Conferences in Research and Practice in
Information Technology

Volume 77

Theory of Computing 2008

Australian Computer Science Communications, Volume 30, Number 4

Theory of Computing 2008

Proceedings of the
Fourteenth Computing: The Australasian Theory Symposium (CATS 2008), Wollongong, NSW, Australia, January 2008

James Harland and Prabhu Manyem, Eds.

Volume 77 in the Conferences in Research and Practice in Information Technology Series.
Published by the Australian Computer Society Inc.

Published in association with the ACM Digital Library.

Theory of Computing 2008. Proceedings of the Fourteenth Computing: The Australasian Theory Symposium (CATS 2008), Wollongong, NSW, Australia, January 2008

Conferences in Research and Practice in Information Technology, Volume 77.

Editors:
James Harland
School of Computer Science and Information Technology
RMIT University
124 La Trobe Street
Melbourne 3001,
Australia
Email: jah@cs.rmit.edu.au

Prabhu Manyem
School of Information Technology and Mathematical Sciences
University of Ballarat
P.O. Box 663
Ballarat Victoria 3353
Australia
Email: p.manyem@ballarat.edu.au

Series Editors:
Vladimir Estivill-Castro, Griffith University, Queensland
John F. Roddick, Flinders University, South Australia
Simeon Simoff, University of Technology, Sydney, NSW
crpit@infoeng.flinders.edu.au

Publisher: Australian Computer Society Inc.
PO Box Q534, QVB Post Office
Sydney 1230
New South Wales
Australia.

Conferences in Research and Practice in Information Technology, Volume 77
ISSN 1445-1336
ISBN 978-1-920682-58-3

Printed December 2007 by Flinders Press, PO Box 2100, Bedford Park, SA 5042, South Australia.
Cover Design by Modern Planet Design, (08) 8340 1361.

The *Conferences in Research and Practice in Information Technology* series aims to disseminate the results of peer-reviewed research in all areas of Information Technology. Further details can be found at http://crpit.com/.

Table of Contents

Proceedings of the Fourteenth Computing: The Australasian Theory Symposium (CATS 2008), Wollongong, NSW, Australia, January 2008

Preface vii

Programme Committee viii

Organising Committee ix

CORE - Computing Research and Education xi

ACSW Conferences and the Australian Computer Science Communications xii

ACSW and CATS 2008 Sponsors xv

Keynote

Chipping Away at P vs NP: How Far Are We from Proving Circuit Size Lower Bounds? 3
Eric Allender

Contributed Papers

Logic and Types

The Inhabitation Problem for Intersection Types 7
Martin Bunder

Weak Parametric Failure Equivalences and Their Congruence Formats 15
Xiaowei Huang, Li Jiao and Weiming Lu

Modelling for Lazy Clause Generation 27
Olga Ohrimenko and Peter Stuckey

Optimisation

The Core Concept for 0/1 Integer Programming 39
Samuel Huston, Jakob Puchinger and Peter Stuckey

An ILP for the metro-line crossing problem 49
Matthew Asquith, Joachim Gudmundsson and Damian Merrick

A Multidimensional Bisection Method for Unconstrained Minimization Problem 57
Elena Morozova

Optimal Joint Vendor-Buyer Inventory Strategy for Deteriorating Items with Salvage Value 63
Nita H. Shah, Ajay S. Gor and Hui Wee

Parameterised Complexity

Tractable Cases of the Extended Global Cardinality Constraint 67
Marko Samer and Stefan Szeider

Parameterized Complexity of the Clique Partition Problem 75
Egbert Mujuni and Frances Rosamond

The Parameterized Complexity of Regular Subgraph Problems and Generalizations 79
Luke Mathieson and Stefan Szeider

Graph Algorithms

Well-covered Graphs and Greedoids 87
Vadim Levit and Eugen Mandrescu

On the Non-existence of Even Degree Graphs with Diameter 2 and Defect 2 93
Mirka Miller, Minh H. Nguyen and Guillermo Pineda-Villavicencio

Graph Classes and the Complexity of the Graph Orientation Minimizing the Maximum Weighted Outdegree 97
Yuichi Asahiro, Eiji Miyano and Hirotaka Ono

Algorithms

Generating Balanced Parentheses and Binary Trees by Prefix Shifts 107
Frank Ruskey and Aaron Williams

Testing Embeddability Between Metric Spaces 117
Ching-Lueh Chang, Yen-Wu Ti and Yuh-Dauh Lyuu

On the Efficiency of Pollard's Rho Method for Discrete Logarithms 125
Shi Bai and Richard P. Brent

Verifying Michael and Scott's Lock-Free Queue Algorithm using Trace Reduction 133
Lindsay Groves

Author Index 143

Preface

The fourteenth *Computing: The Australasian Theory Symposium* (CATS) is being held at the University of Wollongong, Australia during January 22-25, 2008. We received 28 submissions, out of which 17 were accepted. Each paper was thoroughly refereed by at least three reviewers from an international programme committee, followed by a healthy discussion among committee members.

The keynote speech will be delivered by Eric Allender from Rutgers University, New Jersey, USA. Professor Allender is a world renowned authority on Computational Complexity. He is a Fellow of the ACM, as well as being an ACM Distinguished Scientist.

A greater number of academics from around the world participated in the programme committee this year than previous years. The year is also notable in that students made a major contribution to a significant proportion (about one-third) of the accepted papers. This clearly forebodes a bright future for CATS.

We take this opportunity to thank the programme committee members and reviewers for all their hard work, and to the University of Wollongong for hosting the event. Congratulations to all authors whose submissions were accepted for presentation.

Welcome to all speakers, the keynote speaker, and other delegates to CATS. We wish you an enjoyable and productive time at Wollongong, and hope that the meeting serves as a platform for exciting new research initiatives in theoretical computer science.

James Harland
RMIT University
Prabhu Manyem
University of Ballarat

CATS 2008 Programme Chairs
January 2008

Programme Committee

Chairs

James Harland, RMIT University, Melbourne
Prabhu Manyem, University of Ballarat, Australia

Members

Argimiro Arratia, University of Valladolid, Spain
Richard Brent, Australian National University, Canberra
Hajo Broersma, University of Durham, UK
Cristian Calude, University of Auckland, NZ
Jeremy Dawson, Australian National University, Canberra
Thomas Erlebach, University of Leicester, UK
Graham Farr, Monash University, Melbourne
Joachim Gudmundsson, NICTA, Sydney
Venkatesan Guruswami, University of Washington, Seattle
Seokhee Hong, NICTA and the University of Sydney, Sydney
Costas Iliopoulos, Kings College, London
Mike Johnson, Macquarie University, Sydney
Jens Palsberg, UCLA
David Pearce, Victoria University of Wellington, NZ
R. Ramanujam, Institute of Mathematical Sciences, Chennai, India
Joe Ryan, University of Ballarat, Australia
Matthias Stallmann, North Carolina State University, USA
Richard Taylor, Defence Science and Technology Organisation, Canberra
Hans van Ditmarsch, University of Otago, NZ

Additional Reviewers

Shane Culpepper
Rod Downey
Mohammad Farshi
Eldar Hajilarov
Kamal Lodaya
Daniel Marx
Somnath Sikdar
Daniel Morales Silva
Alwen Tiu
Emlyn Williams
Zhiyou Wu
David Yost

Organising Committee

Welcome

I would like to welcome you to the University of Wollongong and ACSW 2008.

The Illawarra is a scenic, yet diverse, band of coastline stretching 85km south from the Royal National Park through to Wollongong, Shellhabour and the seaside town of Kiama. Wollongong has a strong industrial heritage and has attracted people from all around the world. The cosmopolitan nature of Wollongong has made it a truly global city where everyone feels at home. Some of the attractions you must see while in the city include the Nan Tien temple, Wollongong City Gallery, Science Centre and Planetarium.

Established in 1951, the University of Wollongong has forged a distinctive identity among Australian and international universities. An enterprising institution with a personalised style, UOW is confidently building an international reputation for quality research and education. With campuses stretching from Wollongong to Dubai, UOW has a total of 22,754 domestic students and 9,114 international students. The School of Computer Science and Software Engineering is one of the four schools in the Faculty of Informatics and has 38 academic and general staff. The school houses research hubs including Centre for Computer and Information Security Research, Centre for Visual Information Processing and Content Management, Centre for Intelligent Systems Research, and Decision System Laboratory.

ACSW 2008 includes the following conferences:

- Australasian Computer Science Conference (ACSC),
- Australasian Database Conference (ADC),
- Australasian Computer Education Conference (ACE),
- Computing: The Australian Theory Symposium (CATS),
- Asia-Pacific Conference of Conceptual Modelling (APCCM),
- Australasian User Interface Conference (AUIC),
- Australasian Symposium on Grid Computing and Research (AUSGRID),
- Australasian Workshop on Health Data and Knowledge Management (HDKM),
- Australasian Information Security Workshop:Privacy Enhancing Systems (AISW), and the
- Australasian Computing Doctoral Consortium (ACDC).

The nature of ACSW requires the cooperation of many people. I would like to thank all those who have worked to ensure the success of ACSW2008 including the Organizing Committee, the Conference Chairs and Programme Committees, the invited speakers and the delegates.

Professor Philip Ogunbona
Head, School of Computer Science and Software Engineering
University of Wollongong
January, 2008

General Chair

Professor Philip Ogunbona, School of Computer Science and Software Engineering, University of Wollongong

Organising Committee Members

Mrs Meghan Gestos
A/Prof Willy Susilo
A/Prof Yi Mu
Dr Zhiquan Zhou
Prof Aditya Ghose
Dr. Dr Yang-Wai Chow

CORE - Computing Research and Education

CORE welcomes all delegates to ACSW2008 in Wollongong.

ACSW, the Australasian Computer Science Week continues to grow with new conferences becoming entrenched in the week. As the premier annual Computer Science event in Australia and New Zealand, it provides an unparalleled opportunity for the wide community of Computer Science academics and researchers to meet, network, promote IT research and be exposed to the latest research in other areas of IT. The research presented at each conference is of the highest standard and essential for the growth and future of our region, in an ever more competitive world.

