \Psi(G)$, the subgraph H induced by $S \cup N(S)$ is a König-Egerváry graph, and M is a maximum matching in H, then M is a local maximum stable set in the line graph of G.

1 Introduction

Throughout this paper $G = (V, E)$ is a simple (i.e., a finite, undirected, loopless and without multiple edges) graph with vertex set $V = V(G)$ and edge set $E = E(G)$.

If $X \subset V$, then $G[X]$ is the subgraph of G spanned by X. By $G - W$ we mean the subgraph $G[V - W]$, if $W \subset V(G)$. We also denote by $G - F$ the partial subgraph of G obtained by deleting the edges of F, for $F \subset E(G)$, and we write shortly $G - e$, whenever $F = \{e\}$.

If $A, B \subset V$ are disjoint and non-empty, then by (A, B) we mean the set $\{ab : ab \in E, a \in A, b \in B\}$.

The *neighborhood* of a vertex $v \in V$ is the set $N(v) = \{w : w \in V \text{ and } vw \in E\}$. If $|N(v)| = 1$, then v is a *pendant vertex*. We denote the *neighborhood* of $A \subset V$ by $N_G(A) = \{v \in V - A : N(v) \cap A \neq \emptyset\}$ and its *closed neighborhood* by $N_G[A] = A \cup N(A)$, or shortly, $N(A)$ and $N[A]$, if no ambiguity.

K_n, C_n denote respectively, the complete graph on $n \geq 1$ vertices, and the chordless cycle on $n \geq 3$ vertices. A graph having no K_3 as a subgraph is a *triangle-free graph*.

A *stable* set in G is a set of pairwise non-adjacent vertices. A stable set of maximum size will be referred to as a *maximum stable set* of G, and the *stability*

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 127–133, 2009.

number of G, denoted by $\alpha(G)$, is the cardinality of a maximum stable set in G. In the sequel, by $\Omega(G)$ we denote the set of all maximum stable sets of the graph G.

A set $A \subseteq V(G)$ is a *local maximum stable set* of G if A is a maximum stable set in the subgraph spanned by $N[A]$, i.e., $A \in \Omega(G[N[A]])$, [11]. Let $\Psi(G)$ stand for the set of all local maximum stable sets of G.

Clearly, every set S consisting of only pendant vertices belongs to $\Psi(G)$. Nevertheless, it is not a must for a local maximum stable set to contain pendant vertices. For instance, $\{e, g\} \in \Psi(G)$, where G is the graph from Figure 1.

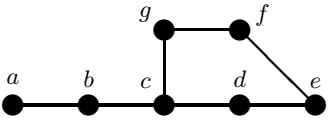


Fig. 1. A graph having various local maximum stable sets

The following theorem concerning maximum stable sets in general graphs, due to Nemhauser and Trotter Jr. [19], shows that some stable sets can be enlarged to maximum stable sets.

Theorem 1. *[19] Every local maximum stable set of a graph is a subset of a maximum stable set.*

Let us notice that the converse of Theorem 1 is true, because $\Omega(G) \subseteq \Psi(G)$. The graph W from Figure 1 has the property that any $S \in \Omega(W)$ contains some local maximum stable set, but these local maximum stable sets are of different cardinalities: $\{a, d, f\} \in \Omega(W)$ and $\{a\}, \{d, f\} \in \Psi(W)$, while for $\{b, e, g\} \in \Omega(W)$ only $\{e, g\} \in \Psi(W)$.

However, there exists a graph G satisfying $\Psi(G) = \Omega(G)$, e.g., $G = C_n$, for $n \geq 4$.

A *matching* in a graph $G = (V, E)$ is a set of edges $M \subseteq E$ such that no two edges of M share a common vertex. A *maximum matching* is a matching of maximum size, denoted by $\mu(G)$. A matching is *perfect* if it saturates all the vertices of the graph. A matching $M = \{a_i b_i : a_i, b_i \in V(G), 1 \leq i \leq k\}$ of a graph G is called *a uniquely restricted matching* if M is the unique perfect matching of $G[\{a_i, b_i : 1 \leq i \leq k\}]$, [5]. Recently, a generalization of this concept, namely, a *subgraph restricted matching* has been studied in [4].

Krogdahl found that a matching M of a bipartite graph is uniquely restricted if and only if M is alternating cycle-free, [9]. This statement was observed for general graphs by Golumbic *et al.* in [5].

In [11], [12], [15], [16], [17] we showed that, under certain conditions involving uniquely restricted matchings, $\Psi(G)$ forms a greedoid on $V(G)$. The classes of graphs, where greedoids were found include trees, bipartite graphs, triangle-free graphs, and well-covered graphs.

Recall that G is a *König-Egerváry graph* provided $\alpha(G) + \mu(G) = |V(G)|$, [2], [21]. It is a known that any bipartite graph is a König-Egerváry graph, [3],

[8]. Properties of König-Egerváry graphs were discussed in a number of papers, e.g., [1], [6], [7], [10], [13], [14], [18], [20]. Let us notice that if S is a stable set and M is a matching in a graph G such that $|S| + |M| = |V(G)|$, it follows that $S \in \Omega(G), M$ is a maximum matching, and G is a König-Egerváry graph, because $|S| + |M| \leq \alpha(G) + \mu(G) \leq |V(G)|$ is true for any graph.

The line graph of a graph $G = (V, E)$ is the graph $L(G) = (E, U)$, where $e_i e_j \in U$ if e_i, e_j have a common endpoint in G.

In this paper we give a sufficient condition in terms of subgraphs of G that ensure that its line graph $L(G)$ has proper local maximum stable sets. In other words, we demonstrate that if: $S \in \Psi(G)$, the subgraph H induced by $S \cup N(S)$ is a König-Egerváry graph, and M is a maximum matching in H, then M is a local maximum stable set in the line graph of G. It turns out that this is also a sufficient condition for a matching of G to be extendable to a maximum matching.

2 Maximum Matchings and Local Maximum Stable Sets

In a König-Egerváry graph, maximum matchings have a special property, emphasized by the following statement.

Lemma 1. *[13] Every maximum matching M of a König-Egerváry graph G is contained in each $(S, V(G) - S)$ and $|M| = |V(G) - S|$, where $S \in \Omega(G)$.*

For example, $M = \{e_1, e_2, e_3\}$ is a maximum matching in the König-Egerváry graph H (from Figure 2), $S = \{a, b, c, d\} \in \Omega(H)$ and $M \subset (S, V(H) - S)$. On the other hand, $M_1 = \{xz, yv\}, M_2 = \{yz, uv\}$ are maximum matchings in the non-König-Egerváry graph G (depicted in Figure 2), $S = \{x, y\} \in \Omega(G)$ and $M_1 \subset (S, V(G) - S)$, while $M_2 \not\subseteq (S, V(G) - S)$.

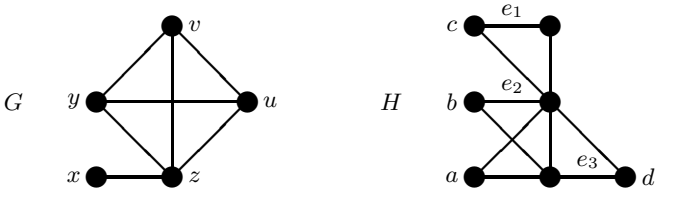


Fig. 2. $\{x, y\} \in \Omega(G)$ and $\{a, b, c, d\} \in \Omega(H)$

Clearly, (maximum) matchings in a graph G correspond to (maximum, respectively) stable sets in $L(G)$ and vice versa. However, not every matching M in G gives birth to a local maximum stable set in $L(G)$, even if M can be enlarged to a maximum matching.

For instance, $M_1 = \{e_1, e_6\}, M_2 = \{e_3, e_6\}$ are both matchings in the graph G from Figure 3, but only M_1 is a local maximum stable set in $L(G)$. Remark that $S_1 = \{v, z\} \in \Psi(G)$, $S_2 = \{x, y\} \notin \Psi(G)$ and each M_i is a maximum matching in $G[N[S_i]]$, for $i \in \{1, 2\}$.

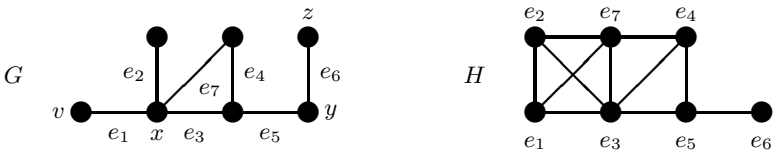


Fig. 3. The graph G and its line-graph $H = L(G)$

Theorem 2. *If $S \in \Psi(G), H = G[N[S]]$ is a König-Egerváry graph, and M is a maximum matching in H, then M is a local maximum stable set in $L(G)$.*

Proof. Let $M = \{e_i = v_i w_i : 1 \leq i \leq \mu(H)\}$. According to Lemma 1, it follows that

$$M \subseteq (S, V(H) - S) \text{ and } |M| = |V(H) - S|,$$

because H is a König-Egerváry graph. Consequently, without loss of generality, we may suppose that

$$\{v_i : 1 \leq i \leq \mu(H)\} \subseteq S, \text{ while } V(H) - S = \{w_i : 1 \leq i \leq \mu(H)\}.$$

Since $N_H(v_i) = N_G(v_i) \subseteq N(S) = V(H) - S$, we have that

$$N_{L(G)}[M] = E(H) \cup \{e = wt \in E : w \in V(H) - S, t \notin S\}.$$

Hence, every $e \in N_{L(G)}[M] - V(L(H))$ is incident in G to some w_i.

Assume that M is not a maximum stable set in $L(G)$, i.e., there exists some stable set $Q \subseteq N_{L(G)}[M]$, such that $|Q| > |M|$. In other words, Q is a matching using edges from

$$E(H) \cup \{e = wt \in E : w \in V(H) - S, t \notin S\},$$

larger than M. Let $F = (M - Q) \cup (Q - M)$. Since M and Q are matchings, every vertex appearing in $G[F]$ has at most one incident edge from each of them, and the maximum degree of a vertex in $G[F]$ is 2. Hence, $G[F]$ consists of only disjoints chordless paths and cycles. Moreover, every path and every cycle in $G[F]$ alternates between edges of Q and edges of M. Since $|Q| > |M|$, it follows that $G[F]$ has a component with more edges of Q than of M. Such a component can only be a path, say $P_{x,y}$, that starts and ends with edges from Q (more precisely, from $Q - M$) and and x, y are not saturated by edges belonging to M. Hence, $P_{x,y}$ must have an odd number of edges.

Case 1. $P_{x,y}$ contains only one edge, namely xy. This is not possible, since at least one of the vertices x, y belongs to $V(H) - S$ and is saturated by M.

Case 2. $P_{x,y}$ contains at least three edges.

Let $xa, by \in Q$ be the first and the last edges on $P_{x,y}$. Clearly, $E(P_{x,y}) \nsubseteq E(H)$, because, otherwise

$$(M - E(P_{x,y})) \cup (E(P_{x,y}) - M)$$

is a matching in H, larger than M, in contradiction with the maximality of M. Hence, $P_{x,y}$ contains edges from M, that alternates with edges from $(E(H) - M) \cup W$, where

$$W = \{wt \in E(G) : w \in V(H) - S, t \in U\},$$

with

$$U = (S - \{v_i : 1 \leq i \leq \mu(H)\}) \cup (V(G) - V(H)\} \neq \varnothing.$$

Therefore, each second vertex on $P_{x,y}$ must belong to $V(H) - S$. Consequently, we infer that also $y \in V(H) - S$, and hence, it is saturated by M, a contradiction.

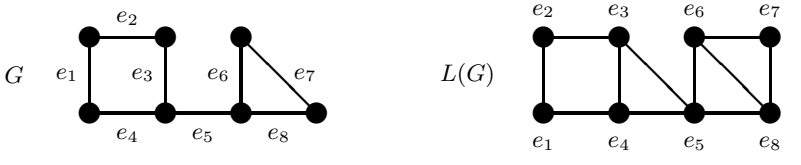


Fig. 4. $M = \{e_5, e_7\}$ is a matching in G and a local maximum stable set in $L(G)$

Notice that $M = \{e_5, e_7\} \in \Psi(L(G))$, while there is no $S \in \Psi(G)$, such that M is a maximum matching in $G[N[S]]$, where G is depicted in Figure 4. In other words, the converse of Theorem 2 is not true.

Clearly, every matching can be enlarged to a maximal matching, which is not necessarily a maximum matching. For instance, the graph G in Figure 5 does not contain any maximum matching including the matching $M = \{e_0, e_1, e_2\}$. The following result shows that, under certain conditions, a matching can be extended to a maximum matching.

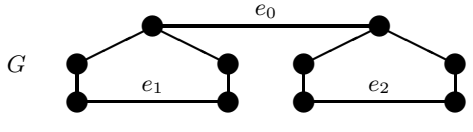


Fig. 5. $\{e_0, e_1, e_2\}$ is a maximal but not a maximum matching

Corollary 1. *If $S \in \Psi(G), H = G[N[S]]$ is a König-Egerváry graph, and M is a maximum matching in H, then there exists a maximum matching M_0 in G such that $M \subseteq M_0$.*

Proof. According to Theorem 2, M is a local maximum stable set in $L(G)$. By Theorem 1, there is some $M_0 \in \Omega(L(G))$, such that $M \subseteq M_0$. Hence, M_0 is a maximum matching in G containing M.

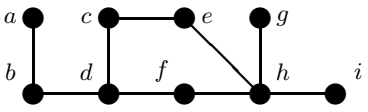


Fig. 6. $M = \{ab, cd, fh\}$ is a maximum matching in $N[\{a, c, f\}]$

Let us notice that Corollary 1 can not be generalized to any subgraph of a non-bipartite König-Egerváry graph.

For instance, the graph G depicted in Figure 6 is a König-Egerváry graph, $S = \{a, c, f\} \in \Psi(G)$, and $M = \{ab, cd, fh\}$ is a maximum matching in $G[N[S]]$, which is not a König-Egerváry graph, but there is no maximum matching in G that includes M.

Since any subgraph of a bipartite graph is also bipartite, we obtain the following result.

Corollary 2. *If G is a bipartite graph, $S \in \Psi(G)$ and M is a maximum matching in $G[N[S]]$, then there exists a maximum matching M_0 in G such that $M \subseteq M_0$.*

3 Conclusions

We showed that there is a connection between $\Psi(G)$ and $\Psi(L(G))$.

Let us notice that there are graphs whose line graphs have no proper local maximum stable sets (see, for example, the graphs in Figure 7).

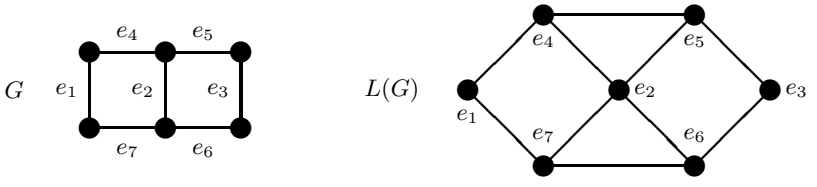


Fig. 7. Both G and its line graph $L(G)$ have no proper local maximum stable sets

Moreover, there are graphs whose iterated line graphs have no proper local maximum stable set, e.g., each C_n, for $n \geq 3$, since C_n and $L(C_n)$ are isomorphic.

An interesting open question reads as follows. Is it true that for a connected graph G the fact that $L(G)$ has no proper local maximum stable sets implies that G itself does not contain proper local maximum stable sets?

References

1. Bourjolly, J.M., Pulleyblank, W.R.: König–Egerváry graphs, 2-bicritical graphs and fractional matchings. Discrete Applied Mathematics 24, 63–82 (1989)
2. Deming, R.W.: Independence numbers of graphs - an extension of the König–Egerváry theorem. Discrete Mathematics 27, 23–33 (1979)

3. Egervary, E.: On combinatorial properties of matrices. Mat. Lapok 38, 16–28 (1931)
4. Goddard, W., Hedetniemi, S.M., Hedetniemi, S.T., Laskar, R.: Generalized subgraph-restricted matchings in graphs. Discrete Mathematics 293, 129–138 (2005)
5. Golumbic, M.C., Hirst, T., Lewenstein, M.: Uniquely restricted matchings. Algorithmica 31, 139–154 (2001)
6. Korach, E.: On Dual Integrality, Min-Max Equalities and Algorithms in Combinatorial Programming, University of Waterloo, Department of Combinatorics and Optimization, Ph.D. Thesis (1982)
7. Korach, E., Nguyen, T., Peis, B.: Subgraph characterization of Red/Blue-Split Graph and König–Egerváry graphs. In: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 842–850. ACM Press, New York (2006)
8. König, D.: Graphen und Matrizen. Mat. Lapok 38, 116–119 (1931)
9. Krogdahl, S.: The dependence graph for bases in matroids. Discrete Mathematics 19, 47–59 (1977)
10. Levit, V.E., Mandrescu, E.: Well-covered and König–Egerváry graphs. Congressus Numerantium 130, 209–218 (1998)
11. Levit, V.E., Mandrescu, E.: A new greedoid: the family of local maximum stable sets of a forest. Discrete Applied Mathematics 124, 91–101 (2002)
12. Levit, V.E., Mandrescu, E.: Local maximum stable sets in bipartite graphs with uniquely restricted maximum matchings. Discrete Applied Mathematics 132, 163–174 (2003)
13. Levit, V.E., Mandrescu, E.: On α^+-stable König–Egervary graphs. Discrete Mathematics 263, 179–190 (2003)
14. Levit, V.E., Mandrescu, E.: On α-critical edges in König–Egervary graphs. Discrete Mathematics 306, 1684–1693 (2006)
15. Levit, V.E., Mandrescu, E.: Triangle-free graphs with uniquely restricted maximum matchings and their corresponding greedoids. Discrete Applied Mathematics 155, 2414–2425 (2007)
16. Levit, V.E., Mandrescu, E.: Well-covered graphs and greedoids, Theory of Computing 2008. In: Harland, J., Manyem, P. (eds.) Proceedings of the Fourteenth Computing: The Australasian Theory Symposium (CATS 2008), Wollongong, NSW. Conferences in Research and Practice in Information Technology, vol. 77, pp. 89–94 (2008)
17. Levit, V.E., Mandrescu, E.: The clique corona operation and greedoids. In: Yang, B., Du, D.-Z., Wang, C.A. (eds.) COCOA 2008. LNCS, vol. 5165, pp. 384–392. Springer, Heidelberg (2008)
18. Lovász, L., Plummer, M.D.: Matching Theory. Annals of Discrete Mathematics 29 (1986)
19. Nemhauser, G.L., Trotter Jr., L.E.: Vertex packings: structural properties and algorithms. Mathematical Programming 8, 232–248 (1975)
20. Paschos, V.T., Demange, M.: A generalization of König–Egerváry graphs and heuristics for the maximum independent set problem with improved approximation ratios. European Journal of Operational Research 97, 580–592 (1997)
21. Sterboul, F.: A characterization of the graphs in which the transversal number equals the matching number. Journal of Combinatorial Theory Series B 27, 228–229 (1979)

On Path Partitions and Colourings in Digraphs

Irith Ben-Arroyo Hartman

Caesarea Rothschild Institute for Interdisciplinary Applications of Computer Science, University of Haifa, Haifa 31905, Israel
irith@cs.haifa.ac.il

Abstract. We provide a new proof of a theorem of Saks which is an extension of Greene's Theorem to acyclic digraphs, by reducing it to a similar, known extension of Greene and Kleitman's Theorem. This suggests that the Greene-Kleitman Theorem is stronger than Greene's Theorem on posets. We leave it as an open question whether the same holds for all digraphs, that is, does Berge's conjecture concerning path partitions in digraphs imply the extension of Greene's theorem to all digraphs (conjectured by Aharoni, Hartman and Hoffman)?

1 Introduction

Dilworth's well known theorem [7] states that in a partially ordered set the size of a maximum antichain equals the size of a minimum chain partition. Greene and Kleitman [13] generalized Dilworth's theorem to a min-max theorem for the maximum cardinality of the union of k antichains ($k \in \mathbb{N}$). Previously, Greene [12] had proved a similar min-max theorem where the role of chains and antichains is interchanged. Linial [15] conjectured that the theorems of Greene-Kleitman and Greene can be extended to all digraphs by replacing the equality by an inequality. Later, Berge [3] made a stronger conjecture than Linial's extending the Greene-Kleitman theorem to all digraphs, and Aharoni, Hartman and Hoffman (AHH) [1] made a similar conjecture which extends Greene's theorem to all digraphs, and is stronger than Linial's conjecture. Both conjectures of Berge and Aharoni-Hartman-Hoffman were proved for all acyclic digraphs (see [15], [6],[17], [5] and [1]). For $k = 1$ Berge's conjecture holds by the Gallai-Milgram theorem [11], and the Aharoni-Hartman-Hoffman conjecture holds by the Gallai-Roy theorem [10,16]. Recently, Berger and Hartman [4] proved Berge's conjecture for $k = 2$. For other values of k (except for the extreme upper values), all of the conjectures mentioned above are open. For a survey of the subject see [14].

The purpose of this paper is reduce the Aharoni-Hartman-Hoffman conjecture for acyclic digraphs to Berge's conjecture. Furthermore, a polynomial algorithm is given proving the AHH Conjecture, based on an oracle for Berge's Conjecture. If the same holds for all digraphs, then it will be sufficient to prove Berge's conjecture, and the rest of the conjectures will follow.

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 134–143, 2009.

2 Preliminaries and the Main Result

Let $G = (V, E)$ be a directed graph and let $|V| = n$. If L is a collection of subsets of V, we set $\bigcup L := \{x; x \in A \text{ for some } A \in L\}$. The cardinality of the set X is denoted by $|X|$.

A *path* P in G is a sequence of distinct vertices $(v_1, v_2, \ldots, v_l)$ such that $(v_i, v_{i+1}) \in E$, for $i = 1, 2, \ldots, l-1$. Let the *cardinality* of P be $|P| = l$. If a path P is of cardinality one, then we say it is *trivial*.

For positive integers q, k, a *q-path system* is a family $\mathcal{P}^q := \{P_1, P_2, \ldots, P_q\}$ of q pairwise disjoint paths, a *k-colouring* is a family $\mathcal{C}^k := \{C_1, C_2, \ldots, C_k\}$ of k pairwise disjoint independent sets, also called *colour classes*.

Denote $\lambda_q := \max |\bigcup \mathcal{P}^q|$ and $\alpha_k := \max |\bigcup \mathcal{C}^k|$ where the maximum is taken over all q-path systems and k-colourings, respectively, and $|\bigcup \mathcal{P}^q|$ ($|\bigcup \mathcal{C}^k|$) denote the number of vertices covered by $\mathcal{P}^q$ ($\mathcal{C}^k$). A q-path system with $|\bigcup \mathcal{P}^q| = \lambda_q$ is called *optimal*.

A family $\mathcal{P}$ of paths is called a *path partition* of G if all its members are pairwise disjoint, and $\cup\mathcal{P} = V$. The k-norm $|\mathcal{P}|_k$ of a path partition $\mathcal{P} = \{P_1, \ldots, P_m\}$ is defined by $|\mathcal{P}|_k := \sum_{i=1}^{m} min\{|P_i|, k\}$. Denote by $\mathcal{P}^0$ the set of trivial paths in $\mathcal{P}$, and by $\mathcal{P}^{\geq k}$ ($\mathcal{P}^{>k}$) the sets of paths in $\mathcal{P}$ of cardinality at least (more than) k. A partition which minimizes $|\mathcal{P}|_k$ is called *k-optimal*. Let $\pi_k(G) = min|\mathcal{P}|_k$ where the minimum is taken over all possible path partitions in G. If k is the cardinality of the smallest non-trivial path on $\mathcal{P}$, then clearly, $|\mathcal{P}|_k = k|\mathcal{P}^{\geq k}| + |\mathcal{P}^0|$.

Similarly, a *colouring* $\mathcal{C}$ is a family of pairwise disjoint independent sets where $\cup\mathcal{C} = V$. The q-norm $|\mathcal{C}|_q$ of a colouring $\mathcal{C} = \{C_1, \ldots, C_m\}$ is defined by $|\mathcal{C}|_q := \sum_{i=1}^{m} min\{|C_i|, q\}$. Let $\chi_q(G) = min|\mathcal{C}|_q$ where the minimum is taken over all possible colourings in G.

Theorem 1 (Greene-Kleitman[13]). *If G is a graph of a poset, then $\alpha_k(G) = \pi_k(G)$ for all $1 \leq k \leq \lambda_1$.*

Theorem 2 (Greene[12]). *If G is a graph of a poset, then $\lambda_q(G) = \chi_q(G)$ for all $1 \leq q \leq \alpha_1$.*

The inequality $\alpha_k(G) \leq \pi_k(G)$ is trivial since any path P (chain) in a poset induces a clique which can meet a k-colouring by at most $\min\{|P_i|, k\}$ vertices. Similarly, every independent set (antichain) C in a poset can meet a q-path system by at most $\min\{|C_i|, q\}$ vertices, implying the inequality $\lambda_q(G) \leq \chi_q(G)$. The other direction of the inequalities is less trivial and was conjectured by Linial to be true for all digraphs:

Conjecture 3 (Linial[15]). *Let G be a digraph, and k, q positive integers. Then*

1. $\alpha_k(G) \geq \pi_k(G)$
2. $\lambda_q(G) \geq \chi_q(G)$

For any graph G, a k-colouring $\mathcal{C}^k$ is *orthogonal* to a path partition if each $P_i \in \mathcal{P}$ meets exactly $min\{|P_i|, k\}$ different colour classes in $\mathcal{C}^k$. Similarly, a colouring $\mathcal{C}$ is *orthogonal* to a q-path system $\mathcal{P}^q$ if each $C_i \in \mathcal{C}$ meets exactly $min\{|C_i|, q\}$ different paths in $\mathcal{P}^q$.

Conjecture 4 (Berge's Strong Path Partition Conjecture [3]). *Let G be a digraph, k a positive integer, and $\mathcal{P}$ a k-optimal path partition. Then there exists a k-colouring $\mathcal{C}^k$ orthogonal to $\mathcal{P}$.*

This conjecture implies Conjecture 3-(1). The following is an equivalent conjecture to Conjecture 4:

Conjecture 5 (Equivalent to Conjecture 4). *Let G be a digraph, k a positive integer, and let $\mathcal{P}$ be some path partition in G. Then either there exists a k-colouring $\mathcal{C}^k$ orthogonal to $\mathcal{P}$, or there exists a path partition $\mathcal{P}'$ with $|\mathcal{P}'|_k < |\mathcal{P}|_k$*

The following conjecture extends Greene's Theorem to all digraphs and implies part 2 of Conjecture 3, in a similar way that Berge's Conjecture extends the Greene-Kleitman Theorem:

Conjecture 6 (Aharoni, Hartman, Hoffman (AHH) [1]). *Let G be a digraph, q a positive integer, and $\mathcal{P}^q$ an optimal q-path system. Then there exists a colouring $\mathcal{C}$ orthogonal to $\mathcal{P}^q$.*

Conjecture 7 (Equivalent to Conjecture 6). *Let G be a digraph, q a positive integer, and $\mathcal{P}^q$ some q-path system in G. Then either there exists a colouring $\mathcal{C}$ orthogonal to $\mathcal{P}^q$, or there exists a q-path system $\mathcal{P}'^q$ with $|\mathcal{P}'^q| > |\mathcal{P}^q|$.*

Conjecture 4 implies Conjecture 3-(1) and it holds for $k = 1$ by the Gallai-Milgram [11] theorem. Conjecture 6 implies Conjecture 3-(2) and it holds for $q = 1$ by the Gallai-Roy [10,16] theorem.

The following definition of Frank [9] helps us in uniting all Conjectures 3-6.

Definition 8. *A q-path system $\mathcal{P}^q = \{P_1, P_2, \ldots, P_q\}$ and a k-colouring $\mathcal{C}^k = \{C_1, C_2, \ldots, C_k\}$ are* orthogonal *if*

1. $V = (\cup\mathcal{P}^q) \bigcup (\cup\mathcal{C}^k)$
2. $|P_i \cap C_j| = 1$ *for* $1 \leq i \leq q, 1 \leq j \leq k$

For a q-path system $\mathcal{P}^q$, the *associated path partition* $\mathcal{P}$ is defined by $\mathcal{P} := \mathcal{P}^q \cup \{\{x\}; x \notin \cup\mathcal{P}^q\}$. Similarly, the *associated colouring* to a k-colouring $\mathcal{C}^k$ is defined by $\mathcal{C} := \mathcal{C}^k \cup \{\{x\}; x \notin \cup\mathcal{C}^k\}$.

Observation 9.
1. *Let $\mathcal{P}^q$ be a q-path system orthogonal to $\mathcal{C}^k$, a k-colouring, for some integers q and k. Then $\mathcal{C}^k$ is orthogonal to the associated path partition $\mathcal{P} := \mathcal{P}^q \cup \{\{x\}; x \notin \cup\mathcal{P}^q\}$ and the associated colouring $\mathcal{C} := \mathcal{C}^k \cup \{\{x\}; x \notin \cup\mathcal{C}^k\}$ is orthogonal to $\mathcal{P}^q$.*
2. *Conversely, if $\mathcal{C}^k$ is orthogonal to some path partition $\mathcal{P}$, then $\mathcal{C}^k$ is orthogonal to $\mathcal{P}^q := \mathcal{P}^{\geq k}$. Note that $\mathcal{C}^k$ is also orthogonal to $\mathcal{P}^q$ where $\mathcal{P}^q$ consists of paths of cardinality more than k, and any number of paths of cardinality exactly k are included in $\mathcal{P}^q$.*

3. *Similarly, if a colouring $\mathcal{C}$ is orthogonal to some q-path system $\mathcal{P}^q$, then $\mathcal{P}^q$ is orthogonal to the k-colouring $\mathcal{C}^k := \mathcal{C}^{\geq q}$, where $\mathcal{C}^{\geq q}$ denotes the set of independent sets in $\mathcal{C}$ of size at least q.*
4. *Furthermore, if $\mathcal{P}^q$ and $\mathcal{C}^k$ are orthogonal then $\alpha_k(G) \geq \pi_k(G)$ and $\lambda_q(G) \geq \chi_q(G)$, implying Linial's conjectures for these values of k and q.*

Theorem 10. *Conjecture 7 can be reduced to Conjecture 5 for acyclic digraphs.*

Corollary 11. *Theorem 1 implies Theorem 2, i.e. Greene-Kleitman's Theorem implies Greene's Theorem.*

Proof of Corollary. Assume Theorem 1 holds. Then any optimal k-colouring $\mathcal{C}^k$ in a graph of a poset must be orthogonal to an optimal path partition $\mathcal{P}$ because in a poset each $P \in \mathcal{P}$ can meet at most $\min\{|P|, k\}$ vertices from $\mathcal{C}^k$. If some P would meet less than $\min\{|P|, k\}$ vertices from $\mathcal{C}^k$, we would have $\alpha_k(G) < \pi_k(G)$, contrary to Theorem 1. Hence Conjecture 4 holds for posets, implying by Theorem 10 that Conjecture 6 holds, and hence Greene's Theorem (Theorem 2) follows.

3 Proof of Theorem 10

3.1 Outline of Proof

We assume that Conjecture 5 is true. Given any path partition $\mathcal{P}$, and positive integer k, we assume that we have some Oracle that either finds a k-colouring $\mathcal{C}^k$ orthogonal to $\mathcal{P}$, or finds a path partition $\mathcal{P}'$ with $|\mathcal{P}'|_k < |\mathcal{P}|_k$. Let $q \geq 1$, and let $\mathcal{P}^q$ be a q-path system. We will show that Conjecture 7 holds for $\mathcal{P}^q$. Let $\mathcal{P}$ be the path partition associated with $\mathcal{P}^q$. We prove that either there exists a k, $1 \leq k \leq \min_{P \in \mathcal{P}^q} |P|$, and a k- colouring $\mathcal{C}^k$ orthogonal to $\mathcal{P}$, implying by Observation 9-(1) and (2) that $\mathcal{C} := \mathcal{C}^k \cup \{\{x\}; x \notin \cup \mathcal{C}^k\}$ is orthogonal to $\mathcal{P}^q$, or we find another q-path system $\mathcal{P}'^q$ with $|\mathcal{P}'^q| > |\mathcal{P}^q|$.

The proof is algorithmic and it uses a network constructed from G as in Frank [9]. We define a flow f which corresponds to the path partition $\mathcal{P}$ associated with $\mathcal{P}^q$. We begin with $k = 1$. If $\mathcal{P}$ is 1-optimal, then by the Gallai -Milgram Theorem (or Conjecture 4 for $k = 1$) there exists an independent set $\mathcal{C}^1$ orthogonal to it, implying that $\mathcal{C} := \mathcal{C}^1 \cup \{\{x\}; x \notin \cup \mathcal{C}^k\}$ is orthogonal to $\mathcal{P}^q$ (by Observation 9) and we are done. Otherwise, $\mathcal{P}$ is not 1-optimal, and the Oracle finds a path partition $\mathcal{P}'$ with $|\mathcal{P}'|_1 < |\mathcal{P}|_1$. Let f' be the flow corresponding to $\mathcal{P}'$. Then $f' - f$ is a feasible flow in the residual network $N_{f'}$ corresponding to f'. Depending on f', we either increase k by one, or we show that $f' - f$ can be used to find a new flow f'' which satisfies two main conditions:

1. f'' corresponds to a path partition $\mathcal{P}''$ in G.
2. $\mathcal{P}''$ contains a q-path system which covers more vertices than $\mathcal{P}^q$, contradicting the optimality of $\mathcal{P}^q$.

We prove that if no k-colouring $\mathcal{C}^k$ exists which is orthogonal to $\mathcal{P}$, for all $1 \leq k \leq \min_{P \in \mathcal{P}^q} |P|$, then a new flow is found, yielding a k-path which covers more vertices than $\mathcal{P}^q$. This will imply Conjecture 7.

In general digraphs, the subgraph $\mathcal{P}''$, corresponding to f'', may contain cycles and the proof may fail.

3.2 Details of Proof

We proceed to define the network N, the flow f corresponding to a path partition $\mathcal{P}$, the residual network N_f corresponding to f, and the criterion for either increasing k, or finding a flow f'' which contradicts the optimality of $\mathcal{P}^q$.

The network. We describe the network N as in [9]. Assume $G = (V, E)$ where $V = \{v_1, v_2, \ldots, v_n\}$, and $k \geq 1$. Associate a network $N = (\overline{V}, \overline{E}, a, c, s, t)$ with G as follows:

Let $\overline{V} = \{s, t, v_1', v_2', \ldots, v_n', v_1'', v_2'', \ldots, v_n''\}$, $\overline{E} = \{(s, v_i'); i = 1, 2, \ldots, n\} \cup \{(v_i'', t); i = 1, 2, \ldots, n\} \cup \{(v_i', v_j''); (v_i, v_j) \in E\} \cup \{(v_i', v_i''); 1 \leq i \leq n\} \cup \{(s, t)\}$.

All of the arc capacities $c(e)$ are equal to one, while the costs $a(e)$ are:

$$a(e) = \begin{cases} 1 \text{ if } e = (v_i', v_i'') \\ k \text{ if } e = (s, t) \\ 0 \text{ otherwise} \end{cases}$$

We denote the value of a feasible flow f by $\text{val}(f)$ and its cost by $\text{cost}(f)$.

Path partitions and flows. Since all the capacities are one we may assume that a feasible flow in N is integral. We define a *full* feasible flow as a flow which satisfies that for each $v_i \in V(G)$, at least one of the edges (s, v_i') or (v_i'', t) has non-zero flow. For example, a maximal flow (i.e. a flow f s.t. there exists no other flow f', $f \leq f'$) is a full flow.

Assume we have a full feasible flow f in N of value v. We associate with it a partition $\mathcal{P} = \mathcal{P}(f)$ of $V(G)$ into paths defined as follows: If $f(v_i', v_j'') = 1$, $i \neq j$, then $(v_i, v_j) \in E[\mathcal{P}]$, and if $f(v_i', v_i'') = 1$ then (v_i) is a trivial path in $\mathcal{P}$. Since f is full each vertex in $V(G)$ is covered by $\mathcal{P}$, and since all the capacities are one, $\mathcal{P}$ is indeed a collection of disjoint paths. If G is not acyclic, then $\mathcal{P}$ is a collection of disjoint paths and cycles.

If $\mathcal{P} = \mathcal{P}(f)$ is a path partition, and k is less than or equal to the smallest non-trivial path in $\mathcal{P}$, then

$$|\mathcal{P}|_k = k|\mathcal{P}^{\geq k}| + |\mathcal{P}^0| = k|\mathcal{P}^{>1}| + |\mathcal{P}^0| = k(n - \text{val}(f)) + \text{cost}(f) \qquad (1)$$

Conversely, given a path partition $\mathcal{P}$ in G, we associate with it the flow $f := f_{\mathcal{P}}$ defined by: If $(v_i) \in \mathcal{P}^0$ then $f(s, v_i') = f(v_i', v_i'') = f(v_i'', t) = 1$. For each $(v_i, v_j) \in E[\mathcal{P}]$ define $f(s, v_i') = f(v_i', v_j'') = f(v_j'', t) = 1$. The flow in all other edges is defined as zero. It is easy to check that $f_{\mathcal{P}}$ is a full feasible flow with $\text{val}(f) = n - |\mathcal{P}^{>1}|$, and $\text{cost}(f) = |\mathcal{P}^0|$. If $k \leq \min_{P \in \mathcal{P}^{>1}} |P|$, then the equation in formula (1) holds.