Despite desperate pleas from industry and government for IT staff, 2007 has again offered little growth in student numbers, particularly undergraduates, in ICT courses. This has affected almost all member departments and resulted in many CORE stalwarts retiring or taking redundancy. Many members have been active in a number of activities designed to address the issue but we do not yet seem to be winning the hearts or minds of potential students, their parents or careers advisors.

ACS, with whom we work closely, has released a new Core Body of Knowledge, CBOK. This provides us with the opportunity to rethink our courses but whether these will attract any more students remains to be seen.

A major activity for CORE this year has been a continuation of the 2006 ranking of ICT conferences and journals in preparation for the RQF. This activity drew considerable interest and input from many members.

Thank you all for your contributions in 2007 and we look forward to an interesting 2008.

CORE Computing Research & Education

Jenny Edwards
President, Computing Research and Education
January, 2008

ACSW Conferences and the Australian Computer Science Communications

The Australasian Computer Science Week of conferences has been running in some form continuously since 1978. This makes it one of the longest running conferences in computer science. The proceedings of the week have been published as the *Australian Computer Science Communications* since 1979 (with the 1978 proceedings often referred to as *Volume 0*). Thus the sequence number of the Australasian Computer Science Conference is always one greater than the volume of the Communications. Below is a list of the conferences, their locations and hosts.

2010 (Proposed). Communications Volume Number 32. Host and Venue - Queensland University of Technology, Brisbane, QLD.
2009. Volume 31. Host and Venue - Victoria University, Wellington, New Zealand.

2008. **Volume 30. Host and Venue - University of Wollongong, NSW.**

2007. Volume 29. Host and Venue - University of Ballarat, VIC. First running of HDKM.
2006. Volume 28. Host and Venue - University of Tasmania, TAS.
2005. Volume 27. Host - University of Newcastle, NSW. APBC held separately from 2005.
2004. Volume 26. Host and Venue - University of Otago, Dunedin, New Zealand. First running of APCCM.
2003. Volume 25. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Adelaide Convention Centre, Adelaide, SA. First running of APBC. Incorporation of ACE. ACSAC held separately from 2003.
2002. Volume 24. Host and Venue - Monash University, Melbourne, VIC.
2001. Volume 23. Hosts - Bond University and Griffith University (Gold Coast). Venue - Gold Coast, QLD.
2000. Volume 22. Hosts - Australian National University and University of Canberra. Venue - ANU, Canberra, ACT. First running of AUIC.
1999. Volume 21. Host and Venue - University of Auckland, New Zealand.
1998. Volume 20. Hosts - University of Western Australia, Murdoch University, Edith Cowan University and Curtin University. Venue - Perth, WA.
1997. Volume 19. Hosts - Macquarie University and University of Technology, Sydney. Venue - Sydney, NSW. ADC held with DASFAA (rather than ACSW) in 1997.
1996. Volume 18. Host - University of Melbourne and RMIT University. Venue - Melbourne, Australia. CATS joins ACSW.
1995. Volume 17. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Glenelg, SA.
1994. Volume 16. Host and Venue - University of Canterbury, Christchurch, New Zealand. CATS run for the first time separately in Sydney.
1993. Volume 15. Hosts - Griffith University and Queensland University of Technology. Venue - Nathan, QLD.
1992. Volume 14. Host and Venue - University of Tasmania, TAS. (ADC held separately at La Trobe University).
1991. Volume 13. Host and Venue - University of New South Wales, NSW.
1990. Volume 12. Host and Venue - Monash University, Melbourne, VIC. Joined by Database and Information Systems Conference which in 1992 became ADC (which stayed with ACSW) and ACIS (which now operates independently).
1989. Volume 11. Host and Venue - University of Wollongong, NSW.
1988. Volume 10. Host and Venue - University of Queensland, QLD.
1987. Volume 9. Host and Venue - Deakin University, VIC.
1986. Volume 8. Host and Venue - Australian National University, Canberra, ACT.
1985. Volume 7. Hosts - University of Melbourne and Monash University. Venue - Melbourne, VIC.
1984. Volume 6. Host and Venue - University of Adelaide, SA.
1983. Volume 5. Host and Venue - University of Sydney, NSW.
1982. Volume 4. Host and Venue - University of Western Australia, WA.
1981. Volume 3. Host and Venue - University of Queensland, QLD.
1980. Volume 2. Host and Venue - Australian National University, Canberra, ACT.
1979. Volume 1. Host and Venue - University of Tasmania, TAS.
1978. Volume 0. Host and Venue - University of New South Wales, NSW.

Conference Acronyms

ACE. Australian/Australasian Computing Education Conference.
ACSAC. Asia-Pacific Computer Systems Architecture Conference (previously Australian Computer Architecture Conference (ACAC).
ACSC. Australian/Australasian Computer Science Conference.
ACSW. Australian/Australasian Computer Science Week.
ADC. Australian/Australasian Database Conference.
AISW. Australasian Information Security Workshop.
APBC. Asia-Pacific Bioinformatics Conference.
APCCM. Asia-Pacific Conference on Conceptual Modelling.
AUIC. Australian/Australasian User Interface Conference.
AusGrid. Australasian Workshop on Grid Computing and e-Research.
CATS. Computing - The Australian/Australasian Theory Symposium.
HDKM. Australasian Workshop on Health Data and Knowledge Management.

Note that various name changes have occurred, most notably the change of the names of conferences to reflect a wider geographical area.

ACSW and CATS 2008 Sponsors

We wish to thank the following sponsors for their contribution towards this conference. For an up-to-date overview of sponsors of ACSW 2008 and CATS 2008, please see `http://www.cs.uow.edu.au/conf/acsw08/`.

University of Wollongong, Australia

AUSTRALIAN
COMPUTER
SOCIETY

Australian Computer Society

CORE - Computing Research and Education

RMIT
UNIVERSITY

School of Computer Science and Information Technology

School of Information Technology and Mathematical Sciences

Keynote

Chipping Away at P vs NP: How Far Are We from Proving Circuit Size Lower Bounds?

Eric Allender

Rutgers, the State University of New Jersey
Piscataway, New Jersey, USA

Many people are pessimistic about seeing a resolution to the P vs NP question any time soon. This pessimism extends also to questions about other important complexity classes, including two classes that will be the focus of this talk: TC^0 and NC^1.

TC^0 captures the complexity of several important computational problems, such as multiplication, division, and sorting; it consists of all problems computable by constant-depth, polynomial-size families of circuits of MAJORITY gates. TC^0_d is the subclass of TC^0 solvable with circuits of depth d. Although TC^0 seems to be a small subclass of P, it is still open if NP = TC^0_3.

NC^1 is the class of problems expressible by Boolean formulae of polynomial size. NC^1 contains TC^0, and captures the complexity of evaluating a Boolean formula.

Any proof that NP is not equal to TC^0 will have to overcome the obstacles identified by Razborov and Rudich in their paper on "Natural Proofs". That is, a "natural" proof that NP is not equal to TC^0 yields a proof that no pseudorandom function generator is computable in TC^0. This is problematic, since some popular cryptographic conjectures imply that such generators do exist. This leads to pessimism about the even more difficult task of separating NC^1 from TC^0.

Some limited lower bounds are within the grasp of current techniques, however. For example, several problems in P are known to require formulae of quadratic size — but this seems to be of little use in trying to prove superpolynomial formula size. Along similar lines, it is known that, for every d, there is a constant $c > 1$ such that the formula evaluation problem (one of the standard complete problems for NC^1) requires TC^0_d circuits of size at least n^c.

It might not seem too outrageous to hope to obtain a slightly stronger lower bound, showing that there is a $c > 1$ such that this same set requires uniform TC^0 circuits of size n^c (regardless of the depth d). We show that this would be sufficient to prove that TC^0 is properly contained in NC^1.

This is joint work with Michal Koucký, Czech Academy of Sciences, Prague.

Contributed Papers

The Inhabitation Problem for Intersection Types

M W Bunder[1]

[1] School of Mathematics and Applied Statistics
University of Wollongong
Wollongong NSW 2522 AUSTRALIA
Email: mbunder@uow.edu.au

Abstract

In the system $\lambda\wedge$ of intersection types, without ω, the problem as to whether an arbitrary type has an inhabitant, has been shown to be undecidable by Urzyczyn in [10]. For one subsystem of $\lambda\wedge$, that lacks the $\wedge$-introduction rule, the inhabitation problem has been shown to be decidable in Kurata and Takahashi [9]. The natural question that arises is: What other subsystems of $\lambda\wedge$, have a decidable inhabitation problem?

The work in [2], which classifies distinct and inhabitation-distinct subsystems of $\lambda\wedge$, leads to the extension of the undecidability result to $\lambda\wedge$ without the (η) rule. By new methods, this paper shows, for the remaining six (two of them trivial) distinct subsystems of $\lambda\wedge$, that inhabitation is decidable. For the latter subsystems inhabitant finding algorithms are provided.

Keywords: Lambda Calculus, Type Theory, Intersection Types, Inhabitation.

1 Introduction

In simple (Curry-style) type theory (see for example Hindley [8]), not every closed lambda term (or combinator) has a type. Coppo and Dezani-Ciancaglini in [7] extended simple type theory to include intersection types and the universal type ω, in their system all $\lambda-$terms have types.

We consider the type assignment system $TA_{\lambda\wedge}$ (or simply $\lambda\wedge$), which is that of [7], without ω, in which all closed $\lambda-$terms with normal form have types. We will be interested in the inhabitation problem which asks if it can be decided whether, for a type α, there is a term X such that $\vdash X : \alpha$ in a given type theory. For $\lambda\wedge$ the inhabitation problem was shown to be undecidable by Urzyczyn in [10]. For a subsystem of $\lambda\wedge$, that lacks the $\wedge$-introduction rule, the inhabitation problem has been shown to be decidable in Kurata and Takahashi in [9]. We detemine which, of the other natural subsystems of $\lambda\wedge$, as identified in [2], have a decidable inhabitation problem, in some cases this follows easily from the work in [9] and [10]. We also provide algorithms which allow us to find an inhabitant X for a type α, in the decidable systems.

Before doing this we need to detail the type systems and list some results from [2].

1.1 Definition (Types)

The set of types T is given by:

1. $a, b, c, ...$, atoms (or type variables) are types.
2. If α and β are types so are $(\alpha \to \beta)$ and $(\alpha \wedge \beta)$.

A type $\alpha \to \beta$ is called an $\to$-type. A type $\alpha \wedge \beta$ is called an $\wedge$-type.

The usual bracketing rules of logic will apply.

1.2 Definition (Statements)

If M is a λ-term and α a type, $M : \alpha$ is a statement.

1.3 Definition (Judgements)

If Δ is a set of statements $\{x_1 : \alpha_1, \ldots, x_n : \alpha_n\}$ where $x_1, \ldots, x_n$ are distinct variables and $M : \alpha$ is a statement, $\Delta \vdash M : \alpha$ is a judgement.

1.4 Definition (Postulates for the Type Assignment System $TA_{\lambda\wedge}$)

(Var) $\quad \Delta, x : \alpha \vdash x : \alpha$

$(\to E)$ $\quad \dfrac{\Delta \vdash M : \alpha \to \beta \qquad \Delta \vdash N : \alpha}{\Delta \vdash MN : \beta}$

$(\to I)$ $\quad \dfrac{\Delta, x : \alpha \vdash M : \beta}{\Delta \vdash \lambda x.M : \alpha \to \beta}$

$(\wedge I)$ $\quad \dfrac{\Delta \vdash M : \alpha \qquad \Delta \vdash M : \beta}{\Delta \vdash M : \alpha \wedge \beta}$

$(\wedge E)$ $\quad \dfrac{\Delta \vdash M : \alpha \wedge \beta}{\Delta \vdash M : \alpha} \qquad \dfrac{\Delta \vdash M : \alpha \wedge \beta}{\Delta \vdash M : \beta}$

(η) $\quad \dfrac{\Delta \vdash \lambda x.Nx : \alpha \qquad x \notin FV(N)}{\Delta \vdash N : \alpha}$

$\Delta \vdash M : \alpha$ (or more formally $\Delta \vdash_{\lambda\wedge} M : \alpha$) will represent: $\Delta \vdash M : \alpha$ can be derived from the above postulates.

The system $TA_{\lambda\wedge}$ will usually be abbreviated to $\lambda\wedge$.

An alternative formulation of $\lambda\wedge$ uses a preorder $\leq$ on T.

1.5 Definition ($\leq$)

Axioms
(1) $\alpha \leq \alpha$

(2) $\alpha \leq \alpha \wedge \alpha$
(3) $\alpha \wedge \beta \leq \alpha$
(4) $\alpha \wedge \beta \leq \beta$
(5) $(\alpha \to \beta) \wedge (\alpha \to \gamma) \leq \alpha \to \beta \wedge \gamma$

Rules
(6) $\alpha \leq \beta \leq \gamma \quad \Rightarrow \quad \alpha \leq \gamma$
(7) $\alpha \leq \alpha' \ \& \ \beta \leq \beta' \quad \Rightarrow \quad \alpha \wedge \beta \leq \alpha' \wedge \beta'$
(8) $\alpha \leq \alpha' \ \& \ \beta \leq \beta' \quad \Rightarrow \quad \alpha' \to \beta \leq \alpha \to \beta'.$

In Definition 1.4 the $(\wedge E)$ and (η) rules can be replaced by:

$$(\leq) \qquad \frac{\Delta \vdash M : \alpha \qquad \alpha \leq \beta}{\Delta \vdash M : \beta}$$

We will be interested in the following subsystems of $\lambda\wedge$.

1.6 Definition (Notation for Type Systems)

We will denote the system involving the judgements of Definition 1.3 for types, with postulates (Var), $(\to E)$ and $(\to I)$, by $\lambda(\ )$ and provability in this system by $\vdash$. Systems with additional rules will be denoted by $\lambda(\wedge I), \lambda(\wedge I, \eta)$ etc and the corresponding provability by $\vdash_{\wedge I}, \vdash_{\wedge I, \eta}$ etc. Clearly $\lambda\wedge$ is $\lambda(\wedge I, \wedge E, \eta)$ or $\lambda(\wedge I, \leq)$.

We will use λ and $\vdash_\lambda$ for Curry's simple type theory. This is $\lambda(\ )$ and $\vdash$ without the use of $\wedge$ in Definition 1.1(iii) and $(\wedge I)$ and $(\wedge E)$ in Definition 1.4.

We will write $A, B, C, \ldots$ for arbitrary type systems.

1.7 Definition (Inhabitation)

If A is one of the type systems of Definition 1.6, we say that a type α is inhabited if $(\exists M) \vdash_A M : \alpha$.

Note that α being inhabited does not imply that there is any algorithm that guarantees to find an inhabitant of α.

1.8 Definition (Inhabitation Problem)

The question as to whether, in a system A, it can be decided if an arbitrary type is inhabited or not is called the inhabitation problem of A.

Urzyczyn showed in [10] that the inhabitation problem for $\lambda\wedge$ is undecidable. Kurata and Takahashi have shown in [9] that the problem is decidable for $\lambda(\leq)$. Note that their method does not include an algorithm for finding an inhabitant for a given type.

In [2] we studied and classified the various subsystems of $\lambda\wedge$. We found that some of the subsystems A and B were equivalent in the sense that:

$$(\forall \alpha, M)\,[\vdash_A M : \alpha \quad \Leftrightarrow \quad \vdash_B M : \alpha] \qquad (1)$$

This is denoted by $A \approx_1 B$.

Additional systems A and B had equivalent inhabitation problems in that

$$(\forall \alpha)\,[(\exists N) \vdash_A N : \alpha \quad \Leftrightarrow \quad (\exists N) \vdash_B N : \alpha] \qquad (2)$$

This is denoted by $A \approx_2 B$.

Any pair of systems satisfying (2) that we found also satisfied

$$\begin{array}{ll} (\forall \alpha, M)\,[\vdash_A M : \alpha & \Rightarrow \quad \vdash_B M : \alpha] \vee \\ (\forall \alpha, M)\,[\vdash_B M : \alpha & \Rightarrow \quad \vdash_A M : \alpha] \end{array} \qquad (3)$$

Work in [2] showed that systems equivalent in the (2) - (3) sense come in the following groups (or inhabitation equivalence classes).

(1) $\lambda\wedge\ [\equiv \lambda(\wedge I, \wedge E, \eta) \approx_1 \lambda(\wedge I, \leq)],\ \lambda(\wedge I, \wedge E)$
(2) $\lambda(\wedge I), \lambda(\wedge I, \eta)$
(3) $\lambda(\leq)\ [\approx_1 \lambda(\leq, \wedge E, \eta) \approx_1 \lambda(\leq, \wedge E)]$
(4) $\lambda(\wedge E), \lambda(\wedge E, \eta)$
(5) $\lambda(\ )\ [\approx_1 \lambda(\eta)].$

Note that $\lambda(\leq)$ and $\lambda(\wedge E, \eta)$ are distinct systems that are both "$\lambda\wedge$ without $(\wedge I)$". $(a \wedge b \to b \wedge a$ is inhabited in $\lambda(\leq)$ but not in $\lambda(\wedge E, \eta)$.)

Urcyczyn's work in [10] for $\lambda\wedge$ and the inhabitation equivalence of the systems in Group 1 lead to:

1.9 Theorem

The inhabitation problems for the systems $\lambda\wedge, \lambda(\wedge I, \leq)$ and $\lambda(\wedge I, \wedge E)$ are undecidable.

The work of Kurata and Takahashi in [9] shows that $\lambda(\leq)$ is decidable. As the systems in Group 3 are equivalent it follows that:

1.10 Theorem

The inhabitation problems for $\lambda(\leq), \lambda(\leq, \wedge E, \eta)$ and $\lambda(\leq, \wedge E)$ are decidable.

The system considered by Kurata and Takahashi was actually $\lambda(\leq)$ with (ω), but the addition or deletion of (ω) does not affect the result.

We will show below, using generation lemmas proved in [2], that inhabitation problems for the systems in Groups 2, 4 and 5 are also decidable. We will in fact provide algorithms to find inhabitants for arbitrary types in these systems.

Note that in systems that do not have both $(\wedge E)$ and $(\wedge I)$ or the full strength of $(\leq)$, we may have

$$\begin{array}{rl} & \Delta \vdash M : \alpha \wedge \beta \\ \text{but not} & \Delta \vdash M : \beta \wedge \alpha \\ \text{and} & \Delta \vdash M : \alpha \wedge (\beta \wedge \alpha) \\ \text{but not} & \Delta \vdash M : (\alpha \wedge \beta) \wedge \alpha. \end{array}$$

Notation We write $\alpha_1 \wedge \ldots \wedge \alpha_n$ to represent one of the possible bracketings of $\alpha_1 \wedge \ldots \wedge \alpha_n$.

Of course, via the formulas as types isomorphism, a type in a type system can be considered as a theorem of a logic and its inhabitant as a proof of that theorem. The logics corresponding to the intersection type systems however, are not particularly simple (see Venneri [11] and Bunder [5] and [6]) and it is easier to examine decidability for the type theories rather than for the corresponding logics.

2 Inhabitation for $\lambda(\ )$

It is easy to show that any valid judgement $\Gamma \vdash \alpha$ in $\lambda(\ )$ can be transformed into a valid judgement $\Gamma' \vdash \alpha'$ in λ by replacing all distinct $\wedge$-types in Γ and α by distinct atoms. Hence as $\lambda(\ ) \approx_2 \lambda(\eta)$:

2.1 Theorem

The inhabitation problems for the systems $\lambda(\ )$ and $\lambda(\eta)$ are decidable.

If α is a type, an inhabitant of α, or a guarantee that there is none in $\lambda(\ )$ and $\lambda(\eta)$, can be provided by an inhabitant finding algorithm, such as that in [3], for λ, applied to the α' . (The methods used in [3] are a simplified version of the Ben-Yelles algorithm (see [4] and Hindley [8]).)

2.2 Example

$\tau = (a \wedge b \to c) \to a \wedge b \to (a \wedge b \to c \to (a \wedge b) \wedge d) \to (a \wedge b) \wedge d$

Let $\tau' = (e \to c) \to e \to (e \to c \to f) \to f$.

Using the algorithm of [3] for λ:

$x_1 : e \to c,\ x_2 : e$ and $x_3 : e \to c \to f$, give $x_1x_2 : c,\ x_3x_2(x_1x_2) : f$.