The residual network. For a given flow f in N, the *residual network* N_f is defined to be:

$$N_f = (\overline{V}, \overline{E_f}, a_f, c_f, s, t)$$

where

$$\overline{E_f} := \{e \in \overline{E}; f(e) < c(e)\} \cup \{\overleftarrow{e}\,; e \in \overline{E} \text{ and } f(e) > 0\}.$$

Here if $e = (u, v)$ then $\overleftarrow{e} = (v, u)$ is its reverse. The *residual capacity* $c_f : \overline{E_f} \to \mathbb{R}^+$ is defined as $c_f \equiv 1$. The cost function $a_f : E_f \to \mathbb{R}$ is defined as: $a_f(e) := a(e)$ for every $e \in \overline{E}$, and $a_f(\overleftarrow{e}) := -a(e)$ for every $e \in \overline{E}$.

Lemma 12 (see[2])

1. *If f is a feasible flow in a network N, and g is a feasible flow in the residual network N_f, then $f + g$ is a feasible flow in the original network N defined as follows: $(f + g)(e) = f(e) + g(e) - g(\overleftarrow{e})$ for every $e \in \overline{E}$. (If $e \notin \overline{E_f}$, $\overleftarrow{e} \notin \overline{E_f}$, we let $g(e) = 0$, $g(\overleftarrow{e}) = 0$, respectively). The flow $f + g$ satisfies* $\mathrm{val}(f + g) = \mathrm{val}(f) + \mathrm{val}(g)$ *and* $\mathrm{cost}(f + g) = \mathrm{cost}(f) + \mathrm{cost}(g)$.
2. *Similarly, if f, f' are two feasible flows in a network N, then $f' - f$ is a feasible flow in the residual network N_f of value* $\mathrm{val}(f') - \mathrm{val}(f)$ *and cost* $\mathrm{cost}(f') - \mathrm{cost}(f)$. *The flow $f' - f$ in N_f is defined as follows: If $e \in E \cap E_f$ then $(f' - f)(e) = (f'(e) - f(e))^+$ where $x^+ = max\{x, 0\}$. Similarly, if $e \in E$, and $\overleftarrow{e} \in E_f$, then $(f' - f)(\overleftarrow{e}) = (f(e) - f('(e))^+$.*
3. *Since all the capacities are one, $f' - f$ can be represented as the sum of flows along $s - t$ paths and cycles in N_f, each having flow value of* $0, 1$ *or* -1.

For a full feasible flow f define

$$w_k(f) := k \cdot \mathrm{val}(f) - \mathrm{cost}(f) \tag{2}$$

If f is the flow corresponding to a path partition $\mathcal{P}$, then by Equation (1), $|\mathcal{P}|_k = kn - w_k(f)$.

Lemma 13. *Let $\mathcal{P}$ and $\mathcal{P}'$ be path partitions with $|\mathcal{P}'|_k < |\mathcal{P}|_k$, and assume that all non-trivial paths in $\mathcal{P}$ and $\mathcal{P}'$ are of cardinality at least k. Let f and f' be feasible flows in N corresponding to $\mathcal{P}$ and $\mathcal{P}'$, respectively. Let $f' - f = f_1 + f_2 + ... + f_m$, where each f_i is an $s - t$-flow in N_f of value* $0, 1$ *or* -1. *Then there exists some f_{i_0} ($1 \le i_0 \le m$) with $w_k(f_{i_0}) > 0$.*

Proof. From $|\mathcal{P}'|_k - |\mathcal{P}|_k < 0$, it follows from (1) and (2) that $w_k(f') - w_k(f) > 0$. By Lemma 12,

$$w_k(f') - w_k(f) = k \cdot \mathrm{val}(f') - \mathrm{cost}(f') - (k \cdot \mathrm{val}(f) - \mathrm{cost}(f)) = k \cdot \mathrm{val}(f' - f) - \mathrm{cost}(f' - f) =$$

$$= w_k(f' - f) = k \cdot \mathrm{val}(\Sigma_{i=1}^m f_i) - \mathrm{cost}(\Sigma_{i=1}^m f_i) = \Sigma_{i=1}^m w_k(f_i) > 0$$

Hence, there must be some f_{i_0}, $(1 \le i_0 \le m)$ with $w_k(f_{i_0}) > 0$. ∎

The Algorithm and proof of correctness

0. **Input:** G , $q \geq 1$, $\mathcal{P}^q$
1. **Initialize:** $k \leftarrow 1$; $\mathcal{P} := \mathcal{P}^q \cup \{\{x\}; x \notin \cup\mathcal{P}^q\}$; $f \leftarrow f_{\mathcal{P}}$
2. **while** $(k \leq \min_{P \in \mathcal{P}^q} |P|)$ **do**
3. **begin**
4. **if** $(\exists \mathcal{C}^k$ orthogonal to $\mathcal{P})$ **then** **Stop**
5. Let $\mathcal{P}'$ with $|\mathcal{P}'|_k < |\mathcal{P}|_k$, and $w_k(f_{i_0}) > 0$. **If** $(\mathrm{val}(f_{i_0}) = 0$ or 1 $)$ **then** find improved $\mathcal{P}'^q$. **Stop**
6. **else** $(\mathrm{val}(f_{i_0}) = -1)$ **then** $k \leftarrow k + 1$
7. **end**

Remarks

1. In line 4, the Oracle, as implied by Conjecture 5, finds a k-colouring $\mathcal{C}^k$ orthogonal to $\mathcal{P}$. By Observation 9, $\mathcal{C} = \mathcal{C}^k \cup \{\{x\}; x \notin \cup\mathcal{C}^k\}$ is orthogonal to $\mathcal{P}^q$, and we are done.
2. Otherwise, (line 5) the Oracle finds a path partition $\mathcal{P}'$ with $|\mathcal{P}'|_k < |\mathcal{P}|_k$. We assume that all non-trivial paths in $\mathcal{P}'$ are of cardinality at least k. (Otherwise, we just break up paths of cardinality less than k into single vertices). Lemma 13 implies the existence of f_{i_0} with $\mathrm{val}(f_{i_0}) = 0, 1$ or -1 and $w_k(f_{i_0}) > 0$

Lemma 14. *Let $\mathcal{P}$ be a path partition and $f = f_{\mathcal{P}}$. If there exist flows f_{j_0} and f_{j_1} in N_f satisfying* $\mathrm{val}(f_{j_0}) = 1, \mathrm{val}(f_{j_1}) = -1$, $\mathrm{cost}(f_{j_0}) + \mathrm{cost}(f_{j_1}) < 0$, *then there exists a flow f' in N_f with* $\mathrm{val}(f') = 0$ *and* $\mathrm{cost}(f') < 0$.

Proof. If f_{j_0} and f_{j_1} are disjoint, then $f' = f_{j_0} + f_{j_1}$ is a flow satisfying $\mathrm{val}(f') = \mathrm{val}(f_{j_0}) + \mathrm{val}(f_{j_1}) = 0$ and $\mathrm{cost}(f') = \mathrm{cost}(f_{j_0}) + \mathrm{cost}(f_{j_1}) < 0$. Otherwise, $f_{j_0} + f_{j_1}$ is a collection of cycles (not necessarily disjoint!) of total negative cost. One of these cycles must have a negative cost, and corresponds to a flow f' in N_f with $\mathrm{val}(f') = 0$ and $\mathrm{cost}(f') < 0$. ■

We are now ready to prove the correctness of the algorithm:

Theorem 15. *Given a q-path system $\mathcal{P}^q$, the algorithm above either finds a colouring $\mathcal{C}$ orthogonal to it, or a q-path system $\mathcal{P}''^q$ with $|\bigcup \mathcal{P}''^q| > |\bigcup \mathcal{P}^q|$.*

Proof. In line 1, k is initialized to 1, the path partition $\mathcal{P}$ associated with $\mathcal{P}^q$ is constructed, and a flow f corresponding to $\mathcal{P}$ is defined in N. If the algorithm stops at line 4, then by Remark (1) above we are done. Otherwise, the Oracle finds a path partition $\mathcal{P}'$ with $|\mathcal{P}'|_k < |\mathcal{P}|_k$. Let f_{i_0} be the flow (of value 0,1 or -1) as implied in Lemma 13. Let $f'' := f + f_{i_0}$, and $\mathcal{P}'' := \mathcal{P}(f'')$.

Case 1: Assume $\mathrm{val}(f_{i_0}) = 0$. Since $w_k(f_{i_0}) > 0$ it follows that $\mathrm{cost}(f_{i_0}) < 0$ and $f'' = f + f_{i_0}$ is a flow with $\mathrm{val}(f'') = \mathrm{val}(f)$ and $\mathrm{cost}(f'') < \mathrm{cost}(f)$.

Then $|\mathcal{P}''^{>1}| = |\mathcal{P}^{>1}| = n - \text{val}(f) \leq q$. However, $|\bigcup \mathcal{P}''^{>1}| = n - \text{cost}(f'') > n - \text{cost}(f) = |\bigcup \mathcal{P}^q|$. If $\mathcal{P}^q$ contains no trivial paths then $\mathcal{P}''^{>1}$ is a q-path system with $|\bigcup \mathcal{P}''^q| > |\bigcup \mathcal{P}^q|$, and we are done. Otherwise, we add the necessary number of trivial paths to $\mathcal{P}''^{>1}$ to make it a q-path system, and we are done again.

Case 2: If $\text{val}(f_{i_0}) = -1$ then k is increased unless $k = \min_{P \in \mathcal{P}^q} |P|$. Assume $k = \min_{P \in \mathcal{P}^q} |P|$. From $w_k(f_{i_0}) > 0$ we deduce that $\text{cost}(f_{i_0}) \leq -k - 1$. If $k = 1$ then $\mathcal{P}$ contains t trivial paths, for some $t \geq 1$, and $q - t$ non-trivial paths. Now $|\mathcal{P}''^{>1}| = n - \text{val}(f'') = n - (\text{val}(f) + \text{val}(f_{i_0})) = n - \text{val}(f) + 1 = |\mathcal{P}^{>1}| + 1 = q - t + 1$. But $\text{cost}(f_{i_0}) \leq -2$, implying that $\mathcal{P}''^{>1}$ in addition to $t - 1$ trivial paths is a q-path system which covers at least one more vertex than $\mathcal{P}^q$.

Assume now that $k \geq 2$. Let $P = (v_1, v_2, \ldots, v_k)$ be the shortest path in $\mathcal{P}^q$. Then $f_{i_1} = (s, v'_k, v''_k, v'_{k-1}, v''_{k-1}, \ldots, v'_1, v''_1, t)$ is a flow in N_f with $\text{val}(f_{i_1}) = 1$ and $\text{cost}(f_{i_1}) = k$. By applying Lemma 14 on the flows f_{i_1} and f_{i_0} we are guaranteed the existence of a flow f' in N_f with $\text{val}(f') = 0$ and $\text{cost}(f') < 0$. We let $f'' := f + f'$. As was shown in Case 1, $\mathcal{P}''^q = \mathcal{P}''^{>1}$ is a q-path system which covers more vertices than $\mathcal{P}^q$.

Case 3.1: If $\text{val}(f_{i_0}) = 1$ and $k = 1$, then $\text{cost}(f_{i_0}) \leq 0$. Then $|\mathcal{P}''^{>1}| = n - \text{val}(f'') = n - \text{val}(f) - 1 \leq q - 1$, and $\mathcal{P}''^{>1}$ covers $n - \text{cost}(f'') \geq n - \text{cost}(f) = |\bigcup \mathcal{P}^q|$ vertices. If we add any path from $G - \bigcup \mathcal{P}''^{>1}$ to the family $\mathcal{P}''^{>1}$, we have a q-path system which covers more vertices than $\mathcal{P}^q$, and we are done.

Case 3.2: Finally, if $\text{val}(f_{i_0}) = 1$ and $k \geq 2$, then k was increased to its current value because there exists a flow f_{i_1} with $\text{val}(f_{i_1}) = -1$ and $w_{k-1}(f_{i_1}) > 0$. From $w_k(f_{i_0}) > 0$, and $w_{k-1}(f_{i_1}) > 0$ we deduce that $\text{cost}(f_{i_0}) < k$ and $\text{cost}(f_{i_1}) < -(k-1)$. By Lemma 14, there exists a flow f' with $\text{val}(f') = 0$ and $\text{cost}(f') < 0$. The rest follows as in Case 1. This completes the proof.

■

4 When G Is Not Acyclic

In an arbitrary digraph, which is not necessarily acyclic, we have no guarantee that $f'' := f + f_{i_0}$ (where f_{i_0} was defined in Lemma 13) corresponds to a path partition. It may correspond to a partition of paths and cycles. However, we do know that $f' = f + \Sigma_{i=1}^m f_i$ does correspond to a path partition (the partition $\mathcal{P}'$!). Perhaps f_{i_0} can be replaced by a collection of flows, thus yielding the following conjecture:

Conjecture 16. *Let G be digraph, $k \geq 1$. Assume $\mathcal{P}$ and $\mathcal{P}'$ are path partitions in G and $|\mathcal{P}'|_k < |\mathcal{P}|_k$. We assume that all non-trivial paths in $\mathcal{P}$ and $\mathcal{P}'$ are of cardinality at least k. Let f (f') be the corresponding flow $f_{\mathcal{P}}$ (respectively $f'_{\mathcal{P}}$) in our network N. Let $f' - f = \sum_{i=1}^m f_i$ be a decomposition of $f' - f$ such that for*

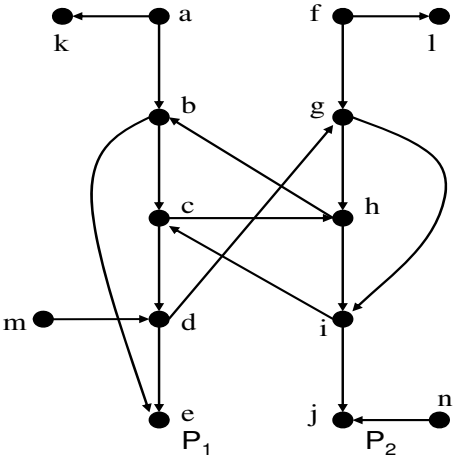


Fig. 1. G. $\mathcal{P}^2 = \{P_1, P_2\}$

any i, $f_i \in N_f$ *and* $\mathrm{val}(f_i) \in \{0, -1, 1\}$. *Then there exists a subset* $I \subseteq \{1, ..., m\}$ *such that the flow* $S = \sum_{i \in I} f_i$ *satisfies:*
(i) $S \in N_f$
(ii) $\mathcal{P}(f + S)$ *is acyclic*
(iii) $w_k(S) > 0$
(iv) $\mathrm{val}(S) \in \{0, -1, 1\}$

Mercier[8] has found a counterexample to this conjecture. Consider the graph in Figure 1. Let $k = 1$, $\mathcal{P} = \{(a, b, c, d, e), (f, g, h, i, j), (k), (l), (m), (n)\}$ and $\mathcal{P}' = \{(a, k), (f, l), (n, j), (m, d, g, i, c, h, b, e)\}$.
Let $f_1 = (t, n'', n', j'', i', c'', b', e'', d', g'', f', l'', l', s)$, and $f_2 = (t, m'', m', d'', c', h'', g', i'', h', b'', a', k'', k', s)$. It is easy to verify that $f' - f = f_1 + f_2$, where f' and f correspond to the path partitions $\mathcal{P}'$ and $\mathcal{P}$, respectively. Since both f_1 and f_2 are flows in N_f from t to s, it follows that $\mathrm{val}(f_1 + f_2) = -2$. However, $\mathcal{P}(f + f_i)$ contains a cycle, for each $i = 1, 2$. This contradicts Conjecture 16.

Acknowledgment. I thank Eli Berger, and Fabien Mercier for their helpful comments on this paper.

References

1. Aharoni, R., Ben-Arroyo Hartman, I., Hoffman, A.J.: Path partitions and packs of acyclic digraphs. Pacific Journal of Mathematics 118(2), 249–259 (1985)
2. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows. Prentice Hall, Englewood Cliffs (1993)

3. Berge, C.: k-optimal partitions of a directed graph. Europ. J. Combinatorics 3, 97–101 (1982)
4. Berger, E., Hartman, I.B.-A.: Proof of Berge's Strong Path Partition Conjecture for $k = 2$. Europ. J. Combinatorics 29(1), 179–192 (2008)
5. Cameron, K.: On k-optimum dipath partitions and partial k-colourings of acyclic digraphs. Europ. J. Combinatorics 7, 115–118 (1986)
6. Cameron, K.: Gallai-type min-max inequalities. In: Proceedings of the Third China-USA International Conference on Graph Theory, Combinatorics, Computing and Applications, pp. 7–16. World Scientific, Singapore (1994)
7. Dilworth, P.: A decomposition theorem for partially ordered sets. Ann. of Math. 51(1), 161–166 (1950)
8. Mercier, F.: Personal communication
9. Frank, A.: On chain and antichain families of a partially ordered set. J. Comb. Theory, Ser. B 29, 176–184 (1980)
10. Gallai, T.: On directed paths and circuits. In: Erdos, P., Katona, G. (eds.) Theory of Graphs, pp. 115–118. Academic Press, New York (1968)
11. Gallai, T., Milgram, A.N.: Verallgemeinerung eines graphentheoretischen satzes von Redei. Acta Sc. Math. 21, 181–186 (1960)
12. Greene, C.: Some partitions associated with a partially ordered set. J. Combinatorial Theory, Ser. A 20, 69–79 (1976)
13. Greene, C., Kleitman, D.J.: The structure of Sperner k-families. J. Combinatorial Theory, Ser. A 20, 41–68 (1976)
14. Hartman, I.B.-A.: Berge's conjecture on directed path partitions - a survey. Discrete Math. 306, 2498–2514 (2006)
15. Linial, N.: Extending the Greene-Kleitman theorem to directed graphs. J. Combinatorial Theory, Ser. A 30, 331–334 (1981)
16. Roy, B.: Nombre chromatique et plus longs chemins, Rev. F1. Automat. Informat. 1, 127–132 (1976)
17. Saks, M.: A short proof of the k-saturated partitions. Adv. In Math. 33, 207–211 (1979)

On Related Edges in Well-Covered Graphs without Cycles of Length 4 and 6

Vadim E. Levit[1] and David Tankus[2]

[1] Ariel University Center of Samaria, Israel
levitv@ariel.ac.il
[2] Sami Shamoon College of Engineering, Israel
davidt@sce.ac.il

Abstract. A graph is *well-covered* if every maximal independent set has the same cardinality. The recognition problem of well-covered graphs is known to be **co-NPC**. The complexity status of the problem is not known if the input is restricted to graphs with no cycles of length 4. We conjecture that the problem is polynomial if the input graph does not contain cycles of length 4 and 6, and prove several theorems supporting our conjecture.

1 Introduction

Throughout this paper $G = (V, E)$ is a simple (i.e., a finite, undirected, loopless and without multiple edges) graph with vertex set $V = V(G)$ and edge set $E = E(G)$.

Let $S \subseteq V$ be a set of vertices, and let $i \in \mathbb{N}$. Then

$$N_i(S) = \{w \in V |\ min_{s \in S}\ d(w, s) = i\},$$

where $d(x, y)$ is the minimal number of edges required to construct a path between x and y. If $i \neq j$ then obviously $N_i(S) \cap N_j(S) = \phi$. If $S = \{v\}$ for some $v \in V$, then $N_i(\{v\})$ is abbreviated to $N_i(v)$.

A set of vertices $S \subseteq V$ is *independent* if for every $x, y \in S$, x and y are not adjacent. It is clear that an empty set is independent. The *independence number* of G, denoted by $\alpha(G)$, is the cardinality of the maximum size independent set in the graph.

A graph is *well-covered* if every maximal independent set has the same cardinality, $\alpha(G)$.

Let $T \subseteq V$. Then S *dominates* T if $S \cup N_1(S) \supseteq T$. If S and T are both empty, then $N_1(S) = \phi$, and therefore S dominates T. If S is a maximal independent set of G, then it dominates the whole graph.

Two adjacent vertices, x and y, in G are said to be *related* if there is an independent set S, containing neither x nor y, such that $S \cup \{x\}$ and $S \cup \{y\}$ are both maximal independent sets in the graph.

The recognition of well-covered graphs is known to be **co-NPC**. The problem remains **co-NPC** even when the input is restricted to $K_{1,4}$-free graphs [3].

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 144–147, 2009.

However, the problem is polynomially solvable for $K_{1,3}$-free graphs [5], [6]; for graphs with girth at least 5 (see [4]); or for graphs with a bounded maximal degree [2].

In [1], Brown, Nowakowski and Zverovich investigate well-covered graphs with no cycles of length 4. They denote such graphs by $\mathcal{WC}(\widehat{C}_4)$, and prove the following theorem.

Theorem 1. *Let $G \in \mathcal{WC}(\widehat{C}_4)$. If xy is an edge in G, but x and y are not related, then $G - xy$ is well-covered and $\alpha(G) = \alpha(G - xy)$.*

In [1] the following open problem is presented.

Problem 1. What is the complexity of determining whether an input graph with no cycles of length 4 is well-covered?

2 Graphs without Cycles of Length 4

In this section we continue the investigation of the structure of well-covered graphs with no cycles of length 4.

Theorem 2. *Let $G = (V, E)$ be a graph which does not contain cycles of length 4, and let $xy \in E$. Suppose $G - xy \in \mathcal{WC}(\widehat{C}_4)$. Then the following two conditions are equivalent:*

1. *There is no independent set of*
   $N_2(x) \cap N_3(y)$, which dominates $N_1(x) \cap N_2(y)$,
   and
   there is no independent set of
   $N_2(y) \cap N_3(x)$, which dominates $N_1(y) \cap N_2(x)$.
2. *$G \in \mathcal{WC}(\widehat{C}_4)$ and $\alpha(G) = \alpha(G - xy)$.*

Proof. Assume that Condition 1 holds. Let S be any maximal independent set of G. It is enough to prove that $|S| = \alpha(G - xy)$.

If $|S \cap \{(x, y)\}| = 0$, then S is a maximal independent set of $G - xy$, and therefore $|S| = \alpha(G - xy)$.

Suppose $|S \cap \{(x, y)\}| = 1$, and assume, without loss of generality, that $x \in S$. Condition 1 implies that y is dominated by $S - \{x\}$. Hence, S is a maximal independent set of $G - xy$, and $|S| = \alpha(G - xy)$.

Assume on the contrary that Condition 2 holds but Condition 1 does not. Assume, without loss of generality, that there exists an independent set $I \subseteq N_2(y) - N_1(x)$ which dominates $N_1(y) - \{x\}$. Let S be any maximal independent set of $G - xy$ which contains $I \cup \{x, y\}$. Hence, $S - \{y\}$ is a maximal independent set of G, which contradicts Condition 2.

3 Graphs without Cycles of Length 4 and 6

In this section our discussion is restricted to graphs which contain neither cycles of length 4 nor cycles of length 6. For such graphs, Condition 1 of the Theorem

2 is equivalent to the claim that x and y are not related. Thus, in this context Theorem 2 proves the opposite direction of Theorem 1.

Let $\mathcal{WC}(\widehat{C}_4, \widehat{C}_6)$ be the set of well-covered graphs which contain neither cycles of length 4 nor cycles of length 6.

Theorem 3. *The following problem can be solved in polynomial time:*

Input: A graph $G = (V, E) \in \mathcal{WC}(\widehat{C}_4, \widehat{C}_6)$, and an edge $xy \in E$.
Question: Are x and y related?

Proof. For every $v \in \{x, y\}$, let $u = \{x, y\} - \{v\}$, and define:
$M_1(v) = N_1(v) \cap N_2(u)$, $M_2(v) = N_1(M_1(v)) - \{v\}$.

The vertices x and y are related if and only if there exists an independent set in $M_2(x) \cup M_2(y)$ which dominates $M_1(x) \cup M_1(y)$.

The fact that the graph does not contain cycles of length 6 implies the following 3 conclusions:

- There are no edges which connect vertices of $M_2(x)$ with vertices of $M_2(y)$.
- The set $M_2(x) \cap M_2(y)$ is independent.
- There are no edges between $M_2(x) \cap M_2(y)$ and other vertices of $M_2(x) \cup M_2(y)$.

Hence, if $S_x \subseteq M_2(x)$ and $S_y \subseteq M_2(y)$ are independent, then $S_x \cup S_y$ is independent, as well. Therefore, it is enough to prove that one can decide in polynomial time whether there exists an independent set in $M_2(v)$ which dominates $M_1(v)$, where $v \in \{x, y\}$.

Let v be any vertex in $\{x, y\}$. Every vertex of $M_2(v)$ is adjacent to exactly one vertex of $M_1(v)$, or otherwise the graph contains a C_4. Every connectivity component of $M_2(v)$ contains at most 2 vertices, or otherwise the graph contains either a C_4 or a C_6. Let $A_1, ..., A_k$ be the connecivity components of $M_2(v)$.

Define a flow network $F_v = \{G_F = (V_F, E_F), s \in V_F, t \in V_F, w : E_F \longrightarrow R\}$ as follows:

Let $V_F = M_1(v) \cup M_2(v) \cup \{a_1, ..., a_k, s, t\}$, where $a_1, ..., a_k, s, t$ are new vertices, s and t are the source and sink of the network, respectively.

The directed edges E_F are:

- the directed edges from s to each vertex of $M_1(v)$;
- all directed edges v_1v_2 s.t. $v_1 \in M_1(v)$, $v_2 \in M_2(v)$ and $v_1v_2 \in E$;
- the directed edges va_i, for each $1 \leq i \leq k$ and for each $v \in A_i$;
- the directed edges a_it, for each $1 \leq i \leq k$.

Let $w \equiv 1$. Invoke any polynomial time algorithm for finding a maximum flow in the network, for example Ford and Fulkerson's algorithm. Let S_v be the set of vertices in $M_2(v)$ in which there is a positive flow. Clearly, S_v is independent. The maximality of S_v implies that $|M_1(v) \cap N_1(S_v)| \geq |M_1(v) \cap N_1(S'_v)|$, for any independent set S'_v of $M_2(v)$.

Let us conclude the proof with a brief sketch of the algorithm as a whole.

For each $v \in \{x, y\}$, build a flow network F_v as described above, and find a maximum flow. Let S_v be the set of vertices in $M_2(v)$ in which there is a positive flow. If S_v does not dominate $M_1(v)$ the algorithm terminates announcing that x and y are not related. Otherwise, let S be any maximal independent set of $G - \{x, y\}$ which contains $S_x \cup S_y$. Each of $S \cup \{x\}$ and $S \cup \{y\}$ is a maximal independent set of G, and x, y are related. This algorithm can be implemented in polynomial time.

4 Conclusions

Our main conjecture reads as follows.

Conjecture 1. The following recognition problem can be solved in polynomial time:

Input: A graph G which does not contain cycles of length 4 or 6.
Question: Is G well-covered?

Moreover, we assume that this recognition problem may be solved by the following algorithm.

Algorithm 4. *Algorithm for deciding whether* $G \in \mathcal{WC}(\widehat{C}_4, \widehat{C}_6)$.

- *For each edge* e *in the graph* G*, decide whether* e *is related.*
  By Theorem 3 this task may be implemented in polynomial time.
- *Delete all non-related edges from the graph* G*.*
  Denote the obtained graph by G^**.*
- *Decide whether* $G^* \in \mathcal{WC}(\widehat{C}_4, \widehat{C}_6)$*.*
- *If* $G^* \notin \mathcal{WC}(\widehat{C}_4, \widehat{C}_6)$ *then* $G \notin \mathcal{WC}(\widehat{C}_4, \widehat{C}_6)$*.*
- *If* $G^* \in \mathcal{WC}(\widehat{C}_4, \widehat{C}_6)$ *then* $G \in \mathcal{WC}(\widehat{C}_4, \widehat{C}_6)$*.*

References

1. Brown, J.I., Nowakowski, R.J., Zverovich, I.E.: The structure of well-covered graphs with no cycles of length 4. Discrete Mathematics 307, 2235–2245 (2007)
2. Caro, Y., Ellingham, N., Ramey, G.F.: Local structure when all maximal independent sets have equal weight. SIAM Journal on Discrete Mathematics 11, 644–654 (1998)
3. Caro, Y., Sebő, A., Tarsi, M.: Recognizing greedy structures. Journal of Algorithms 20, 137–156 (1996)
4. Finbow, A., Hartnell, B., Nowakowski, R.: A characterization of well-covered graphs of girth 5 or greater. Journal of Combinatorial Theory Ser. B 57, 44–68 (1993)
5. Tankus, D., Tarsi, M.: Well-covered claw-free graphs. Journal of Combinatorial Theory Ser. B 66, 293–302 (1996)
6. Tankus, D., Tarsi, M.: The structure of well-covered graphs and the complexity of their recognition problems. Journal of Combinatorial Theory Ser. B 69, 230–233 (1997)

On the Cubicity of AT-Free Graphs and Circular-Arc Graphs*

L. Sunil Chandran[1], Mathew C. Francis[1], and Naveen Sivadasan[2]

[1] Dept. of Computer Science and Automation, Indian Institute of Science, Bangalore–560 012, India
{mathew,sunil}@csa.iisc.ernet.in
[2] Advanced Technology Centre, TCS, Deccan Park, Madhapur, Hyderabad–500 081, India
s.naveen@atc.tcs.com

Abstract. A unit cube in k dimensions (k-cube) is defined as the Cartesian product $R_1 \times R_2 \times \cdots \times R_k$ where R_i(for $1 \leq i \leq k$) is a closed interval of the form $[a_i, a_i + 1]$ on the real line. A graph G on n nodes is said to be representable as the intersection of k-cubes (cube representation in k dimensions) if each vertex of G can be mapped to a k-cube such that two vertices are adjacent in G if and only if their corresponding k-cubes have a non-empty intersection. The *cubicity* of G denoted as cub(G) is the minimum k for which G can be represented as the intersection of k-cubes.

An interesting aspect about cubicity is that many problems known to be NP-complete for general graphs have polynomial time deterministic algorithms or have good approximation ratios in graphs of low cubicity. In most of these algorithms, computing a low dimensional cube representation of the given graph is usually the first step.

We give an $O(bw \cdot n)$ algorithm to compute the cube representation of a general graph G in $bw + 1$ dimensions given a bandwidth ordering of the vertices of G, where bw is the *bandwidth* of G. As a consequence, we get $O(\Delta)$ upper bounds on the cubicity of many well-known graph classes such as AT-free graphs, circular-arc graphs and cocomparability graphs which have $O(\Delta)$ bandwidth. Thus we have:

1. cub(G) $\leq 3\Delta - 1$, if G is an AT-free graph.
2. cub(G) $\leq 2\Delta + 1$, if G is a circular-arc graph.
3. cub(G) $\leq 2\Delta$, if G is a cocomparability graph.

Also for these graph classes, there are constant factor approximation algorithms for bandwidth computation that generate orderings of vertices with $O(\Delta)$ width. We can thus generate the cube representation of such graphs in $O(\Delta)$ dimensions in polynomial time.

Keywords: Cubicity, bandwidth, intersection graphs, AT-free graphs, circular-arc graphs, cocomparability graphs.

* The work done by the first and second authors was partially supported by a DST grant SR/S3/EECE/62/2006.

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 148–157, 2009.

1 Introduction

Let $\mathcal{F} = \{S_x \subseteq U : x \in V\}$ be a family of subsets of a universe U, where V is an index set. The intersection graph $\Omega(\mathcal{F})$ of $\mathcal{F}$ has V as vertex set, and two distinct vertices x and y are adjacent if and only if $S_x \cap S_y \neq \emptyset$. Representations of graphs as the intersection graphs of various geometrical objects is a well studied topic in graph theory. Probably the most well studied class of intersection graphs are the *interval graphs*, where each S_x is a closed interval on the real line. A restricted form of interval graphs, that allow only intervals of unit length, are *indifference graphs*, also called *unit interval graphs*. Other characterizations of these graphs including their equivalence with *proper interval graphs* are detailed in [1].

A well known concept in this area of graph theory is the *cubicity*, which was introduced by F. S. Roberts in 1969 [2]. This concept generalizes the concept of indifference graphs. A unit cube in k dimensions (k-cube) is a Cartesian product $R_1 \times R_2 \times \cdots \times R_k$ where R_i (for $1 \leq i \leq k$) is a closed interval of the form $[a_i, a_i + 1]$ on the real line. Two k-cubes, $(x_1, x_2, \ldots, x_k)$ and $(y_1, y_2, \ldots, y_k)$ are said to have a non-empty intersection if and only if the intervals x_i and y_i have a non-empty intersection for $1 \leq i \leq k$. For a graph G, its *cubicity* is the minimum dimension k, such that G is representable as the intersection graph of k-cubes. We denote the cubicity of a graph G by $\text{cub}(G)$. The graphs of cubicity at most 1 are exactly the class of indifference graphs.

If we require that each vertex correspond to a k-dimensional axis-parallel box $R_1 \times R_2 \times \cdots \times R_k$ where R_i (for $1 \leq i \leq k$) is a closed interval of the form $[a_i, b_i]$ on the real line, then the minimum dimension required to represent G is called its *boxicity* denoted as $\text{box}(G)$. Clearly $\text{box}(G) \leq \text{cub}(G)$ for any graph G because cubicity is a stricter notion than boxicity.

It has been shown that deciding whether the cubicity of a given graph is at least 3 is NP-hard [3].

In many algorithmic problems related to graphs, the availability of certain convenient representations turn out to be extremely useful. Probably, the most well-known and important examples are the tree decompositions and path decompositions. Many NP-hard problems are known to be polynomial time solvable given a tree(path) decomposition of the input graph that has bounded width. Similarly, the representation of graphs as intersections of "disks" or "spheres" lies at the core of solving problems related to frequency assignments in radio networks, computing molecular conformations etc. For the maximum independent set problem which is hard to approximate within a factor of $n^{(1/2)-\epsilon}$ for general graphs, a PTAS is known for disk graphs given the disk representation [4,5] and an FPTAS is known for unit disk graphs [6]. In a similar way, the availability of cube or box representation in low dimension make some well known NP hard problems like the max-clique problem, polynomial time solvable since there are only $O((2n)^k)$ maximal cliques if the boxicity or cubicity is at most k. Though the complexity of finding the maximum independent set is hard to approximate within a factor $n^{(1/2)-\epsilon}$ for general graphs, it is approximable to a $\log n$ factor

for boxicity 2 graphs (the problem is NP-hard even for boxicity 2 graphs) given a box or cube representation [7,8].

It is easy to see that the problem of representing graphs using k-cubes can be equivalently formulated as the following geometric embedding problem. Given an undirected unweighted graph $G = (V, E)$ and a threshold t, find an embedding $f : V \rightarrow \mathbb{R}^k$ of the vertices of G into a k-dimensional space (for the minimum possible k) such that for any two vertices u and v of G, $||f(u) - f(v)||_\infty \leq t$ if and only if u and v are adjacent where $||\ ||_\infty$ denotes the L_∞ norm. Clearly, a k-cube representation of G yields the required embedding of G in the k-dimensional space. The minimum dimension required to embed G as above under the L_2 norm is called the *sphericity* of G. Refer [9] for applications where such an embedding under L_∞ norm is argued to be more appropriate than embedding under L_2 norm. The connection between cubicity and sphericity of graphs were studied in [10,11].

As far as we know, the only known upper bound for the cubicity of general graphs (existential or constructive) is by Roberts [2], who showed that $\text{cub}(G) \leq 2n/3$ for any graph G on n vertices. The cube representation of special class of graphs like hypercubes and complete multipartite graphs were investigated in [2,11,12]. A lower bound for the cubicity of general graphs was given [13]. The cubicity of interval graphs was shown to be bounded above by $\lceil \log \Delta \rceil + 4$ in [14].

Linear Ordering and Bandwidth. Given an undirected graph $G = (V, E)$ on n vertices, a *linear ordering* of G is a bijection $f : V \rightarrow \{1, \ldots, n\}$. The *width* of the linear ordering f is defined as $\max_{(u,v)\in E} |f(u) - f(v)|$. The *bandwidth minimization problem* is to compute f with minimum possible width. The *bandwidth* of G denoted as $\text{bw}(G)$ is the minimum possible width achieved by any linear ordering of G. A *bandwidth ordering* of G is a linear ordering of G with width $\text{bw}(G)$. Our algorithm to compute the cube representation of a graph G takes as input a linear ordering of G. The smaller the width of this ordering, the lesser the number of dimensions of the cube representation of G computed by our algorithm. It is NP-hard to approximate the bandwidth of G within a ratio better than k for every $k \in \mathbb{N}$ [15]. Feige [16] gives a $O(\log^3(n)\sqrt{\log n \log\log n})$ approximation algorithm to compute the bandwidth (and also the corresponding linear ordering) of general graphs. For bandwidth computation, several algorithms with good heuristics are known that perform very well in practice [17].

1.1 Our Results

We summarize below the results of this paper.

1. For any graph G, $\text{cub}(G) \leq \text{bw}(G) + 1$
2. For an AT-free graph G with maximum degree Δ, $\text{cub}(G) \leq 3\Delta - 1$
3. For a circular-arc graph G with maximum degree Δ, $\text{cub}(G) \leq 2\Delta + 1$
4. For a cocomparability graph G with maximum degree Δ, $\text{cub}(G) \leq 2\Delta$

1.2 Definitions and Notations

All the graphs that we consider will be simple, finite and undirected. For a graph G, we denote the vertex set of G by $V(G)$ and the edge set of G by $E(G)$. For a vertex $u \in V(G)$, let $d(u)$ denote its degree (the number of outer neighbors of u). The maximum degree of G is denoted by $\Delta(G)$ or simply Δ when the graph under consideration is clear. For a vertex $u \in V(G)$, we denote the set of neighbours of u by $N_G(u)$. By definition, $N_G(u) = \{v \in V(G) \mid (u, v) \in E(G)\}$. Again, for ease of notation, we use $N(u)$ instead of $N_G(u)$ when there is no scope for ambiguity. Let G' be a graph such that $V(G') = V(G)$. Then G' is a *supergraph* of G if $E(G) \subseteq E(G')$. We define the *intersection* of two graphs as follows. If G_1 and G_2 are two graphs such that $V(G_1) = V(G_2)$, then the intersection of G_1 and G_2 denoted as $G = G_1 \cap G_2$ is the graph with $V(G) = V(G_1) = V(G_2)$ and $E(G) = E(G_1) \cap E(G_2)$. Let $\text{bw}(G)$ denote the bandwidth of G.