So $\vdash \lambda x_1x_2x_3.x_3x_2(x_1x_2) : \tau'$ and

$\vdash \lambda x_1x_2x_3.x_3x_2(x_1x_2) : \tau$.

Our proof of the decidability of the inhabitation problem for $\lambda(\wedge E)$ requires some additional notation and a number of preliminary lemmas.

3 Notation

3.1 Definition (Long Subterms)

An occurrence of a subterm N of a term M is said to be long in M if (i) $N \equiv x_iX_1 \dots X_n$ and the occurrence is not part of NX_{n+1} or (ii) if $N \equiv \lambda x_1 \dots x_k.Q$ and the occurrence is not part of $\lambda x_0.N$.

3.2 Definition (Positive and Negative Subtypes)

1. τ is a positive subtype of τ.
2. If $\alpha \to \beta$ is a positive (negative) subtype of τ then α is a negative (positive) subtype of τ and β is a positive (negative) subtype of τ.
3. If $\alpha \wedge \beta$ is a positive (negative) subtype of τ, α and β are positive (negative) subtypes of τ.

3.3 Definition (Long Subtypes)

An occurrence of a subtype α of a type τ is said to be a long $\to$-subtype of τ if the occurrence is not the α in a $\beta \to \alpha$ in τ.

An occurrence of a subtype α in τ is a long $\wedge$-subtype of τ if the occurrence is not the α in an $\alpha \wedge \beta$ or $\beta \wedge \alpha$ in τ.

3.4 Example

$\tau = (a \wedge b \to (c \to d) \to e) \wedge (f \to g)$.

τ and c are long positive $\to$ and $\wedge$ subtypes of τ ($\to \wedge$-subtypes).

$a \wedge b, c \to d$ and f are long negative $\to \wedge$-subtypes of τ.

$a \wedge b \to (c \to d) \to e$ and $f \to g$ are long positive $\to$-subtypes of τ.

a and b are long negative $\to$-subtypes of τ.

$c,\ e$ and g are long positive $\wedge$-subtypes of τ.

d is a long negative $\wedge$-subtype of τ.

3.5 Definition (Nontrivial Intersections)

A nontrivial intersection is any one other than one of the form $\alpha \wedge \dots \wedge \alpha$.

4 The Generation Lemma for $\lambda(\wedge E)$

The Generation Lemma follows directly from the work in [2], modified using Definition 1.11 and Lemma 4.3(v) of [2].

4.1 Lemma Generation Lemma for $\lambda(\wedge E)$

If

$$\Delta \vdash_{\wedge E} M : \alpha \tag{4}$$

then one of the following holds:

1. $M \equiv x,\ (\exists \beta)\ x : \beta\, \epsilon\, \Delta\ \&\ \beta \equiv \beta_1 \wedge \dots \wedge \alpha \wedge \dots \wedge \beta_n$.
2. $M \equiv PQ,\ (\exists \beta, \gamma)\ \Delta \vdash_{\wedge E} P : \gamma \to \beta$, $\Delta \vdash_{\wedge E} Q : \gamma$

   where the derivations are shorter than those of (4) and $\beta \equiv \beta_1 \wedge \dots \wedge \alpha \wedge \dots \wedge \beta_n$.
3. $M \equiv \lambda x.N,\ (\exists \beta, \gamma)\ \Delta, x : \beta \vdash_{\wedge E} N : \gamma$

where the derivation is shorter than that of (4) and $\alpha \equiv (\beta \to \gamma)$.

5 The Main Lemma for $\lambda(\wedge E)$

A derivation is said to have a cut if it has a use of $(\to E)$, as in Definition 1.4, where $\Delta \vdash M : \alpha \to \beta$ is derived by $(\to I)$, or a use of $(\wedge I)$ followed immediately by a use of $(\wedge E)$ (or an equivalent use of $(\leq)$). A derivation is normalised if it has no cuts.

It is well known (see [1]) that all derivations in $\lambda\wedge$ with ω can be normalised. All terms appearing in such derivations are in normal form. This result clearly applies to $\lambda\wedge$ and its subsystems as well.

5.1 Definition

The type of a variable x_m in a derivation of

$$x_1 : \tau_1, \dots, x_n : \tau_n \vdash_A N : \alpha$$

will be defined to be τ_m, (i) if $1 \leq m \leq n$, or, (ii) if $\lambda x_m.M$ is introduced into N by $(\to I)$ from

$$x_1 : \tau_1, \dots, x_m : \tau_m \vdash_A M : \beta.$$

In the derivation β is defined to be the type of the occurrence of M in N.

For systems without $(\wedge I)$ the type of a variable and the type of a term introduced, as a subterm, into a normalised derivation are uniquely defined. For systems with $(\wedge I)$ an occurrence of a term may have a finite set of types in a derivation.

5.2 Lemma

If $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash_{\wedge E} N : \alpha$, there exists a term M in β-normal form such that no two distinct variables of M have the same type and also:

1. $$x_{j_1} : \tau_{j_1}, \ldots, x_{j_\ell} : \tau_{j_\ell} \vdash_{\wedge E} M : \alpha \qquad (5)$$
   where $\{j_1, \ldots, j_\ell\} \subseteq \{1, \ldots, n\}$.
2. For every occurrence of a long subterm P of M with
   $$FV(P) = \{x_{i_1}, \ldots, x_{i_k}\}$$
   there are types $\tau_{i_1}, \ldots, \tau_{i_k}$ and β such that
   $$x_{i_1} : \tau_{i_1}, \ldots, x_{i_k} : \tau_{i_k} \vdash_{\wedge E} P : \beta \qquad (6)$$

where:

(I) $\tau_{i_1}, \ldots, \tau_{i_k}$ are long negative $\rightarrow\wedge$-subtypes of $\tau = \tau_1 \rightarrow \ldots \rightarrow \tau_n \rightarrow \alpha$.

(II) If P is of the form $\lambda x_r ... x_t.R$, β has a long positive $\rightarrow \wedge$-occurrence in α or a long negative $\rightarrow \wedge$-occurrence in one of $\tau_{i_1}, \ldots, \tau_{i_k}$.

(III) If P is of the form $x_r P_1 \ldots P_t, (t \geq 0)$, β has a long positive $\wedge$-occurrence in α or a long negative $\wedge$-occurrence in one of $\tau_{i_1}, \ldots, \tau_{i_k}$. Also β has a negative occurrence in α or a positive occurrence in one of $\tau_{i_1}, \ldots, \tau_{i_k}$. .

Proof (i) If N' is the β-normal form of N there is a normalised derivation of

$$x_1 : \tau_1, \ldots, x_n : \tau_n \vdash_{\wedge E} N' : \alpha. \qquad (7)$$

If in (7) a long subterm $x_s Q_1 \ldots Q_t$ of N' has type $\alpha_1 \rightarrow \ldots \rightarrow \alpha_u \rightarrow \gamma$, where γ is an atom or an intersection, this is replaced by $\lambda x_q \ldots x_{q+u-1}.x_s Q_1 \ldots Q_t x_q \ldots x_{q+u-1}$, where $x_q, \ldots x_{q+u-1}$ are variables not in N', $q > n$ and $\tau_{q+i-1} = \alpha_i$ for $q = 1, \ldots, u$. When all such changes to N' are made call the result N''.

Next free or bound variables x_p and x_q, with the same type in N'', are identified. For example $\lambda x_q.C_1[\lambda x_p.C[x_q, x_p]]$ becomes $\lambda x_p.C_1[\lambda x_p.C[x_p, x_p]]$. If $1 \leq p, q \leq n$, x_q can be omitted from the left hand side of the $\vdash$ in (7). None of these changes alter any subtypes of τ.

When all such changes have been made we have M and (7) becomes (5).

(ii) Let the variables in M other than $x_1, \ldots, x_n$ be $x_{n+1}, \ldots, x_m$ and their types be $\tau_{n+1}, \ldots, \tau_m$.

Case 1 $M \equiv P$.

In this case $\beta \equiv \alpha$ and (6) is (5) with any $x_j \notin FV(M)$ omitted. (This can always be done in a normalised derivation). (I) and (II) clearly hold and if P is of the form $x_r P_1 ... P_t$ it follows from Lemma 4.1(ii) that β has a positive occurrence in τ_r, so (III) holds.

We now prove the remaining cases by induction on M.

Case 2 $M \equiv x_i M_1 \ldots M_p$. $(p \geq 0)$ and P is, or is in, an M_j.

By Lemma 4.1(ii) applied p times we have:

$x_{j_1} : \tau_{j_1}, \ldots, x_{j_\ell} : \tau_{j_\ell} \vdash_{\wedge E} x_i : \alpha_1 \rightarrow \ldots \rightarrow \alpha_p \rightarrow \xi$

and

$x_{j_1} : \tau_{j_1}, \ldots, x_{j_\ell} : \tau_{j_\ell} \vdash_{\wedge E} M_j : \alpha_j$

where $\xi = \xi_1 \wedge \ldots \wedge \alpha \wedge \ldots \wedge \xi_t$ and $1 \leq j \leq p$.

We have (6) and (I) by the induction hypothesis, after leaving out variables not free in P. Also by the induction hypothesis, if P is of the form $\lambda x_r ... x_t.R$ and β does not have a long negative $\rightarrow \wedge-$occurrence in one of $\tau_{i_1}, \ldots, \tau_{i_k}$, it has a long positive $\rightarrow \wedge$-occurrence in α_j and so a long negative one in τ_i and a long positive one in α. Thus (II) holds.

If P is of the form $x_r P_1 ... P_t$, β has a long positive $\wedge$-occurrence in α_j (and so a long negative one in τ_i) or a long negative $\wedge$-occurrence in one of $\tau_{i_1}, ..., \tau_{i_k}$. Also β has a negative occurrence in α_j (and so a positive one in τ_i) or a positive occurrence in one of $\tau_{i_1}, ..., \tau_{i_k}$. Thus (III) holds.

Case 3 $M \equiv \lambda x_{n+1}.Q$, where P is, or is in, Q.

By Lemma 4.1 (iii)

$$x_{j_1} : \tau_{j_1}, \ldots, x_{j_\ell} : \tau_{i_\ell},\ x_{n+1} : \tau_{n+1} \vdash_{\wedge E} Q : \xi$$

where $\alpha \equiv \tau_{n+1} \rightarrow \xi$.