An indifference graph is an interval graph which has an interval representation that maps the vertices to unit length intervals on the real line such that two vertices are adjacent in the graph if and only if the intervals mapped to them overlap.

Definition 1 (Unit interval representation). *Given an indifference graph $I(V, E)$, the unit interval representation is a mapping $f : V(I) \to \mathbb{R}$ such that for any two vertices u, v, $|f(u) - f(v)| \leq 1$ if and only if $(u, v) \in E(I)$.*

Note that this is equivalent to mapping each vertex of I to the unit interval $[f(u), f(u) + 1]$ so that two vertices are adjacent in I if and only if the unit intervals mapped to them overlap. Now, consider the mapping $g : V(I) \to \mathbb{R}$ given by $g(u) = xf(u)$ where $x \in \mathbb{R}$. It can be easily seen that for any two vertices u, v, $|g(u) - g(v)| \leq x$ if and only if (u, v) is an edge in I. g thus corresponds to an interval representation of I using intervals of length x. We call such a mapping g a *unit interval representation of I with interval length x*. We can thus alternatively define indifference graphs as follows: a graph $G(V, E)$ is an indifference graphs if and only if there exists a function $f : V(G) \to \mathbb{R}$ and a constant t such that for $u, v \in V(G)$, $(u, v) \in E(G) \Leftrightarrow |f(u) - f(v)| \leq t$.

Definition 2 (Indifference graph representation). *The indifference graphs $I_1, \ldots, I_k$ constitute an indifference graph representation of a graph G if $G = I_1 \cap \cdots \cap I_k$.*

Theorem 1 (Roberts[2]). *A graph G has $\text{cub}(G) \leq k$ if and only if it has an indifference graph representation with k indifference graphs.*

2 Cubicity and Bandwidth

2.1 The Construction

We show that given a linear ordering of the vertices of G with width b, we can construct an indifference graph representation of G using $b + 1$ indifference graphs.

Theorem 2. *If G is any graph with bandwidth b, then* $\mathrm{cub}(G) \leq b+1$.

Proof. Let n denote $|V(G)|$ and let $\mathcal{A} = u_0, u_1, \ldots, u_{n-1}$ be a linear ordering of the vertices of G with width b. We will assume that $n \geq b$; otherwise, Roberts' result[2] gives us $\mathrm{cub}(G) \leq 2n/3 \leq 2b/3$ giving us a better bound than required. Since $\mathcal{A}$ has width b, if $(u_j, u_k) \in E(G)$, then $|j-k| \leq b$. For two vertices $u_j, u_k \in V(G)$, we will abuse notation to say that $u_j < u_k$ if $j < k$ and $u_j > u_k$ if $j > k$. The relations $\leq$ and $\geq$ on $V(G)$ are also defined similarly.

We construct $b+1$ indifference graphs $I_0, I_1, \ldots, I_{b-1}$ and H, such that $G = I_0 \cap I_1 \cap \cdots \cap I_{b-1} \cap H$.

Construction of indifference graph H

The vertex set of H is $V(G)$ and let its edge set be denoted by $E(H)$. Since H has to be a supergraph of G, we have to make sure that every edge in $E(G)$ has to be present in $E(H)$. b being the bandwidth of the linear ordering $\mathcal{A}$ of vertices taken, a vertex u_j is not adjacent in G to any vertex u_k when $|j-k| > b$. Let the function $h : V(G) \to \mathbb{R}$ be the unit interval representation for H with interval length b, i.e., for $u_j, u_k \in V(G)$, $(u_j, u_k) \in E(H) \Leftrightarrow |h(u_j) - h(u_k)| \leq b$. We construct h in such a way that $E(H) = \{(u_j, u_k) \mid |j-k| < b\} \cup \{(u_j, u_k) \mid |j-k| = b \text{ and } (u_j, u_k) \in E(G)\}$. h is defined as:
Let $\epsilon = 1/n^2$.

$$
\begin{aligned}
&h(u_j) = j, \text{ for } j < b \\
&h(u_j) = h(u_{j-b}) + b, \text{ for } j \geq b \text{ and } (u_{j-b}, u_j) \in E(G) \\
&h(u_j) = h(u_{j-b}) + b + \epsilon, \text{ for } j \geq b \text{ and } (u_{j-b}, u_j) \notin E(G)
\end{aligned}
$$

The definition of h can be explained as the following iterative procedure. We first assign the interval $[j, j+b]$ to vertex u_j, for all j. This makes sure that u_j is not adjacent to any vertex u_k, if $k > j+b$. Now, each vertex is adjacent in H to exactly the b vertices preceding and following it in $\mathcal{A}$. Now, for each vertex u_j where $j \geq b$, we shift $h(u_j)$, the interval for u_j, slightly to the right (by ϵ) if u_j is not adjacent to u_{j-b} in G so that $h(u_j)$ becomes disjoint from $h(u_{j-b})$. Along with $h(u_j)$, all the intervals that start after $h(u_j)$ are also shifted right by ϵ. This procedure is done for vertices $u_b, \ldots, u_{n-1}$ in that order. Our choice of a small value for ϵ ensures that h is still a supergraph of G, as it will be shown later.

Note that for a vertex u_j,

$$
\begin{aligned}
h(u_j) &\leq h(u_{j-b}) + b + \epsilon \\
&\leq h(u_{j-2b}) + 2b + 2\epsilon \leq \cdots \leq h(u_{j \mod b}) + \lfloor j/b \rfloor b + \lfloor j/b \rfloor \epsilon \\
&= j \mod b + \lfloor j/b \rfloor b + \lfloor j/b \rfloor \epsilon \\
&= j + \lfloor j/b \rfloor \epsilon \\
&\leq j + n\epsilon = j + 1/n
\end{aligned}
$$

Claim. H is a supergraph of G.

Proof. First we observe that for any vertex u_j, $j \leq h(u_j) \leq j + 1/n$. Now, consider an edge (u_j, u_k) of G where $j < k$. Since the width of the input linear ordering $\mathcal{A}$ is b, we have $k - j \leq b$. Now we consider the following two cases. If $k - j \leq b - 1$ then $h(u_k) - h(u_j) \leq k + 1/n - j \leq b - 1 + 1/n < b$. Since each interval in H has length b, it follows that $(u_j, u_k) \in E(H)$. If $k - j = b$ then from the definition of h, it follows that $h(u_k) = h(u_{k-b}) + b = h(u_j) + b$. Thus $h(u_k) - h(u_j) = b$ implying that $(u_j, u_k) \in E(H)$. Every edge in G is therefore present in H, or in other words, H is a supergraph of G.

Construction of I_i, for $0 \leq i \leq b - 1$.
The vertex set of the indifference graph I_i is $V(G)$ and let $E(I_i)$ denote the edge set of I_i. I_i is constructed as follows. Let $v_0, v_1, \ldots, v_{k-1}$ be a subsequence of $\mathcal{A}$ of k vertices such that $v_0 = u_i$, $v_1 = u_{i+b}$, $v_2 = u_{i+2b}$, ..., $v_j = u_{i+jb}$ and so on where $k = \lceil \frac{n-i}{b} \rceil$. We define v_k as a dummy vertex with the property that $\forall u \in V(G), u < v_k$. We now define f_i, the unit interval representation for I_i with interval length 2, as follows:

$$f_i(u) = 1, \text{ if } u < u_i$$

If u be a vertex such that $u \geq u_i$:

$$\begin{aligned} f_i(u) &= t, \text{ if } u = v_t \\ &= t + 2, \text{ if } v_t < u < v_{t+1} \text{ and } (u, v_t) \in E(G) \\ &= t + 3, \text{ if } v_t < u < v_{t+1} \text{ and } (u, v_t) \notin E(G) \end{aligned}$$

Claim. I_i for $0 \leq i \leq b - 1$ is a supergraph of G.

Proof. Consider the indifference graph I_i. Let (x, y) be any edge in $E(G)$. We assume without loss of generality that $x < y$.
Case $x < u_i = v_0$: Then $y < u_{i+b} = v_1$. Thus, $f_i(x) = 1$ and $0 \leq f_i(y) \leq 3$. Therefore, $|f_i(x) - f_i(y)| \leq 2$ which implies that $(x, y) \in E(I_i)$ (since f_i is a unit interval representation with interval length 2).
Case $x = v_t$ for some t: Then $y \leq v_{t+1}$, therefore $f_i(x) = t$ and $f_i(y) = t + 1$ (if $y = v_{t+1}$) or $t + 2$ (if $y < v_{t+1}$). In either case, $|f_i(x) - f_i(y)| \leq 2$ and therefore, $(x, y) \in E(I_i)$.
Case $v_t < x < v_{t+1}$ for some t: Then $y < v_{t+2}$. Therefore, $f_i(x)$ can take values in $\{t+2, t+3\}$ while $f_i(y)$ can take values in $\{t+1, t+2, t+3, t+4\}$. Therefore, $|f_i(x) - f_i(y)| \leq 2$, which implies that $(x, y) \in E(I_i)$.

Since all the cases have been considered, it follows that any edge in $E(G)$ is also an edge in $E(I_i)$.

It remains to show that $G = I_0 \cap \cdots \cap I_{b-1} \cap H$. To do this, it suffices to show that any $(x, y) \notin E(G)$ is not present in at least one of the indifference graphs $I_0, \ldots, I_{b-1}, H$. Let $x = u_j$ and $y = u_k$ and we will assume without loss of generality that $j < k$ (i.e. $x < y$). Consider the case $k - j \geq b$. In this case, we claim that $(x, y) \notin E(H)$. This is because of the following. If $k - j = b$ then clearly $h(x) - h(y) = h(u_k) - h(u_j) = b + \epsilon$ and thus $(x, y) \notin E(H)$ (recall that

h is a unit interval representation with interval length b). Now, if $k - j \geq b + 1$ then $h(u_k) - h(u_j) \geq k - j - 1/n \geq (b + 1) - 1/n > b$ (since $h(u_k) \geq k$ and $h(u_j) \leq j + 1/n$). Thus $(u_j, u_k) \notin E(H)$. Now the remaining case is $k - j < b$. Consider the graph I_l where $l = j \mod b$. Let $t = \lfloor j/b \rfloor$ and let $v_r = u_{l+rb}$, for $r = 0, 1, 2, \ldots$. Then $v_t = u_j$. Since $k - j < b$, $u_k < v_{t+1}$. Thus we have $f_l(u_j) = t$ and $f_l(u_k) = t + 3$. Thus, $|f_l(u_j) - f_l(u_k)| > 2$ and hence $(u_j, u_k) \notin E(I_l)$ as required.

Thus $I_0, \ldots, I_{b-1}, H$ is a valid indifference graph representation of G using $b + 1$ indifference graphs which establishes that $\text{cub}(G) \leq b + 1$. ■

Tightness of the result: Though the bound of $\text{cub}(G) \leq \text{bw}(G)+1$ might seem far from being tight for many graphs such as complete graphs, there are several graphs for which the bound becomes almost tight. For example, the bandwidth and cubicity of paths are both equal to 1 and for cycles, the bandwidth and cubicity are both equal to 2 – our bound is thus tight but for an additive constant of 1. A Roberts' graph is the graph obtained by removing a perfect matching from a complete graph. It can be seen from the results of [2] that the cubicity of a Roberts' graph on n vertices is $n/2$. The bandwidth of the Roberts' graph can be seen to be $n - 2$ upon observation. Thus our bound is tight upto a factor of 2 for Roberts' graphs.

2.2 The Algorithm

Our algorithm to compute the cube representation of G in $b+1$ dimensions given a linear ordering of the vertices of G with width b constructs the indifference supergraphs of G, namely, $I_0, \ldots, I_{b-1}, H$ using the constructive procedure used in the proof of Theorem 2. It is easy to verify that this algorithm runs in $O(b \cdot n)$ time where b is the width of the input linear arrangement and n is the number of vertices in G. However, it has to be noted that obtaining a linear ordering of the vertices of a given graph with width equal to the bandwidth of the graph is an NP-hard problem and thus limits the use of our algorithm for general graphs. Feige[16] provides a polynomial time algorithm to generate a linear ordering of the vertices of a graph with width at most $O(\log^4 n)$ times the bandwidth of the graph. Moreover, for bandwidth computation, several algorithms with good heuristics are known that perform very well in practice [17].

3 Applying Our Results

Theorem 2 can be used to derive upper bounds for the cubicity of several special classes of graphs such as circular-arc graphs, cocomparability graphs and AT-free graphs. We find upper bounds for the bandwidth of these graph classes in terms of the maximum degree and consequently obtain upper bounds on the cubicity. Bandwidth of circular-arc graphs have been studied in [18,19], that of AT-free graphs in [20] and that of cocomparability graphs in [21]. The following lemmas can also be proved using certain properties given in [18,20,21].

Lemma 1. *If G is a circular-arc graph,* $\text{bw}(G) \leq 2\Delta$*, where Δ is the maximum degree of G.*

Proof. Let an arc on a circle corresponding to a vertex u be denoted by $[h(u), t(u)]$ where $h(u)$(called the *head* of the arc) is the starting point of the arc when the circle is traversed in the clockwise order and $t(u)$ (called the *tail* of the arc) is the ending point of the arc when traversed in the clockwise order. We assume without loss of generality that the end-points of all the arcs are distinct and that no arc covers the whole circle. If any of these cases occur, the end-points of the arcs can be shifted slightly so that our assumption holds true.

Choose a vertex $v_1 \in V(G)$. Start from $h(v_1)$ and traverse the circle in the clockwise order. We order the vertices of the graph (other than v_1) as $v_2, \ldots, v_n$ in the order in which the heads of their corresponding arcs are encountered during this traversal. Now, we define an ordering $f : V(G) \rightarrow \{1, \ldots, n\}$ of the vertices of G as follows:

$f(v_j) = 2j$, if $1 \leq j \leq \lfloor n/2 \rfloor$.
$f(v_j) = 2(n-j)+1$, if $\lfloor n/2 \rfloor < j \leq n$.

We now prove that the width of this ordering is at most 2Δ.

We claim that if $h(v_j)$ and $h(v_k)$ are two consecutive heads encountered during a clockwise traversal of the circle, $|f(v_j) - f(v_k)| \leq 2$. To see this, we will consider the different cases that can occur:
Case : When $1 \leq j < j+1 = k \leq \lfloor n/2 \rfloor$. Here, $f(v_j) = 2j$ and $f(v_k) = 2(j+1)$. Therefore, $|f(v_j) - f(v_k)| = 2$.
Case : When $\lfloor n/2 \rfloor < j < j+1 = k \leq n$. In this case, $f(v_j) = 2(n-j)+1$ and $f(v_k) = 2(n-(j+1))+1$, which means that $|f(v_j) - f(v_k)| = 2$.
Case : When $j = \lfloor n/2 \rfloor < j+1 = k$,
Subcase : If n is even. $f(v_j) = 2j = n$ and $f(v_k) = 2(n-(j+1))+1 = 2n-2j-1 = n-1$.
Subcase : If n is odd, $f(v_j) = 2j = n-1$ and $f(v_k) = 2n-2j-1 = n$.
In both these cases, $|f(v_j) - f(v_k)| = 1$.
Case : When $j = n$ and $k = 1$. We then have $f(v_j) = 1$ and $f(v_k) = 2$. Therefore, $|f(v_j) - f(v_k)| = 1$.

Now, consider any edge $(v_j, v_k) \in E(G)$. Assume without loss of generality that $h(v_j)$ occurs first when we traverse the circle in clockwise direction starting from $h(v_1)$. Now, if we traverse the arc corresponding to v_j from $h(v_j)$ to $t(v_j)$, we will encounter at most $\Delta - 1$ heads $h(u_1), h(u_2), \ldots, h(u_{\Delta-1})$ before we reach $h(v_k)$ since v_j can be connected to at most Δ vertices in G. We already know that $|f(v_j) - f(u_1)| \leq 2$ and $|f(u_i) - f(u_{i+1})| \leq 2$, for $1 \leq i \leq \Delta - 2$. Also, $|f(u_{\Delta-1} - f(v_k)| \leq 2$. It follows that $|f(v_j) - f(v_k)| \leq 2\Delta$. Thus f is an ordering of the vertices of G with width at most 2Δ and therefore we have $\text{bw}(G) \leq 2\Delta$. □

Corollary 1. *If G is a circular-arc graph with maximum degree Δ, then* $\text{cub}(G) \leq 2\Delta + 1$.

Proof. Follows from Theorem 2 and Lemma 1. □

Lemma 2. *If G is a cocomparability graph, then* $\mathrm{bw}(G) \leq 2\Delta - 1$*, where Δ is the maximum degree of G.*

Proof. Let V denote $V(G)$ and let $|V| = n$. Since $\overline{G}$ is a comparability graph, there exists a partial order $\prec$ in $\overline{G}$ on the node set V such that $(u, v) \in E(\overline{G})$ if and only if $u \prec v$ or $v \prec u$. This partial order gives a direction to the edges in $E(\overline{G})$. We can run a topological sort on this partial order to produce a linear ordering of the vertices, say, $f : V \to \{1, \ldots, n\}$. The topological sort ensures that if $u \prec v$, then $f(u) < f(v)$. Now, let $(u, v) \in E(G)$ and let w be a vertex such that $f(u) < f(w) < f(v)$. We will show that w is adjacent to either u or v in G. Suppose not. Then $(u, w), (w, v) \in E(\overline{G})$ and therefore $u \prec w$ and $w \prec v$. Now, by transitivity of $\prec$, this implies that $u \prec v$, which means that $(u, v) \in E(\overline{G})$ – a contradiction. Therefore, any vertex w such that $f(u) < f(w) < f(v)$ in the ordering f is adjacent to either u or v. Since the maximum degree of G is Δ, there can be at most $2\Delta - 2$ vertices between with $f(\cdot)$ value between $f(u)$ and $f(v)$. Thus, the width of the ordering given by f is at most $2\Delta - 1$ and therefore, $\mathrm{bw}(G) \leq 2\Delta - 1$. □

Corollary 2. *If G is a cocomparability graph with maximum degree Δ, then* $\mathrm{cub}(G) \leq 2\Delta$*.*

Proof. Follows from Theorem 2 and Lemma 2. □

A *caterpillar* is a tree such that a path (called the *spine*) is obtained by removing all its leaves. In the proof of Theorem 3.16 of [20], Kloks et al. show that every connected AT-free graph G has a spanning caterpillar subgraph T, such that adjacent nodes in G are at a distance at most four in T. Moreover, for any edge $(u, v) \in E(G)$ such that u and v are at distance exactly four in T, both u and v are leaves of T. Let $p_1, \ldots, p_k$ be the nodes along the spine of G.

Lemma 3. *If G is an AT-free graph,* $\mathrm{bw}(G) \leq 3\Delta - 2$*, where Δ is the maximum degree of G.*

Proof. Let L_i denote the set of leaves of T adjacent to p_i. Clearly, $|L_i| \leq \Delta$ and $L_i \cap L_j = \emptyset$ for $i \neq j$. For any set S of vertices, let $\langle S \rangle$ denote an arbitrary ordering of the vertices in set S. Let $\langle u \rangle$ denote ordering with just one vertex u in it. If $\alpha = u_1, \ldots, u_s$ and $\beta = v_1, \ldots, v_t$ are two orderings of vertices in G, then let $\alpha \diamond \beta$ denote the ordering $u_1, \ldots, u_s, v_1, \ldots, v_t$. Let $\mathcal{A} = \langle L_1 \rangle \diamond \langle p_1 \rangle \diamond \langle L_2 \rangle \diamond \langle p_2 \rangle \diamond \cdots \diamond \langle L_k \rangle \diamond \langle p_k \rangle$ be a linear ordering of the vertices of G. One can use the property of T stated in the previous paragraph to easily show that $\mathcal{A}$ is a linear ordering of the vertices of G with width at most $3\Delta - 2$. Therefore, $\mathrm{bw}(G) \leq 3\Delta - 2$. □

Corollary 3. *If G is an AT-free graph with maximum degree Δ, then* $\mathrm{cub}(G) \leq 3\Delta - 1$*.*

Proof. Follows from Theorem 2 and Lemma 3. □

References

1. Golumbic, M.C.: Algorithmic Graph Theory And Perfect Graphs. Academic Press, New York (1980)
2. Roberts, F.S.: On the boxicity and cubicity of a graph. In: Recent Progresses in Combinatorics, pp. 301–310. Academic Press, New York (1969)
3. Yannakakis, M.: The complexity of the partial order dimension problem. SIAM Journal on Algebraic Discrete Methods 3, 351–358 (1982)
4. Erlebach, T., Jansen, K., Seidel, E.: Polynomial-time approximation schemes for geometric intersection graphs. SIAM Journal on Computing 34(6), 1302–1323 (2005)
5. Afshani, P., Chan, T.: Approximation algorithms for maximum cliques in 3d unit-disk graphs. In: Proc. 17th Canadian Conference on Computational Geometry (CCCG), pp. 6–9 (2005)
6. van Leeuwen, E.J.: Approximation algorithms for unit disk graphs. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 351–361. Springer, Heidelberg (2005)
7. Agarwal, P.K., van Kreveld, M., Suri, S.: Label placement by maximum independent set in rectangles. Comput. Geom. Theory Appl. 11, 209–218 (1998)
8. Berman, P., DasGupta, B., Muthukrishnan, S., Ramaswami, S.: Efficient approximation algorithms for tiling and packing problems with rectangles. J. Algorithms 41, 443–470 (2001)
9. Michael, T., Quint, T.: Sphere of influence graphs and the l_∞-metric. Discrete Applied Mathematics 127, 447–460 (2003)
10. Fishburn, P.C.: On the sphericity and cubicity of graphs. Journal of Combinatorial Theory, Series B 35(3), 309–318 (1983)
11. Maehara, H.: Sphericity exceeds cubicity for almost all complete bipartite graphs. Journal of Combinatorial Theory, Series B 40(2), 231–235 (1986)
12. Michael, T., Quint, T.: Sphericity, cubicity, and edge clique covers of graphs. Discrete Applied Mathematics 154(8), 1309–1313 (2006)
13. Chandran, L.S., Mannino, C., Orialo, G.: On the cubicity of certain graphs. Information Processing Letters 94, 113–118 (2005)
14. Chandran, L.S., Francis, M.C., Sivadasan, N.: On the cubicity of interval graphs. Electronic Notes in Discrete Mathematics 29, 315–319 (2007)
15. Unger, W.: The complexity of the approximation of the bandwidth problem. In: Proceedings of the 39th IEEE Annual Symposium on Foundations of Computer Science, pp. 82–91 (November 1998)
16. Feige, U.: Approximating the bandwidth via volume respecting embeddings. In: Prceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, pp. 90–99. ACM Press, New York (1998)
17. Turner, J.: On the probable performance of heuristics for bandwidth minimization. SIAM journal on computing 15, 561–580 (1986)
18. Kratsch, D., Stewart, L.: Approximating bandwidth by mixing layouts of interval graphs. SIAM Journal on Discrete Mathematics 15(4), 435–449 (2002)
19. Kloks, T., Kratsch, D., Borgne, Y.L., Müller, H.: Bandwidth of split and circular permutation graphs. In: Brandes, U., Wagner, D. (eds.) WG 2000. LNCS, vol. 1928, pp. 243–254. Springer, Heidelberg (2000)
20. Kloks, T., Kratsch, D., Müller, H.: Approximating the bandwidth of asteroidal triple-free graphs. Journal of algorithms 32(1), 41–57 (1999)
21. Kratsch, D., Stewart, L.: Domination on cocomparability graphs. SIAM Journal on Discrete Mathematics 6(3), 400–417 (1993)

$O(m \log n)$ Split Decomposition of Strongly Connected Graphs

Benson L. Joeris[1], Scott Lundberg[2], and Ross M. McConnell[2]

[1] Trinity College, Cambridge, CB2 1TQ, UK
blj24@cam.ac.uk
[2] Computer Science Department, Colorado State University, Fort Collins, CO 80523-1873
{lundberg,rmm}@cs.colostate.edu

Abstract. In the early 1980's, Cunningham described a unique decomposition of a strongly-connected graph. A linear time bound for finding it in the special case of an undirected graph has been given previously, but up until now, the best bound known for the general case has been $O(n^3)$. We give an $O(m \log n)$ bound.

Keywords: split decomposition, join decomposition, strongly connected graphs.

1 Introduction

Split decomposition is a unique decomposition of arbitrary strongly-connected digraphs described by Cunningham in 1982 [3]. Because undirected graphs are a special case of strongly-connected digraphs, a special case of the decomposition applies to arbitrary undirected graphs. Also known as join decomposition, it is useful in many areas ranging from recognition of certain graph classes [5] to optimizations of NP-hard problems [9]. It is a proper generalization of the well known modular decomposition, also called substitution decomposition [8,7].

As a convention we denote the number of vertices of a graph as $n = |V|$ and the number of edges $m = |E|$.

Cunningham gave the first algorithm for computing the decomposition on arbitrary strongly-connected digraphs, which runs in $O(n^4)$ time [3]. Bouchet improved this to $O(n^3)$ [2]. This solves an interesting special case, which is determining whether a graph is *prime* with respect to the split decomposition, which means that it can be decomposed only in trivial ways (explained further below). Spinrad gave an $O(n^2)$ algorithm for determining whether an arbitrary strongly-connected directed graph is prime [10], but not for finding the decomposition tree if it has a nontrivial decomposition. Since then, much work has focused on the special case of undirected graphs. This work includes an $O(nm)$ algorithm by Gabor, Supowit, and Hsu [5], an $O(n^2)$ algorithm by Ma and Spinrad [6], and, finally, a linear-time $(O(n + m))$ algorithm by Dahlhaus [4].

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 158–171, 2009.

This leaves open the possibility of improving on the previous best bound of $O(n^3)$ for finding the decomposition of strongly-connected digraphs. In this paper, we give an $O(m \log n)$ bound.

Our approach borrows generously from techniques developed by Ma and Spinrad for their $O(n^2)$ algorithm for undirected graphs [6]. In particular, we make similar use of a technique called *graph partitioning* or *partition refinement*.

As an historical note, it is worth noting that techniques for implementing partition refinement efficiently, and many well-known applications of it, were first described by Spinrad [11], [14], [13]. We get the improvement to Ma and Spinrad's time bound, a generalization to strongly-directed graphs, and a substantial simplification of their approach, by modifying a clever charging argument for partition refinement, called *halving before re-use*. The use of this trick for modular decomposition, transitive orientation, and many related problems was circulated widely in the mid-1980's in a working manuscript, also written by Spinrad [12]. He obtained the trick by showing how to modify a related one, developed by Hopcroft, for state minimization in deterministic finite automata [1]. These techniques and applications have been mistakenly attributed in the literature to subsequent papers that, like the present paper, borrow heavily from Spinrad's early work on the subject.

2 Split Sets and Split Decomposition Trees

We view an undirected graph as the special case of a directed graph where every undirected edge is a directed two-cycle. If X is a nonempty subset of V, by $G[X]$ we denote the subgraph of G induced by X. If $x \in V$, by $deg(x)$, we denote the degree of x. If G is a directed graph, then we let $deg(x)$ denote the number of in-neighbors plus the number of out-neighbors.

A *split* in a directed graph is a partition of the vertices into two sets, X and Y, where the edges directed from X to Y forms a cartesian product $X' \times Y'$, for some $X' \subseteq X$ and some $Y' \subseteq Y$, and the edges directed from Y to X form a cartesian product $Y'' \times X''$, for some $Y'' \subseteq Y$ and some $X'' \subseteq X$. For example, in Figure 1, $X' = \{a\}$, $Y' = \{c, d, e\}$, $X'' = \{b\}$, and $Y'' = \{c, d, e\}$. X and Y are *split sets* if they satisfy these requirements. X' is the set of *outgoing connectors of* X, and X'' is the set of *incoming connectors of* X. Y' is the set of *outgoing neighbors of* X, and Y'' is the set of *incoming neighbors of* X. Applying these definitions to Y reverses the roles of outgoing vs. incoming and connectors vs. neighbors. In a directed graph, it is not necessary that $X' = X''$ or $Y' = Y''$ or that they be disjoint, but in an undirected graph, symmetry dictates that $X' = X''$ and $Y' = Y''$, in which case we can refer to X' and Y' simply as the *connectors* and *neighbors* of X.

Figure 1 shows how, if we find a split, we can represent the graph with two *quotients*, where, in one quotient, Y is replaced with a single *marker vertex* whose out-neighbors and in-neighbors are the in-connectors and out-connectors of X, and X is similarly replaced with a marker vertex in the other. This process is invertible; we can reconstruct the original graph by a *composition* of the two quotients.

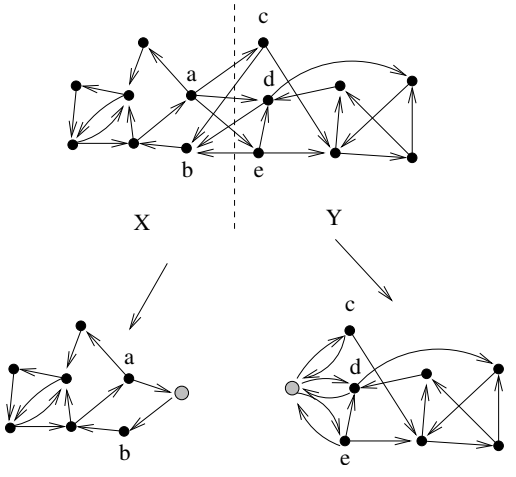


Fig. 1. Two quotients formed by a split in a directed graph

This differs from modular decomposition, where a graph is broken into a quotient, which receives a marker, and a factor, which does not. This asymmetry is due to the fact that, unlike split decomposition, where the complement of a split set is a split set, the complement of a module in modular decomposition is not a module.

The quotient consisting of X and its marker y can be considered a many-to-one mapping of V to $X \cup \{y\}$, where each element of X maps to itself and each element of Y maps to y. This mapping has the property that any split set of G maps to a split set of the quotient, and the inverse image of any split set of the quotient is a split set of G. Let us call this the *homomorphic rule*.

There can be an exponential number of split sets (consider the complete graph, where all nonempty proper subsets of V are split sets). However, they can be represented implicitly in $O(n)$ space with a *split decomposition tree*, which is an unrooted tree whose leaves are the vertices of G. We first describe how this is accomplished for an undirected graph. If u is an internal node, let a *neighbor set of* u be the set of leaves reachable through a neighbor v of u. That is, they are the set of vertices of G that are in v's component of the tree if the tree edge uv is removed. Each internal node of the tree is labeled *prime* or *degenerate*. A set is a split set if and only if it is a neighbor set of an internal node, a union of all but one neighbor set of an internal node, or any union of at least one and fewer than all neighbor sets of a degenerate node. Cunningham showed that there is a unique tree with these properties.

For instance, Figure 2 gives the split decomposition of an undirected graph. The neighbor sets of node 3 are $\{c\}$, $\{y, x, w, v\}$, $\{a\}$, $\{e, d, b, f\}$, and $\{g, h, i\ , j, k,$ $n, p,\ q, r,\ s, t\}$. This is a degenerate node, since every union of these neighbor sets is a split set. For instance, the union of $\{u, v, w, x, y\}$, $\{c\}$, and $\{b, d, e, f\}$ is a

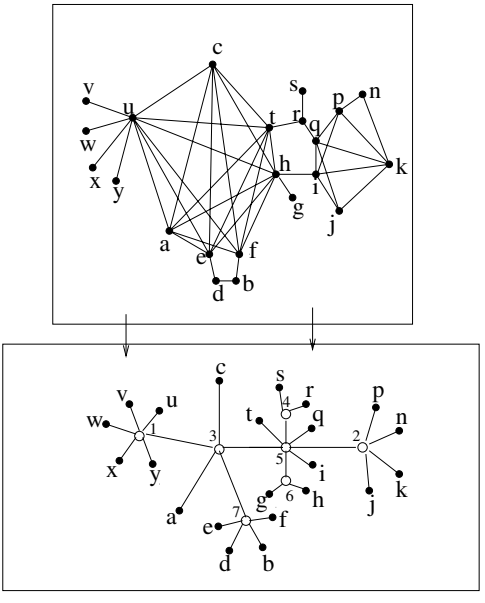


Fig. 2. The split decomposition of an undirected graph

split set with connectors $\{u, c, e, f\}$. Node 5, on the other hand is prime: the only unions of its neighbor sets that are split sets are one neighbor set or all but one neighbor set.

The *associated quotient* at an internal node u is the quotient G' of G obtained by replacing each of u's neighbor sets with a marker vertex. For simplicity, if V' is the neighbor set reachable through a neighbor v, we can consider v to *be* the marker for V'. For instance, the associated quotient at node 3 of Figure 2 is a complete graph on vertices $\{1, a, 7, 5, c\}$. The associated quotient at node 5 is a graph on vertex set $\{3, 6, i, 2, q, 4, t\}$ and edge set $\{\{6, i\}, \{i, q\}, \{q, 4\}, \{4, t\}, \{t, 6\}, \{3, t\}, \{3, 6\}, \{2, q\}, \{2, i\}\}$.

A graph is *prime* if its only split sets are the one-element subsets and their complements, and *degenerate* if all nonempty proper subsets of its vertices are split sets. By the homomorphic rule, the quotient associated with a prime node is a prime graph and the quotient associated with a degenerate node is a degenerate graph. The only degenerate quotients associated with nodes of the decomposition of a connected undirected graph are stars and complete graphs. It is easy to see that the process of decomposing G into the quotients at the internal nodes of the decomposition tree is invertible; G can be reconstructed by composition using the tree and its associated quotients.

If the graph is a strongly-connected digraph, the decomposition tree is similar, except that, in addition to prime and degenerate nodes, it may have *circular nodes*. At a circular node, there is a cyclic ordering of its neighbor sets, and a union of neighbor sets is a split set if and only if it is a union of at least one and fewer than all of the neighbor sets *that are consecutive in the cyclic ordering*. By

the homomorphic rule, it must be the case that the quotient associated with a circular node is a graph with a cyclic ordering on its vertices, such that a set of vertices is a split set if and only if it is a nonempty proper subset of the vertices that is consecutive in the circular ordering. Let us call such a graph a *circular graph*.

A *cycle of transitive tournaments* is a cyclic ordering of transitive tournaments, where the sink of each is identified with the source of the next. Figure 3 gives an example. Cunningham showed that a graph with at least four vertices is a circular graph if and only if it is a cycle of transitive tournaments. By the homomorphic rule, therefore, the quotient associated with a circular node of degree at least four must be a cycle of transitive tournaments.

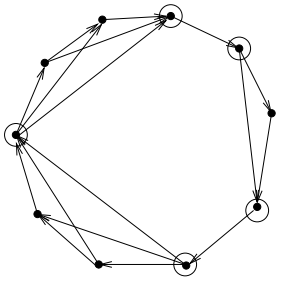


Fig. 3. The quotient associated with a circular node is a cycle of transitive tournaments. The sink of each transitive tournament is the source of the next (circled vertices), and a nonempty set of vertices is a split set if and only if the vertices are a proper subset of the vertices that is consecutive in the cyclic order.

So far, our distinction between prime, degenerate, and circular nodes is ambiguous for nodes of degree three. We therefore consider a node of degree three to be circular if its associated quotient is a cycle of transitive tournaments.

2.1 The Strategy

Let a, b, c be vertices of G. We have two methods at our disposal whose implementation is described below: $S(a, b, G)$, which finds the maximal split sets in G that don't contain a or b, and $L(a, b, c, G)$ which finds the maximal split set in G that doesn't contain a or b but does contain c. We show below that $S(a, b, G)$ is a partition of $V \setminus \{a, b\}$, and, because $L(a, b, c, G)$ is the member of $S(a, b, G)$ that contains c, it is unique. We could get $L(a, b, c, G)$ by running $S(a, b, G)$ and discarding all returned sets except the one that contains c, but we use a separate procedure that omits the unnecessary steps for efficiency reasons. Before showing how to compute $S(a, b, G)$ and $L(a, b, c, G)$, we show how to reduce finding the split decomposition to calls to these procedures.

Because a, b, c are vertices of G, they must be leaf nodes of G's split decomposition tree. We don't yet know what the G's decomposition tree looks like, but we do know that the paths connecting a, b, and c in the tree must intersect

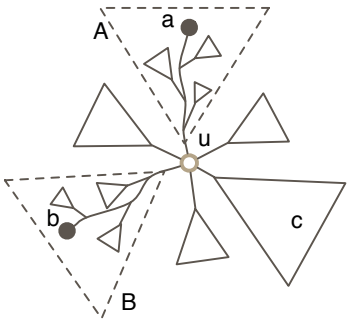


Fig. 4. What we get from calling $S(a, b, G)$

at single internal node, which we will call u. (See Figure 4.) Let A, B, and C denote the neighbor sets of u that contain a, b, and c, respectively.

It is easy to see from the relationship between the split sets and neighbor sets of prime nodes that if all nodes of the split decomposition are prime, $S(a, b, G)$ returns precisely the neighbor sets of nodes on the path from a to b that do not contain a or b. It follows that $S(a, b, G)$ is a partition of $V \setminus \{a, b\}$, that each neighbor set of u, other than A and B, is a member of $S(a, b, G)$, and that $A \setminus \{a\}$ and $B \setminus \{b\}$ each are unions of the remaining members of $S(a, b, G)$, since they are the neighbor sets of the nodes other than u on the path from a to b. A call to $L(b, c, a, G)$ finds A and a call to $L(a, c, b, G)$ finds B. We now know all neighbor sets of u.

To find the remainder of the decomposition tree, we replace each neighbor set of u with a quotient graph, recursively find the decomposition trees of these quotients, and then join them by identifying their markers with u (see Figure 5). The correctness of this algorithm when each node of the decomposition tree is prime is immediate from the homomorphic rule.

Introducing degenerate and circular nodes. From the relationship between the split sets and the decomposition tree, it is easy to see that once the possibility of degenerate and circular nodes is introduced, the following are the members of $S(a, b, G)$ for some node v on the path from a to b: a neighbor set of v that does not contain a or b if v is prime, the union of all neighbor sets that do not contain a or b if v is degenerate, the union of neighbor clockwise from A and counterclockwise from B or the union of neighbor sets clockwise from A and counterclockwise from B if v is a circular node. (In the case of a circular node, one of the two sets is empty if A and B are adjacent in the cyclic order.)

As before, we find A and B by calls to $L(b, c, a, G)$ and $L(a, c, b, G)$, and recursively find the decomposition trees of the quotients for A, B, and those members of $S(a, b, G)$ that are not subsets of A or B, that is, that are unions of one or more neighbor sets of u. If u is prime, this requires one recursive call for each neighbor set of u. If u is degenerate, this requires three recursive calls, one for A, one for B, and one for the union of all other neighbor sets. If u is circular,

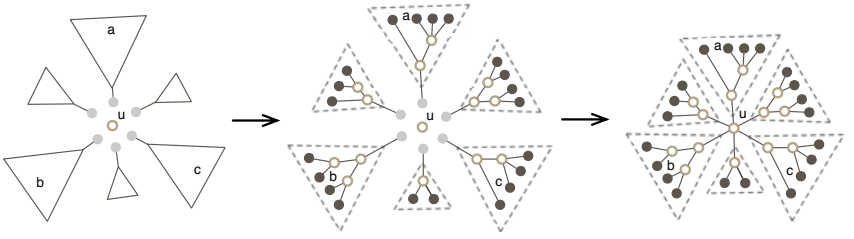


Fig. 5. Completing the decomposition tree through recursion

it requires up to four recursive calls, as explained above. We then identify the markers with u to perform a composition of these trees as before.

If u is degenerate, let X be the union of neighbor sets of u other than A and B returned by $S(a, b, G)$. By induction, we may assume that a recursive call on the quotient consisting of X and a marker produces the split decomposition of this graph. Identifying u and the marker for X, and doing the same for the results of recursive calls on A and B makes u a tree node of degree three. If X is the union of more than one neighbor set of u in the actual decomposition of G, then this is incorrect, because u should have one neighbor for each of these. Note that w carries the missing neighbor sets that should be u's. Since u has degree at most four, it takes $O(1)$ time to find its associated quotient and determine whether it is a complete graph, a star, or a circular graph (cycle of transitive tournaments). By induction, we assume that this has been determined for w by the recursive call. If u and w both have complete or circular quotients, the composition of their quotients obtained by identifying the markers corresponding to u and w is a larger complete or circular quotient. We accomplish this by contracting the edge uw in the tree, letting u stand for the resulting node. This allows u to inherit its individual neighbor sets from w. Similarly, if both of u and w have associated quotients that are stars, we can determine in $O(1)$ time whether the composition of their two stars is a larger star, and perform the contraction if it is.

2.2 Implementation of $S(a, b, G)$ and $L(a, b, c, G)$

$S(a, b, G)$ works by starting with an initial partition $\{\{a\}, \{b\}, V \setminus \{a, b\}\}$ of V, and successively refining the partition classes until they give $\{a\}$, $\{b\}$, and $S(a, b, G)$.

For simplicity, let us first assume that G is undirected. We find $S(a, b, G)$ by selecting a vertex as a *pivot vertex*, p, and perform a *pivot operation* on it, which may refine the partition by splitting partition classes that do not contain p. The *outsiders* of a partition class S are those vertices $V \setminus S$. For a vertex v that starts out in $V \setminus \{a, b\}$ the only initial outsiders of its class are a and b. Every time the class containing v is split, it finds itself in a smaller class that has more outsiders. If S is a partition class, the *known outsiders* are those on

which we have performed a pivot since the point when they became outsiders of S, and the *known connectors* of S are those vertices of S that have edges to known outsiders.

We show that if the following invariants apply before a pivot, then they also apply after the pivot:

- **Splitting invariant:** A split set that started out as a subset of a single partition class remains a subset of a single partition class after each refinement.

- **Pivot invariant:** If P is the set of finished outsiders of S, S is a split set in $G[P \cup S]$.

The invariants apply to the initial partition because no partition class has been split and no partition class has any finished outsiders. When all outsiders of every partition class are known, it follows by the pivot invariant that each partition class is a split set. Since the initial partition is $\{\{a\}, \{b\}, V \setminus \{a, b\}\}$, the splitting invariant ensures that the final partition is $\{a\}, \{b\}$, and the maximal split sets that do not contain a or b, in other words, the final partition is $\{\{a\}, \{b\}\} \cup S(a, b, G)$.

Implementation of a pivot. We first describe the implementation of a pivot operation in the undirected case. Pivoting on a pivot vertex p moves it from the unfinished to the finished outsiders of each partition class except the one, P, that contains p and is not allowed to split during the pivot. In $S(a, b, G)$, for each partition class S other than P, we identify the following *founding sets* for subclasses that S will be subdivided into during the pivot: the known connectors of S that are non-neighbors of p, the known connectors that are neighbors of p, and the members of S that were not known connectors, but that are neighbors of p.

Some of these founding sets may be empty, and if S has at most one non-empty founding set, it will still satisfy the pivot invariant once p is included among its known outsiders. Otherwise, members of different founding sets in S have different sets of known outside neighbors, so they cannot remain in the same partition class without violating the pivot invariant. They become the known connectors of new subclasses that S will be partitioned into. It remains to determine which other members of S can be placed with which founding sets. We select a vertex z that has already been claimed for a subclass S' of S, and have it claim all of its neighbors in S for S', except for those neighbors that have already been claimed. Let us call z a *claim staker*. A second claim-staking by z during a split of S cannot claim any new vertices for its subclass, so each vertex is selected at most once to stake claims during a pivot. Once all members of S have been claimed for a subclass, these subclasses become the refinement of S.

Except for the requirement that a vertex already be claimed before it is selected as a claim-staker, the order in which we select vertices as claim-stakers does not matter in maintaining the splitting invariant, which can be seen as follows. Let S'' be a split set that is a subset of S. If it intersects the known connectors of S, then the connectors of S'' must be its intersection with the known

connectors of S, and since its connectors have the same neighbors outside of S, its known connectors are a subset of a single founding set. All paths into S'' contain one of its connectors, the claiming of vertices in S'' is initiated at its connectors, and claims to other members of S'' are sealed off from claims to other members of S''. All of S'' ends up in the subclass founded by this founding set. Similarly, if S'' doesn't intersect a founding set, its connectors are all claimed at once by the first outsider of S'' that claims vertices for its class, since a neighbor of one of its outsiders is a neighbor of all of them. This again seals off the rest of S'' from claims by competing classes, and S'' ends up in a single partition class, as required.

After the split of S, each subclass's founding set is its set of known connectors, and they all have the same set of known outsiders that are neighbors, including p. The pivot invariant is preserved.

$L(a, b, c, G)$ is implemented using pivots also, with the only difference being that we keep only the partition class that contains c. Every time this class is split by a pivot, we discard the resulting subclasses that do not contain p.

For a strongly-connected digraph, we run the same procedure for a pivot on p, with the only difference that we reinterpret the neighbors of a vertex to be its out-neighbors. Let us call this the *outward pivot*. This refines the partition, but does not refine the class containing p. We then re-run the procedure on the refined partition, this time reinterpreting the neighbors of a vertex to be its in-neighbors. This is the *inward pivot*. A *pivot* on p consists of an outward pivot and then an inward pivot. Using trivial variants of the above arguments, and the fact that the graph is strongly-connected, which means that every nonempty proper subset of V has both incoming and outgoing directed edges, it is easy to see that the splitting and pivot invariants are maintained.

2.3 The $O(m \log n)$ Time Bound

When the algorithm generates a recursive call, it adds a marker to it. That recursive call may, in turn, generate others, the marker is passed to one of them, and a new marker is added to that call. This shows that multiple markers can occur in a single call deep in the recursion.

To avoid proliferation of markers in any one call, when we select a, b, and c, we give priority to markers in selecting a and b. We claim that this ensures that there are at most two markers in any recursive call. The proof is by induction on the depth of the call. Suppose that it is true for a recursive call on a quotient, G. Since priority is given to markers in selecting a and b, no other vertices are markers. Since a and b are passed to different recursive calls, each recursive call, which receives a new marker, still has at most two markers.

Let the *non-marker degree sum* in a recursive call be the sum of in-degrees and out-degrees of the non-markers. Because each recursive call has at most two non-markers, there are at most two directed edges that fail to be incident to a marker. The following is immediate:

Lemma 1. *If k is the non-marker degree sum in a recursive call, then the total degree sum of all vertices in the call is $O(k)$.*

We keep a list of in-neighbors and a list of out-neighbors on each vertex. We implement each partition class with a doubly-linked list, where each element has a pointer to the front of the list that supports identifying its partition class in $O(1)$ time. In addition, we keep a doubly-linked list of the known in-connectors and a doubly-linked list of the known out-connectors, and mark the vertices in these lists according to which they are members of, which may be both of them.

During the outward pivot, this representation allows creating lists of the founding sets of all partition classes that don't contain p, in time proportional to $deg(p)$, by traversing p's list of out-neighbors, identifying the neighbor's class S, removing it from S, and from the list of incoming connectors if it is an incoming connector, and putting it in one of two doubly-linked lists of founding sets for the subclasses of the class. Which founding set it goes into depends on whether the neighbor is already a known inward connector. This gives at most two founding sets for each partition class S: neighbors of p that were not already know inward connectors and neighbors of p that were not. The third founding set for S, the non-neighbors of p that were already known inward connectors of S, is what's left of the doubly-linked list of inward connectors after neighbors of p have been removed, and we have not touched them. We therefore spend $O(deg(p))$ time getting a doubly-linked list of the members of each founding set for all partition classes that don't contain p. If S has only one founding set, then it is unnecessary to split it to get it to continue to satisfy the pivot invariant, so this founding set is just restored as the known inward connectors of S, and no further work is done on it.

If S does not contain p and has more than one founding set, then it is split. When $z \in S$ is selected as a claim-staker, it traverses its list of out-neighbors, and ignores the neighbor if it is either marked or a member of a different class from S. Otherwise it marks it and moves it to its own class. We conclude that, during an outward pivot, we spend $O(deg(z))$ time for each member z of a class that is properly split by the pivot on p. A similar analysis applies to the inward pivot.

The insight that leads to the strategy for getting the $O(m \log n)$ bound is to suppose for the moment that every time a class S is split into a set of subclasses by a pivot on p, each of these side classes has at most half the non-marker degree sum of S, and that we select a vertex p as a pivot only if it is *ripe*, that is, only if the non-marker degree sum of the class that contains it is at most half that of the class that contained it the last time it was used as a pivot. Then, by the foregoing, we spend $O(deg(x))$ time on each vertex each time the degree sum of its class halves, for a total of $O(deg(x) \log m) = O(deg(x) \log n)$ over all uses of the vertex as a pivot or a claim staker. The total time over all vertices is $O(m \log n)$.

Let us now repair this strategy once we consider the possibility that if a class is split, one of the resulting subclasses it is split into may have greater than half

the non-marker degree sum of the original. We begin by bounding the cost of all calls to $S(a, b, G)$ over all recursive calls.

The cost of calls to $S(a, b, G)$. When vertices in up to three subclasses of a class S that is being split lay claim to vertices for their own class, we work on the subclasses in parallel, keeping the degree sum of the vertices that have made claims so far equal among the three calls. A class becomes *closed* if all of its members have attempted to lay claim to new members, and has no hope of capturing further members. When two classes become closed, we can put the remaining list of unclaimed members at the front of the list of the third class without touching them individually, and update the data structures for the three classes in time proportional to the time we've spent so far on the class. One of the other two subclasses has non-marker degree sum at most half of that of S, so we can charge all costs to the non-marker degree sum of this subclass, plus the degree of p. *We can still charge all costs to the degree sum of vertices that find themselves in classes whose non-marker degree sum has halved since the last time they were charged.*

We must also bound the cost of identifying ripe pivots. When we pivot on a vertex, we pivot on all vertices in the class. We label each class with the non-marker degree sum of the most recent class that contained its members. When we split a class, each of the subclasses inherits this label, and the ones whose current degree sum is half this label are put in a list of ripe classes.

A risk in adhering to the discipline of only pivoting on ripe vertices might cause the partition refinement to halt, due to the absence of ripe vertices, before the partition classes are split sets. The key insight is that if we ever run out of ripe vertices to pivot on, the class X with the largest non-marker degree sum is a split set. This is because every other partition class Y has at most the non-marker degree sum that X has, so Y has at most half the non-marker degree sum as the most recent partition class that contained both X and Y. Y has become ripe at some point since it was separated from X, and since it is not ripe, all of its vertices have been used as pivots since that time. All members of Y are finished outsiders of X. Applying this to all partition classes $Y \neq X$, we see that all members of $V \setminus X$ are finished outsiders of X. By the pivot invariant, X is a split set. We then pivot once on a connector from this X (it doesn't matter which one, since they all have the same neighbors outside of X), and then remove X from consideration. This pivot may split some more classes, generating more ripe vertices, or else we may repeat the argument and remove another class X from consideration. Despite the new constraint, the procedure for finding $S(a, b, G)$ halts when we have found and removed every member of $\{\{a\}, \{b\}\} \cup S(a, b, G)$, and we get $S(a, b, G)$ in $O(m \log n)$ time.

It remains to show that the cost of calls to $S(a, b, G)$ over all recursive calls of the split decomposition algorithm is also $O(m \log n)$ time. Other than A and B, each recursive call is a member D of $S(a, b, G)$. In the recursive call on D, when we choose a new a and b, this creates two ripe vertices that can be used to re-start the partitioning on D without violating the constraint that we only pivot on ripe vertices. Unfortunately, A and B are each a union of members of

$S(a, b, G)$, each of which may have smaller non-marker degree sum than A or B. However, as depicted in Figure 5, the members of $S(a, b, G)$ that are adjacent to vertices on the subpath from a to u are just $S(a, u, G_A)$, where G_A is quotient passed to the recursive call on A. We avoid overcharging vertices in A for calls to $S()$ by passing in the members of $S(a, u, G_A)$, which are computed as a side-effect of finding $S(a, b, G)$, instead of making a call to find $S(a, u, G_A)$. This avoids the need to re-charge vertices in A before they become ripe again.

The cost of calls to $L(a, c, b, G)$ and $L(b, c, a, G)$. In a call $L(a, c, b, G)$, we keep only one of the partition classes that we would during a call to $S(a, c, G)$, namely, the one that contains b. We make use of a pivot p exactly once, when it is no longer in the partition class S that contains b. We ignore neighbors of p that are not in S, obtaining the founding sets of S, as before, in $O(deg(p))$ time. If S has only one founding set, we do no work on it, as before. Otherwise, since S is the only class of interest, we make use of a vertex to stake claims only if it is in S. If the subclass S' that contains b has more than half of the non-marker degree sum of S, then we charge the cost of the inward pivot to vertices in a different subclass. If it is less than half of the non-marker degree sum of S, we could charge the cost to the degree sum of S', but since the degree sum of the other subclasses exceeds this, we can still charge it to vertices in other subclasses. The invariant this maintains is that at no point have we charged any costs to the degrees of vertices in the class that contains b. This gives the following:

Lemma 2. *If B is the set returned by $L(a, c, b, G)$, the cost of the call to $L(a, c, b, G)$ is the degree sum of $V \setminus B$.*

When it is time to call $L(a, c, b, G)$ and $L(b, c, a, G)$, we already know $S(a, b, G)$. If some element D of $S(a, b, G)$ has non-marker degree sum larger than that half that of G, we replace D with a marker before the calls to avoid charging elements internal to D. By the homomorphic rule and the fact that the sets returned by $L(a, c, b, G)$ and $L(b, c, a, G)$ are unions of sets in $\{\{a\}, \{b\}\} \cup S(a, b, G)$, this does not affect the result of the calls. Touching the marker for D can be charged to the recursive call on D, and since there are $O(n)$ recursive calls overall, this contributes $O(n) = O(m)$ to the total running time. The edges incident to the marker can be found by looking elements of D in all adjacency lists of vertices outside of G, and we are allowed $O(1)$ charges for each such adjacency-list element, since it is not in D.

If there is no such D, it may be that A or B, say, B has more than half the non-marker degree sum of G. If we knew this in advance, we could avoid charging to elements internal to B by running $L(a, c, b, G)$ to find B first, charging this to elements in $V \setminus B$. By Lemma 2, we could then replace B with a marker b' to create a quotient graph G', and since A is disjoint from B, a call to $L(b', c, a, G')$ returns the same result as $L(b, c, a, G)$.

Unfortunately, we don't know in advance which of A and B might turn out to have more than half the degree sum of G. We therefore we run the calls $L(a, c, b, G)$ and $L(b, c, a, G)$ in parallel, keeping the degree sum of vertices used as pivots and claim-stakers equal. If $L(a, c, b, G)$ halts first, we have touched

all edges incident to vertices in $V \setminus B$ during the call, so we can subtract their non-marker degree sum from that of G to obtain that of B. If this is more than half the degree sum of G, then since both calls have spent the same amount of time, we can charge the cost of both calls so far to elements of $V \setminus B$, even if the call to $L(b, c, a, G)$ has touched elements internal to B. We then replace B with a marker, just as we did with D, above, before finishing out the call to $L(b, c, a, G)$. This prevents elements internal to B from getting charged.

Summarizing, every time a vertex or an element of its adjacency list is charged during a call to $L()$, the non-marker degree sum of the partition class that contains the vertex is at most half what it the previous time these elements were charged, and the charges pay for all costs of running the algorithm. The sum of these charges is $O(m \log m) = O(m \log n)$.

3 What Goes Wrong If G Is Not Strongly-Connected

One question that Cunningham did not dwell on is what goes wrong when a digraph is not strongly-connected. Using the existence of Cunningham's decomposition tree, we have shown that the maximal split sets that do not contain a or b is a partition of $V \setminus \{a, b\}$. Figure 6 shows that this is not the case when G is not strongly-connected, and therefore, the representation with the unique decomposition tree as we have described it does not exist. Cunningham's proof that it does exist makes use of the fact that, in a strongly-connected graph, for every nonempty proper subset S of V, there is an edge from an element of S to an element of $V \setminus S$, and this is not true for arbitrary directed graphs.

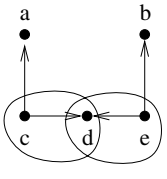


Fig. 6. A digraph that is not strongly-connected, and its maximal split sets that do not contain a or b

Nevertheless, a split is well-defined on an arbitrary directed graph, and the question of whether an arbitrary directed graph is prime is also well-defined. Spinrad gives an $O(n^2)$ algorithm for testing whether a strongly-connected graph is prime. As far as we know, there has been no work on the problem of determining whether an arbitrary directed graph is prime.

Acknowledgments

We would like to acknowledge Haim Kaplan for helpful discussions and feedback on this work.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
2. Bouchet, A.: Digraph decompositions and eulerian systems. SIAM Journal on Algebraic and Discrete Methods 8 (1987)
3. Cunningham, W.H.: Decomposition of directed graphs. SIAM J. Algebraic Discrete Methods 3, 214–228 (1982)
4. Dahlhaus, E.: Parallel algorithms for hierarchical clustering, and applications to split decomposition and parity graph recognition. Journal of Algorithms 36, 205–240 (2000)
5. Gabor, C.P., Supowit, K.J., Hsu, W.-L.: Recognizing circle graphs in polynomial time. Journal of the ACM 36, 435–473 (1989)
6. Ma, T.H., Spinrad, J.: An $O(n^2)$ algorithm for undirected split decomposition. Journal of Algorithms 16, 145–160 (1994)
7. Möhring, R.H.: Algorithmic aspects of the substitution decomposition in optimization over relations, set systems and boolean functions. Annals of Operations Research 4, 195–225 (1985)
8. Möhring, R.H., Radermacher, F.J.: Substitution decomposition for discrete structures and connections with combinatorial optimization. Annals of Discrete Mathematics 19, 257–356 (1984)
9. Rao, M.: Solving some NP-complete problems using split decomposition. Discrete Applied Mathematics (2007)
10. Spinrad, J.: Prime testing for the split decomposition of a graph. SIAM Journal on Discrete Mathematics 2, 590–599 (1989)
11. Spinrad, J.P.: Two Dimensional Partial Orders. Ph.D thesis, Princeton University (1982)
12. Spinrad, J.P.: Graph partitioning (1985) (unpublished manuscript)
13. Spinrad, J.P.: On comparability and permutation graphs. Siam J. Comput. 14, 658–670 (1985)
14. Spinrad, J.P., Valdes, J.: Recognition and isomorphism of two-dimensional partial orders. In: Proceedings of the 10th Colloquium on Automata, Languages and Programming. LNCS, pp. 676–686. Springer, Berlin (1983)

Path-Bicolorable Graphs
(Extended Abstract)

Andreas Brandstädt[1], Martin C. Golumbic[2], Van Bang Le[1], and Marina Lipshteyn[2]

[1] Institut für Informatik, Universität Rostock, D-18051 Rostock, Germany
{ab,le}@informatik.uni-rostock.de
[2] Caesarea Rothschild Institute and Department of Computer Science, University of Haifa, Haifa, Israel
{golumbic,marinal}@cs.haifa.ac.il

Abstract. In this paper, we introduce the notion of path-bicolorability that generalizes bipartite graphs in a natural way: For $k \geq 2$, a graph $G = (V, E)$ is P_k-*bicolorable* if its vertex set V can be partitioned into two subsets (i.e., colors) V_1 and V_2 such that for every induced P_k (i.e., path with exactly $k-1$ edges and k vertices) in G, the two colors alternate along the P_k, i.e., no two consecutive vertices of the P_k belong to the same color V_i, $i = 1, 2$. Obviously, a graph is bipartite if and only if is P_2-bicolorable, every graph is P_k-bicolorable for some k and if G is P_k-bicolorable then it is P_{k+1}-bicolorable. The notion of P_k-bicolorable graphs is motivated by a similar notion of cycle-bicolorable graphs introduced in connection with chordal probe graphs. Moreover, P_3- and P_4-bicolorable graphs are closely related to various other concepts such as 2-subcolorable graphs, P_4-bipartite graphs and alternately orientable graphs.

We give a structural characterization of P_3-bicolorable graphs which also implies linear time recognition of these graphs. Moreover, we give a characterization of P_4-bicolorable graphs in terms of forbidden subgraphs.

Keywords: P_3-bicolorable graphs; P_4-bicolorable graphs; bipartite graphs; linear time recognition.

1 Introduction

Let $k \geq 2$ be an integer. We call a graph $G = (V, E)$ P_k-*bicolorable* if its vertex set V can be partitioned into two disjoint subsets V_1 and V_2 (the *color classes*) such that for every induced P_k (i.e., a path with exactly $k-1$ edges and k vertices) in G, no two consecutive vertices of the P_k belong to the same color class. Obviously, the P_2-bicolorable graphs are exactly the bipartite graphs. Let $\mathcal{P}_k$ denote the class of all P_k-bicolorable graphs. Then $\mathcal{P}_k \subset \mathcal{P}_{k+1}$.

This notion is motivated by various other concepts which were previously studied.

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 172–182, 2009.

1. In connection with chordal probe graphs, first studied in [13] and [23], Berry, Golumbic and Lipshteyn [3] define the following class of graphs: A graph $G = (V, E)$ is *cycle-bicolorable* if its vertex set V can be partitioned into two subsets V_1 and V_2 (the *color classes*) such that for every chordless cycle C in G, no two consecutive vertices in C belong to the same color class, i.e., the two colors alternate in each chordless cycle of G. Bipartite graphs and chordal graphs as well as chordal probe graphs are cycle-bicolorable, and cycle-bicolorable graphs are odd-hole-free and antihole-free [13,23] from which it follows by the famous Strong Perfect Graph Theorem [8] (i.e., a graph is perfect if and only if it contains no odd hole and no odd antihole) that cycle-bicolorable graphs are perfect. Berry, Golumbic and Lipshteyn [3] characterized cycle-bicolorable graphs and gave a polynomial time recognition algorithm for the class.
2. Hoàng [15] defined two related notions: A graph $G = (V, E)$ is *alternately orientable* if the edges of G can be oriented in such a way that in every chordless cycle of G, the orientations of two consecutive edges ab and bc alternate, i.e., if ab is directed $a \to b$ then bc is directed $c \to b$. A graph $G = (V, E)$ is *alternately colorable* if the edges of G can be colored by two colors in such a way that in every chordless cycle of G, the colors of consecutive edges alternate. Hoàng [15] showed that both, alternately orientable and alternately colorable graphs, are classes of perfect graphs by using the important fact that in a minimal imperfect graph, every P_3 extends into a chordless cycle (which is shown in [15]).

It is easy to see that cycle-bicolorable graphs are alternately orientable (if the two colors are red and green, orient every edge from red to green) but not vice versa. This gives a proof of perfection of cycle-bicolorable graphs without using the Strong Perfect Graph Theorem.

It is clear that every P_3-bicolorable graph is cycle-bicolorable but not vice versa as the example of the bull shows (see Figure 1).

Other papers related to P_3-bicolorable graphs are [1,7,11]: A graph $G = (V, E)$ is 2-*subcolorable* if V admits a partition into disjoint subsets V_1 and V_2 such that every P_3 in G contains vertices from both V_1 and V_2. Clearly, P_3-bicolorable graphs are 2-subcolorable but not vice versa. In [7,11], among others, it is shown that recognizing 2-subcolorable graphs is NP-complete, even for restricted graph classes.

A paper related to P_4-bicolorable graphs is [16]: A graph $G = (V, E)$ is P_4-*bipartite* if V admits a partition into disjoint subsets V_1 and V_1 such that every P_4 in G contains vertices from both V_1 and V_2. Clearly, P_4-bicolorable graphs are P_4-bipartite graphs but not vice versa. In [16], it was shown that recognizing P_4-bipartite graphs is NP-complete. As we will see (cf. Proposition 1), recognizing P_4-bicolorable graphs is solvable in polynomial time. While no non-trivial characterization of P_4-bipartite graphs is known, we are able to give a characterization of P_4-bicolorable graphs in terms of forbidden induced subgraphs.

Other well-investigated graph classes related to P_4-bicolorable graphs are cographs (i.e., P_4-free graphs) [9,10], P_4-reducible graphs [18,19] and P_4-sparse graphs [14,20], to mention just a few of the many papers dealing with generalizations of cographs (see [6] for further examples).

Our main results are structural characterizations of P_3- as well as of P_4-bicolorable graphs. Due to space limitations, the longest and most complicated proofs are omitted.

2 Basics and Terminology

Throughout this paper, let $G = (V, E)$ be a finite undirected graph without self-loops and multiple edges with vertex set V and edge set E, and let $|V| = n$, $|E| = m$. For a vertex $v \in V$, let $N(v) = \{u \mid uv \in E\}$ denote the (*open*) *neighborhood* of v in G, and let $N[v] = \{v\} \cup \{u \mid uv \in E\}$ denote the *closed neighborhood* of v in G. We also write $x \sim y$ for $xy \in E$ and $x \not\sim y$ for $xy \notin E$. A *clique* is a set of vertices which are mutually adjacent. A *stable set* is a set of vertices which are mutually nonadjacent. For subset X of vertices, write $N(X) = \cup_{x \in X} N(x) - X$. As usual, the *complement graph* $\overline{G}$ of $G = (V, E)$ is the graph with vertex set V and for $x \neq y$, xy is an edge in $\overline{G}$ if and only if $xy \notin E$. Sometimes we also denote $\overline{G}$ by co-G.

For $U \subseteq V$, let $G[U]$ denote the subgraph of G induced by U. Throughout this paper, all subgraphs are understood to be induced subgraphs. Let $\mathcal{F}$ denote a set of graphs. A graph G is *$\mathcal{F}$-free* if none of its induced subgraphs is in $\mathcal{F}$.

For two disjoint sets X and Y of vertices in a graph, we write X①Y (X⓪Y) for the fact that every vertex in X is adjacent (nonadjacent) to every vertex in Y.

For $k \geq 1$, let P_k denote a chordless path with k vertices and $k-1$ edges, and for $k \geq 3$, let C_k denote a chordless cycle with k vertices and k edges. A *hole* is a C_k with $k \geq 5$. An *odd hole* is a C_{2k+1} with $k \geq 2$. An *antihole* is the complement $\overline{C_k}$ for $k \geq 5$. An *odd antihole* is the complement of an odd hole. A graph is *chordal* if it contains no induced C_k, $k \geq 4$. $G = (V, E)$ is *bipartite* if there is a partition $V = X \cup Y$ with stable sets X and Y. A bipartite graph $G = (X, Y, E)$ is a *complete bipartite* graph if X①Y.

Two vertices $x, y \in V$ are *true twins* if they have the same closed neighborhood, i.e., $N[x] = N[y]$. The *true twin operation* adds a new vertex y to graph G which is a true twin to an already existing vertex x in G.

A vertex subset $U \subseteq V$ is a *module* in G if for all $v \in V \setminus U$, either v is adjacent to all vertices of U or v is adjacent to none of them. We note that the modules of G are indentical to the modules of $\overline{G}$. A module U is *nontrivial* if $U \neq V$ and $|U| > 1$. A *homogeneous set* is a nontrivial module. A graph G is *prime* if all its modules are trivial. A *clique module* in G is a module which induces a clique in G. Obviously, true twins are a clique module.

It is well known (see e.g. [6]) that in a connected and co-connected graph G, the maximal homogeneous sets are pairwise disjoint, and if G^* denotes the result of contracting each maximal homogeneous set to a corresponding single

vertex (the so-called *characteristic graph* of G) then G^* is prime. The *modular decomposition* of a graph is based upon this property; see e.g. [24].

A result of Hoàng and Reed [17, Claim 3.5], describes how in a prime graph, a C_4 extends to a larger subgraph; let A be the graph consisting of six vertices $v_1, \ldots, v_6$ such that vertices v_i, $i \in \{1, 2, 3, 4\}$ induce a C_4 with edges $v_i v_{i+1}$ (index arithmetic modulo 4), and v_5 (v_6, respectively) is only adjacent to v_1 (v_4, respectively). The *domino* is the graph with the same six vertices as A where additionally to A, the pair $v_5 v_6$ is an edge. The graph called *house* (see Figure 1) is the complement of P_5.

Lemma 1 ([17]). *If a prime graph contains an induced C_4 then it contains an induced house, A or domino.*

In its complement version, Lemma 1 says that if a prime graph contains $2K_2$ (i.e., $\overline{C_4}$) then it contains P_5 or $\overline{A}$ or co-domino.

Other small graphs which play an important role in this paper are shown in Figure 2 where the house is called G_1; the graph G_2 is also called *net*, and its complement G_3 is also called 3-*sun* or S_3. Note that $G_6 = \overline{A}$, $\overline{G_4} = G_7$ and $\overline{G_{12}} = G_{13}$.

Let $k \geq 2$ be a fixed integer. For a graph $G = (V, E)$, the *reduced graph* $\mathrm{r}_k(G)$ of G is obtained from G by removing all edges in G that do not belong to an induced P_k in G. The 2-CNF formula $\mathrm{F}_k(G)$ associated to G is defined as follows: The boolean variables are the vertices of G, and for each induced P_k $x_1 x_2 \ldots x_{k-1} x_k$ of G, $(x_i \vee x_{i+1})$ and $(\overline{x_i} \vee \overline{x_{i+1}})$, $1 \leq i \leq k-1$, are $2k-2$ clauses, the P_k-*clauses* for that P_k. The formula $F_k(G)$ is then the conjunction of all P_k-clauses for all P_k in G and of all one-literal clauses (v) for each vertex v that is not contained in any induced P_k in G.

Proposition 1. *Let $k \geq 2$ be an integer. The following statements are equivalent for any graph G.*

(i) *G is P_k-bicolorable;*
(ii) *$\mathrm{r}_k(G)$ is bipartite;*
(iii) *$\mathrm{F}_k(G)$ is satisfiable.*

Proof. (i) $\Rightarrow$ (ii), (iii): Let $G = (V, E)$ be P_k-bicolorable, and let $V = V_1 \cup V_2$ be a P_k-bicoloring of G. Then, first, all edges of G connecting two vertices in V_1 or two vertices in V_2 are not contained in any induced P_k in G, hence V_1 and V_2 are stable sets in $\mathrm{r}_k(G)$, and (ii) holds. Next, assign all $x \in V_1$ to `true` and all $y \in V_2$ to `false`. Clearly, $\mathrm{F}_k(G)$ is satisfied by this truth assignment, and (iii) holds.

(ii) $\Rightarrow$ (i): Let $V = V_1 \cup V_2$ be a bipartition of V into disjoint subsets V_1 and V_2 that are stable sets in $\mathrm{r}_k(G)$. Since every induced P_k in G is also an induced P_k in $\mathrm{r}_k(G)$, each edge of any P_k in G must connect a vertex in V_1 and a vertex in V_2, i.e., G is P_k-bicolorable.

(iii) $\Rightarrow$ (i): Let $\mathrm{F}_k(G)$ be satisfied by a truth assignment, and let V_1 consist of all true vertices and V_2 consist of all false vertices. Then, by definition of $\mathrm{F}_k(G)$, every edge of any induced P_k in G must connect a vertex in V_1 and a vertex in V_2, i.e., G is P_k-bicolorable. □

Corollary 1. *For every fixed $k \geq 2$, P_k-bicolorable graphs can be recognized in polynomial time.*

Indeed, if t_k is the time needed to listing all P_k in G, $t_0 := O(1)$, then Proposition 1 gives a recognition algorithm for testing if G is P_k-bicolorable in time $O(t_{k-2} \cdot m)$ where m is the number of edges of G.

3 P_3-Bicolorable Graphs

The following observation is easy to see.

Observation 1. *Odd holes and the house, bull, dart, gem and W_4 (see Figure 1) are not P_3-bicolorable.*

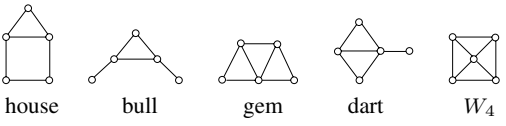


Fig. 1. Minimal odd-hole-free non-P_3-bicolorable graphs

Theorem 1. *For all graphs G, the following statements are equivalent:*

(i) *G is P_3-bicolorable;*
(ii) *G is (odd-hole, house, bull, dart, gem, W_4)-free;*
(iii) *Every induced subgraph of G is bipartite or has true twins;*
(iv) *G results from a bipartite graph B by substituting cliques into the vertices of B.*

Proof. (i) $\Rightarrow$ (ii): By Observation 1.

(ii) $\Rightarrow$ (iii): It suffices to prove that G itself is bipartite or has true twins since all induced subgraphs of G also satisfy (ii). Suppose that G is not bipartite. Then, since G is odd-hole-free, G has a maximal clique Q with at least three vertices. If Q has no neighbor in $G - Q$ then clearly, every two vertices in Q form true twins. Now, let $v \in G - Q$ be adjacent to a vertex $q_1 \in Q$. As Q is a maximal clique, there is a vertex $q_2 \in Q$ such that v is nonadjacent to q_2. Consider a vertex $q_3 \in Q \setminus \{q_1, q_2\}$.

If v is nonadjacent to q_3 then q_2, q_3 form true twins since otherwise, if a vertex u distinguishes q_2 and q_3, say u is adjacent to q_2 and nonadjacent to q_3 then u, v, q_1, q_2, q_3 would induce a bull or dart or gem or house.

If v is adjacent to q_3 then q_1, q_3 form true twins since otherwise, if a vertex u distinguishes q_1 and q_3, say u is adjacent to q_1 and nonadjacent to q_3 then u, v, q_1, q_2, q_3 would induce a dart or gem or W_4.

(iii) $\Rightarrow$ (iv): If G is not bipartite, let x, y be true twins in G. Then (iv) follows easily by the induction hypothesis for $G - y$.

(iv) $\Rightarrow$ (i) is obvious. □

Theorem 1 implies further structural properties of P_3-bicolorable graphs that lead to a linear time recognition algorithm as we will explain below. A *universal vertex* is a vertex adjacent to all other vertices.

Corollary 2. *Let G be a connected graph with disconnected complement $\overline{G}$. Then the following statements are equivalent:*

(i) *G is P_3-bicolorable;*
(ii) *G is (P_4, dart, W_4)-free;*
(iii) *G results from a complete bipartite graph by substituting cliques into the vertices;*
(iv) *G has exactly two co-connected components each of which induces a P_3-free subgraph in G, or the set C of all universal vertices in G is a non-empty clique and $G - C$ is P_3-free.*

Proof. (i) $\Rightarrow$ (ii): By noting that if G has a P_4, then, as $\overline{G}$ is disconnected, G has a gem; thus (ii) follows by Theorem 1 (ii).