By the induction hypothesis and the omission of variables that are duplicated or not free in Q, (6) and (I) hold. If P is of the form $\lambda x_r ... x_t.R$, β has a long positive $\wedge$-occurrence in ξ, and so in α, or a long negative $\wedge$-occurrence in one of $\tau_1, ..., \tau_{n+1}$. If this is in τ_{n+1} it has a long positive $\wedge$-occurrence in α. (Note that P can't be Q, in this case, as then P is not long in M.) Thus (II) holds.

If P is of the form $x_r P_1 ... P_t$, β has a long positive $\wedge$-occurrence in ξ (and so in α) or a long negative $\wedge$-occurrence in one of $\tau_{i_1}, \ldots, \tau_{i_k}$. If this is in τ_{n+1} this is a positive $\wedge$-occurrence in α. Also β has a negative occurrence in ξ (and so in α) or a positive occurrence in one of $\tau_{i_1}, \ldots, \tau_{i_k}$. If this is τ_{n+1}, this is negative in α. Thus (III) holds.

Note 1. In many modern trteatments of $\lambda-$calculus, clashes of bound variables, as introduced in part (i) of the proof, though strictly allowed, are avoided. The identification of variables with the same types, in this proof, and that of Lemma 8.2, simplifies the proof and leads to finitely bounded inhabitation search algorithms for $\lambda(\wedge E)$ and $\lambda(\wedge I)$ in Sections 6 and 9.

2. In the lemma corresponding to 5.2 in [3] (and also in the Ben-Yelles algorithm in Hindley [8]), we could assume that M was in long normal form, which meant that every long subterm of M, formed by application had an atomic type. Here we can only assume that M must have an atomic or an intersection type. For example $M = x_1 x_2$ in:

$$x_1 : a \rightarrow (b \rightarrow c) \wedge (e \rightarrow f), x_2 : a,$$

$$x_3 : (b \rightarrow c) \wedge (e \rightarrow f) \rightarrow g \vdash_{\wedge E} x_3(x_1 x_2) : g$$

cannot be expanded to $\lambda x_4.x_1 x_2 x_4$ where $x_1 x_2 x_4$ has an atomic type.

5.3 Lemma

If

$$\Delta \vdash_{\wedge E} M : \tau, \qquad (8)$$

N appears in M and is introduced into the derivation of (8) by

$$\Delta' \vdash_{\wedge E} N : \alpha \qquad (9)$$

where $\Delta \subseteq \Delta'$, then if

$$\Delta' \vdash_{\wedge E} P : \alpha \qquad (10)$$

where $FV(P) \subseteq FV(N)$, we have

$$\Delta \vdash_{\wedge E} [P/N]M : \tau \qquad (11)$$

where in $[P/N]M$ only the given occurrence of N with type α, introduced in (9), is replaced by P.

Proof By induction, on M.

Case 1 $M \equiv N$ then $\Delta' \equiv \Delta$ and (11) is (10).

Case 2 $M \equiv RQ$ where N is (in) R or Q
By Lemma 4.1 (ii)

$$\Delta \vdash R : \gamma \to \beta$$

$$\Delta \vdash Q : \gamma$$

where $\beta \equiv \beta_1 \wedge \ldots \wedge \tau \wedge \ldots \wedge \beta_n$.
By the induction hypothesis we have

$$\Delta \vdash_{\wedge E} [P/N]R : \gamma \to \beta$$

or

$$\Delta \vdash_{\wedge E} [P/N]Q : \gamma$$

and $(\to E)$ and $(\wedge E)$ gives (11).

Case 3 $M \equiv \lambda x.R$ where N is (in) R
By Lemma 4.1 (iii) we have

$$\Delta, x : \beta \vdash_{\wedge E} R : \gamma$$

where $\beta \to \gamma \equiv \tau$.
The result follows by the induction hypothesis and $(\to I)$.

6 The Type Inhabitant Search Algorithm for $\lambda(\wedge E)$

Aim Given a $\to \wedge$-type τ, to find a λ-term M such that

$$\vdash_{\wedge E} M : \tau$$

Step 1 To each distinct long negative $\to \wedge$-subtype of τ assign a distinct variable, giving a list:

$$x_1 : \tau_1, \ldots, x_m : \tau_m$$

Step 2 For each type $\tau_i \equiv \alpha \wedge \beta$ from Step 1 write $x_i : \alpha, x_i : \beta$, repeat the procedure if α or β are intersections. Identical types for the same x_i may be omitted, identical types may also be obtained for distinct $x_i s$.

Step 3 For each set $A \subseteq \{x_1, \ldots, x_m\}$ and for each β that has both a long positive $\wedge$- and a negative occurrence in τ find an N , by application and $(\wedge E)$, such that $FV(N) \subseteq A$ and $N : \beta$, if there is not already such an N.

Step 4 For each set $A \subseteq \{x_1, \ldots, x_m\}$ and each β which has both a long positive and a long negative $\to \wedge$-occurrence in τ, if possible, find a term N by abstraction with respect to some or all of the variables found in Step 1 such that $FV(N) \subseteq A$ and $N : \beta$, if there isn't already such an N. (More than one abstraction with respect to the same variable is allowed.)

If after a Step 4 a closed $M : \tau$ is found stop. If not continue with further applications of Steps 3 and 4 until no new terms are created. If that happens, without forming $M : \tau, \tau$ has no inhabitants. This same algorithm, without Step 2, can be used to find inhabitants in $\lambda(\ )$ of λ.

6.1 Example

$\tau \equiv (a \to b \wedge (c \to d)) \to a \wedge c \to d$

Step 1 $x_1 : a \to b \wedge (c \to d),\ x_2 : a \wedge c$

Step 2 $x_2 : a,\ x_2 : c$

Step 3 $x_1 x_2 : b \wedge (c \to d),\ x_1 x_2 : c \to d,\ x_1 x_2 x_2 : d$

Step 4 $\lambda x_1 x_2 . x_1 x_2 x_2 : \tau$.

6.2 Example

$\tau = [(a \to b \to c) \wedge a \to b \to b \to c]$

Step 1 $x_1 : (a \to b \to c) \wedge a,\ x_2 : b$

Step 2 $x_1 : a \to b \to c,\ x_1 : a$

Step 3 $x_1 x_1 : b \to c,\ x_1 x_1 x_2 : c$

Step 4 $\lambda x_1 x_2 . x_1 x_1 : \tau$ and $\lambda x_1 x_2 x_2 . x_1 x_1 x_2 : \tau$.

6.3 Theorem

Given a type τ, the Type Inhabitant Search Algorithm for $\lambda(\wedge E)$ will produce an inhabitant in β-normal form for τ in $\lambda(\wedge E)$ and $\lambda(\wedge E, \eta)$, or show that there is no such inhabitant in either system.

Proof By Lemma 5.2, if τ has an inhabitant, it has one M in β-normal form with no two variables of the same type.

Also by Lemma 5.2, Step 1 of the algorithm provides us with a finite set of typed variables which is the largest set that need appear in M. Step 2 provides each of these variables with a finite (possibly empty) set of additional types which includes all the types these variables need take in M.

Lemma 5.2 also provides us with all the composite types subterms of M can have and these again form a finite set. By Lemma 5.3, once we have a subterm for M with a certain set of free variables, there is no need to look for another with a superset of this set of free variables. Hence the number of terms that can be formed is finite and these are systematically constructed by Steps 3 and 4.

As the inhabitant Search Algorithm for $\lambda(\wedge E)$ is inherently finite, if it terminates without having found an inhabitant for τ, then τ has none.

Given that $\lambda(\wedge E) \approx_2 \lambda(\wedge E, \eta)$ and that $\lambda(\wedge E)$ is a subsystem of $\lambda(\wedge E, \eta)$, this algorithm also finds inhabitants of τ in $\lambda(\wedge E, \eta)$ or shows that there are none.

7 The Generation Lemma for $\lambda(\wedge I)$

The Generation Lemma follows directly from the work in [2], modified using Definition 1.11 and Lemma 4.3(iv) of [2].

7.1 Lemma Generation Lemma for $\lambda(\wedge I)$

If

$$\Delta \vdash_{\wedge I} M : \alpha \tag{12}$$

where M is in normal form, then one of the following holds:

1. $M \equiv x,\ (\exists \beta)\ x : \beta\ \in \Delta\ \&\ \alpha \equiv \beta \wedge \ldots \wedge \beta$.

2. $M \equiv PQ,\ (\exists\beta_1, \alpha_1, \ldots \beta_k, \alpha_k)\Delta \vdash_{\wedge I} P : \beta_i \to \alpha_i$
$$\Delta \vdash_{\wedge I} Q : \beta_i.$$
for $1 \leq i \leq k$, where $\alpha \equiv \alpha_1 \wedge \ldots \wedge \alpha_k$ and the derivations, together, are shorter than those of (12).

3. $M \equiv \lambda x.N,\ (\exists\beta_1, \gamma_1, \ldots, \beta_k, \gamma_k)\ \Delta, x : \beta_i \vdash_{\wedge I} N : \gamma_i$

for $1 \leq i \leq k$, where $\alpha \equiv (\beta_1 \to \gamma_1) \wedge \ldots \wedge (\beta_k \to \gamma_k)$ and each derivation is shorter than that of (12).

8 The Main Lemmas for $\lambda(\wedge I)$

The two lemmas below are generalisations of two lemmas used in [3] to prove that the inhabitation finding algorithm for simple type theory, that appears there, is valid. The situation here is little more complex because one occurrence of a subterm of X in $\Delta \vdash_{\wedge I} X : \tau$ can have more than one type in the derivation.

For example with $x : \beta$ we might prove:

$$\Delta \vdash_{\wedge I} \lambda x.M : \beta \to \alpha$$

and with $x : \gamma$, we might prove:

$$\Delta \vdash_{\wedge I} \lambda x.M : \gamma \to \delta,$$

and so by $(\wedge I)$:

$$\Delta \vdash_{\wedge I} \lambda x.M : (\beta \to \alpha) \wedge (\gamma \to \delta).$$

So not only $\lambda x.M$, but also x, can have more than one type in a derivation.