(ii) $\Rightarrow$ (iii): Let G satisfy (ii). Then G clearly satisfies (iii) in Theorem 1, hence by Theorem 1 (iv), G is obtained from a bipartite graph B by substituting cliques into vertices of B. As G is connected and P_4-free, B is connected and P_4-free. Thus, (iii) follows by noting that connected P_4-free bipartite graphs are exactly the complete bipartite graphs.

(iii) $\Rightarrow$ (iv): Let G satisfy (iii) with a complete bipartite graph $B = (X, Y, E)$. Now, if $|X| \geq 2$ and $|Y| \geq 2$, then clearly the first case in condition (iv) occurs, and if $|X| = 1$ or $|Y| = 1$, the second case.

(iv) $\Rightarrow$ (i) is obvious. □

Corollary 3. *Let G be a connected graph with connected complement $\overline{G}$. Then G is P_3-bicolorable if and only if its characteristic graph G^* is bipartite and each homogeneous set in G induces a P_3-free subgraph.*

Proof. ($\Rightarrow$): Let $G = (V, E)$ be P_3-bicolorable. First, as G^* is (isomorphic to) an induced subgraph of G, G^* is also P_3-bicolorable, hence, by Theorem 1 (iii), G^* is bipartite. Next, consider a homogeneous set M in G, and let $R = V - (N(M) \cup M)$. As G is connected, $N(M) \neq \emptyset$, and as $\overline{G}$ is connected, $R \neq \emptyset$. Moreover, $N(R) \cap N(M) \neq \emptyset$. Thus, any P_3 in $G[M]$ together with a vertex $u \in N(M)$ and a vertex $v \in R$ adjacent to u would induce a dart. Hence $G[M]$ must be P_3-free.

($\Leftarrow$): This direction is clear by noting that G is obtained from G^* by substituting P_3-free graphs into vertices of G^*. □

Theorem 2. *P_3-bicolorable graphs can be recognized in linear time.*

Proof. By Corollaries 2 and 3, any linear time algorithm for modular decomposition (see e.g. [24]) gives a linear time algorithm for recognizing whether a given graph G is P_3-bicolorable: If G is disconnected then decompose G into its connected components. If $\overline{G}$ is disconnected then check whether G satisfies (iv) in Corollary 2, and finally, if G is connected and co-connected then check the conditions of Corollary 3. Clearly, all this can be done in linear time. □

4 3-Leaf Powers and P_3-Bicolorable Graphs

A graph $G = (V, E)$ is a 3-*leaf power* (see [25]) if there exists a tree T such that (i) all leaves of T are exactly the vertices of G and (ii) two distinct vertices in G are adjacent if and only if they are at distance at most three in T. In [5], among others, the following is shown:

Theorem 3 ([5]). *A connected graph is a 3-leaf power if and only if it is obtained from a tree T by substituting the vertices of T by cliques.*

From Theorems 1 and 3, we conclude

Corollary 4. *For any connected chordal graph G, the following statements are equivalent:*

(i) *G is P_3-bicolorable;*
(ii) *G is (bull, dart, gem)-free;*
(iii) *G is a 3-leaf power;*
(iv) *G is the result of substituting cliques into the vertices of a suitable tree.*

A linear-time recognition algorithm for 3-leaf powers is given in [5] (that improves the time bound $O(n^3)$ given in [25]). Corollary 4 means that the linear-time recognition algorithm for P_3-bicolorable graphs in Theorem 2 provides another way to recognize 3-leaf powers in linear time.

5 P_4-Bicolorable Graphs

The main result of this section, namely Theorem 4, gives a characterization of P_4-bicolorable graphs in terms of forbidden induced subgraphs. The following observation is easy to verify.

Observation 2. *Odd holes and the graphs $G_1, \ldots, G_{13}$ depicted in Figure 2 are not P_4-bicolorable.*

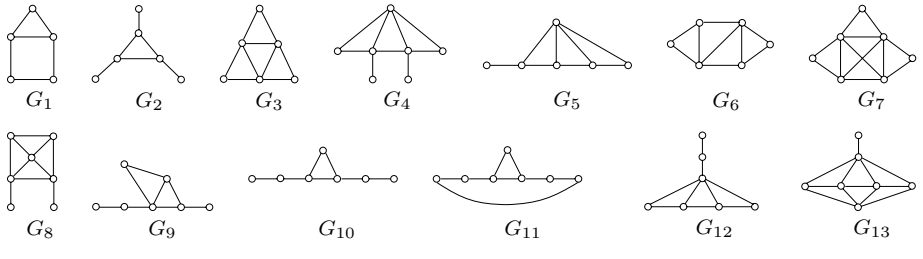


Fig. 2. Minimal non-P_4-bicolorable graphs $G_1, \ldots, G_{13}$

It turns out that the following notion of P_4-connectedness introduced in [21] is helpful in studying P_4-bicolorable graphs. A graph $G = (V, E)$ is *P_4-connected*, if for every partition of V into nonempty disjoint sets V_1, V_2 there exists a P_4 containing vertices from both V_1 and V_2, a *crossing* P_4. The maximal induced subgraphs in G which are P_4-connected are the *P_4-connected components* of G.

Observation 3. *Let G be a P_4-connected (G_{12}, G_{13})-free graph. Then every homogeneous set in G induces a cograph.*

Proof. Let M be a homogeneous set in $G = (V, E)$. As G is P_4-connected, there exists an induced P_4, say P, crossing M and $V - M$. Assuming Q is an induced P_4 in $G[M]$, $Q \cup (P - M)$ induces a G_{12} or a G_{13}. Thus, $G[M]$ is P_4-free. □

The subsequent Observation 4 follows from the definition of P_4-connectedness and from Observation 3; note that if G is P_4-connected then G and $\overline{G}$ are connected, and recall that G^* is obtained from G by contracting each maximal homogeneous set to a single vertex.

Observation 4

(i) *G is P_4-bicolorable if and only if each P_4-connected component of G is P_4-bicolorable.*
(ii) *Let G be a P_4-connected graph. Then G is P_4-bicolorable if and only if each homogeneous set in G induces a cograph and G^* is P_4-bicolorable.*

Due to space limitations, the proof of the main result of this section, namely of Theorem 4, is omitted. It makes use of the following Lemmas 2 and 3.

A *co-pan* is a graph with five vertices, say a, b, c, d, e and edges ab, bc, cd, ce, de; the edge de is called the *cross edge* of the co-pan. Similarly, a *co-chair* is a graph with five vertices, say a, b, c, d, e and edges ab, bc, bd, cd, ce, de; the edge cd is called the *cross edge* of the co-chair.

Lemma 2. *Let G be an odd-hole-free and $(G_1, \ldots, G_{13})$-free graph. Then no cross edge of a co-pan in G is contained in a P_4 of G.*

Lemma 3. *Let G be an odd-hole-free and $(G_1, \ldots, G_{13})$-free graph. Then no cross edge of a co-chair in G is contained in a P_4 of G.*

Theorem 4. *A graph is P_4-bicolorable if and only if it is odd-hole-free and $(G_1, \ldots, G_{13})$-free.*

6 P_4-Bicolorable Graphs and Related Graph Classes

We already mentioned that there are other well-investigated graph classes related to P_4-bicolorable graphs: Obviously, every cograph (i.e., P_4-free graph) [9,10] is P_4-bicolorable. *P_4-sparse graphs* [14,20] are those graphs where each collection of five vertices induces at most one P_4. By definition, G is P_4-sparse if and only if it is $(C_5, P_5, \overline{P_5}$,chair, co-chair,pan,co-pan)-free, and a result of [14] characterizes the prime P_4-sparse graphs as the spiders. G is *P_4-reducible* [18,19] if G is P_4-sparse and $(S_3, \overline{S_3})$-free. Then Theorem 4 implies:

Corollary 5

(i) *P_4-reducible graphs are P_4-bicolorable.*
(ii) *A P_4-sparse graph is P_4-bicolorable if and only if it is a P_4-reducible graph.*

Another interesting result in connection with P_4-bicolorable graphs is a characterization of those graphs which are interval graphs and whose complement is an interval graph. It is well known [12] that a graph is a split graph if and only if it is $(2K_2, C_4, C_5)$-free.

Theorem 5 ([2]). *A graph and its complement is an interval graph if and only if it is a (G_2, G_3, G_4, G_7)-free split graph.*

Recall that $G_3 = \overline{G_2}$, $G_7 = \overline{G_4}$ and $G_{13} = \overline{G_{12}}$. Moreover, $G_6 = \overline{A}$.

By Theorem 4 we have:

Corollary 6. *G and $\overline{G}$ are P_4-bicolorable if and only if G is $(C_5, P_5, \overline{P_5}, G_2, \overline{G_2}, G_4, \overline{G_4}, A, \overline{A}, G_{12}, \overline{G_{12}})$-free.*

Theorem 6. *For a graph G, G and $\overline{G}$ are P_4-bicolorable if and only if for each of its P_4-conncected components H of G,*

(i) *the homogeneous sets of H induce cographs, and*
(ii) *the characteristic graph H^* of H is an interval and co-interval graph.*

Proof. "$\Longrightarrow$": By Observation 3, the homogeneous sets of P_4-connected components H induce cographs. H^* is prime and thus, by Lemma 1, contains no $2K_2$ and no C_4, i.e., it is a split graph which moreover is $(G_2, \overline{G_2}, G_4, \overline{G_4})$-free. Thus, by Theorem 5, H^* is an interval and co-interval graph.

"$\Longleftarrow$": By Observation 4 (i), G is P_4-bicolorable if and only if each P_4-connected component of G is P_4-bicolorable. Thus it suffices to show that each P_4-conncected component H is P_4-bicolorable. If the characteristic graph H^* of H is an interval and co-interval graph then by Theorem 5 and by Theorem 4, H^* is P_4-bicolorable. Moreover, by substituting cographs as homogeneous sets, this property is maintained, i.e., a P_4-bicoloring of H^* remains a P_4-bicoloring of H. □

Another interesting class was studied in [26]:

Theorem 7 ([26]). *A graph is superbrittle if and only if it is $(C_5, P_5, \overline{P_5}, A, \overline{A}, G_{12}, \overline{G_{12}})$-free.*

Theorem 8 ([4]). *A graph G is superbrittle if and only if for each of its P_4-conncected components H of G,*

(i) *the homogeneous sets of H induce cographs, and*
(ii) *the characteristic graph H^* of H is a split graph.*

Corollary 7. *If G and its complement $\overline{G}$ are P_4-bicolorable then G is superbrittle.*

7 Conclusion

In this extended abstract, we defined the classes of P_k-bicolorable graphs for $k \geq 2$. The P_2-bicolorable graphs are precisely the bipartite graphs, and we have characterized the classes of P_3-bicolorable and P_4-bicolorable graphs in terms of forbidden induced subgraphs. We have also shown that P_3-bicolorable graphs can be recognized in linear time. Note that Proposition 1 gives a recognition algorithm for P_4-bicolorable graphs in $O(m^2)$ time. We pose the question whether the P_4-bicolorable graphs can be recognized in linear time.

Obviously, for every fixed k the recognition of P_k-bicolorable graphs can be done in polynomial time. If k is part of the input, this is no longer clear: T. Kaiser [22], based on an idea of J. Kratochvíl, has shown that a given graph G with n vertices and a specified vertex v has a Hamiltonian path starting at v if and only if a simple derived graph G' is *not* P_{2n+2}-bicolorable.

References

1. Albertson, M.O., Jamison, R.E., Hedetniemi, S.T., Locke, S.C.: The subchromatic number of a graph. Discrete Math. 74, 33–49 (1989)
2. Benzaken, C., Hammer, P.L., de Werra, D.: Split graphs of Dilworth number 2. Discrete Math. 55, 123–128 (1985)
3. Berry, A., Golumbic, M.C., Lipshteyn, M.: Recognizing chordal probe graphs and cycle-bicolorable graphs. SIAM J. Discrete Math. 21, 573–591 (2007)
4. Brandstädt, A., Le, V.B.: Split-Perfect Graphs: Characterizations and Algorithmic Use. SIAM J. Discrete Math. 17, 341–360 (2004)
5. Brandstädt, A., Le, V.B.: Structure and linear time recognition of 3-leaf powers. Information Processing Letters 98, 133–138 (2006)
6. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. SIAM Monographs on Discrete Math. Appl., vol. 3. SIAM, Philadelphia (1999)
7. Broersma, H., Fomin, F.V., Nešetřil, J., Woeginger, G.: More about subcolorings. Computing 69, 187–203 (2002)
8. Chudnovsky, M., Robertson, N., Seymour, P., Thomas, R.: The strong perfect graph theorem. Annals of Mathematics 64, 51–229 (2006)
9. Corneil, D.G., Lerchs, H., Stewart-Burlingham, L.: Complement reducible graphs. Discrete Appl. Math. 3, 163–174 (1981)
10. Corneil, D.G., Perl, Y., Stewart, L.K.: A linear recognition algorithm for cographs. SIAM J. Computing 14, 926–934 (1985)
11. Fiala, J., Jansen, K., Le, V.B., Seidel, E.: Graph subcolorings: Complexity and Algorithms. SIAM J. Discrete Math. 16, 635–650 (2003)
12. Földes, S., Hammer, P.L.: Split graphs. Congressus Numerantium 19, 311–315 (1977)
13. Golumbic, M.C., Lipshteyn, M.: Chordal probe graphs. Discrete Applied Mathematics 143, 221–237 (2004)
14. Hoàng, C.T.: Perfect Graphs, Ph.D. Thesis, School of Computer Science, McGill University Montreal (1985)
15. Hoàng, C.T.: Alternating orientation and alternating colouration of perfect graphs. Journal of Combinatorial Theory (B) 42, 264–273 (1987)

16. Hoàng, C.T., Le, V.B.: P_4-free colorings and P_4-bipartite graphs. Discrete Math. and Theoretical Computer Science 4, 109–122 (2001)
17. Hoàng, C.T., Reed, B.A.: Some classes of perfectly orderable graphs. Journal Graph Theory 13, 445–463 (1989)
18. Jamison, B., Olariu, S.: P_4-reducible graphs–a class of uniquely tree representable graphs. Studies in Appl. Math. 81, 79–87 (1989)
19. Jamison, B., Olariu, S.: A linear-time recognition algorithm for P_4-reducible graphs. Theoretical Computer Science 145, 329–344 (1995)
20. Jamison, B., Olariu, S.: Recognizing P_4-sparse graphs in linear time. SIAM J. Computing 21, 381–406 (1992)
21. Jamison, B., Olariu, S.: p-components and the homogeneous decomposition of graphs. SIAM J. Discrete Math. 8, 448–463 (1995)
22. Kaiser, T.: Communication (2007)
23. Lipshteyn, M.: Structured Families of Graphs: Properties, Algorithms, and Representations, Ph.D. Thesis, University of Haifa (2005)
24. McConnell, R.M., Spinrad, J.P.: Modular decomposition and transitive orientation. Discrete Math. 201, 189–241 (1999)
25. Nishimura, N., Ragde, P., Thilikos, D.M.: On graph powers for leaf-labeled trees. J. Algorithms 42, 69–108 (2002)
26. Preissmann, M., de Werra, D., Mahadev, N.V.R.: A note on superbrittle graphs. Discrete Math. 61, 259–267 (1986)

Path Partitions, Cycle Covers and Integer Decomposition
(Lecture Note)

András Sebő*

CNRS, Laboratoire G-SCOP, 46, Avenue Félix Viallet,
38000 Grenoble 38031 Grenoble, Cedex 1, France
Andras.Sebo@g-scop.inpg.fr

Abstract. A polyhedron P has the *integer decomposition property*, if every integer vector in kP is the sum of k integer vectors in P. We explain that the projections of polyhedra defined by totally unimodular constraint matrices have the integer decomposition property, in order to deduce the same property for coflow polyhedra defined by Cameron and Edmonds. We then apply this result to the convex hull of particular stable sets in graphs. Therebye we prove a generalization of Greene and Kleitman's well-known theorem on posets to arbitrary digraphs which implies recent and classical purely graph theoretical results on cycle covers, is closely related to conjectures of Berge and Linial on path partitions, and implies these for some particular values of the parameters.

1 Introduction

Partitioning the vertex-set of a graph by a minimum number of paths is one of the most natural problems concerning graphs. Minimizing the number of paths in such a partition contains the Hamiltonian Path problems both in the directed and undirected case.

For undirected graphs some variants involve matching theory some others the connectivity of graphs. Some results concern only particular classes of graphs. The only general result about minimum partitions of the vertex-set into paths in undirected graphs concerns intersection graphs of paths in a tree, by Monma and Wei [27], a class later generalized in [19], [20].

For digraphs, a classical theorem of Gallai and Milgram *relates the problem to the stability number of a graph*. Path partitions in digraphs have been treated both with elegant graph theory, network flows, and polyhedral combinatorics, but have not yet revealed all of their secrets:

Conjectures of Berge [4] and Linial [26] about the relation of maximum sets of vertices inducing a k-chromatic subgraph and particular path partitions resist

* Research sponsored and highly motivated by the France-Israel Binational Collaboration Grant "Recognition, Decomposition and Optimization Problems in Graph Theory".

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 183–199, 2009.

through the decades. Hartman's excellent survey [22] witnesses of the variety of the methods that have been tried out with a lot of partial results but no breakthrough as far as the general conjectures are concerned. Some other conjectures are less well-known or have not yet been stated.

Led by analogies, we ask and answer in this talk more questions, and point at some connections.

Section 2 states analogous pairs of theorems on path partitions and cycle covers.

Section 3 presents the results concerning cycle covers deducing all from a general theorem proved with the help of the property of a corresponding polyhedron: the integer decomposition property.

Section 4 presents some results on path partitions, and some connections of these to cycle covers.

Notation and Terminology: Let $G = (V, E)$ a digraph. A *path* of a digraph is an ordered set $P = (v_1, \ldots, v_{|P|})$ of vertices, all different, so that $v_i v_{i+1} \in E$ $(i = 1, \ldots, |P| - 1)$. We will denote $\mathrm{ini}(P) := v_1$ the *initial* (first) vertex of a path and $\mathrm{ter}(P) := v_{|P|}$ the *terminal* (last) vertex of it. For us a path will be a vertex-set, that is, with an abuse of notation we will apply set-operations involving a path P, and in this case P is just the set of its vertices. If the first and last points are equal it is called a *cycle* which also included one element sets (even if there is no incident loop). A *subpartition* is just a family of disjoint subsets of V.

For a family of sets $\mathcal{P}$, $R(\mathcal{P}) := V \setminus \cup\mathcal{P}$. This complementation concerns the complement *with respect to the graph in which it is defined*. If we delete some vertices and the new vertex set is V', for a subpartition $\mathcal{P}'$ of V', $R(\mathcal{P}')$ is defined as $R(\mathcal{P}') := V' \setminus \cup\mathcal{P}'$. We apply this notation only when the vertex-set to which we apply it is clear. A subpartition of paths is a *path partition* if and only if $R(\mathcal{P}) = \emptyset$.

If we do not say otherwise, $G = (V, E)$ is a digraph, $n := |V|$ and $m := |E|$.

A *stable* set is a subset of vertices that does not induce any edge. The maximum size of a stable set of a graph G is denoted by $\alpha = \alpha(G)$. The maximum size of a k-chromatic induced subgraph (equivalently, the union of k stable sets) is denoted by α_k; $\alpha_1 = \alpha$. The *chromatic number*, denoted by $\chi = \chi(G)$, is the minimum of k such that $\alpha_k = n$. The minimum size of a partition into paths is denoted by $\pi = \pi(G)$, and the minimum cover by cycles is $\zeta = \zeta(G)$. By convention $\{v\}$ is also a cycle for all $v \in V$ (and it is of course a path too). So $\pi, \zeta \leq n$. The number of vertices of the longest path is denoted by $\lambda = \lambda(G)$.

Two families of sets are called *orthogonal* if taking any set of each, the intersection is always 1. A family is said to *cover* a set, if the union of its members contains the set.

The subgraph induced by a set $X \subseteq V$ will be denoted by $G(X)$, just replaced by X to avoid double parentheses, for instance $\alpha(X) := \alpha(G(X))$;"strongly connected" will sometimes be replaced by *strong*.

2 Pairs of Assertions

2.1 Tournaments ($\alpha = 1$)

A *tournament* is an oriented complete graph.

Theorem 1 (Rédei [28])**.** *Let G be a tournament. Then it has a Hamiltonian Path.*

Theorem 2 (Camion [12])**.** *Let G be a strong tournament. Then it has a Hamiltonian Cycle.*

We state "loose" and "tight" versions of some assertions. The former refers to inequalities that generalize the nontrivial inequalities of minmax theorems (of Dilworth's, of Greene-Kleitman's or of some more recent ones), and the latter generalize "complementary slackness" (the sstructure implied by the equality in these inequalities).

2.2 Stability ($k = 1$, loose)

Theorem 3 (Gallai, Milgram [17])**.** *Let G be an arbitrary digraph. Then $\alpha \geq \pi$.*

Theorem 4 (Bessy, Thomassé [7], Gallai's conjecture [16])**.** *Let G be strong . Then $\alpha \geq \zeta$.*

Specializing these to acyclic transitive digraphs both imply the nontrivial part of Dilworth's theorem stating equality in the former theorem for acyclic transitive digraphs (posets), see for instance [33]. To deduce it from the latter theorem, we first have to make an acyclic transitive digraph strongly connected. This can be done for instance by adding a "supersource" and joining it to all the vertices of 0 indegree (sources), adding a "supersink" and joining all the vertices of 0 outdegree (sinks) to it, and adding an arc from the supersink to the supersource. (Just one vertex joined to and from all vertices is also a possible choice.)

2.3 Stability ($k = 1$, tight)

Any proof of the Gallai-Milgram theorem obviously provides the following:

Theorem 5. *Let G be an arbitrary digraph. Then there exists a path partition $\mathcal{P}$ of G and a stable set orthogonal to $\mathcal{P}$.*

Since Theorem 4 is the weakening of a min-max theorem, the condition of equality ("complementary slackness") easily implies:

Theorem 6. *Let G be a strong digraph. Then there exists a cycle cover $\mathcal{C}$ of G and a stable set S orthogonal to $\mathcal{C}$, where each element of S is covered by exactly one member of $\mathcal{C}$.*

Of course these statements can also be specialized to Dilworth's theorem, with the same reduction as before.

2.4 Coloring ($k = \lambda$, loose)

Theorem 7 (Gallai, Roy [18], [29]). *Let G be an arbitrary digraph. Then there exists a path of size at least χ.*

Theorem 8 (Bondy [8]). *Let G be a strong digraph. Then there exists a cycle of size at least χ.*

2.5 Coloring ($k = \lambda$, tight)

Any proof of the Gallai-Roy theorem obviously provides the following:

Theorem 9. *[8] Let G be an arbitrary digraph. Then for any longest path there exists a colouring whose color classes are orthogonal to the path.*

Bondy's theorem does correspond to an LP duality theorem, but the vertices are not stable sets and cycles ; the corresponding polyhedron does have fractional vertices [31], and it is not evident how it would imply an analogous theorem for the longest cycle in a strongly connected graph. Nevertheless, this tight version of Bondy's theorem is true, and this is actually what Bondy proved:

Theorem 10. *[8] Let G be a strong digraph. Then there exists a cycle and a colouring so that the color classes are orthogonal to the cycle.*

Here is a somewhat different structural sharpening of the Gallai-Roy theorem:

Conjecture 1 (Laborde, Payan, Xuong [25]). *Let G be an arbitrary digraph. Then there exists a stable set in G that meets every longest path.*

Could this be true replacing "longest path" by "longest cycle" in strongly connected graphs ?

2.6 General (Loose)

Conjecture 2 (Linial [26]). *Let G be a digraph. Then* $\alpha_k \geq \min_{X \subseteq V}\{|X| + k\pi(V \setminus X)\}$.

Theorem 11 (Sebő [31]). *Let G be a strong digraph. Then* $\alpha_k \geq \min_{X \subseteq V}\{|X| + k\zeta(V \setminus X)\}$.

Corollary 1 *(*Greene-Kleitman [21]). Let G be a transitive acyclic digraph. Then $\alpha_k = \min_{X \subseteq V}\{|X| + k\zeta(V \setminus X)\}$.

Indeed, for transitive acyclic digraphs " $\leq$ " is easy, and to prove the nontrivial inequality of the Greene-Kleitman theorem, the reduction of Subsection 2.2 to strongly connected graphs works again. So the corollary indeed follows from the preceding theorem. Note that the right hand side of the Greene-Kleitman theorem or of Linial's conjecture is usually written as $\min\{\sum_{P \in \mathcal{P}} \min\{|P|, k\} : \mathcal{P}$ *is a path partition*$\}$, and this sum is called the *k-norm* of $\mathcal{P}$.

2.7 General (Tight)

The following is a simple already unknown version of Berge's conjecture. It implies Linial's conjecture. (The two conjectures have been stated independently.)

Conjecture 3 (Berge [4]). *Let G be a digraph, and $k \in \mathbb{N}$, $k \geq 1$. Then there exists $X \subseteq V$, a path partition $\mathcal{P}$ of $V \setminus X$, and k disjoint stable sets orthogonal to $\mathcal{P}$ whose union contains X.*

We prove now the cycle cover version of this conjecture. Note that there are several options here for replacing "there exists" by "for all". Berge originally stated *for all $X \subseteq V$, and path partition $\mathcal{P}$ of $V \setminus X$ minimizing $|X| + k|\mathcal{P}|$.* We will call this *strongest conjecture* since it implies the above weaker assertion which implies in turn Linial's conjecture.

Theorem 12 (Sebő [31]). *Let G be a strong digraph, and $k \in \mathbb{N}$, $k \geq 1$. Then there exists $X \subseteq V$, a cycle cover $\mathcal{C}$ of $V \setminus X$, and k disjoint stable sets covering X, all orthogonal to $\mathcal{C}$. Furthermore, each element of the stable sets is covered by at most one member of $\mathcal{C}$.*

This last theorem easily implies all that has been previously stated about covers in strongly connected graphs:

It shows k stable sets whose union U satisfies $|U| = |X| + k|\mathcal{C}|$ for some $X \subseteq V$ and cycle cover $\mathcal{C}$ of $V \setminus X$. Theorem 11 follows since $\alpha_k \geq |U|$. Theorem 12 and Theorem 11 are central in our presentation. We show here how they can be proved through the integer decomposition property of coflow polyhedra, and how they can be useful.

Theorem 8 follows from Theorem 12 because choosing k to be the size of the longest cycle, $|X| + k|\mathcal{C}| \geq |X| + |\cup \mathcal{C}| \geq n$, so the union of the k disjoint stable sets provided by Theorem 12 is at least n. So G can be colored with k colors.

The $k = 1$ special case of Theorem 12 is Theorem 6, itself implying Theorem 4. Indeed, in this case the elements in X can be replaced by 1-element cycles.

Theorem 12 will, in turn, be proved in Section 3.3.

3 Cycle Covers

The ultimate goal of this section is to prove the second theorem of each subsection of Section 2. They have already been proved from Theorem 12, and here we will prove this latter. There are some interesting tools on the way, and they will lead us further: the integrality and integer decomposition property of *coflow* polyhedra, and the *coherent orders* of Knuth, Bessy and Thomassé. For the notations and basic notions from polyhedral combinatorics (including TDI, integer decomposition, etc.) we refer in this extended abstract to [32], [33]. The talk will be self-contained.

3.1 Coflows and Integer Decomposition

We wish to introduce here a ready to use helpful treatment of node-capacitated circulation problems. The idea is well-known: node-capacities can be reduced to edge-capacities by splitting each vertex v into two copies, an in-copy v_{in} and an out-copy v_{out} and adding the arc $v_{\text{in}}v_{\text{out}}$ with the given vertex-capacities (possibly lower and upper), see [32], [11]. It is less well-known that relevant cycle-cover or cycle packing problems arise in this way and have useful properties, such as box total dual integrality, primal integrality if the parameters are integers [11], furthermore integer decomposition (below). This elegant tool defined by Cameron and Edmonds is defined as follows:

The *coflow system of inequalities* $\mathcal{Q}(G, a, b, c)$, where $G = (V, E)$ is a digraph, $a, b : V(G) \to \mathbb{Z}$, $c : E \to \mathbb{Z}$ is the following system in $n := |V|$ variables x_v $(v \in V)$:

$$x(V_C) \leq c(E_C) \text{ for every cycle } C \text{ with vertex-set } V_C \text{ and edge-set } E_C \ ,$$

$$a \leq x \leq b.$$

The set of points $x \in \mathbb{R}^V$ satisfying the coflow inequalities $\mathcal{Q}(G, a, b, c)$ is called the *coflow polyhedron*, and is denoted by $Q(G, a, b, c)$. The *coflow (primal) problem* $P(G, a, b, c, w)$, where G, a, b, c are as before, and $w : V(G) \to \mathbb{Z}$, is the following:

$$\max\{w^\top x : x \in Q(G, a, b, c)\}$$

$D(G, a, b, c, w)$ will denote the dual linear program, and $\text{opt}(G, a, b, c, w)$ the common optimum of the primal and the dual (which can also be infinite).

In [31] the coflow approach is followed by applying the splitting of vertices case by case, without stating any general theorem. (Coflows have been so far absent from books and surveys - we hope to contribute to their inclusion.) Both in [11] and [31], primal integrality is deduced from the TDI property (the above Lemma) through Edmonds and Giles' theorem. However, [11] observes that primal integrality can be directly deduced proving that the coflow polyhedron is the projection of the dual of a circulation problem, and in [31] the dual of the stated flow problem provides an integer primal solution to coflows.

We express all this in a slightly simpler way using Charbit's matrix from [13], which is smaller and simpler than the network matrices from [11] (or [31]) :

Given a digraph $G = (V, E)$, $n := |V|$, $m := |E|$, let A denote the $2n \times m$ matrix whose first n rows consist of the usual incidence matrix of G (one $+1$ and one -1 per column, the rest is 0) and the second m rows the same, except that the $+1$ are replaced by 0. It is easy to see that this matrix A is totally unimodular (as a submatrix of a network matrix, or see [13] end of Section 3.2).

Lemma 1. *The coflow polyhedron $Q(G, a, b, c)$ can be written as*

$$Q(G, a, b, c) = \{(y_{n+1}, \ldots, y_{2n}) : y \in \mathbb{R}^{2n}, yA \leq c\}. \tag{1}$$

Proof. (see [13] proof of Theorem 3.3 Claim 2, but our context here is simpler.) Indeed, $(y_1, \ldots, y_n)$ is a potential for the weight function $c_e - \sum\{y_{n+i} :$ vertex i is the tail of edge e$\}$. However, a potential exists if and only if there is no negative cycle, that is, along every negative cycle the sum of the y_{n+i} does not exceed the sum of the c_e on the same cycle. □

Lemma 2 *(Coflow Theorem [9],[11]). Any system of coflow inequalities $\mathcal{Q}(G, a, b, c)$ is TDI.*

Proof. First proof: For every $w : V(G) \to \mathbb{Z}$ the dual problem $D(G, a, b, c, w)$ is a problem of covering vertices by cycles which is a flow problem with w as lower capacities on the vertices. Second proof: The dual solutions are the same as the primal solutions of the LP with TU coefficient matrix in the preceding lemma. □

Surprisingly, these two three-line proofs are sufficient for getting our general graph theory results in a straightforward way.

Lemma 3 *(Baum, Trotter [3]). If A is totally unimodular, $\{y : yA \leq c\}$ has the integer decomposition property.*

Proof. Indeed, suppose $\bar{y}A \leq kc$. We have to show an integer vector y_k, $y_k A \leq c$ for which

$$(\bar{y} - y_k)A \leq (k-1)c.$$

Then the statement follows by induction on k. We have to find a linear solution to

$$\bar{y}A - (k-1)c \leq y_k A \leq c,$$

where $\bar{y}$, A, k, c are fixed and the entries of y_k are the variables. This system of linear inequalities has a solution, since $y_k := (1/k)\bar{y}$ is a solution. Since A is unimodular, then it also has an integer solution, and the claim is proved. □

We mimic now the same proof once more for handling projections:

Lemma 4. *If $Q \subseteq \mathbb{R}^{m'}$, $Q = \{y : yA \leq c\}$ where A is totally unimodular, and $m < m'$, then*

$$P := \{(y_1, \ldots, y_m) : y \in Q\}$$

has the integer decomposition property.

Proof. Let $x \in kP \cap \mathbb{Z}^m$, that is, $x/k \in P$. By definition there exists $(x_{m+1}, \ldots, x_{m'}$, so that $(x/k, x_{m+1}, \ldots, x_{m'}) \in Q$. So So $(x, kx_{m+1}, \ldots kx_{m'}) \in kQ$. It is sufficient to show that $x' := (kx_{m+1}, \ldots kx_{m'})$ can be chosen to be integer, because then by the integer decomposition property of Q:

$$(x, x') = y^1 + \ldots + y^k,\ y^i \in Q, \text{ and } y^i \text{ is an integer vector,}$$

and letting x^i be the vector formed by the first m entries of y^i, we get

$$x = x^1 + \ldots + x^k,\ x^i \in P \cap \mathbb{Z}^m \quad (i = 1, \ldots k).$$

Now x' is a feasible solution of the equation $xB + x'A' \leq c$, where B is the matrix formed by the first n rows of A, and where A' is the rest of A. Since A is totally unimodular, A' is also totally unimodular, so the equation

$$x'A' \leq kc - xB,$$

– where k, c, x, B are fixed, all integer, and x' is variable –, also has an integer solution x' (it does have a feasible solution x', A' is totally unimodular, and the right hand side is integer). □

In [31] flows are applied in each special case separatly: c takes there only two different values in all of these– 0 and $k \in \mathbb{N}$, which makes the proofs and algorithms simpler. *The integer decomposition property of $Q(G, a, b, c)$ is also proved for a special case, and turns out to be crucial for proving Theorem 11 and 12.*

Integer decomposition makes possible the inclusion of *unions of vertices of* $0-1$ *coflow polyhedra, establishing that these also form coflow polyhedra, analogously with a similar matroid property.*

It is unfortunate that these special cases were proved one by one in [31], [13], without knowing about coflows. Several colleagues advised a similar unified treatment – Attila Bernáth made a very concrete suggestion. Then I learned about coflows from Irith Hartman, but the nontrivial graph theoretic proof of the integer decomposition property in particular cases – where the tree is hiding the forest – persisted. I have realized Lemma 4 only recently, and that proving this property for coflows in general provides a much simpler proof of Theorem 15 than the proof we followed in [31]:

Theorem 13 *(Coflow ID). Coflow polyhedra have the integer decomposition property.*

Proof. By Lemma 3, coflow polyhedra are of the form of the condition of Lemma 4, therefore, by this latter Lemma, they have the integer decomposition property. □

3.2 Coherence

Graph theory courses characterize strongly connected graphs with the existence of an "ear decomposition" [33, Theorem 6.9]. Knuth's characterization is then at hand, and provides considerably more information:

Theorem 14 *(Knuth). Let $G = (V, E)$ be a strong digraph. Then for every $v \in V$ there exists an order $v_1, \ldots, v_n$ on V such that $v_1 = v$, and*

(i) Every $e \in E$ is contained in a cycle C with at most one backward arc.
(ii) Every $v \in V$ can be reached from v_1 using only forward arcs.

A *backward arc* (with respect to a given order of the vertices) is an arc $v_iv_j \in E$, $i > j$, the other arcs are *forward arcs.*

Bessy and Thomassé [7] found the relevant part (i) independently, and developed it as a key to their proof of Gallai's conjecture Theorem 4. (A second ingredient was Dilworth's theorem that has been traded for circulations in [31].) Following them we call an order satisfying (i) *coherent*. They proved, equivalently to (i), that every strong digraph has a coherent order. The equivalence of this fact with Knuth's theorem has been realized by Iwata and Matsuda [23].

Four simple proofs of the existence of coherent orders in strongly connected graphs, each providing its own insight, can be found in [7], [23], [24], [31].

Knuth proved this theorem as an application of his "Wheels within Wheels" theorem [24]. Iwata and Matsuda found Knuth's theorem in the archives, and proved it shortly and constructively using the ear decomposition of strongly connected graphs providing a measurable computational progress as well: it takes $O(nm)$ time to construct the order in Theorem 14, whereas a construction in [31] based on different ideas takes $O(n^2m^2)$ time. To prove the statement by induction (ii) is useful.

Fixing an order, the *index* (or winding) $\mathrm{ind}(C)$ of a cycle C is the number of its backward arcs, except for cycles $\{v\}$ $(v \in V)$ for which we define $\mathrm{ind}(\{v\}) = 1$ (like if it had one "backward loop"). If $\mathcal{C}$ is a set of cycles, we denote

$$\mathrm{ind}(\mathcal{C}) := \sum_{C \in \mathcal{C}} \mathrm{ind}(C).$$

For any cycle C in any graph with any order, $\mathrm{ind}(C) \geq 1$, so for any set of cycles $\mathcal{C}$, we have $\mathrm{ind}(\mathcal{C}) \geq |\mathcal{C}|$.

3.3 Topping

At the end of Section 2 we deduced the Greene-Kleitman theorem, and well-known results on cycle-covers, from Theorem 12. It is now the turn of Theorem 12 itself, completed by the following topping:

Given a graph $G = (V, E)$ with an order on the vertex set, let us call a set S satisfying

$$\text{(COMB)} \qquad |S \cap C| \leq \mathrm{ind}(C), (i = 1, \ldots k).$$

a *cyclic stable set*. (This notion is equivalent to a geometric notion of Bessy and Thomassé [7]. The equivalence is proved in [31, (5)].)

The only thing we need here about cyclic stable sets though is that they *are indeed stable sets provided* $G = (V, E)$ *with the given order is coherent*. This is true, because by coherence every arc $e = ab \in E$ $(a, b \in V)$ is contained in a cycle C with $\mathrm{ind}(C) = 1$, so we get for the sets S satisfying (COMB): $|S \cap \{a, b\}| \leq |S \cap C| = 1$. So for every arc $ab \in E$, S can contain at most one of a and b.

Theorem 15 *([31] Theorem 3.1). Let G be a strong digraph given with a coherent order.*

$$\max\{|S_1 \cup \ldots \cup S_k| : \ S_i \ (i{=}1,\ldots k) \ \textit{is a cyclic stable set} \ \} =$$
$$= \min\{|R(\mathcal{C})| + k\,\mathrm{ind}(\mathcal{C}) : \ \mathcal{C} \ \textit{is a set of cycles}\}.$$

Proof. Let $G = (V, E)$ be strong and $k \in \mathbb{Z}$. Apply Theorem 14 to G and fix the coherent order it provides. Let B be the set of backward arcs, and define $c_{B,k}(e) := k$ if $e \in B$, and 0 otherwise $a := 0 \in \mathbb{Z}^n$, $b := w := 1 \in \mathbb{Z}^n$ (constant 0 and constant 1 vectors, that we will simply denote by 0 and 1). Then $\mathcal{Q}(G, 0, 1, c_{B,k})$ is the following system:

(kBT) $\quad x(V_C) \leq k\,\mathrm{ind}(C)$ for every cycle C with vertex-set V_C, $0 \leq x \leq 1$.

Claim 1: The optimum of $P(G, 0, 1, c_{B,k}, 1)$ is $\max|S_1 \cup \ldots \cup S_k|$, $S_i \subseteq V$ $(i = 1, \ldots k)$ satisfies (COMB).

Such a union defines a primal solution, so the optimum is at least this quantity. To prove the equality, we show that the optimum x_{opt} of $P(G, 0, 1, c_{B,k}, 1)$ can be written in this form. Note $Q(G, 0, \infty, c_{B,k}) = kQ(G, 0, \infty, c_{B,1})$, so $P(G, 0, 1, c_{B,k}, 1)$ is the problem

$$\max 1^\top x, \ \text{subject to } x = (x_1, \ldots, x_n) \in kQ(G, 0, \infty, c_{B,1}), \quad x \leq 1.$$

Because of Theorem 13 applied to $Q(G, 0, \infty, c_{B,1})$:

$$x_{\mathrm{opt}} = x^1 + \ldots + x^k, \ x^i \in Q(G, 0, \infty, c_{B,1}) \ \text{ for all } i = 1, \ldots k,$$

and because of $x_{\mathrm{opt}} \in \{0, 1\}^n$ and $x^i \geq 0\,(i = 1, \ldots k)$ *we have* $x^i \in \{0, 1\}^n$, that is, x_i is the incidence vector of a set S_i satisfying (COMB).

Claim 2: The optimum of $D(G, 0, 1, c_{B,k}, 1)$ is

$$\min\{|R(\mathcal{C})| + k\,\mathrm{ind}(\mathcal{C}) : \ \mathcal{C} \text{ is a set of cycles}\}.$$

Indeed, (kBT) is a TDI system (Lemma 2), and therefore the optimum of $D(G, 0, 1, c_{B,k}, 1)$ is a $0 - 1$ vector. Since for a given dual solution $R(\mathcal{C}) := \{v \in V : \text{ the dual variable for } v \text{ is } 1\ \}$, the dual optimum of (kBT) is as claimed.

We have arrived at the end of the proof now: by the duality theorem of linear programming $\mathrm{OPT}(G, 0, 1, c_{B,k}, 1)$ is equal to both the quantities in Claim 1 and Claim 2, and by (COMB) the sets S_i $(i = 1, \ldots, k)$ are all cyclic stable sets. □

Theorem 11 is an immediate corollary since $\mathrm{ind}(\mathcal{C}) \geq |\mathcal{C}|$.

The stable sets S_i $(i = 1, \ldots k)$ and the set of cycles $\mathcal{C}$ provided by the theorem satisfy by complementary slackness (get it directly from the equalities of the theorem or in its proof) : $|S_i \cap C| = \mathrm{ind}(C) \geq 1$ $(i = 1, \ldots k)$, $C \in \mathcal{C}$, so we can delete from each S_i all but one of the elements of $S_i \cap C$, *finishing the proof of Theorem 12 as well.*

The original proof of theorems 15, 11, 12 was quite tedious – the integer decomposition property was proved through a complicated graph theory argument using potentials (arriving at a geometric surplus though). Theorem 13 provides

a shorter way. (Which can also be converted into an algorithm.) Theorem 15 is actually the most general result we can prove for the union of k stable sets. It is similar to sums of matroids.

Let us finally deduce from Theorem 15 the two fundamental results of Bessy and Thomassé [7] originally proved with two entirely different methods. They are both minmax theorems, so "structural versions" follow by complementary slackness.

Corollary 2 *([7] Theorem 1). Let G be a strong digraph given with a coherent order.*

$$\max\{|S| : \ S \text{ is a cyclic stable set } \} = \min\{\operatorname{ind}(\mathcal{C}) : \ \mathcal{C} \text{ covers } V\}.$$

Proof. Apply Theorem 15 to $k = 1$, noting that the one-element subsets of X can be replaced by a cycle of index 1. □

For $k = 1$ the integer decomposition actually becomes simply flow integrality and we get back the simple proof of Theorem 6 in the introduction of [31] (Subsection 0.3).

Corollary 3 *([7], combination of Lemma 3 and Theorem 3). The minimum of k such that G can be colored with k cyclic stable sets is equal to the maximum of $\lceil |C|/\operatorname{ind}(C) \rceil$ over all cycles of C.*

Proof. Apply Theorem 15 to $k := \max\{\lceil |C|/\operatorname{ind}(C) \rceil : \ C \text{ is a cycle of } G \}$. Then $k \operatorname{ind}(C) \geq |C|$ for every cycle, and therefore the right hand side in Theorem 15 is n. Less cyclic stable sets are not enough, since $k-1$ cyclic stable sets meet each cycle C in at most $(k-1)\operatorname{ind}(C)$ elements, which is less than $|C|$ for the cycles for which the above maximum is reached. □

These two corollaries showed the way: they are the two ice-cream balls, the theorem is the topping.

4 Path Partitions

We prove Berge's conjecture in the following cases:

4.1 Long Paths

Theorem 16. *Let $G = (V, E)$ be a digraph, $k, m \in \mathbb{N}$ and $\mathcal{P} = \{P_1, \ldots, P_m\}$ a path partition. Then there exists*

(i) either k disjoint stable sets orthogonal to $\mathcal{P}$,
(ii) or a subpartition $\mathcal{Q} = \{Q_1, \ldots, Q_{m-1}\}$ of paths s.t. $\operatorname{ini}(\mathcal{Q}) \subseteq \operatorname{ini}(\mathcal{P})$, $\operatorname{ter}(\mathcal{Q}) \subseteq \operatorname{ter}(\mathcal{P})$, *and*

$$|R(\mathcal{Q})| \leq k - 1.$$

Proof. We prove the statement by induction on $n := |V|$. Suppose it holds for all $n' < n$ with all values of m and k, and prove it for G.

We can suppose that $|P| \geq k$ for all $P \in \mathcal{P}$, because if say $|P_m| < k$, define $Q_i := P_i$ for all $i = 1, \ldots, m-1$. We see that (ii) holds: $|V \setminus (Q_1 \cup \ldots \cup Q_{m-1})| = |P_m| \leq k-1$.

Let $a_i := \mathrm{ini}(P_i)$, and let a_i' be the second vertex of P_i, and $P_i' := P_i \setminus \{a_i\}$, that is, $\mathrm{ini}(P_i') = a_i'$ $(i = 1, \ldots, m)$.

We distinguish now two cases:

Case 1: $\mathrm{ini}(\mathcal{P})$ is not a stable set, that is, say $a_1a_2 \in E$.

If $|P_1| = k$, we can replace $\mathcal{P}$ by $\mathcal{P} \setminus \{P_1\}$, and add a_1 to P_2 as first vertex: we see that (ii) holds then.

So suppose $|P_1| \geq k+1$, and apply the induction hypothesis to $G - a_1$ and the same path partition restricted to $V \setminus \{a_1\}$, that is, with the only change of replacing P_1 by $P_1 \setminus \{a_1\}$. If now (i) holds, then, using also that $|P_1 \setminus \{a_1\}| \geq k$, (i) also holds for G. So suppose (ii) holds for $G - a_1$, and let $\mathcal{Q}' = \{Q_1', \ldots, Q_{m-1}'\}$ be the path partition satisfying (ii). Since $\mathrm{ini}(\mathcal{Q}')$ is an $m-1$ element subset of $\{a_1', a_2, \ldots, a_m\}$, it contains a path Q_1' with $\mathrm{ini}(Q_1') = a_1'$ or $\mathrm{ini}(Q_1') = a_2$. Adding a_1 as a first vertex to Q_1' we get a path partition of G that satisfies (ii).

Case 2: $\mathrm{ini}(\mathcal{P})$ is a stable set.

Apply the statement to $G' := G - \mathrm{ini}(\mathcal{P})$, $k' := k-1$.

If then (i) holds, then adding the stable set $\mathrm{ini}(\mathcal{Q})$ to the provided $k-1$ stable sets, we get that (i) holds for G and $\mathcal{P}$ with parameter k.

Otherwise alternative (ii) holds for $G' = (V', E')$ with $\mathcal{P}'$ and k', that is, we have a subpartition of paths $\mathcal{Q}' = \{Q_1', \ldots, Q_{m-1}'\}$, in G' such that $\mathrm{ini}(\mathcal{Q}') \subseteq \mathrm{ini}(\mathcal{P}')$ (and $\mathrm{ter}(\mathcal{Q}') \subseteq \mathrm{ter}(\mathcal{P}') = \mathrm{ter}(\mathcal{P})$), that is, with an appropriate choice of the notation $\mathrm{ini}(\mathcal{Q}') = \{a_1', \ldots, a_{m-1}'\}$, furthermore $\mathrm{ini}(Q_i') = \{a_i'\}$ for all $i = 1, \ldots, m-1$, and

$$|R(\mathcal{Q}')| = |V' \setminus (Q_1' \cup \ldots \cup Q_{m-1}')| \leq k-2.$$

Define now $Q_i := Q_i' \cup a_i$ $(i = 1, \ldots, m-1)$. Clearly, $\mathcal{Q} := \{Q_1, \ldots, Q_{m-1}\}$ is a subpartition of paths in G, and

$$|R(\mathcal{Q})| = |V \setminus (Q_1 \cup \ldots \cup Q_{m-1})| = |V' \setminus (Q_1' \cup \ldots \cup Q_{m-1}')| + 1 \leq k-1,$$

since $V \setminus (Q_1 \cup \ldots \cup Q_{m-1}) = (V' \setminus (Q_1' \cup \ldots \cup Q_{m-1}')) \cup \{a_m\}$, proving that alternative (ii) holds for G with $\mathcal{P}$ and k. □

This implies Berge's conjecture when there exists an optimal path partition with only paths of length at least k, which turns out to be equivalent to a result of Aharoni, Hartman and Hoffman [2]. Their proof is based on improving paths, and probably implies all the claims of the Theorem, in a more involved way though.

Corollary 4. *[2] If G is a digraph and $\mathcal{P}$ is a path partition where $k|\mathcal{P}| = \min\{|X| + k\pi(V \setminus X) : X \subset V\}$, then there exist k disjoint stable sets orthogonal to $\mathcal{P}$.*

4.2 Acyclic Digraphs

For acyclic digraphs Berge's and Linial's conjectures are consequences of Theorem 13 on the lines of, and more simply than the proof of Theorem 15, but I do not see how to deduce them directly from the statement of Theorem 15. The results have been proved in [1], [2], [10], [14], [26], [30]. Let us show how Theorem 13 replaces all the difficulties:

Let G be acyclic, and $1, \ldots, n$ an order of the vertices with only forward arcs. Add all backward arcs, that is $\hat{G} := G \cup B$, $B := \{ij, i > j\}$. Note that this order is coherent for $\hat{G}$, and a cycle with β backward arcs is the disjoint union of vertex-sets of β cycles each having 1 backward arc.

Consider now the polyhedron $Q(\hat{G}, -\infty, 1, c_{B,k}) \subseteq kQ(\hat{G}, -\infty, \infty, c_{B,1})$. It can have negative vertices! (We cannot avoid this, since since we want equality constraints for the dual problem.) By Theorem 13 $Q(\hat{G}, -\infty, \infty, c_{B,1})$ – which is now simply $\{x \in \mathbb{R}^n : x(P) \leq 1$ for every path $P\}$ – has the integer decomposition property again, and by Theorem **??** $\mathcal{Q}(G, -\infty, 1, c_{B,k})$ is TDI. Now we can finish using Theorem 13 exactly like in the proof of Theorem 15. (Negative variables do not disturb, since by complementary slackness a primal optimal solution $x \in Q(\hat{G}, -\infty, 1, c_{B,k})$ satisfies $x(P) = k$ for all paths P of an optimal path partition; because of $x \leq 1$, x has at least k different 1 entries; because of Theorem 13 we have $x = x^1 + \ldots + x^k$, $x^i \in Q(\hat{G}, -\infty, \infty, c_{B,1})$, and then the positive coordinates of the x_i meet every path, and in different vertices for $i \neq j = 1, \ldots, n$.)

4.3 Corollaries for Path Partitions

Berger and Hartman studied the two next-to-extreme cases of Berge's conjecture [5], [6]:"$k = 2$" and "$k = \lambda - 1$" – the $k = 1$ and $k = \lambda$ cases being completely settled, see the subsections 2.2, 2.3, 2.4, 2.5. It is somewhat discouraging for the continuation that the path partition and cycle cover versions are so far completely unrelated even in the Gallai-Milgram case $k = 1$.

However, the following theorems show some connections at the other extreme, and for strongly connected graphs a larger interval can be allowed for k. These are the starting steps of a research with Irith Hartman in the frame of the French-Israeli collaboration project, intending to prove Berge's conjecture.

The following result is a direct corollary of Theorem 15, and it provides a common statement and proof of the Gallai-Roy theorem, and a theorem of Berger and Hartman [6] according to which Berge's strongest conjecture (and therefore Linial's conjecture as well) is true if $k = \lambda - 1$. Their original proof is quite involved. Note that we prove only the version Conjecture 3 ignoring short ($< k$) paths that are not singletons, still implying Linial's conjecture.

Theorem 17. *Let $G = (V, E)$ be a digraph, and $k \in \mathbb{N}$, $k \geq \lambda - 1$. For any subpartition $\mathcal{P}$ of paths minimizing $|R(\mathcal{P})| + k|\mathcal{P}|$, there exists k disjoint stable sets orthogonal to $\mathcal{P}$ whose union contains $R(\mathcal{P})$.*

Proof. Let $\mathcal{P}$ satisfy the condition. The number $|R(\mathcal{P})| + k|\mathcal{P}|$ is called the *k-norm* of $\mathcal{P}$ [22].

Add a new vertex v_0 to the graph, a cycle C_0 through v_0 with $k+1$ new vertices besides v_0, and add all the edges v_0v, vv_0 ($v \in V$). Denote $\hat{V} := V \cup V(C_0)$. Order V starting with v_0, continuing on C_0 until the vertex before v_0, then continuing with the other members of $\mathcal{P}$ in some order, from the sources to the sinks, and finally adding the vertices of $R(\mathcal{P})$ in arbitrary order. Let $\hat{G} = (\hat{V}, \hat{E})$ be the constructed graph.

Note: C_0 serves the goal of covering v_0 in a predictable way, and the presence of C_0 will also have the useful consequence that we will never color v_0. Adding v_0 without adding C_0 may slightly change the problem and cause technical difficulties. Let $\mathcal{C} := \{C_0\} \cup \{v_0 \cup P : P \in \mathcal{P}\}$.

The defined order is coherent, since for all $uv \in E : v_0, u, v$ is a cycle with one backward arc, and C_0 is also a cycle that has one backward arc.

All cycles in $\mathcal{C}$ have one backward arc, so $\operatorname{ind}(\mathcal{C}) = |\mathcal{C}| = |\mathcal{P}|+1$. We show *that $\mathcal{C}$ minimizes the right hand side of Theorem 15* (see after Claim 2), and then this theorem will provide the statement. Let $\mathcal{Q} = \{Q_0, \dots, Q_m\}$ ($Q_0, \dots, Q_m$ are cycles) minimize the right hand side of this theorem, and among the possible choices $R(\mathcal{Q})$ be maximum. In fact we will show

$$|R(\mathcal{C})| + k \operatorname{ind}(\mathcal{C}) \le |R(\mathcal{Q})| + k \operatorname{ind}(\mathcal{Q}), \tag{1}$$

by showing through claims 1, 2 a path partition $\mathcal{P}'$ in G, $R(\mathcal{P}') = R(\mathcal{Q})$, $|\mathcal{P}'| = |\mathcal{Q}| - 1$, and then by the minimality of the k-norm of $\mathcal{P}$ we have

$$|R(\mathcal{P})| + k|\mathcal{P}| \le |R(\mathcal{P}')| + k|\mathcal{P}'|, \tag{2}$$

implying (1): indeed, the left hand side of (1) is k plus the left hand side of (2), and according to the following, the right hand side of (1) is *at least k plus the right hand side of (2)*:

$$|R(\mathcal{P}')| + k|\mathcal{P}'| + k = |R(\mathcal{Q})| + k|\mathcal{Q}| \le |R(\mathcal{Q})| + k|\operatorname{ind}(\mathcal{Q})|.$$

Claim 1: There exists $i \in \{1, \dots, m\}$ so that $Q_i = C_0$, and therefore $Q_0 = C_0$ can be supposed.

Indeed, if C_0 does not occur, then the vertices of C_0 different of v_0 cannot occur in $\mathcal{Q}$ at all. Since their number is $k+1$, by adding C_0 to $\mathcal{Q}$ we decrease $|R(\mathcal{Q})|$ by $k+1$ and $k|\mathcal{Q}|$ increases only by k ($|\mathcal{Q}|$ increases by 1), contradicting the optimal choice of $\mathcal{Q}$.

Claim 2: $P_i' := Q_i \setminus v_0$ ($i = 0, 1, \dots, m$) are pairwise disjoint.

Indeed, $Q_i \setminus v_0 \le \lambda \le k+1$, so if it meets $Q_j \setminus v_0$ ($j \ne i$), then $\mathcal{Q} \setminus \{Q_i\}$ contradicts the choice of $\mathcal{Q}$, because $|R(\mathcal{Q} \setminus \{Q_i\})| \le |R(\mathcal{Q})| + k$: $v_0 \notin R(\mathcal{Q})$ by Claim 1, and the possible other common point is not in $R(\mathcal{Q})$ either. On the other hand $k \operatorname{ind}(\mathcal{Q} \setminus \{Q_i\}) = k \operatorname{ind}(\mathcal{Q}) - k \operatorname{ind}(Q_i) \le k \operatorname{ind}(\mathcal{Q}) - k$, so the right hand side of the formula of Theorem 15 does not increase, $R(\mathcal{Q})$ increases, again contradicting the choice of $\mathcal{Q}$.

Now by Claim 2, $\mathcal{P}' := \{P'_1, \ldots, P'_m\}$ is a set of disjoint paths satisfying the promised relations $R(\mathcal{P}') = R(\mathcal{Q})$, $|\mathcal{P}'| = |\mathcal{Q}| - 1$, so (1) is satisfied and $\mathcal{C}$ is an optimal set of cycles in Theorem 15.

Theorem 15 provides now exactly what we want, unless v_0 is a colored vertex. However, one can suppose that v_0 is contained in at least 2 members of $\mathcal{C}$, since in case of $\mathcal{C} = \{C_0\}$ the set $\mathcal{C}' = \{C_0, \{v_0\} \cup P\}$, where P is a longest path is a set of cycles which also minimizes the right hand side. Then by complementary slackness in Theorem 15, v_0 is not contained in any of the sets S_i provided by the theorem. □

For $k = \lambda - 1$ the theorem can be restated as follows:

Corollary 5. *Let $G = (V, E)$ be a directed graph. If $\mathcal{P}$ is a maximum number of disjoint maximum paths of G, there exists a set $U \subseteq V$ consisting of exactly one vertex of each $P \in \mathcal{P}$, and a (complete) coloring of $G - U$ where each color class is orthogonal to $\mathcal{P}$.*

Indeed, $|R(\mathcal{P})| + (\lambda - 1)|\mathcal{P}|$ is minimum provided $\mathcal{P}$ is a maximum set of disjoint maximum paths.

Applying the theorem to $k = \lambda$ we can reformulate it into the following very similar form where U can in addition be chosen to be a cyclic stable set (it is one of the colors) the paths are not necessarily part of a maximum packing, however all of the colors may have to meet $R(\mathcal{P})$ (while U did not). Both corollaries extend the Gallai-Roy theorem.

Corollary 6. *Let $G = (V, E)$ be a directed graph. If $\mathcal{P}$ is any number of disjoint maximum paths of G, there exists a coloring of G where the color classes are orthogonal to $\mathcal{P}$.*

The following result exploits some simple properties of paths, but the application of these prevents to use the gadget reductions of the previous proof and we cannot avoid assuming strong connectivity.

Theorem 18. *Conjecture 3 is true provided G is strongly connected and $k \geq \lambda - \sqrt{\lambda}$.*

Proof. Let G be strongly connected, $k \geq \lambda - \sqrt{\lambda}$, and choose a coherent order. Apply Theorem 15 and let $S_1, \ldots, S_k$ the stable sets in the maximum, X and $\mathcal{C}$ the set and cycle cover in the minimum, moreover, suppose that among the possible choices, $|X|$ is biggest possible. As in the previous proof, $k \geq \lambda/2$ easily implies that $\mathrm{ind}(C) = 1$ for all $C \in \mathcal{C}$. The problem is that the cycles in $\mathcal{C}$ are not necessarily disjoint.

If a cycle has at most $\lambda - \lfloor\sqrt{\lambda}\rfloor$ vertices not covered by any other cycle, then delete it from $\mathcal{C}$ and add to X the vertices that get now uncovered, contradicting the choice of X. So the difference of any two cycles has size larger than $\lambda - \lfloor\sqrt{\lambda}\rfloor$. *If for two intersecting cycles C_1, C_2 we have $|C_1 \setminus C_2|, |C_2 \setminus C_1| > \lambda - \lfloor\sqrt{\lambda}\rfloor$, then their union contains a path with more than λ vertices.* (This bound is essentially tight.)

Indeed, we have then $|C_1 \cap C_2| \leq \lfloor\sqrt{\lambda}\rfloor$. But then $C_1 \cap C_2$ divides both cycles into paths, and one of these paths has at least $\lfloor\sqrt{\lambda}\rfloor$ vertices outside C_1, say. (If all these subpaths of C_2 have at most $\sqrt{\lambda} - 1$ vertices outside C_1, then $|C_2 \setminus C_1| \leq \sqrt{\lambda}(\sqrt{\lambda} - 1) = \lambda - \sqrt{\lambda}$.) Take such a path P for instance in C_2. Then $|C_1 \cup P| > \lambda - \lfloor\sqrt{\lambda}\rfloor + \lfloor\sqrt{\lambda}\rfloor = \lambda$. It is easy to see that $|C_1 \cup P|$ contains a Hamiltonian path, that is, a path of length larger than λ, contradicting the definition of λ.

So the cycles in $\mathcal{C}$ are pairwise disjoint and then we are done again by complementary slackness. □

Acknowledgment. Many thanks are due to Irith Hartman and an anonymous referee for a very thorough reading of the originally submitted manuscript and a lot of helpful corrections.

References

1. Aharoni, R., Ben-Arroyo Hartman, I., Hoffman, A.: Path Partitions and Packs of Acyclic digraphs. Pacific Journal of Mathematics 118(2), 249–259 (1985)
2. Aharoni, R., Ben-Arroyo Hartman, I.: On Greene-Kleitman's theorem for general digraphs. Discrete Mathematics 120, 13–24 (1993)
3. Baum, S., Trotter, L.: Integer Rounding and Polyhedral decomposition of totally unimodular systems. In: Henn, Korte, Oettli (eds.) Proc. Bonn 1977, Optimizaton and Operations Research. Lecture Notes in Economics and Math Systems, vol. 157, pp. 15–23. Springer, Berlin (1977)
4. Berge, C.: k-optimal partitions of a directded graph. European J. of Combinatorics 3, 97–101 (1982)
5. Berger, E., Ben-Arroyo Hartman, I.: Proof of Berge's strong path partition conjecture for k=2. European Journal of Combinatorics 29(1), 179–192 (2008)
6. Berger, E., Ben-Arroyo Hartman, I.: Proving Berge's Path Partition Conjecture for k = $\lambda-$ 1 (manuscript)
7. Bessy, S., Thomassé, S.: Spanning a strong digraph by α circuits: a proof of Gallai's conjecture. Combinatorica 27, 659–667 (2007)
8. Bondy, A.: Diconnected orientations and a conjecture of Lasvergnas. J. London Math. Soc. 14(2) (1976)
9. Cameron, K.: Polyhedral and algorithmic ramifications of antichains, Ph.D. Thesis, University of Waterloo, Waterloo, Canada (1982)
10. Cameron, K.: On k-optimum dipath partitions and partial k-colorings of acyclic digraphs. Europ. J. Combinatorics 7, 115–118
11. Cameron, K., Edmonds, J.: Coflow Polyhedra. Discrete Mathematics 101, 1–21 (1992)
12. Camion, P.: Chemins et circuits Hamiltoniens des graphes complets. C. R. Acad. Sci., Paris (1959)
13. Charbit, P., Sebő, A.: Cyclic Orders: Equivalence, and Duality. Combinatorica 28(2), 131–143 (2008)
14. Felsner, S.: Orthogonal structures in directed graphs. J. Comb. Theory, Ser. B 57, 309–321 (1993)
15. Frank, A.: On chain and antichain families of a partially ordered set. J. of Comb. Theory, Series B 29, 251–261 (1980)

16. Gallai, T.: Problem 15. In: Fiedler, M. (ed.) Theory of Graphs and its Applications, p. 161. Czech Acad. Sci., Prague (1964)
17. Gallai, T., Milgram, A.N.: Verallgemeinerung eines graphentheoretischen Satzes von Rédei. Acta Sc. Math. 21, 181–186 (1960)
18. Gallai, T.: On directed paths and circuits. In: Erdős, P., Katona, G. (eds.) Theory of Graphs, pp. 115–118. Academic Press, New York (1968)
19. Golumbic, M.C., Rotem, D., Urrutia, J.: Comparability graphs and intersection graphs. Discrete Mathematics 43(1) (1983)
20. Golumbic, M.C., Lipshteyn, M., Stern, M.: The k-edge intersection graphs of paths in a tree. Discrete Applied Mathematics 156(4), 451–461 (2008)
21. Greene, C., Kleitman, D.J.: The structure of Sperner k-families. Journal of Combinatorial Theory, Series A 20, 41–68 (1976)
22. Hartman, I.B.-A.: Berge's Conjecture on Directed Path Partitions - A Survey, volume in honor of Claude Berge. Discrete Mathematics 306, 2582–2592 (2006)
23. Iwata, S., Matsuda, T.: Finding coherent cyclic orders in strong digraphs. Combinatorica 28(1), 83–88 (2008)
24. Knuth, D.E.: Wheels within Wheels. Journal of Combinatorial Theory/B 16, 42–46 (1974)
25. Laborde, J.M., Payan, C., Xuong, N.H.: Independent sets and longest directed paths in digraphs. In: Graphs and other combinatorial topics (Prague 1982), Teubner, Leipzig, pp. 173–177 (1983)
26. Linial, N.: Extending the Greene-Kleitman theorem to directed graphs. J. of Combinatorial Theory, Ser. A 30, 331–334 (1981)
27. Monma, C., Wei, V.K.: Intersection graphs of paths in a tree. Journal of Combinatorial Theory B 41, 141–181 (1986)
28. Rédei, L.: Ein Kombinatorischer Satz. Acta Litt. Sci. Szeged 7, 39–43 (1934)
29. Roy, B.: Nombre chromatique et plus longs chemins. Rev. F1, Automat. Informat. 1, 127–132 (1976)
30. Saks, M.: A short proof of the k-saturated partitions. Adv. In Math. 33, 207–211 (1979)
31. Sebő, A.: Minmax Theorems in Cyclically Ordered graphs. Journal of Combinatorial Theory /B 97(4), 518–552 (2007)
32. Schrijver, A.: Theory of Linear and Integer Programming. Wiley, Chichester (1986)
33. Schrijver, A.: Combinatorial Optimization. Springer, Heidelberg (2003)

Properly Coloured Cycles and Paths: Results and Open Problems

Gregory Gutin[1] and Eun Jung Kim[2]

[1] Department of Computer Science, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK
gutin@cs.rhul.ac.uk
[2] Department of Computer Science, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK
eunjung@cs.rhul.ac.uk

Abstract. In this paper, we consider a number of results and six conjectures on properly coloured (PC) paths and cycles in edge-coloured multigraphs. We overview some known results and prove new ones. In particular, we consider a family of transformations of an edge-coloured multigraph G into an ordinary graph that allow us to check the existence of PC cycles and PC (s,t)-paths in G and, if they exist, to find shortest ones among them. We raise a problem of finding the optimal transformation and consider a possible solution to the problem.

1 Introduction

The class of edge-coloured multigraphs generalize directed graphs. There are several other generalizations of directed graphs such as arc-coloured digraphs, hypertournaments and star hypergraphs, but the class of edge-coloured multigraphs has been given the main attention in graph theory literature because many concepts and results on directed graphs can be extended to edge-coloured multigraphs and there are several important applications of edge-coloured multigraphs. For instance, in [10, 11] Dorninger considers chromosome arrangement in a cell of an eukaryotic organism by using the 2-edge-coloured multigraphs. For a more extensive treatment of this topic, see [6, 7].

In this paper we overview some known results on properly coloured (PC) cycles and paths in edge-coloured multigraphs, prove new ones and consider several open problems on the topic. In Section 2 we briefly consider a problem of whether an edge-coloured graph has a PC cycle. In Sections 3 and 4, we offer a useful tool to study edge-coloured multigraphs. In investigating problems on PC subgraphs of edge-coloured multigraphs, it is convenient to transform an edge-coloured graph into an ordinary graph. We suggest a new technique that somewhat automates this transformation. Moreover, by proving some new results, we illustrate how the proposed technique allows us to obtain more efficient algorithms for PC cycle and PC (s,t)-path problems by reducing the order and

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 200–208, 2009.

size of the transformed graph. We raise a problem of determining the minimum order and size of the transformed graph, and describe the family of graphs that may be the solution to the problem.

In Section 5 we study long PC cycles and paths in arbitrary edge-coloured multigraphs and Section 6 is devoted to longest (mostly Hamilton) PC cycles in edge-coloured complete graphs.

An m**-path-cycle subgraph** F of a multigraph G is a vertex-disjoint union of m paths and a number of cycles in G (some cycles can be of length 2). If $m = 0$, we call F a **cycle subgraph** of G. For a vertex set X of a multigraph G, $G\langle X\rangle$ denotes the subgraph of G induced by X. For a pair s, t of distinct vertices of G, a path between s and t is called an (s,t)-**path**.

We consider **edge-coloured multigraphs**, i.e., undirected multigraphs in which each edge has a colour, but no parallel edges have the same colour. If an edge-coloured multigraph G has c colours, we assume that the colours are $1, 2, \ldots, c$ and we call G a c-**edge-coloured** multigraph. We denote the colour of an edge e of an edge-coloured multigraph G by $\chi(e)$. When G has no parallel edges, we call G an **edge-coloured graph**.

Let G be a c-edge-coloured multigraph and let $v \in V(G)$. By $N_i(v)$ we denote the set of neighbours of v adjacent to v by an edge of colour i; let $d_i(x) = |N_i(x)|$. The **maximum** (**minimum**) **monochromatic degree** of $G = (V, E)$ is defined by

$$\Delta_{mon}(G) = \max\{d_j(v) : \ v \in V, \ 1 \leq j \leq c\}$$
$$(\delta_{mon}(G) = \min\{d_j(v) : \ v \in V, \ 1 \leq j \leq c\}).$$

Let $\chi(v) = \{i : \ 1 \leq i \leq c, N_i(v) \neq \emptyset\}$. A path or cycle Q of G is **properly coloured (PC)** if every two adjacent edges of Q are of different colours.

2 Existence of PC Cycles

Since a pair of parallel edges in a c-edge-coloured multigraph ($c \geq 2$) forms a PC cycle, in this section, we consider only c-edge-coloured graphs.

It is easy to see that the problem of checking whether a c-edge-coloured graph has a PC cycle is more general (even for $c = 2$) than the simple problem of verifying whether a digraph contains a directed cycle. Indeed, consider a digraph D and, to obtain a 2-edge-coloured graph G from D, replace each arc xy of D with edges xz_{xy} and $z_{xy}y$ of colours 1 and 2, where z_{xy} is a new vertex ($z_{xy} \neq z_{x'y'}$ provided $xy \neq x'y'$). Observe that G has a PC cycle if and only if D has a directed cycle.

The following theorem by Yeo [21] provides a simple recursive way of checking whether a c-edge-coloured graph has a PC cycle. (For $c = 2$, Theorem 2.1 was first proved by Grossman and Häggkvist [14].)

Theorem 2.1. *Let G be a c-edge-coloured graph, $c \geq 2$, with no PC cycle. Then, G has a vertex $z \in V(G)$ such that no connected component of $G - z$ is joined to z with edges of more than one colour.*

Let us consider the following function introduced by Gutin [15]: $d(n,c)$, the minimum number k such that every c-edge-coloured graph of order n and minimum monochromatic degree at least k has a PC cycle. It was proved in [15] that $d(n,c)$ exists and that

$$d(n,c) \leq \frac{1}{\lfloor c/2 \rfloor}(\log_2 n - \frac{1}{3}\log_2 \log_2 n + \Theta(1)). \quad (1)$$

Abouelaoualim et al. [1] stated a conjecture which implies that $d(n,c) = 1$ for each $c \geq 2$. Using a recursive construction inspired by Theorem 2.1 of c-edge-coloured graphs with minimum monochromatic degree p and without PC cycles, Gutin [15] showed that

$$d(n,c) \geq \frac{1}{c}(\log_c n - \log_c \log_c n) \quad (2)$$

and, thus, the conjecture does not hold. The bounds (1) and (2) imply that $d(n,c) = \Theta(\log_2 n)$ for every fixed $c \geq 2$.

Conjecture 2.2. *[15] There is a function $s(c)$ dependent only on c such that $d(n,c) = s(c)\log_2 n(1+o(1))$.*

In particular, it would be interesting to determine $s(2)$.

3 P-Gadgets

We consider gadget constructions which generalize some known constructions mentioned below. The P-gadget graphs G^* and G^{**} of an edge-coloured multigraph G described in the next section allow one to transform several problems on properly coloured subgraphs of G into perfect matching problems in G^* or G^{**}.

Let G be an edge-coloured multigraph and let $G' = G - \{x \in V(G) : |\chi(x)| = 1\}$. For each $x \in V(G')$ let G_x be an arbitrary (non-edge-coloured) graph with the following four properties:

P1 $\{x_q : q \in \chi(x)\} \subseteq V(G_x)$;
P2 G_x has a perfect matching;
P3 For each $p \neq q \in \chi(x)$, if the graph $G_x - \{x_p, x_q\}$ is not empty, it has a perfect matching;
P4 For each set $L \subseteq \chi(x)$ with at least 3 elements; if the graph $G_x - \{x_l : l \in L\}$ is not empty, it has no perfect matching.

Each G_x with the properties P1-P4 is called a **P-gadget**. Let us consider the following three P-gadgets; the first two are known in the literature and the third one is new.

1. One P-gadget is due to Szeider [19]:

$$V(G_x) = \{x_i, x_i' : \ i \in \chi(x)\} \cup \{x_a'', x_b''\} \text{ and}$$

$$E(G_x) = \{x_i'x_a'', x_i'x_b'', x_ix_i' : \ i \in \chi(x)\} \cup \{x_a''x_b''\}.$$

   We will call this the **SP-gadget**.
2. Another gadget is due to Bang-Jensen and Gutin [4]:

$$V(G_x) = \{x_j : \ j \in \chi(x)\} \cup \{y_j : \ j \in \chi(x) \setminus \{m, M\}\},$$

   where $m = \min \chi(x)$, $M = \max \chi(x)$, and

$$E(G_x) = \{x_jy_k : \ j \in \chi(x), \ k \in \chi(x) \setminus \{m, M\}\} \cup \{x_jx_k : \ j \neq k \in \chi(x)\}.$$

   We will call this the **BJGP-gadget**.
3. The following new gadget is a sort of crossover of the above two and is called the **XP-gadget**:

$$V(G_x) = \{x_j : \ j \in \chi(x)\} \cup \{y_j : \ j \in \chi(x) \setminus \{m, M\}\},$$

   where m and M are defined above, and

$$E(G_x) = \{x_mx_M\} \cup \{x_jy_j, \ x_my_j, \ x_My_j : \ j \in \chi(x) \setminus \{m, M\}\}.$$

It is not difficult to verify that the three P-gadgets indeed satisfy P1-P4. Let $z = \chi(x)$. Observe that the SP-gadget has $2z + 2$ vertices and $3z + 1$ edges, the BJGP-gadget $2z-2$ vertices and $z(3z-5)/2$ edges, the XP-gadget $2z-2$ vertices and $3z-5$ edges. Thus, the XP-gadget has the minimum number of vertices and edges among the three P-gadgets. It is not difficult to verify that the XP-gadget has the minimum number of vertices and edges among all possible P-gadgets for $z = 2, 3, 4$. Perhaps, this is true for any z.