8.1 Lemma

If X is in normal form, U is an occurrence of a subterm of X which has types $\alpha_1 \ldots \alpha_k$ in the derivation of $\Delta \vdash_{\wedge I} X : \tau$, then if V, with $FV(V) \subseteq FV(U)$, can also be assigned types $\alpha_1, \ldots, \alpha_k$, given Δ, and if the types of $FV(V)$ are the same as they were for $FV(U)$, then

$$\Delta \vdash_{\wedge I} X[U := V] : \tau.$$

Proof By induction on X, as in Lemma 5.3.

8.2 Lemma

If
$$x_1 : \tau_1, \ldots, x_\ell : \tau_\ell \vdash_{\wedge I} Z : \tau \qquad (13)$$
then there is an $X =_\beta Z$ in β-normal form such that:

1. No two distinct variables of X have the same set of types.
2. For some $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, \ell\}$,
$$x_{i_1} : \tau_{i_1}, \ldots, x_{i_k} : \tau_{i_k} \vdash_{\wedge I} X : \tau \qquad (14)$$
3. Each type of each variable x_{i_j} in $\lambda x_{i_1} \ldots x_{i_k}.X$ that is used in a derivation of (14), will have a long negative $\to$ - occurrence in $\tau' = \tau_{i_1} \to \ldots \to \tau_{i_k} \to \tau$.
4. Each occurrence of a composite subterm Y of X, that is, in the derivation of (14), an abstraction, or is long and formed by application, will have a type which is a long positive $\wedge$-occurrence in τ'.

Proof (i), (ii). Theorem 4.13 of [1] proves that every Z satisfying (13), with $\lambda\wedge$ for $\lambda(\wedge I)$, has a β-normal form. Clearly this also holds for $\lambda(\wedge I)$. Subject reduction can be proved for $\lambda(\wedge I)$, by standard means, so (13) holds with $nf(Z)$ for Z.

If $nf(Z)$ contains two variables x_p and x_q, with the same set of types, we can change, by Lemma 8.1, all occurrences of x_q to x_p, without altering any types. Also we can drop one of two, now identical $x_p : \tau_p$ and $x_q : \tau_q$ from $x_1 : \tau_1, \ldots, x_\ell : \tau_\ell$. We then have (14), so (i) and (ii) hold.

(iii), (iv) By induction on the length of X.

If X, which is in normal form, is formed by application, $X \equiv x_{i_t} X_1 \ldots X_m$ then by Lemma 7.1(ii) and (i) we have:

$$\begin{array}{lll} x_{i_1} : \tau_{i_1}, \ldots, x_{i_k} : \tau_{i_k} & \vdash_{\wedge I} & x_{i_t} : \beta_1 \to \ldots \to \beta_m \to \beta \\ x_{i_1} : \tau_{i_1}, \ldots, x_{i_k} : \tau_{i_k} & \vdash_{\wedge I} & X_r : \beta_r \end{array} \qquad (15)$$

for $1 \leq r \leq m$, where $\tau = \beta \wedge \ldots \wedge \beta$, $1 \leq t \leq k$ and $\tau_{i_t} = \beta_1 \to \ldots \to \beta_m \to \beta$.

If $x_{i_j} = x_{i_t}, \tau_{i_j} = \tau_{i_t}$ has a long negative $\to$ - occurrence in τ', so (iii) holds.

If x_{i_j} is in X_r, for some r, then, by the induction hypothesis (iii), τ_{i_j} has a long negative $\to$ - occurrence in $\tau_{i_1} \to \ldots \to \tau_{i_k} \to \beta_r$.

If this occurrence is in $\tau_{i_1} \to \ldots \to \tau_{i_k} \to$, this is a long negative $\to$ - occurrence in τ'. If it is in β_r, then τ_{i_j} has a long positive $\to$ - occurrence in τ_{i_t} and so a long negative $\to$ - occurrence in τ'. Hence (iii) holds.

If Y is a long composite subterm of X formed by application, it may be X itself, in which case (iv) holds with τ as the type of Y.

Otherwise Y is (in) an X_r. By the induction hypothesis (iv), applied to (15), we have that this occurrence of Y has a type with a long positive $\wedge$ - occurrence in $\tau_{i_1} \to \ldots \to \tau_{i_k} \to \beta_r$. If the occurrence is in $\tau_{i_1} \to \ldots \to \tau_{i_k} \to$, it is also one in τ'. If the occurrence is in β_r, as β_r has a long negative $\to$ - occurrence in τ_{i_t} and so a positive one in τ', each type of Y has a long positive $\wedge$ - occurrence in τ', i.e. (iv) holds.

If X is formed by abstractions, $X = \lambda x_{i_{k+1}}.V$, then by Lemma 7.1 (iii)

$$x_{i_1} : \tau_{i_1}, \ldots, x_{i_k} : \tau_{i_k}, x_{i_{k+1}} : \tau^s \vdash_{\wedge I} V : \beta^s \qquad (16)$$

for $1 \leq s \leq u$ and $\tau = (\tau^1 \to \beta^1) \wedge \ldots \wedge (\tau^u \to \beta^u)$.

As the derivation of (14) can come via (16), for $1 \leq s \leq u$, and $(\wedge I)$, any types τ_{i_j} of the variable x_{i_j} used in the derivation of (14) , must be used in the derivation of (16) for at least one value of s.

By the induction hypothesis (iii), applied to (16), we have that each τ_{i_j} has a long negative $\to$ - occurrence in $\tau_{i_1} \to \ldots \to \tau_{i_k} \to \tau^s \to \beta^s$ and so in τ'. Thus (iii) holds.

If Y is formed by application and is long in X, it is also long in V, so, by the induction hypothesis (iv), each occurrence of Y, in the derivation of (16), for some $s, 1 \leq s \leq u$, has a type with a long positive $\wedge$ - occurrence in $\tau_{i_1} \to \ldots \to \tau_{i_k} \to \tau^s \to \beta^s$ and so in τ'. So (iv) holds.

If Y is formed by abstraction and is (in) V, the result (iv) holds by induction hypothesis (iv).

If Y is X, (iv) holds as the type of Y is τ.

The $\lambda(\wedge I)$ Inhabitant Search Algorithm, given below, is a generalised version of the algorithm for λ given in [4]. The latter, in turn, is a simplified version of the Ben-Yelles Algorithm for λ (see Hindley [8]).

9 The $\lambda(\wedge I)$ Inhabitant Search Algorithm

Aim To find a λ-term X such that, for all $i, 1 \leq i \leq k$

$$x_1 : \tau_1^i, \ldots, x_n : \tau_n^i \vdash_{\wedge I} X : \delta_i \tag{17}$$

where $Xx_1 \ldots x_n$ has (i) a minimal number of distinct variables and (ii) a minimal total number of occurrences of these variables.

Notes 1. To find an inhabitant X of a type τ, we only need to solve (17) for $n = 0, k = 1$, and $\delta_1 = \tau$, but we require to solve more general versions of (17) in the process.

2. For no value of m and ℓ $(1 \leq m < \ell \leq n)$ is $\tau_\ell^i = \tau_m^i$ for all $i, 1 \leq i \leq k$, as otherwise $X[x_\ell := x_m]$ will be a solution of (17) where $X[x_\ell := x_m]x_1 \ldots x_{\ell-1}x_{\ell+1} \ldots x_n$ has fewer distinct variables than $Xx_1 \ldots x_n$.

3. We assume that no two instances of (17), for distinct values of i, are identical.

4. We will call a set of judgements, such as (17), for $1 \leq i \leq k$, with a common set of variables on the left of the $\vdash$ and a common unknown λ-term, such as X, on the right of the $\vdash$, a **simultaneous set of judgements (ssj)**.

5. An ssj of the form

$$x_1 : \tau_1^i, \ldots, x_n : \tau_n^i \vdash_{\wedge I} Y : \delta_i$$

for $1 \leq i \leq k$ is said to be **equivalent** to the ssj (17).

6. In the algorithm we will construct a tree where the nodes are ssj's, with (17) at the root.

Step 1 If in (17) $\tau_j^i = \delta_i$ for some j $(1 \leq j \leq n)$ and all $i, (1 \leq i \leq k)$, then the tree consists only of the root, the ssj (17), and the algorithm stops with $X = x_j$.

Otherwise, if $\tau_j^i = \beta_1^i \to \ldots \to \beta_r^i \to \delta_i$ for all i, $1 \leq i \leq k$, construct a group of ssj's

$$x_1 : \tau_1^i, \ldots, x_n : \tau_n^i \vdash_{\wedge I} X_{t_1} : \beta_{t_1}^i \tag{18}$$

each with $1 \leq i \leq k$, provided there is no ssj equivalent to (18) for any $t_1 (1 \leq t_1 \leq r)$ in the branch from (17) to the root of the tree. If there is such an equivalent ssj, (17) has no solution.

For each τ_j^i of the appropriate form, that is not excluded in this way, there is a group of ssj's of the form (18) with $1 \leq t_1 \leq r$, that appears in the tree directly below (17).

If all of the ssj's in a group have a solution, found by going back to Step 1, then (17) has as solution $X = x_j X_1 \ldots X_r$.

If there is no solution of (18) for any t_1, there is no solution of (17) for that value of j, and the tree is not extended below the ssj's in the group with that value of j.

If there is no solution of (17) for any value of j or no τ_j^i is of the right form and if $\delta_i = \tau_{n+1}^i \to \gamma_i$ for all i $(1 \leq i \leq k)$, go to Step 2. If $\delta_i = \alpha_i \cap \gamma_i$, for some i, go to Step 3.

Step 2 If $\tau_{n+1}^i \neq \tau_j^i$ for some i and all j the ssj

$$x_1 : \tau_1^i, \ldots, x_{n+1} : \tau_{n+1}^i \vdash_{\wedge I} X' : \gamma_i \tag{19}$$

with $1 \leq i \leq k$, appears directly below (17) in the tree (as a singleton group), provided no equivalent ssj appears in the branch from (17) to the root of the tree. (19) is then solved by returning to Step 1. If there is a solution then the solution to (17) is $\lambda x_{n+1}.X'$.