Conjecture 3.1. *The XP-gadget has the minimum number of vertices and edges among all possible P-gadgets for every $z \geq 2$.*

We will see in the next section why minimizing the numbers of vertices and edges in P-gadgets is important for speeding up some algorithms on edge-coloured multigraphs.

4 P-Gadget Graphs

Let G be a c-edge-coloured multigraph and let G_x be a P-gadget for $x \in V(G')$. The graph G^* is defined as follows: $V(G^*) = \cup_{x \in V(G')} V(G_x)$ and $E(G^*) = E_1 \cup E_2$, where $E_1 = \cup_{x \in V(G')} E(G_x)$ and $E_2 = \{y_qz_q : \ y, z \in V(G'), \ yz \in E(G), \ \chi(yz) = q, \ 1 \leq q \leq c\}$.

Let s, t be a pair of distinct vertices of G and let $H = G - \{s, t\}$. Let G^{**} be constructed from H^* by adding s and t and edges $E_3 = \{sx_i : \ sx \in E(G), \chi(sx) = i\} \cup \{tx_i : \ tx \in E(G), \chi(tx) = i\}$.

We will denote the number of vertices and edges in multigraphs G, G^* and G^{**} by n, m, n^*, m^*, n^{**} and m^{**}, respectively.

The following result relates perfect matchings of G^* with PC cycle subgraphs of G. PC cycle subgraphs are important in several problems on edge-coloured multigraphs (for example, for the PC Hamilton cycle problem described in Section 6), see [6]. Recall that $G' = G - \{x \in V(G) : \ |\chi(x)| = 1\}$.

Theorem 4.1. *Let G be a connected edge-coloured multigraph such that G' is non-empty. Then G has a PC cycle subgraph with r edges if and only if G^* has a perfect matching with exactly r edges in E_2.*

Proof: Let M be a perfect matching of G^* with exactly edges

$$x^1_{p_1} y^1_{q_1}, \ldots, x^r_{p_r} y^r_{q_r}$$

in E_2. For a vertex x of G', let Q_x be the set of edges in E_2 adjacent to G_x. By P2, each G_x has even number of vertices ($x \in V(G')$) and since M is a perfect matching in G^*, there is even number of edges in Q_x. By P4, Q_x has either no edges or two edges for each $x \in V(G')$. Let X be the set of all vertices $x \in V(G')$ such that $|Q_x| = 2$. Then, by the definition of G^*, $G\langle X\rangle$ contains a PC cycle factor. It remains to observe that $|X| = r$.

Now let F be a PC cycle subgraph of G with r edges. Observe that the edges of F correspond to a set Q of r independent edges of G^* and that either no edges or two edges of Q are adjacent to G_x for each $x \in V(G')$. Now delete the vertices adjacent with Q from each G_x and observe that each remaining non-empty gadget has a perfect matching by P2 and P3. Combining the perfect matchings of the non-empty gadgets with Q, we get a perfect matching of G^* with exactly r edges from E_2. □

The first part of the next assertion generalizes a result from [4]. The second part is based on an approach which leads to a more efficient algorithm than in [2].

Corollary 4.2. *One can check whether an edge-coloured multigraph G has a PC cycle and, if it does, find a maximum PC cycle subgraph of G in time $O(n^* \cdot (m^* + n^* \log n^*))$. Moreover one can find a shortest PC cycle in G in time $O(n \cdot n^* \cdot (m^* + n^* \log n^*))$.*

Proof: We may assume that G is connected and that G' is not empty. By Theorem 4.1, it is enough to find a perfect matching of G^* containing the maximum number of edges from E_2. Assign weight 0 (1, respectively) to edges of G^* in E_1 (E_2, respectively). Now we need to find a maximum weight perfect matching of G^* which can be done in time $O(n^* \cdot (m^* + n^* \log n^*))$ by a matching algorithm in [13].

To find a shortest PC cycle in G, choose a vertex $x \in V(G')$. We will find a shortest PC cycle in G traversing x. By Theorem 4.1, it is enough to find a perfect matching of G^* containing the minimum number of edges from E_2 while containing at least one edge from E_2 so that the corresponding PC cycle in G

should be non-trivial. We define the weights on edges of G^* as follows. Assign M, where M is a sufficiently large number, to each edge in E_2 incident with G_x. For all other edges, assign weight 1 (0, respectively) to edges of G^* in E_1 (E_2, respectively). A maximum weight perfect matching of G^* contains exactly two edges of weight M by P4, and contains the minimum number of edges in E_2. Finding a maximum weight perfect matching of G^* can be done in time $O(n^* \cdot (m^* + n^* \log n^*))$ and we iterate the process for each $x \in V(G')$. □

The proof of the following result is analogous to the proof of Theorem 4.1.

Theorem 4.3. *Let G be an edge-coloured multigraph and let s, t be a pair of distinct vertices of G. If G^{**} is non-empty, then G has a PC 1-path-cycle subgraph with r edges in which the path is between s and t if and only if G^{**} has a perfect matching with exactly r edges not in E_1.*

The next assertion generalizes a result from [2].

Corollary 4.4. *Let G be an edge-coloured multigraph. One can check whether there is a PC (s,t)-path in G in time $O(m^{**})$ and if G has one, a shortest PC (s,t)-path can be found in time $O(n^{**} \cdot (m^{**} + n^{**} \log n^{**}))$.*

Proof: Let L be a graph. Given a matching M in L, a path P in L is $M-$**augmenting** if, for any pair of adjacent edges in P, exactly one of them belongs to M and the first and last edges of P do not belong to M. Consider a perfect matching M of H^*, where $H = G - \{s,t\}$, which is a collection of perfect matchings of G_x for all $x \in V(G')$. The existence of a perfect matching in G_x is guaranteed by P2. Observe that G has a PC (s,t)-path if and only if there is an $M-$augmenting (s,t)-path P in G^{**}. Since an $M-$augmenting path P can be found in time $O(m^{**})$ (see [20]), we can find a PC (s,t)-path in G, if one exists, in time $O(m^{**})$.

To find a shortest PC (s,t)-path, we assign each edge in $\bigcup_{x \in V(G')} E(G_x)$ weight 0 and every other edge of G^{**} weight 1. Observe that a minimum weight perfect matching Q in the weighted graph G^{**} corresponds to a shortest PC (s,t)-path. Finding a minimum weight perfect matching can be done in time $O(n^{**} \cdot (m^{**} + n^{**} \log n^{**}))$. □

5 Long PC Cycles and Paths

The following interesting result and conjecture were obtained by Abouelaoualim, Das, Fernandez de la Vega, Karpinski, Manoussakis, Martinhon and Saad [1].

Theorem 5.1. *[1] Let G be a c-edge-coloured multigraph G with n vertices and with $\delta_{mon}(G) \geq \lceil \frac{n+1}{2} \rceil$. If $c \geq 3$ or $c = 2$ and n is even, then G has a Hamilton PC cycle. If $c = 2$ and n is odd, then G has a PC cycle of length $n-1$.*

Conjecture 5.2. *[1] Theorem 5.1 holds if we replace $\delta_{mon}(G) \geq \lceil \frac{n+1}{2} \rceil$ by $\delta_{mon}(G) \geq \lceil \frac{n}{2} \rceil$.*

We cannot replace $\delta_{mon}(G) \geq \lceil \frac{n+1}{2} \rceil$ by $\delta_{mon}(G) \geq \lceil \frac{n-1}{2} \rceil$ due to the following example. Let H_1 and H_2 be c-edge-coloured complete multigraphs (for each pair x, y of vertices and each $i \in \{1, 2, \ldots, c\}$ and $j \in \{1, 2\}$, H_j has a edge between x and y of colour i) of order $p+1$ that have precisely one vertex in common. Clearly, a longest PC cycle in $H_1 \cup H_2$ is of length $p+1$.

Since the longest PC path problem is $\mathcal{NP}$-hard, it makes sense to study lower bounds on the length of a longest PC path. The following result was proved by Abouelaoualim et al. [1].

Theorem 5.3. *Let G be a c-edge-coloured graph of order n with $\delta_{mon}(G) = d \geq 1$. Then G has a PC path of length at least* $\min\{n-1, 2\lfloor \frac{c}{2} \rfloor d\}$.

The authors of [1] raised the following two conjectures.

Conjecture 5.4. *Let G be a c-edge-coloured graph of order n and let $d = \delta_{mon}(G) \geq 1$. Then G has a PC path of length at least* $\min\{n-1, 2cd\}$.

They also conjectured the following analog of Theorem 5.3 for multigraphs:

Conjecture 5.5. *Let G be a c-edge-coloured multigraph of order n with $\delta_{mon}(G) = d \geq 1$. Then G has a PC path of length at least* $\min\{n-1, 2d\}$.

6 Longest PC Cycles and Paths in Edge-Coloured Complete Graphs

Let K_n^c denote a c-edge-coloured complete graph with n vertices.

Feng, Giesen, Guo, Gutin, Jensen and Rafiey [12] proved the following:

Theorem 6.1. *A K_n^c ($c \geq 2$) has a PC Hamilton path if and only if K_n^c contains a PC spanning 1-path-cycle subgraph.*

This theorem was first proved by Bang-Jensen and Gutin [4] for the case $c = 2$ and they conjectured that Theorem 6.1 holds for each $c \geq 2$. Theorem 6.1 implies that the maximum order of a PC path in K_n^c equals the maximum order of a PC 1-path-cycle subgraph of K_n^c.

As a result, the problem of finding a longest PC path in K_n^c is polynomial-time solvable for arbitrary $c \geq 2$. To see that a PC 1-path-cycle subgraph of K_n^c can be found in polynomial time, add a pair x, y of new vertices to K_n^c together with all edges needed to have a complete graph on $n+2$ vertices. Let the colour of all edges between x and y, and K_n^c be $c+1$ and let the colour of xy be $c+2$. Observe that the maximum order of a PC 1-path-cycle subgraph of K_n^c is less by exactly two than the maximum order of a PC cycle subgraph of the $c+2$-edge-coloured complete graph described above. It remains to apply Corollary 4.2.

The problem of finding a longest PC cycle K_n^c has not been solved yet for $c \geq 3$ as we will see below. For $c = 2$, Saad [17] found a characterization for longest PC cycles using the following notions. A pair of distinct vertices x, y of G are **colour-connected** if there exist PC (x, y)-paths P and Q such that

$\chi(f_P) \neq \chi(f_Q)$ and $\chi(\ell_P) \neq \chi(\ell_Q)$, where f_P and f_Q are the first edges of P and Q, respectively, and ℓ_P and ℓ_Q are the last edges of P and Q, respectively. We say that G is **colour-connected** if every pair of distinct vertices of G is colour-connected. Saad's characterization is as follows.

Theorem 6.2. *The length of a longest PC cycle in a colour-connected K_n^2 is equal to the maximum order of a PC cycle subgraph of K_n^2.*

Colour-connectivity for K_n^c is an an equivalence relation (see [6]). Using Theorem 6.2, Saad [17] showed that the problem of finding a longest PC cycle in K_n^2 has a polynomial-time randomized algorithm. Using a special case of Corollary 4.2, Bang-Jensen and Gutin [5] proved that the problem is, in fact, polynomial-time solvable. Theorem 6.2 implies the following:

Corollary 6.3. *[17] A K_n^2 has a PC Hamilton cycle if and only if K_n^2 is colour-connected and contains a PC cycle factor.*

There is another characterization of K_n^2 with a PC Hamilton cycle due to Bankfalvi and Bankfalvi, see [6]. The straightforward extension of Corollary 6.3 is not true for any $c \geq 3$ [6]. In fact, no characterization of K_n^c with a PC Hamilton cycle is known for any fixed $c \geq 3$ and it is a very interesting problem to obtain such a characterization. Even the following problem by Benkouar, Manoussakis, Paschos and Saad [8] is still open.

Problem 6.4 *Determine the complexity of the PC Hamilton cycle problem for c-edge-coloured complete graphs when $c \geq 3$.*

We conjecture that the PC Hamilton cycle problem for c-edge-coloured complete graphs when $c \geq 3$ is polynomial-time solvable.

In absence of characterization of K_n^c with a PC Hamilton cycle, sufficient conditions are interest. Manoussakis, Spyratos, Tuza and Voigt [16] proved the next result.

Proposition 6.5. *If $c \geq \frac{1}{2}(n-1)(n-2)+2$, then every K_n^c has a PC Hamilton cycle.*

Let $\Delta_{mon}(K_n^c)$ denote the largest monochromatic degree of K_n^c. Bollobás and Erdős [9] posed the following:

Conjecture 6.6. *Every K_n^c with $\Delta_{mon}(K_n^c) \leq \lfloor n/2 \rfloor - 1$ has a PC Hamilton cycle.*

Improving some previous results on this conjecture, Shearer [18] showed that if $7\Delta_{mon}(K_n^c) < n$, then K_n^c has a PC Hamilton cycle. So far, the best asymptotic estimate was obtained by Alon and Gutin [3].

Theorem 6.7. *[3] For every $\epsilon > 0$ there exists an $n_0 = n_0(\epsilon)$ so that for each $n > n_0$, every K_n^c satisfying $\Delta_{mon}(K_n^c) \leq (1 - \frac{1}{\sqrt{2}} - \epsilon)n$ contains a PC Hamilton cycle.*

References

[1] Abouelaoualim, A., Das, K.C., Fernandez de la Vega, W., Manoussakis, Y., Martinhon, C.A., Saad, R.: Cycles and paths in edge-colored graphs with given degrees (2007) (manuscript)

[2] Abouelaoualim, A., Das, K.C., Faria, L., Manoussakis, Y., Martinhon, C.A., Saad, R.: Paths and Trails in Edge-Colored Graphs. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 723–735. Springer, Heidelberg (2008)

[3] Alon, N., Gutin, G.: Properly colored Hamilton cycles in edge colored complete graphs. Random Struct. & Alg. 11, 179–186 (1997)

[4] Bang-Jensen, J., Gutin, G.: Alternating cycles and paths in edge-coloured multigraphs: a survey. Discrete Math. 165-166, 39–60 (1997)

[5] Bang-Jensen, J., Gutin, G.: Alternating cycles and trails in 2-edge-coloured multigraphs. Discrete Math. 188, 61–72 (1998)

[6] Bang-Jensen, J., Gutin, G.: Digraphs: Theory, Algorithms and Applications. Springer, London (2000), `www.cs.rhul.ac.uk/books/dbook/`

[7] Bang-Jensen, J., Gutin, G.: Digraphs: Theory, Algorithms and Applications, 2nd edn. Springer, London (in preparation)

[8] Benkouar, A., Manoussakis, Y., Paschos, V., Saad, R.: On the Complexity of Some Hamiltonian and Eulerian Problems in Edge-colored Complete Graphs. In: Hsu, W.-L., Lee, R.C.T. (eds.) ISA 1991. LNCS, vol. 557, pp. 190–198. Springer, Heidelberg (1991)

[9] Bollobás, B., Erdős, P.: Alternating Hamiltonian cycles. Israel J. Math. 23, 126–131 (1976)

[10] Dorninger, D.: On permultations of chromesomes, Contributions to General Algebra, vol. 5, pp. 95–103. Teubner-Verlag, Stuttgart (1987)

[11] Dorninger, D.: Hamiltonian circuits determining the order of chromosomes. Discrete Appl. Math. 50, 159–168 (1994)

[12] Feng, J., Giesen, H.-E., Guo, Y., Gutin, G., Jensen, T., Rafiey, A.: Characterization of edge-colored complete graph s with properly colored Hamilton paths. J. Graph Theory 53, 333–346 (2006)

[13] Gabow, H.N.: Data structures for weighted matching and nearest common ancestors with linking. In: Proc. SODA 1990, pp. 434–443 (1990)

[14] Grossman, J.W., Häggkvist, R.: Alternating cycles in edge-partitioned graphs. J. Combin. Theory Ser. B 34, 77–81 (1983)

[15] Gutin, G.: Note on edge-colored graphs and digraphs without properly colored cycles. Austral. J. Combin. (to appear)

[16] Manoussakis, Y., Spyratos, M., Tuza, Z., Voigt, M.: Minimal colorings for properly colored subgraphs. Graphs & Combin. 12, 345–360 (1996)

[17] Saad, R.: Finding a longest alternating cycle in a 2-edge-coloured complete graph is in RP. Combin. Prob. & Comput. 5, 297–306 (1996)

[18] Shearer, J.: A property of the colored complete graph. Discrete Math. 25, 175–178 (1979)

[19] Szeider, S.: Finding paths in graphs avoiding forbidden transitions. Discrete Appl. Math. 126, 261–273 (2003)

[20] Tarjan, R.E.: Data structures and network algorithms. SIAM, Philadelphia (1983)

[21] Yeo, A.: A note on alternating cycles in edge-coloured graphs. J. Combin. Theory Ser. B 69, 222–225 (1997)

Recognition of Antimatroidal Point Sets

Yulia Kempner[1] and Vadim E. Levit[2]

[1] Holon Institute of Technology, Israel
yuliak@hit.ac.il
[2] Ariel University Center of Samaria, Israel
levitv@ariel.ac.il

Abstract. The notion of "antimatroid with repetition" was conceived by Bjorner, Lovasz and Shor in 1991 as a multiset extension of the notion of antimatroid [2]. When the underlying set consists of only two elements, such two-dimensional antimatroids correspond to point sets in the plane. In this research we concentrate on efficient representation of antimatroidal point sets. We define a set of corner points that concisely represents a given antimatroidal point set and show how to reconstruct the antimatroidal point set from a proper set of corner points. We also present an algorithm allowing the given set of points to be recognized as a set of corner points of some antimatroidal point set.

1 Preliminaries

An antimatroid is an accessible set system closed under union [3]. An algorithmic characterization of antimatroids based on the language definition was introduced in [4]. Another algorithmic characterization of antimatroids that depicted them as set systems was developed in [6].

While classical examples of antimatroids connect them with posets, chordal graphs, convex geometries, etc., game theory gives a framework in which antimatroids are interpreted as permission structures for coalitions [1].

There are also rich connections between antimatroids and cluster analysis [8].

Recently, in mathematical psychology, antimatroids turned out to be useful tool for knowledge representation. Actually, antimatroids are used to describe feasible states of knowledge of a human learner [5].

In this paper we investigate the geometry properties of antimatroidal point sets in the plane that allows the original antimatroidal point set to be reconstructed from the compressed data. We show that antimatroidal point sets are defined by a pair of monotone paths made only with north and east steps. Moreover, it is enough to know only some extreme points on these paths to reconstruct a given antimatroidal point set.

Let E be a finite set. A *set system* over E is a pair $(E, \mathcal{F})$, where $\mathcal{F}$ is a family of sets over E, called *feasible* sets. We will use $X \cup x$ for $X \cup \{x\}$, and $X - x$ for $X - \{x\}$.

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 209–216, 2009.

Definition 1. *[10]A finite non-empty set system $(E, \mathcal{F})$ is an antimatroid if*
$(A1)$ for each non-empty $X \in \mathcal{F}$, there exists $x \in X$ such that $X - x \in \mathcal{F}$
$(A2)$ for all $X, Y \in \mathcal{F}$, and $X \not\subseteq Y$, there exists $x \in X - Y$ such that $Y \cup x \in \mathcal{F}$.

Any set system satisfying $(A1)$ is called *accessible.*

In addition, we use the following characterization of antimatroids.

Proposition 1. *[10] For an accessible set system $(E, \mathcal{F})$ the following statements are equivalent:*
(i) $(E, \mathcal{F})$ is an antimatroid
(ii) $\mathcal{F}$ is closed under union $(X, Y \in \mathcal{F} \Rightarrow X \cup Y \in \mathcal{F})$

A set system $(E, \mathcal{F})$ satisfies the *chain property* if for all $X, Y \in \mathcal{F}$, and $X \subset Y$, there exists a chain $X = X_0 \subset X_1 \subset ... \subset X_k = Y$ such that $X_i = X_{i-1} \cup x_i$ and $X_i \in \mathcal{F}$ for $0 \leq i \leq k$.

It is easy to see that the chain property follows from $(A2)$, but these properties are not equivalent.

A poly-antimatroid [11] is a generalization of the notion of the antimatroid to multisets. *A poly-antimatroid* is a finite non-empty multiset system (E, S) that satisfies the antimatroid properties $(A1)$ and $(A2)$.

Let $E = \{x, y\}$. In this case each point $A = (x_A, y_A)$ in the digital plane $\mathbb{Z}^2$ may be considered as a multiset A over E, where x_A is a number of repetitions of an element x, and y_A is a number of repetitions of an element y in multiset A. Consider a set of points in the digital plane $\mathbb{Z}^2$ that satisfies the properties of an antimatroid. That is a two-dimensional poly-antimatroid, or an antimatroidal point set.

For example, see an antimatroidal point set in Figure 1.

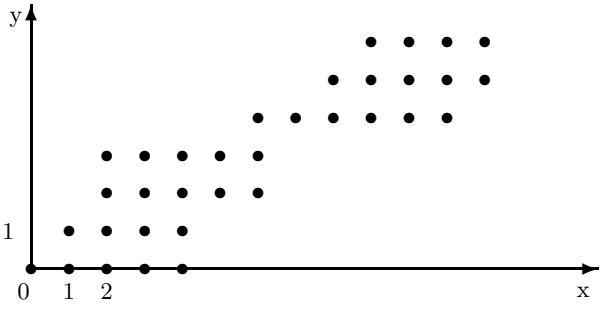


Fig. 1. An antimatroidal point set

2 Representation of Antimatroidal Point Sets

In this section we give a geometric characterization of antimatroidal point sets. The following notation [9] is used. If $A = (x, y)$ is a point in a digital plane, the

4-neighborhood $N_4(x, y)$ is the set of points

$$N_4(x, y) = \{(x-1, y), (x, y-1), (x+1, y), (x, y+1)\}$$

and *8-neighborhood* $N_8(x, y)$ is the set of points

$$N_8(x, y) = \{(x-1, y), (x, y-1), (x+1, y), (x, y+1), (x-1, y-1),$$
$$(x-1, y+1), (x+1, y-1), (x+1, y+1)\}.$$

Let m be any of the numbers 4 or 8. A sequence $A_0, A_1, ..., A_n$ is called an N_m-*path* if $A_i \in N_m(A_{i-1})$ for each $i = 1, 2, ...n$. Any two points $A, B \in S$ are said to be N_m-connected in S if there exists an N_m-path $A = A_0, A_1, ..., A_n = B$ from A to B such that $A_i \in S$ for each $i = 1, 2, ...n$. A digital set S is an N_m-*connected* set if any two points P,Q from S are N_m-connected in S. An N_m-*connected component* of a set S is a maximal subset of S, which is N_m-connected.

An N_m-path $A = A_0, A_1, ..., A_n = B$ from A to B is called *a monotone increasing* N_m-path if $A_i \subset A_{i+1}$ for all $0 \leq i < n$, i.e.,

$$(x_{A_i} < x_{A_{i+1}}) \wedge (y_{A_i} \leq y_{A_{i+1}}) \text{ or } (x_{A_i} \leq x_{A_{i+1}}) \wedge (y_{A_i} < y_{A_{i+1}}).$$

The chain property and the fact that the family of feasible sets of an antimatroid is closed under union mean that for each two points A, B: if $B \subset A$, then there is a monotone increasing N_4-path from B to A, and if A is non-comparable with B, then there is a monotone increasing N_4-path from both A and B to $A \cup B = (\max(x_A, x_B), \max(y_A, y_B))$. In particular, for each $A \in S$ there is a monotone decreasing N_4-path from A to 0. So, we can conclude that an antimatroidal point set is an N_4-connected component in the digital plane $\mathbb{Z}^2$.

A point set $S \subseteq \mathbb{Z}^2$ is defined to be *orthogonally convex* if, for every line L that is parallel to the x-axis ($y = y^*$) or to the y-axis ($x = x^*$), the intersection of S with L is empty, a point, or a single interval

$$([(x_1, y^*), (x_2, y^*)] = \{(x_1, y^*), (x_1 + 1, y^*), ..., (x_2, y^*)\}).$$

It follows immediately from the chain property that any antimatroidal point set S is an orthogonally convex connected component.

Lemma 1. *[7] An antimatroidal point set in the plane is closed under intersection, i.e., if two points $A = (x_A, y_A)$ and $B = (x_B, y_B)$ belong to an antimatroidal point set S, then the point*

$$A \cap B = (min(x_A, x_B), min(y_A, y_B)) \in S.$$

Lemma 1 implies that every antimatroidal point set is a union of rectangles built on each pair of non-comparable points.

Theorem 1. *[7] A set of points S in the digital plane $\mathbb{Z}^2$ is an antimatroidal point set if and only if it is an orthogonally convex N_4-connected set that is bounded by two monotone increasing N_4-paths between $(0, 0)$ and $(x_{\max}, y_{\max})$.*

To describe these two monotone increasing N_4-paths between $(0,0)$ and (x_{max}, y_{max}) we introduce the following notions.

A point A in set S is called an *interior point* in S if $N_8(A) \in S$. A point in S which is not an interior point is called a *boundary point*. All boundary points of S constitute the boundary of S. We can see an antimatroidal point set with its boundary in Figure 2.

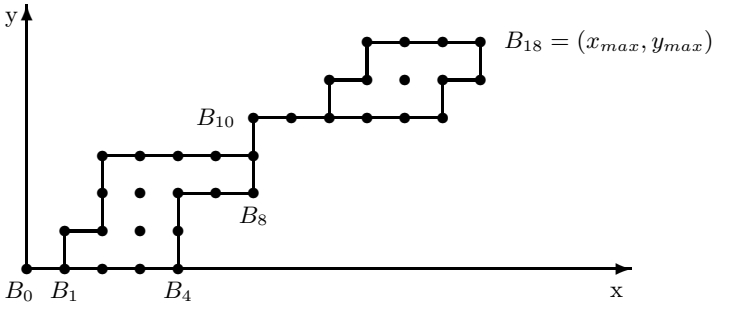


Fig. 2. A boundary of an antimatroidal point set

Since antimatroidal sets in the plane are closed under union and under intersection, there are six types of boundary points that we divide into two sets – lower and upper boundary:

$$\mathcal{B}_{lower} = \{(x,y) \in S : (x+1,y) \notin S \vee (x,y-1) \notin S \vee (x+1,y-1) \notin S\}$$

$$\mathcal{B}_{upper} = \{(x,y) \in S : (x-1,y) \notin S \vee (x,y+1) \notin S \vee (x-1,y+1) \notin S\}$$

Note, that $(0,0) \in \mathcal{B}_{lower} \cap \mathcal{B}_{upper}$, but it is possible that $\mathcal{B}_{lower} \cap \mathcal{B}_{upper}$ includes other points. For example, the point B_{10} in Figure 2 belongs to both the lower and upper boundaries.

Lemma 2. *[7] The boundaries $\mathcal{B}_{lower}$ and $\mathcal{B}_{upper}$ are monotone increasing N_4-paths between $(0,0)$ and $(x_{\max}, y_{\max})$ that bound an antimatroidal point set.*

Corollary 1. *Any antimatroidal point set S may be represented by its boundary in the following form:*

$$S = \mathcal{B}_{lower} \vee \mathcal{B}_{upper} = \{X \cup Y : X \in \mathcal{B}_{lower}, Y \in \mathcal{B}_{upper}\}$$

This result shows that the *convex dimension* of a two-dimensional poly-antimatroid is two. Convex dimension [10] $c\dim(S)$ of any antimatroid S is the minimum number of maximal chains $\emptyset = X_0 \subset X_1 \subset ... \subset X_k = X_{\max}$ with $X_i = X_{i-1} \cup x_i$ whose union gives the antimatroid S.

Since an antimatroidal point set is closed under union, it can be obtained as a union of all its irreducible elements. A multiset $A \in S$ is called *irreducible* if $A = X \cup Y$, where $X, Y \in S$, implies $A = X$ or $A = Y$. It is easy to see that

each interior point $A = (x, y) \in S$ is not irreducible, since from $N_8(A) \in S$ it follows that $A = (x-1, y) \cup (x, y-1)$. There exist both irreducible boundary points (see B_4 and B_8 in Fig. 2) and non-irrreducible boundary points (see B_{18} in Fig. 2).

A particular case of the Krein-Milman theorem states that given a convex polygon, one needs to know only the corners (extreme points) of the polygon to recover its shape. A straightforward analogy can be made between irreducible points and extreme points. Our observation is that the geometrical structure of antimatroidal point sets motivates another notion of the extreme point, which is based on the definition of extreme points in convex geometry. In the following we define an *extreme* point of an antimatroidal point set S as a point $X \in S$ such that $S - X$ is an antimatroidal point set as well.

For example, in Fig. 2 the boundary points B_4 and B_8 are extreme, while the boundary points B_1 and B_{10} are not extreme. In the general case, extreme points do not allow us to reconstruct an antimatroidal point set. For instance, the antimatroidal point set in Fig. 2 can not be reconstructed without the point B_{10}, which is not extreme.

A *corner point* of an antimatroidal point set S is a point $(x, y) \in S$ such that either $(x, y-1), (x+1, y) \notin S$ or $(x-1, y), (x, y+1) \notin S$.

Now, B_{10} is a corner point. Clearly, the set of corner points includes the set of extreme points. Moreover, each corner point is irreducible as well.

It easy to see that a corner point (x, y) with $(x, y-1), (x+1, y) \notin S$ belongs to $\mathcal{B}_{lower}$, and a corner point (x, y) with $(x-1, y), (x, y+1) \notin S$ belongs to $\mathcal{B}_{upper}$.

We call the corner points of the first group the *lower corner points*, and the corner points of the second group - the *upper corner points*.

Theorem 2. *Any poly-antimatroid is uniquely determined by its set of corner points.*

Proof. Given a set of corner points, our intention is to reconstruct the boundaries $\mathcal{B}_{lower}$ and $\mathcal{B}_{upper}$.

Let $C_1 = (x_1, y_1), C_2 = (x_2, y_2), ..., C_L = (x_L, y_L)$ be the sequence of lower corner points ordered by value of x.

Consider two neighboring points C_i and $C_{i+1}$with $i > 0$. Let

$$B = \min_x \max_y (C_i, C_{i+1}) = (x_i, y_{i+1}),$$

then the monotone increasing N_4-path C_iBC_{i+1} belongs to $\mathcal{B}_{lower}$. Indeed, since $(x_{i+1}, y_{i+1} - 1) \notin S$, by accessibility, the point $(x_{i+1} - 1, y_{i+1}) \in S$. If $x_{i+1} - 1 > x_i$, then the point $(x_{i+1} - 1, y_{i+1} - 1) \notin S$. If not, then there is a corner point with $x = x_{i+1} - 1$, i.e., there is a corner point between C_i and C_{i+1}. This implies that the point $(x_{i+1} - 1, y_{i+1}) \in \mathcal{B}_{lower}$ and the point $(x_{i+1} - 2, y_{i+1}) \in S$. Repeating this process, we obtain the monotone increasing N_4-path BC_{i+1} belonging to $\mathcal{B}_{lower}$. Hence, the monotone increasing N_4-path C_iB also belongs to $\mathcal{B}_{lower}$.

Note that $x_L = x_{\max}$. Suppose $C_L \neq (x_{\max}, y_{\max})$ then since $A_{\max} = (x_{\max}, y_{\max})$ is not a lower corner point, the point $(x_{\max}, y_{\max} - 1) \in S$. So, there is

a maximal k for which a point $(x_{\max}, y_{\max} - k) \in S$. Then, this point is the last lower corner point, i.e., it is the point C_L. Hence the monotone increasing N_4-path $C_L A_{\max}$ is the last segment of the lower boundary.

Similarly, if $C_1 \neq (0,0)$ then the monotone increasing N_4-path $(0,0)C_1$ completes the building of the lower boundary.

In the same way the set of upper corner points determines the $\mathcal{B}_{upper}$.

The boundary and corner points of any antimatroidal point set may be found with the use of the Upper and Lower boundary tracing algorithms.

Algorithm 3. ***Upper boundary tracing algorithm***

1. $i := 0$; $j := 0$; $x := x_{\max}$; $y := y_{\max}$;
2. $B_i := (x, y)$;
3. do
3.1 if $(x-1, y) \in S$ *then* $x := x - 1$
else
$j := j + 1$;
$C_j := (x, y)$;
$y := y - 1$;
3.2 $i := i + 1$;
3.3 $B_i := (x, y)$;
until $B_i = (0, 0)$
4. Return the sequence $\mathcal{B} = B_i, B_{i-1}, ..., B_0$
and the sequence $\mathcal{C} = C_j, C_{j-1}, ..., C_1$.

It is easy to check that Algorithm 3 returns a monotone increasing N_4-path $\mathcal{B}$ that passes only over the upper boundary points from $\mathcal{B}_{upper}$. Since each upper corner point $(x, y) \in \mathcal{B}_{upper}$ and characterizes by the fact that $(x-1, y) \notin S$, the sequence $\mathcal{C}$ is the set of all upper corner points without the point $(0,0)$. To verify if $(0,0)$ is an upper corner point it remains to check if $B_{i-1} = (1,0)$.

The Lower boundary tracing algorithm differs from Algorithm 3 in the search order only: (y, x) instead of (x, y). Step 3.1 of Algorithm 3 reads as follows:

3.1 if $(x, y-1) \in S$ then $y := y - 1$, else $x := x - 1$.

Correspondingly, the Lower boundary tracing algorithm returns the lower boundary of antimatroidal point sets and the set of lower corner points.

3 Recognition of Corner Points

We present a scheme of the algorithm that for a given set of points decides if it is a set of corner points for some antimatroidal set. If there exists a proper antimatroidal point set, then the algorithm divides the points of the given set

into two chains: the lower corner points and the upper corner points. Now the proof of Theorem 2 allows reconstruction of boundaries of a proper antimatroidal point set.

The algorithm is based on the following observations.

1. Let $C_1 = (x_1, y_1), C_2 = (x_2, y_2), ..., C_L = (x_L, y_L)$ be the sequence of lower corner points ordered by value of x, then $y_1 = 0$. Similarly, if $C_1 = (x_1, y_1)$, $C_2 = (x_2, y_2), ..., C_U = (x_U, y_U)$ is the sequence of upper corner points ordered by value of x, then $x_1 = 0$.
2. The last lower corner point C_L and the last upper corner point C_U are different, while $(x_L \geq x_U)$ and $(y_L \leq y_U)$.

Let $C_1 = (x_1, y_1), C_2 = (x_2, y_2), ..., C_n = (x_n, y_n)$ be an $\{x, y\}$-lexicographical order of the sequence of all corner points.

3. Let $C_1 = (x_1, y_1), C_2 = (x_2, y_2), ..., C_k = (x_k, y_k)$ be k first points of the ordered sequence of all corner points, $C_{i_1}, C_{i_2}, ..., C_{i_L}$ be a subsequence of lower corner points, and $C_{j_1}, C_{j_2}, ..., C_{j_U}$ be a subsequence of upper corner points. Then $x_{k+1} > x_{j_U}$, and if $y_{k+1} > y_{j_U}$, then $C_{k+1} \in \mathcal{B}_{upper}$.
4. Let $i < j$ and (x_i, y_i), (x_j, y_j) be two neighboring lower corner points. If (x_m, y_m) is the last upper corner point with x-coordinate not exceeding x_i, then $y_m \geq y_j$.

Algorithm 4. *Corner points recognition*

Input*: $\mathcal{C}$=$(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ be an $\{x, y\}$-lexicographical order of points in $\mathbb{Z}^2$.*

Output*: if $\mathcal{C}$ is the set of corner points of some antimatroidal point set, the algorithm returns the ordered sequences of lower corner points and upper corner points, otherwise it reports on the negative result.*

1. Initialization of two stacks S_upper and S_lower.
2. Based on Observation 1 *we check the first corner point:*
if it is a lower corner point we put it to S_lower
and the second corner point have to be an upper corner point;
if it is an upper corner point we put it to S_upper;
otherwise the algorithm returns the negative result.
3. Based on Observations 3 *we check each next point*
whether it is a corner point, and what boundary it belongs to.
If it is not an upper corner point we check it based on Observation 4
whether it is a lower corner point.
Otherwise the algorithm returns the negative result.
4. We check two last points C_L and C_U in accordance
with the condition of Observation 2*.*

Consider the complexity of Algorithm 4. On each step we check the next point, but verification of observation 4 involves traversing the stack *S_upper*, i.e., it requires $O(n)$ time. Thus the overall complexity of Algorithm 4 is $O(n^2)$.

4 Conclusion

In this paper we have emphasized the importance of the two-dimensional case as a first step in understanding the geometrical structure of poly-antimatroids. Some progress has been made, but we would like to study in more detail both a geometrical structure and an algorithmic description of three-dimensional antimatroidal point sets.

References

1. Bilbao, J.M.: Cooperative games under augmenting systems. SIAM Journal of Discrete Mathematics 17, 122–133 (2003)
2. Björner, A., Lovász, L., Shor, P.R.: Chip-firing games on graphs. European Journal of Combinatorics 12, 283–291 (1991)
3. Björner, A., Ziegler, G.M.: Introduction to greedoids. In: White, N. (ed.) Matroid applications. Cambridge Univ. Press, Cambridge (1992)
4. Boyd, E.A., Faigle, U.: An algorithmic characterization of antimatroids. Discrete Applied Mathematics 28, 197–205 (1990)
5. Eppstein, D.: Upright-Quad Drawing of st-planar learning spaces. Journal of Graph Algorithms and Applications 12(1), 51–72 (2008)
6. Kempner, Y., Levit, V.E.: Correspondence between two antimatroid algorithmic characterizations. The Electronic Journal of Combinatorics 10 (2003)
7. Kempner, Y., Levit, V.E.: Geometry of antimatroidal point sets, arXiv:0806.2096 [math.CO] (2008)
8. Kempner, Y., Muchnik, I.: Clustering on antimatroids and convex geometries. WSEAS Transactions on Mathematics 2(1), 54–59 (2003)
9. Klette, R., Rosenfeld, A.: Digital geometry: geometric methods for digital picture analysis. Morgan Kaufmann, San Francisco (2004)
10. Korte, B., Lovász, L., Schrader, R.: "Greedoids". Springer, New York (1991)
11. Nakamura, M.: Characterization of polygreedoids and poly-antimatroids by greedy algorithms. Oper. Res. Lett. 33(4), 389–394 (2005)

Tree Projections: Game Characterization and Computational Aspects

Georg Gottlob[1], Gianluigi Greco[2], Zoltán Miklós[3],
Francesco Scarcello[2], and Thomas Schwentick[4]

[1] Oxford University,
georg.gottlob@comlab.ox.ac.uk
[2] University of Calabria,
ggreco@mat.unical.it
scarcello@deis.unical.it
[3] École Polytechnique Fédérale de Lausanne,
zoltan.miklos@epfl.ch
[4] Universität Dortmund,
thomas.schwentick@udo.edu

Abstract. The notion of tree projection provides a natural generalization for various structural decomposition methods, which have been proposed in the literature in order to single out classes of nearly-acyclic (hyper)graphs. In this paper, the mathematical properties of the notion of tree projection are surveyed, and the complexity of the basic tree projection problem (of deciding the existence of a tree projection) is pinpointed. In more details, a game-theoretic characterization (in terms of the Robber and Captain game [15]) for tree projections is described, which yields a simple argument for the membership in NP of the tree projection problem. Eventually, the main ideas proposed in [14] and underlying the proof of NP-hardness of the tree projection problem are discussed.

Keywords: hypergraphs, tree projections, computational complexity.

1 Introduction

Many NP-hard problems in different application areas, ranging, e.g., from AI [19] and Database Theory [4] to Game theory [6], are known to be efficiently solvable when restricted to instances whose underlying structures can be modeled via acyclic graphs or hypergraphs. Indeed, on these kinds of instances, solutions can usually be computed via dynamic programming, by incrementally processing the acyclic (hyper)graph, according to some of its topological orderings.

Unfortunately, structures arising from real applications are hardly precisely acyclic. Yet, they are often not very intricate and, in fact, tend to exhibit only some limited degree of cyclicity, which suffices to retain most of the nice properties of acyclic structures. Therefore, several efforts have been made to investigate invariants that are best suited to identify nearly-acyclic graph/hypergraphs, leading to the definition of a number of so-called *structural decomposition methods*. These methods aim at transforming a given cyclic (hyper)graph into an acyclic one, by organizing its edges or its nodes

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 217–226, 2009.

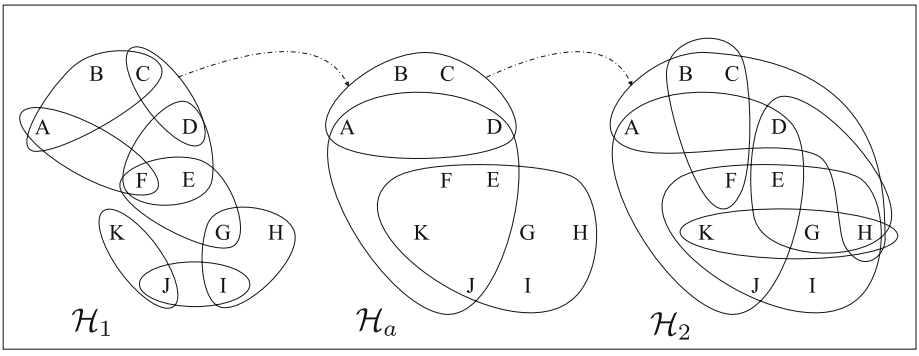


Fig. 1. A Tree Projection $\mathcal{H}_a$ of $\mathcal{H}_1$ with respect $\mathcal{H}_2$; e.g., $\{C, D\} \subseteq \{A, B, C, D\} \subseteq \{A, B, C, D, H\}$

into a polynomial number of clusters, and by suitably arranging these clusters as a tree, called decomposition tree. The original problem instance can then be evaluated over the decomposition tree, with a cost that is usually exponential in the cardinality of the largest cluster, also called *width* of the decomposition.

In fact, a number of powerful decomposition methods have been proposed for hypergraphs, such as the *hypertree decomposition* [12], the *generalized hypertree decomposition* [13], and the *fractional hypertree decomposition* [17]. By abstracting from the peculiarities of the various proposals, it has been observed that decompositions computed according to any of the methods above can, indeed, be viewed as solutions to a more general kind of problem, known as the *tree projection problem*: Given a pair of hypergraphs $(\mathcal{H}_1, \mathcal{H}_2)$, a tree projection of $\mathcal{H}_1$ w.r.t. $\mathcal{H}_2$ is an acyclic hypergraph $\mathcal{H}_a$ such that each hyperedge of $\mathcal{H}_1$ is contained in some hyperedge of $\mathcal{H}_a$, that is in its turn contained in a hyperedge of $\mathcal{H}_2$, which is called the *resource hypergraph*—see, Figure 1 for an illustration.

The importance of tree projections for database theory is well known since many years [11,22]. For instance, it has been shown that if a (natural-join) query is answered by a database program (consisting of joins, semijoins, and projections), then this program induces a tree projection of the original query with respect to the resource hypergraph consisting of all relations (schemas) computed by the program. And, vice versa, if there is a tree projection then the query may be correctly answered by the program computing such resources, suitably augmented with a linear number of semijoins [11]. Note that this result considers programs that are independent of the database size. Whether it holds or not for arbitrary programs (possibly with cycles) is an intriguing long-standing open question.

Notwithstanding its interest, many aspects of tree projections were obscure for long time, and some are still unknown, at the moment. In particular, the precise complexity of deciding the existence of a tree projection was not known. Neither there existed a game-theoretic characterization comparable to the robber-and-cops or robber-and-marshals game for tree decompositions and hypertree decompositions, respectively.

In this paper, we focus on the latter questions about tree projections. In more details, we describe a game-theoretic characterization (in terms of the Robber and Captain game) for tree projections, which has recently been presented in [15]. This characterization provides also a simple argument to show that the tree projection problem is actually in NP. In addition, the main ideas underlying the proof of intractability (NP-hardness) in [14] are also overviewed, to complete the complexity picture. We believe that these recent contributions may be very helpful for proving further results on tree projections, possibly sheding some light on some aspects that are still obscure, and may be relevant for applications in database theory and other related fields.

2 Preliminaries

Hypergraphs and Acyclicity. A *hypergraph* $\mathcal{H}$ is a pair (V, H), where V is a finite set of nodes and H is a set of hyperedges such that, for each $h \in H$, $h \subseteq V$. For the sake of clarity, we always denote V and H by $\mathcal{N}(\mathcal{H})$ and $\mathcal{E}(\mathcal{H})$, respectively. For any set of nodes $X \subseteq V$, the sub-hypergraph of $\mathcal{H}$ induced by X is the hypergraph (X, H') where $H' = \{h \cap X \mid h \in H\}$.

A hypergraph $\mathcal{H}$ is *acyclic* iff it has a join tree [4]. A *join tree* $\mathcal{JT}(\mathcal{H})$ for a hypergraph $\mathcal{H}$ is a tree whose vertices are the hyperedges of $\mathcal{H}$ such that, whenever the same node $X \in V$ occurs in two hyperedges h_1 and h_2 of $\mathcal{H}$, then h_1 and h_2 are connected in $\mathcal{JT}(\mathcal{H})$, and X occurs in each vertex on the unique path linking h_1 and h_2 in $\mathcal{JT}(\mathcal{H})$.

(Generalized) Hypertree Decomposition. A *hypertree for a hypergraph* $\mathcal{H}$ is a triple $\langle T, \chi, \lambda \rangle$, where $T = (N, E)$ is a rooted tree, and χ and λ are labeling functions which associate each vertex $p \in N$ with two sets $\chi(p) \subseteq \mathcal{N}(\mathcal{H})$ and $\lambda(p) \subseteq \mathcal{E}(\mathcal{H})$. If $T' = (N', E')$ is a subtree of T, we define $\chi(T') = \bigcup_{v \in N'} \chi(v)$. In the following, for any rooted tree T, we denote the set of vertices N of T by $vertices(T)$, and the root of T by $root(T)$. Moreover, for any $p \in N$, T_p denotes the subtree of T rooted at p.

A *generalized hypertree decomposition* of a hypergraph $\mathcal{H}$ is a hypertree $HD = \langle T, \chi, \lambda \rangle$ for $\mathcal{H}$ such that: (1) for each edge $h \in \mathcal{E}(\mathcal{H})$, there exists $p \in vertices(T)$ such that $h \subseteq \chi(p)$; (2) for each node $Y \in \mathcal{N}(\mathcal{H})$, the set $\{p \in vertices(T) \mid Y \in \chi(p)\}$ induces a (connected) subtree of T; and (3) for each $p \in vertices(T)$, $\chi(p) \subseteq \mathcal{N}(\lambda(p))$. The *width* of a generalized hypertree decomposition $\langle T, \chi, \lambda \rangle$ is $max_{p \in vertices(T)} |\lambda(p)|$. The *generalized hypertree width* $ghw(\mathcal{H})$ of $\mathcal{H}$ is the minimum width over all its generalized hypertree decompositions.

A *hypertree decomposition* [12] of $\mathcal{H}$ is a generalized hypertree decomposition $HD = \langle T, \chi, \lambda \rangle$ where: (4) for each $p \in vertices(T)$, $\mathcal{N}(\lambda(p)) \cap \chi(T_p) \subseteq \chi(p)$. The *hypertree width* $hw(\mathcal{H})$ of $\mathcal{H}$ is the minimum width over all its hypertree decompositions. Note that, for any hypergraph $\mathcal{H}$, it is the case that $ghw(\mathcal{H}) \leq hw(\mathcal{H}) \leq 3 \times ghw(\mathcal{H}) + 1$ [3]. Moreover, for any fixed natural number $k > 0$, deciding whether $hw(\mathcal{H}) \leq k$ is feasible in polynomial time (and, actually, is highly-parallelizable) [12], while deciding whether $ghw(\mathcal{H}) \leq k$ is NP-complete (for $k > 2$) [14].

Tree Projections. For two hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$ over the same set of nodes, we write $\mathcal{H}_1 \leq \mathcal{H}_2$ iff each hyperedge of $\mathcal{H}_1$ is contained in some hyperedge of $\mathcal{H}_2$. Let $\mathcal{H}_1 \leq \mathcal{H}_2$; then, a *tree projection* (short: TP) of $\mathcal{H}_1$ with respect to $\mathcal{H}_2$ is an acyclic

hypergraph $\mathcal{H}_a$ such that $\mathcal{H}_1 \leq \mathcal{H}_a \leq \mathcal{H}_2$. Whenever such a hypergraph $\mathcal{H}_a$ exists, we say that the pair of hypergraphs $(\mathcal{H}_1, \mathcal{H}_2)$ has a TP. The problem of deciding whether a pair of hypergraphs has a TP is called the *tree projection problem*, and it has recently been proven to be NP-complete [14].

Note that the notion of tree projection is more general than every structural decomposition method. For instance, consider the generalized hypertree decomposition approach: given a hypergraph $\mathcal{H}$ and a natural number $k > 0$, let $\mathcal{H}^k$ denote the hypergraph over the same set of nodes as $\mathcal{H}$, and whose set of hyperedges is given by all possible unions of k hyperedges in $\mathcal{H}$, i.e., $\mathcal{E}(\mathcal{H}^k) = \{h_1 \cup h_2 \cup \cdots \cup h_k \mid \{h_1, h_2, \ldots, h_k\} \subseteq \mathcal{E}(\mathcal{H})\}$. Then, it is well-known and easy to see that $\mathcal{H}$ has generalized hypertree width at most k if and only if there is a TP of $\mathcal{H}$ with respect to $\mathcal{H}^k$.

Similarly, for fractional hypertree decomposition (the most general known decomposition approach) [17], we may define a hypergraph $\mathcal{H}^{fk}$ over the same set of nodes as $\mathcal{H}$, and whose set of hyperedges is given by all possible unions of hyperedges in $\mathcal{H}$ having fractional width at most k, i.e., $\mathcal{E}(\mathcal{H}^{fk}) = \{\bigcup_{h \in H} h \mid H \subseteq \mathcal{E}(\mathcal{H}) \text{ and } fw(H) \leq k\}$. Again, it is easy to see that $\mathcal{H}$ has fractional hypertree width at most k iff there is a TP of $\mathcal{H}$ with respect to $\mathcal{H}^{fk}$.

3 The Robber and Captain Game

In this section, we overview the game-theoretic characterization of tree projections, recently introduced in [15]. This characterization is based on the *Robber and Captain* game that is played on a pair of hypergraphs $(\mathcal{H}_1, \mathcal{H}_2)$ over the same set of vertices $\mathcal{N}(\mathcal{H}_1) = \mathcal{N}(\mathcal{H}_2) = \mathcal{V}$, by a Robber and a Captain controlling some squads of cops, in charge of the surveillance of a number of strategic targets. The Robber stands on a vertex and can run at great speed along the hyperedges of $\mathcal{H}_1$; however, he is not permitted to run trough a vertex that is controlled by a cop. Each move of the Captain involves one squad of cops, which is encoded as a hyperedge $h \in \mathcal{E}(\mathcal{H}_2)$. The Captain may ask any cops in the squad h to run in action, as long as they occupy vertices that are currently reachable by the Robber, thereby blocking an escape path for the Robber. Thus, "second-lines" cops cannot be activated by the Captain. Note that the Robber is fast and may see cops that are entering in action. Therefore, while cops move, the Robber may run trough those positions that are left by cops or not yet occupied. The goal of the Captain is to place a cop on the vertex occupied by the Robber, while the Robber tries to avoid her capture.

For comparison, observe that this game is somewhere between the Robber and Marshals game of [13], where the marshals occupy a full hyperedge at each move, and the Robber and Cops game of [23], where each cop stands on a vertex and thus, if there are enough cops, any subset of any hyperedge can be blocked at each move. Instead, the Captain cannot employ "second-lines" cops, but only cops in charge of positions under possible Robber attacks.

To define the Robber and Captain game more formally, we need some notation. Let V and W two sets of nodes, and $X, Y \in \mathcal{V}$. Then, X is called $[V]$-adjacent to Y if there exists a hyperedge $h_1 \in \mathcal{E}(\mathcal{H}_1)$ such that $\{X, Y\} \subseteq (h_1 - V)$. Thus, here and in the following, the prefix $[V]$- indicates a graph property that holds after removing V.

A $[V]$-path from X to Y is a sequence $X = X_0, \ldots, X_\ell = Y$ of nodes such that X_i is $[V]$-adjacent to X_{i+1}, for each $i \in [0...\ell\text{-}1]$. We say that X $[V]$-touches Y if X is $[\emptyset]$-adjacent to $Z \in \mathcal{V}$, and there is a $[V]$-path from Z to Y; similarly, X $[V]$-touches the set W if X $[V]$-touches some node $Y \in W$. We say that W is $[V]$-connected if for all $X, Y \in W$ there is a $[V]$-path from X to Y. A $[V]$-component of $\mathcal{H}_1$ is a maximal $[V]$-connected non-empty set of nodes $U \subseteq (\mathcal{V} - V)$. For any $[V]$-component C, let $\mathcal{E}(C) = \{h \in \mathcal{E}(\mathcal{H}_1) \mid h \cap C \neq \emptyset\}$, and for a set of hyperedges $H \subseteq \mathcal{E}(\mathcal{H}_1)$, let $\mathcal{N}(H)$ denote the set of nodes occurring in H, that is $\mathcal{N}(H) = \bigcup_{h \in H} h$.

Definition 1 (R&C Game). The Robber and Captain game on $(\mathcal{H}_1, \mathcal{H}_2)$ (short: R&C$(\mathcal{H}_1, \mathcal{H}_2)$ game) is formalized as follows. A *position* for the Captain is a set M of vertices such that $M \subseteq h_2$, for some hyperedge (squad) $h_2 \in \mathcal{E}(\mathcal{H}_2)$. A *configuration* is a pair (M, v), where M is a position for the Captain, and $v \in \mathcal{V}$ is the vertex where the Robber stands. The initial configuration is $(\emptyset, v_0)$, where v_0 is a vertex arbitrarily picked by the Robber.

Let (M_i, v_i) be the configuration at step i. This is a capture configuration, where the Captain wins, if $v_i \in M_i$. Otherwise, the Captain activates the cops in a novel position M_{i+1} such that: for every $X \in M_{i+1}$, the node X $[M_i]$-touches v_i; afterwards, the Robber selects a node v_{i+1} such that there is a $[M_i \cap M_{i+1}]$-path from v_i to v_{i+1}. If the game continues forever, the Robber wins. □

Note that it does not make sense for the Captain to assume that the Robber is on a particular vertex, given the ability of the Robber of changing her positions before the cops land. Thus, given a configuration (M_i, v_i), we may assume w.l.o.g. that the next Captain's move is only determined by the $[M_i]$-component that contains v_i, rather than by v_i itself. Accordingly, positions can equivalently be written as (M_i, C_i), where C_i is an $[M_i]$-component. In this case, capture configurations have the form $(M, \emptyset)$, and the initial configuration has the form $(\emptyset, \mathcal{V})$.

Definition 2 (Strategies). A *strategy* σ (for R&C$(\mathcal{H}_1, \mathcal{H}_2)$) is a function σ that encodes the *moves* of the Captain, i.e., given a configuration (M_i, C_i), with $C_i \neq \emptyset$, σ returns a position M_{i+1} such that X $[M_i]$-touches C_i, for every X in M_{i+1}.

A *game-tree* for σ is a rooted tree $T(\sigma)$ defined over configurations as follows. Its root is the configuration $(\emptyset, \mathcal{V})$. Let (M_i, C_i) be a node in $T(\sigma)$ and let $M_{i+1} = \sigma(M_i, C_i)$. Then, (M_i, C_i) contains exactly one child (M_{i+1}, C_{i+1}), for each $[M_{i+1}]$-component C_{i+1} such that $C_i \cup C_{i+1}$ is $[M_i \cap M_{i+1}]$-connected; we call such a C_{i+1} an $[(M_i, C_i), M_{i+1}]$-option for the Robber. No further edge or node is in $T(\sigma)$.

Then, σ is said a *winning* strategy if $T(\sigma)$ is a finite tree. Moreover, define a position M_{i+1} to be a *monotone* move of the Captain in (M_i, C_i), if for each $[(M_i, C_i), M_{i+1}]$-option C_{i+1}, $C_{i+1} \subseteq C_i$. We say that σ is a *monotone* strategy if it only involves monotone moves. □

It has been observed in [15] that tree projections enjoy two fundamental properties, which are briefly discussed below.

Monotonicity. Firstly, there is no incentive for the Captain to play a strategy σ that is not monotone, since it is always possible for her to construct and play a monotone strategy σ' that is equivalent to σ, i.e., such that σ' is winning if and only if σ is winning.

This crucial property conceptually relates this game with the *Robber and Cops game* characterizing the *treewidth* [23], and differentiates it from most of the hypergraph-based games in the literature, in particular, from the *Robber and Marshals game*, whose monotone strategies characterize hypertree decompositions [13], while non-monotone strategies do not correspond with valid decompositions [1].

Theorem 1 ([15]). *On the* R&C$(\mathcal{H}_1, \mathcal{H}_2)$ *game, the existence of a winning strategy implies the existences of a monotone winning strategy.*

Normal Forms. The second important property pertains minimal tree projections. Formally, a tree projection $\mathcal{H}_a$ of $\mathcal{H}_1$ with respect to $\mathcal{H}_2$ is minimal if there is no tree projection $\mathcal{H}'_a \neq \mathcal{H}_a$ where for each hyperedge $h' \in \mathcal{E}(\mathcal{H}'_a) - \mathcal{E}(\mathcal{H}_a)$, a hyperedge $h \in \mathcal{E}(\mathcal{H}_a) - \mathcal{E}(\mathcal{H}'_a)$ exists with $h' \subseteq h$. In fact, it turns out that minimal tree projections enjoy some properties that are usually required for normal form decompositions in various notions of structural decomposition methods (see, e.g., [12]). In particular, there is a one-to-one correspondence between components and subtrees of the join tree.

For any hyperedge $h \in \mathcal{E}(\mathcal{H})$, let $\mathcal{JT}[h]$ denote the rooted tree obtained by rooting $\mathcal{JT}$ at the vertex h, let $\mathcal{JT}[h]_{h'}$ denote the subtree of $\mathcal{JT}[h]$ rooted at $h' \in \mathcal{E}(\mathcal{H})$, and let $\mathcal{N}(\mathcal{JT}[h]_{h'})$ denote the set of nodes occurring in the vertices of $\mathcal{JT}[h]_{h'}$.

Theorem 2 ([15]). *Let $\mathcal{H}$ be a minimal* TP *for $(\mathcal{H}_1, \mathcal{H}_2)$, and let $h \in \mathcal{E}(\mathcal{H})$. Then, there is a join tree $\mathcal{JT}$ for $\mathcal{H}$ having the following properties:*

(subtrees $\mapsto$ components). *For each pair $h_r, h_s \in \mathcal{E}(\mathcal{H})$ with h_s child of h_r in $\mathcal{JT}[h]$, there is exactly one $[h_r]$-component $comp^{\uparrow}(h_s)$ in $\mathcal{H}$ with $\mathcal{N}(\mathcal{JT}[h]_{h_s}) = C_r \cup (h_s \cap h_r)$. In addition, $h_s \cap comp^{\uparrow}(h_s) \neq \emptyset$ and $h_s \subseteq \mathcal{N}(\mathcal{E}(comp^{\uparrow}(h_s)))$.*

(components $\mapsto$ subtrees). *For each vertex h_r in $\mathcal{JT}[h]$ and for each $[h_r]$-component C_r in $\mathcal{H}$ such that $C_r \subseteq comp^{\uparrow}(h_r)$ (with $comp^{\uparrow}(h) \equiv \mathcal{N}(\mathcal{H})$), there is exactly one child h_s of h_r such that: $\mathcal{N}(\mathcal{JT}[h]_{h_s}) = C_r \cup (h_s \cap h_r)$, $h_s \cap C_r \neq \emptyset$, and $h_s \subseteq \mathrm{Fr}(C_r)$.*

3.1 Tree Projections and the C&R Game

The two properties above have been used in [15] to show that the Captain and Robber game R&C$(\mathcal{H}_1, \mathcal{H}_2)$ precisely characterizes the tree projection problem $(\mathcal{H}_1, \mathcal{H}_2)$, in the sense that a winning strategy for R&C$(\mathcal{H}_1, \mathcal{H}_2)$ exists if and only if $(\mathcal{H}_1, \mathcal{H}_2)$ has a TP. Hence, any decomposition technique that can be restated in terms of tree projections is in turn characterized by R&C games.

Theorem 3 ([15]). *A winning strategy exists in* R&C$(\mathcal{H}_1, \mathcal{H}_2)$ *iff* $(\mathcal{H}_1, \mathcal{H}_2)$ *has a* TP.

Proof (Idea). (only-if part) Let σ be a winning strategy. W.l.o.g., σ can be assumed to be monotone (cf. Theorem 1). Based on σ we can build a hypergraph $\mathcal{H}_a(\sigma)$, where for each vertex (M, C) in $T(\sigma)$, $\mathcal{E}(\mathcal{H}_a(\sigma))$ contains the hyperedge M; and, no further hyperedge is in $\mathcal{E}(\mathcal{H}_a(\sigma))$. Note that, by construction, $\mathcal{H}_a(\sigma) \leq \mathcal{H}_2$, since each position M is such that $M \subseteq h_2$ for some hyperedge $h_2 \in \mathcal{E}(\mathcal{H}_2)$.

Let h_1 be a hyperedge in $\mathcal{E}(\mathcal{H}_1)$. Since σ is a winning strategy, we may conclude that the Captain has necessarily covered in a complete form h_1 in some position. Thus, $\mathcal{H}_1 \leq \mathcal{H}_a(\sigma)$. Eventually, the fact that $\mathcal{H}_a(\sigma)$ is a tree projection follows after observing that by construction $\mathcal{H}_a(\sigma)$ is acyclic.

(if-part) W.l.o.g., let $\mathcal{H}_a$ be a minimal tree projection. Let us build a strategy σ as follows. Let $\mathcal{JT}[h]$ be a join tree for $\mathcal{H}_a$ rooted at an arbitrary hyperedge h, satisfying conditions in Theorem 2. The Captain initially selects the hyperedge h that is the root of $\mathcal{JT}[h]$, and at any times moves on the hyperedges in $\mathcal{H}_a$. Let $h_0 = \emptyset$ and $C_0 = \mathcal{N}(\mathcal{H}_1)$. Then, given the current position h_i, for each $[(h_{i-1}, C_{i-1}), h_i]$-option C_i in $\mathcal{H}_1$, $\sigma(h_i, C_i)$ coincides with the root of $\mathcal{JT}[h_i]_{C_i}$. Eventually, one may observe that σ is indeed a winning strategy, which in addition is monotone. In particular, this is firstly based on observing that on minimal tree projections, for each hyperedge $h \in \mathcal{E}(\mathcal{H}_a)$, C is a $[h]$-*component* in $\mathcal{H}_a \Leftrightarrow C$ is a $[h]$-*component* in $\mathcal{H}_1$. □

4 On the Complexity of the Tree Projection Problem

The complexity of the tree projection problem was raised by Goodman and Shmueli in 1984, and mentioned as an open problem in many papers [10,11,18,22].

4.1 Membership in NP

Consider the problem of deciding the existence of a tree projection of $\mathcal{H}_1$ w.r.t. $\mathcal{H}_2$. Observe that, in principle, every subset of any edge of $\mathcal{H}_2$ may belong to the acyclic hypergraph $\mathcal{H}_a$ we are looking for. Therefore $\mathcal{H}_a$ may well consist of an exponential number of hyperedges (w.r.t. to the size of $\mathcal{H}_1$ and $\mathcal{H}_2$).

However, recall the game-theoretic characterization of Theorem 3, and the fact that on such a game the existence of a winning strategy implies the existences of a monotone winning strategy in normal form (see Theorem 1 and Theorem 2). It is easy to conclude from these results that the existence of a tree projection entails the existence of a tree projection with polynomially many hyperedges (which in fact corresponds to an efficient winning strategy).

Theorem 4. *The tree projection problem is in* NP.

Proof. Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be two hypergraphs. A polynomial-time non-deterministic Turing machine with $(\mathcal{H}_1, \mathcal{H}_2)$ as its input may just guess a hypergraph $\mathcal{H}_a$, and then check that it is a tree projection of $\mathcal{H}_1$ w.r.t. $\mathcal{H}_2$. Since the latter task is feasible in deterministic polynomial-time, and from the discussion above a tree projection with polynomially many hyperedges always exists, this machine correctly solves the problem. □

4.2 NP-Hardness

The hardness of the tree projection problem was recently pinpointed in [14]. In this section, we precisely discuss the main ideas underlying the proof in [14], which relies on a reduction from 3SAT. Let us illustrate this reduction, by preliminary introducing some useful notation.

Notation and Reduction. For $n, m \geq 1$, we denote $\{1, \ldots, n\} \times \{1, \ldots, m\}$ by $[n; m]$. For each $p \in [n; m]$, we write $p \oplus 1$ for the successor of p in the usual lexicographic order on pairs, i.e. the order $(1,1), \ldots, (1,m), (2,1), \ldots, (n,1), \ldots, (n,m)$. We refer to the first element of such a set, i.e. to $(1,1)$, by $\min$ and the maximum element (n,m) by $\max$. Then, $[n;m]^-$ denotes the set $[n;m]$ without the maximal element, i.e. $[n;m]^- = [n;m] \setminus \{\max\}$.

Let $\varphi = \bigwedge_{j=1}^{m} (L_j^1 \vee L_j^2 \vee L_j^3)$ be a 3SAT formula with m clauses and variables $x_1, \ldots, x_n$. We construct two hypergraphs $\mathcal{H}_1 = (V, \mathcal{E}_1)$ and $\mathcal{H}_2 = (V, \mathcal{E}_2)$ such that $\mathcal{H}_1$ has a tree projection with respect to $\mathcal{H}_2$ if and only if φ is satisfiable. The vertex set of $\mathcal{H}_1$ and $\mathcal{H}_2$ is as follows.

$$V = \{y_i, y_i' \mid i \leq n\} \cup \{a_p \mid p \in [2n+3; m]\}$$

We use the following notation:

$$Y = \{y_1, \ldots, y_n\} \qquad Y_i = Y - \{y_i\}, \text{ for each } i \leq n$$
$$Y' = \{y_1', \ldots, y_n'\} \qquad Y_i' = Y' - \{y_i'\}, \text{ for each } i \leq n$$

For $p = (i,j)$, L_p^k denotes (a copy of) the literal $L_{i,j}^k$. The hyperedges of $\mathcal{H}_1$ and $\mathcal{H}_2$ are listed in Table 1.

Table 1. The hyperedge sets in the construction of $\mathcal{H}_1$ and $\mathcal{H}_2$. Here, e_p is in $\mathcal{E}_1$, $\forall\ p \in [2n+3; m]^-$.

$\mathcal{E}_1$	$\mathcal{E}_2$
$e = \{a_{\min}\} \cup Y$	$f = \{a_{\min}\} \cup Y \cup Y'$
$e_i = \{y_i, y_i'\}, i \leq n$	
$e_p = \{a_p, a_{p\oplus 1}\};$	$f_p^k = \begin{cases} \{a_p, a_{p\oplus 1}\} \cup Y \cup Y_l', \text{ if } p \in [2n+3; m]^-, L_p^k = x_l \\ \{a_p, a_{p\oplus 1}\} \cup Y_l \cup Y', \text{ if } p \in [2n+3; m]^-, L_p^k = \neg x_l \end{cases}$
$e' = \{a_{\max}\} \cup Y'$	$f' = \{a_{\max}\} \cup Y \cup Y'$

Proof Idea. The basic idea of the proof is as follows. The join tree T for the possible acyclic hypergraph $\mathcal{H}_a$ between $\mathcal{H}_1$ and $\mathcal{H}_2$ is intended to be a path consisting of $2n+3$ subpaths each of length m, corresponding to $2n+3$ copies of the sequence of m literals. We make use of elements $y_1, \ldots, y_n$ (corresponding to positive assignments to $x_1, \ldots, x_n$) and $y_1', \ldots, y_n'$ (corresponding to negative assignments to $x_1, \ldots, x_n$). The first node of the path should contain the elements $y_1, \ldots, y_n$, the last node $y_1', \ldots, y_n'$. Furthermore, the hyperedges e_i ensure that each y_i has to co-occur with y_i' in some node. Thus, each node on the path has to contain, for each i one of y_i or y_i'. However, the nodes will be able to contain such a variable only if it does not contradict the corresponding literal. E.g., a node corresponding to L_5 can not contain y_3 if $\neg x_3$ (and not x_3) occurs in L_5. Thus, such a join tree will exist if and only if φ is satisfiable.

Armed with the intuitions above, we can now illustrate the main steps of the proof. To show that a tree projection exists if φ is satisfiable let us define the set Z as

$Z = \{y_i \mid \rho(x_i) = 1\} \cup \{y'_i \mid \rho(x_i) = 0\}$, where ρ is a satisfying truth assignment for φ. Let $\mathcal{H}_a$ be a hypergraph with the following hyperedges:

- $\{a_{\min}\} \cup Y \cup Z$
- $\{a_p, a_{p\oplus 1}\} \cup Z$, for each $p \in [2n+3; m]^-$, and
- $\{a_{\max}\} \cup Y' \cup Z$.

One can verify that the hypergraph $\mathcal{H}_a$ has a join tree, thus it is acyclic and $\mathcal{H}_a$ is also a tree projection of $\mathcal{H}_1$ w.r.t. $\mathcal{H}_2$.

For the other direction, we have to show that if there exists a solution for the tree projection problem then the formula φ is satisfiable. In particular:

1. Since the vertex a_{min} is only contained in the hyperedge e of $\mathcal{H}_1$ and f of $\mathcal{H}_2$, any solution must have a hyperedge e_a, which is "between" e and f, i.e. $e \subseteq e_a \subseteq f$. (Similarly, a_{max} is only contained in the hyperedge e' of $\mathcal{H}_1$ and f' of $\mathcal{H}_2$, so any solution must have a hyperedge, which is "between" e' and f'.)
2. Any join tree of $\mathcal{H}_a$, if exists, must contain a path between two specific nodes, labeled with the hyperedges mentioned above.
3. Using the connectedness condition of the join tree one can derive a satisfying truth assignment for φ.

Putting it all together, the following can be proven.

Theorem 5 ([14]). *The tree projection problem is* NP*-hard.*

5 Conclusion

In this paper, we have overviewed some recent results on tree projections, namely, the game theoretic characterization discussed in [15] and the NP-completeness of the tree projection problem shown in [14].

As far as the complexity analysis is concerned, we remark here that the tree projection problem has some similarity to the *graph and hypergraph sandwich problems* surveyed in [8,9]. However, note that the latter definitions are slightly different from the problem considered in this paper. In particular, the definition of the acyclic hypergraph sandwich problem assumes that the three hypergraphs $\mathcal{H}_1$, $\mathcal{H}_2$, and $\mathcal{H}_a$ have the same number m of hyperedges, which may be labeled by a set of indices $\{1, \ldots, m\}$ such that $e_i(\mathcal{H}_1) \subseteq e_i(\mathcal{H}_a) \subseteq e_i(\mathcal{H}_2)$, for each $i \in \{1, ..., m\}$, whereas no such a restriction exists in the *tree projection problem*. In fact, it is still open whether the acyclic hypergraph sandwich problem is NP-hard or not.

References

1. Adler, I.: Marshals, monotone marshals, and hypertree-width. Journal of Graph Theory 47(4), 275–296 (2004)
2. Atserias, A., Bulatov, A., Dalmau, V.: On the Power of k -Consistency. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 279–290. Springer, Heidelberg (2007)

3. Adler, I., Gottlob, G., Grohe, M.: Hypertree-Width and Related Hypergraph Invariants. European Journal of Combinatorics 28, 2167–2181 (2007)
4. Bernstein, P.A., Goodman, N.: The power of natural semijoins. SIAM Journal on Computing 10(4), 751–771 (1981)
5. Bodlaender, H.L., Fomin, F.V.: A Linear-Time Algorithm for Finding Tree Decompositions of Small Treewidth. SIAM Journal on Computing 25(6), 1305–1317 (1996)
6. Daskalakis, C., Papadimitriou, C.H.: Computing pure nash equilibria in graphical games via markov random fields. In: Proc. of ACM EC 2006, pp. 91–99 (2006)
7. Fraigniaud, P., Nisse, N.: Connected Treewidth and Connected Graph Searching. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 479–490. Springer, Heidelberg (2006)
8. Golumbic, M.C., Kaplan, H., Shamir, R.: Graph sandwich problems. Journal of Algorithms 19, 449–473 (1995)
9. Golumbic, M.C., Wassermann, A.: Complexity and algorithms for graph and hypergraph sandwich problems. Graphs and Combinatorics 14, 223–239 (1998)
10. Goodman, N., Shmueli, O.: Syntactic characterization of tree database schemas. Journal of the ACM 30(4), 767–786 (1983)
11. Goodman, N., Shmueli, O.: The tree projection theorem and relational query processing. JCSS 28(1), 60–79 (1984)
12. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decompositions and tractable queries. J. of Computer and System Sciences 64(3), 579–627 (2002)
13. Gottlob, G., Leone, N., Scarcello, F.: Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. J. of Computer and System Sciences 66(4), 775–808 (2003)
14. Gottlob, G., Miklós, Z., Schwentick, T.: Generalized hypertree decompositions: NP-hardness and tractable variants. In: Proc. of PODS 2007, pp. 13–22 (2007)
15. Greco, G., Scarcello, F.: Tree Projections: Hypergraph Games and Minimality. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 736–747. Springer, Heidelberg (2008)
16. Grohe, M.: The complexity of homomorphism and constraint satisfaction problems seen from the other side. Journal of the ACM 54(1) (2007)
17. Grohe, M., Marx, D.: Constraint solving via fractional edge covers. In: Proc. of SODA 2006, pp. 289–298 (2006)
18. Lustig, A., Shmueli, O.: Acyclic hypergraph projections. Journal of Algorithms 30(2), 400–422 (1999)
19. Pearson, J., Jeavons, P.G.: A Survey of Tractable Constraint Satisfaction Problems, CSD-TR-97-15, Royal Holloway, Univ. of London (1997)
20. Robertson, N., Seymour, P.D.: Graph minors III: Planar tree-width. Journal of Combinatorial Theory, Series B 36, 49–64 (1984)
21. Ruzzo, W.L.: Tree-size bounded alternation. Journal of Cumputer and System Sciences 21, 218–235 (1980)
22. Sagiv, Y., Shmueli, O.: Solving queries by tree projections. ACM Transactions on Database Systems 18(3), 487–511 (1993)
23. Seymour, P.D., Thomas, R.: Graph searching and a min-max theorem for tree-width. Journal of Combinatorial Theory, Series B 58, 22–33 (1993)
24. Subbarayan, S., Reif Andersen, H.: Backtracking Procedures for Hypertree, HyperSpread and Connected Hypertree Decomposition of CSPs. In: Proc. of IJCAI 2007, pp. 180–185 (2007)

Author Index