If $\tau_{n+1}^i = \tau_j^i$ for some j and all $i, 1 \leq i \leq k$, the ssj

$$x_1 : \tau_1^i, \ldots, x_n : \tau_n^i \vdash_{\wedge I} X' : \gamma_i \tag{20}$$

appears directly below (17) in the tree, provided no equivalent ssj appears in the branch from (17) to the root.

(20) is solved by returning to Step 1. If there is a solution, then $X = \lambda x_j.X'$.

Step 3 We assume that (17) is ordered so that this $i = k$. In each of the four cases below we add a new ssj directly below (17) in the tree (as a singleton group), provided no equivalent ssj has appeared in the branch from (17) to the root.

1. If $\tau_1^k, \ldots, \tau_n^k, \alpha_k$ and $\tau_1^k, \ldots, \tau_n^k, \gamma_k$ are both distinct from each $\tau_1^j, \ldots, \tau_n^j, \delta_j$ for $1 \leq j < k$, and from each other the ssj is

   $$x_1 : \tau_1^i, \ldots, x_n : \tau_n^i \vdash_{\wedge I} X : \delta_i \tag{21}$$

   for $1 \leq i \leq k+1$, where $\delta_k = \alpha_k$, $\delta_{k+1} = \gamma_k$, and $\tau_t^k = \tau_t^{k+1}$ for $1 \leq t \leq n$.

2. If $\tau_1^k, \ldots, \tau_n^k, \alpha_k$ is distinct from each $\tau_1^j, \ldots, \tau_n^j, \delta_j$ but $\tau_1^k, \ldots, \tau_n^k, \delta_k, \gamma_k \equiv \tau_1^j, \ldots, \tau_n^j, \delta_j$ for some $j, (1 \leq j \leq k)$ or $\equiv \tau_1^k, \ldots, \tau_n^k, \alpha_k$ the new ssj is similar to (17) but with α_k instead of δ_k.

3. If $\tau_1^k, \ldots, \tau_n^k, \gamma_k$ is distinct from each $\tau_1^j, \ldots, \tau_n^j, \delta_j$, but $\tau_1^k, \ldots, \tau_n^k, \alpha_k \equiv \tau_1^j, \ldots, \tau_n^j, \delta_j$ for some $j, (1 \leq j \leq k)$ the new ssj is similar to (17) but with γ_k for δ_k.

4. If $\tau_1^k, \ldots, \tau_n^k, \gamma_k \equiv \tau_1^r, \ldots, \tau_n^r, \delta_r$ and $\tau_1^k, \ldots, \tau_n^k, \alpha_k \equiv \tau_1^j, \ldots, \tau_n^j, \delta_j$ for some j and $r, 1 \leq j, r \leq k$, the new ssj is similar to (17), but with $1 \leq i \leq k-1$.

In each case now go back to Step 1.

We now prove that the algorithm is effective.

9.1 Theorem

The $\lambda(\wedge I)$ Inhabitation Search Algorithm provides a solution X for (17), for all $i (1 \leq i \leq k)$ or a guarantee that there is no solution, by generating a tree with an ssj at each node and (17) at the root. No node will have more than $rs + 1$ nodes directly below it and no branch is longer than $2^{s(r-n)!}$ where r is the number of long negative $\to$-subtypes of

$$\alpha = (\tau_1^1 \to \ldots \to \tau_n^1 \to \delta_1) \wedge \ldots \wedge (\tau_1^k \to \ldots \to \tau_n^k \to \delta_k)$$

and s is the number of long positive $\wedge$-subtypes of α.

Proof In the algorithm, Step 1 considers all the possible ways in which (17) can be derived by (Var) or using $(\to E)$ as the final step, Step 2 considers the ways in which (17) can be derived by $(\to I)$ as a final step and Step 3 the ways in which (17) is derived using $(\wedge I)$ as a final step. Any X satisfying (17), for $1 \leq i \leq k$, must be found in an indefinitely extended tree.

Coming down any branch of the tree from the root, the algorithm has each unknown λ-term as a subterm of the ones above it. If the step above it is Step 1 (other than $\tau_j^i = \delta_i$ for $1 \leq i \leq k$) or Step 2, it will be

a proper subterm of the terms higher in the branch. If on a branch there is an ssj

$$x_1 : \tau_1^i, \ldots, x_p : \tau_p^i \vdash_{\wedge I} X_{t_1 \ldots t_s} : \beta^i_{t_1 \ldots t_s} \tag{22}$$

for $1 \leq i \leq q$, appearing below an equivalent ssj

$$x : \tau_1^i, \ldots, x_p : \tau_p^i \vdash_{\wedge I} X_{t_1 \ldots t_v} : \beta^i_{t_1 \ldots t_v} \tag{23}$$

(i.e. $\beta^i_{t_1 \ldots t_s} = \beta^i_{t_1 \ldots t_v}$ for $1 \leq i \leq q$), then $X_{t_1 \ldots t_s}$ is a proper part of $X_{t_1 \ldots t_v}$, as there must be at least one nontrivial Step 1 or a Step 2 between the ssj's. This however means that $X_{t_1 \ldots t_s}$ is a shorter solution than $X_{t_1 \ldots t_v}$ of the ssj (23), so the branch from (23) to (22) does not lead to an X satisfying (i) and (ii). So the algorithm rightly does not search branches below (22).

We now show that the tree must be finite.

By Lemma 7.1 (17) holds for $1 \leq i \leq k$, if and only if

$$\vdash_{\wedge I} \lambda x_1 \ldots x_n . X : \alpha.$$

By Lemma 8.2 the variables in $\lambda x_1 \ldots x_n . X$, and so in X, have types which are long negative $\rightarrow$-subtypes of α. Let there be r of these.

In a node (i.e. an ssj)

$$x_1 : \tau_1^i, \ldots, x_m : \tau_m^i \vdash_{\wedge I} Y : \beta_i \tag{24}$$

for $1 \leq i \leq \ell$, as by the algorithm the types of different variables are distinct and $n < m \leq r$, there can be at most $(r-n)!$ different sequences $\tau_{n+1}, \ldots, \tau_m$.

Also by Lemma 8.2, each β_i is a long positive $\wedge$-subtype of α. Let there be s of these. Then there can be at most $s(r-n)!$ distinct judgements, of the form (24) in the tree generated by the algorithm and as each node (i.e. each ssj) in the tree consists of a subset of this set of judgements, there can be no more than $2^{s(r-n)!}$ distinct nodes (ssj's) in the tree.

Given that there can be no two equivalent ssj's on any branch, no branch can be longer than $2^{s(r-n)!}$.

Below any ssj, such as (24), there can be at most m groups of ssj's resulting from Step 1, where $m \leq r$. Each group can have no more than s members. Also below (24) there can be an ssj stemming from Step 2 or 3. Thus there can be no more than $rs + 1$ nodes below any node in the tree.

9.2 Corollary

The inhabitation problem for $\lambda(\wedge I)$is decidable.

9.3 Example

To find X such that

$$\vdash_{\wedge I} X : (a \to a \to a \wedge a) \wedge ((a \to b) \to (b \to a) \to a \to b)$$

$$\Big|$$

Step 3

$$\Big|$$

$$\vdash_{\wedge I} X : a \to a \to a \wedge a$$

$$\vdash_{\wedge I} X : (a \to b) \to (b \to a) \to a \to b$$

$$\Big|$$

Step 2 $\quad (X = \lambda x_1 . X')$

$$\Big|$$

$$x_1 : a \vdash_{\wedge I} X' a \to a \wedge a$$

$$x_1 : a \to b \vdash_{\wedge I} X' (b \to a) \to a \to b$$

$$\Big|$$

Step 2 $\quad (X' = \lambda x_2 . X'')$

$$\Big|$$

$$x_1 : a,\ x_2 : a \vdash_{\wedge I} X'' : a \wedge a$$

$$x_1 : a \to b,\ x_2 : b \to a \vdash_{\wedge I} X'' : a \to b$$

$$\Big|$$

Step 3

$$\Big|$$

$$x_1 : a,\ x_2 : a \vdash_{\wedge I} X'' : a$$

$$x_1 : a \to b,\ x_2 : b \to a \vdash_{\wedge I} X'' : a \to b$$

$$\Big|$$

Step 1 $\quad X'' = x_1$

So $\quad X = \lambda x_1 x_2 . x_1.$

10 Conclusion

The inhabitation problem is decidable for the systems $\lambda(\leq), \lambda(\leq, \wedge E, \eta), \lambda(\leq, \wedge E), \lambda(\wedge I), \lambda(\wedge I, \eta), \lambda(\wedge E), \lambda(\wedge E, \eta), \lambda()$, and $\lambda(\eta)$ and undecidable for $\lambda\wedge, \lambda(\wedge I, \leq)$ and $\lambda(\wedge I, \wedge E)$.

We have given inhabitant finding algorithms for $\lambda(), \lambda(\eta), \lambda(\wedge E), \lambda(\wedge E, \eta), \lambda(\wedge I)$ and $\lambda(\wedge I, \eta)$.

References

Barendregt, H.P., Coppo, M., Dezani, M. (1983), 'A filter lambda model and the completeness of type assignment', *Journal of Symbolic Logic* **48**, 931–940.

Bunder, M.W. (2002), 'A classification of intersection type system', *Journal of Symbolic Logic* **67**, 353–362.

Bunder, M.W. (2000), 'Proof finding algorithms for implicational logics', *Theoretical Computer Science* **232**, 165-186.

Bunder, M.W. (1995), 'Ben-Yelles-type algorithms and the generation of proofs in implicational logics', *University of Wollongong, Department of Mathematics Preprint Series* **3/95**.

Bunder, M.W. (2002), 'Intersection type for lambda terms and combinators and their logics', *Journal of the Interest Group in Propositional Logic* **10**, 357–378.

Bunder, M.W. (2003), 'Intersection type systems and logics related to the Meyer-Routley system B+', *Australasian Journal of Logic* **1**, 43–55.

Coppo, M. & Dezani, M. (1978), 'A new type-assignment for lambda terms', *Archiv Math. Logik* **19**(2), 139–156.

Hindley, J.R. (1987), *Basic Simple Type Theory*, Cambridge University Press.

Kurata, T. & Takahashi, M. (1995), Mining association rules between sets of items in large databases, *in* 'Lecture notes in Computer Science', Vol. 902, TLCA '95. M. Dezani and G. Plotkin (eds) pp. 297–311.

Urzyczyn, P. (1999), 'The emptiness problem for intersection types', *Journal of Symbolic Logic* **64**, 1195–1215.

Venneri, B. (1994), 'Intersection types as logical formulae', *Journal of Logic and Computation* **4**, 109–124.

Weak Parametric Failure Equivalences and Their Congruence Formats

Xiaowei Huang[1] **Li Jiao**[2] **Weiming Lu**[1]

[1] Academy of Mathematics and System Science,
Chinese Academy of Sciences, P.R. China.
Email: xwhuang@amss.ac.cn

[2] State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences, P.R. China.

Abstract

Weak equivalences are important behavioral equivalences in the course of specifying and analyzing the reactive systems using process algebraic languages. In this paper, we propose a series of weak equivalences named weak parametric failure equivalences, which take two previously-known behavioral equivalences, i.e., the weak failure equivalence and the weak impossible future equivalence, as their special cases. More importantly, based on the idea of the structural operational semantics, a series of rule formats are further presented to congruence format for their corresponding weak parametric failure equivalences, i.e., a specific equivalence is further congruent in any languages satisfying its corresponding congruence format. This series of rule formats reflect the gradual changes in the weak parametric failure equivalences. We conclude that, when the weak parametric failure equivalences become coarser, their corresponding rule formats turn tighter.

Keywords: weak failure equivalence, rule formats, structural operational semantics

1 Introduction

When using process algebraic languages to specify the distributed systems, a suitable semantic equivalence is usually necessary for reasoning and analyzing. There exist various semantic equivalences to be applied in various situations. An equivalence relation is reflexive, symmetric, and transitive.

Behavioral equivalences are based on the observability and thus equivalences may differ by the notions of observability (van Glabbeek, 2001, 1993). A natural classification of behavioral equivalences is that a given behavioral equivalence may be strong or weak. Their difference mostly exists in the ways of dealing with the internal transitions, which are generally denoted as τ transitions. Strong equivalences regard τ transitions the same as the observable actions. Weak equivalences, on the other hand, suppose them unobserved by the outer-world. Therefore, when the given distributed systems are further reactive systems, the weak equivalences/preorders are more suitable than the strong equivalences/preorders. A reactive systems can be seen as a black box, which computes by reacting to the stimuli, e.g., input and output, from its environments, and thus no internal transitions can be witnessed from the outside. In this paper, we will focus on the weak equivalences.

Among various weak semantic equivalences, the weak failure equivalence and the weak impossible future equivalence are two interesting semantic equivalences. The weak impossible future equivalence is strictly finer than the weak failure equivalence. The weak failure semantic is usually denoted, by its denotational characterization, as a set of weak failure pairs. Two processes are weak failure equivalent iff they have the same set of weak failure pairs. Likewise, the weak impossible future semantic is usually denoted as a set of weak impossible future pairs and two processes are weak impossible future equivalent iff they have the same set of weak impossible future pairs.

Looking into the two pairs, we find that their first parameters both express the abilities: some process p executes a sequence of observable actions and evolves into another process p'. The difference exists in their second parameters. The second parameter of a weak failure pair is a set of actions which are not enabled by p', and the second parameter of a weak impossible future pair is a set of action sequences which are not enabled by p'.

Based on these observations, we define a series of weak equivalences, called weak parametric failure equivalences. Like the above two weak equivalences, a weak i-failure pair with $i \in \mathbb{N} \cup \{\omega\}$ is only different in its second parameter with the weak failure pair, where $\mathbb{N}$ is the set of natural numbers and ω is the cardinality of $\mathbb{N}$. Its second parameter is a set of action sequences which are not enabled by p' and the lengthes of these action sequences do not exceed i. Therefore, the plain weak failure equivalence is the weak 1-failure equivalence in our framework and the weak impossible future equivalence is the weak ω-failure equivalence. Furthermore, with the increasing of the parametric i, the weak i-failure equivalence becomes finer.

Structural Operational Semantics (SOSs) (Plotkin, 2004) have been widely used in defining the meanings of the operators in various process algebraic language, such as CCS (Milner, 1989) and ACP (Baeten, 1990). The main idea of SOSs is: at first, each process is represented by a closed term and has some out-going transitions to communicate with outer world; then, these processes are coordinated by some specified rules, which are called transition rules, to get a higher-level process. As a result, the out-going transitions of this higher-level process are determined by the out-going transitions of its sub-processes.

Transition System Specifications (TSSs) (Groote, 1992), which borrowed from logic programming, form a theoretical basis for SOSs. By imposing some syntactic restrictions on TSS, one can retrieve so-called rule formats. From a specified rule format, one may deduce some interesting properties. Among these properties, one of the most important is whether or not a behavioral equivalence is congruent for a TSS in this rule format. A congruence is an essential equivalent property - namely that we can 'substitute equals for equals' (Milner, 1999). We will use language as an alias of the TSS. Up to now, some rule formats have been presented to meet the behavioral equivalences, for examples, GSOS format (Bloom, 1995) and ntyft/nxyft format (Groote, 1993) have been proved to

be congruent on strong bisimulation, de Simone (Simone, 1985) format was proved to be congruent on failure equivalence, and so on.

However, more works have been done on pursuing a suitable rule format for a given strong equivalence. On the contrary, much less attentions were paid on the rule formats for weak equivalences. More specifically, to our knowledge, no congruence formats have been presented for the weak failure equivalence or the weak impossible future equivalence.

In the paper, we will propose a series of rule formats for the newly defined weak parametric failure equivalences. In fact, weak 1-failure format is presented for the weak 1-failure equivalence, weak finite failure format is for the weak i-failure equivalences with $1 < i < \omega$, and weak ω-failure format is for the weak ω-failure equivalence. Then, we prove that the weak parametric failure equivalence can be preserved after composition if the language is in its corresponding rule formats, i.e., these rule formats are all congruence formats for their corresponding equivalences.

Here, we want to sketch out two critical points in pursuing these rule formats:

The first critical point is on the feasibility of allowing the rules with τ-conclusion. Rules with τ-conclusion are an important class of rules in classical process algebraic languages, notable examples include hiding operator of CSP and parallel composition operator of CCS. However, not all behavioral equivalences can be preserved under these rules, as is pointed out in Rensink and Vogler (Rensink, 2007) that the acceptance testing equivalence may not be preserved under the hiding operator. In this paper, we will take a close look into these rules. In fact, the weak i-failure equivalences with $i < \omega$ may not be preserved under these rules, but the weak ω-failure equivalence, i.e., the impossible future equivalence, can survive these rules.

The second critical point is whether or not the patience rules for receiving arguments are prerequisite in the rule formats for a given weak parametric failure equivalence. Patience rules, which are used to smooth the evolvement of τ transitions of subprocesses, are usually necessary in rule formats for weak equivalences. However, since patience rules are defined in accordance with the arguments of an operator, they can be divided into three classes: patience rules for active arguments, patience rules for receiving arguments and patience rules for other arguments. Though patience rules for active arguments are generally needed, patience rules for receiving arguments are not necessary for some rule formats. We find that, for the weak 1-failure format, patience rules for receiving arguments are not prerequisite by the help of the exclusion of rules with τ-conclusion. On the other hand, they are prerequisite for the weak i-failure format with $i > 1$.

As a result, the weak finite failure format is tighter than the weak ω-failure format because the rules with τ-conclusion should be excluded from the language in the weak finite failure format, and the weak 1-failure format is tighter than the weak finite failure format since it may further exclude the patience rules for receiving arguments from the language. Therefore, we can conclude that, when the weak parametric failure equivalences become coarser, their corresponding rule formats turn tighter.

Finally, we want to say more on the newly-proposed weak i-failure equivalences with $1 < i < n$. In fact, we have not found their niche applications, though they can be used in most applications of the 1-failure equivalences. The reasons that we introduce these intermediate weak equivalences are that

1) they can smooth the changes between the weak 1-failure equivalence and the weak ω-failure equivalence,

2) their congruence format is also an intermediate format between the weak 1-failure format and the weak ω-failure format, and

3) most importantly, we want to make clear the technical reasons why there exist differences between the weak 1-failure format and the weak ω-failure format. Take it more concrete, from the weak ω-failure format to the weak 1-failure format, the reason why the rules with τ conclusion are excluded is that the parameter i degrades from infinite to finite, and the reason why the patience rules for receiving arguments are not necessary is that, no matter how they are presented in the language, only the set of next one, but not next finite or infinite, observable actions remains unique.

The structure of this paper is: in the section 2, we will introduce some preliminaries, mainly on the behavioral equivalences and the rule formats in Structural Operational Semantics. Then in Section 3, we will put forward the formal definitions on the weak parametric failure equivalences. Intuitive motivations on their rule formats will be exhibited with examples in Section 4. Section 5 is devoted to the formal definitions of the rule formats, and the proofs on the congruence theorems. And then, in Section 6, we will conclude the paper.

2 Preliminaries on Behavioral Equivalences and Rule Formats

Let Act denote a set of names which will be used to label on events and Act^* be the set of all action sequences. We usually use $a, b, ...$ to range over the actions in Act, and use $A, B, ...$ to range over subsets of actions in Act. τ is generally used to denote the internal action which can not be observed by the outer world, and we use $\alpha, \beta, ...$ to range over the actions in $Act \cup \{\tau\}$. $\delta, \mu, \sigma, ...$ is to range over the sequences of actions. $\Phi, \Psi, ...$ is to range over the sets of sequences. $p, q, ...$ will be used to represent processes.

Any behavioral semantics of some process p can be characterized by a function $O(p)$ (van Glabbeek, 2001). $O(p)$ constitutes the observable behaviors of p. The equivalence relation $\sim_O$ can be defined by $p \sim_O q \Longleftrightarrow O(p) = O(q)$. The readers are referred to van Glabbeek (van Glabbeek, 2001, 1993) for comprehensive reviews of the behavioral equivalences.

SOS has been widely accepted as a tool to define operational semantics of processes. A TSS is a formalization of SOS (Plotkin, 2004). The readers are referred to Aceto, Fokkink and Verhoef (Aceto, 2001) for a comprehensive review on SOS.

Definition 2.1 (Aceto, 2001) Let $V = \{x_1, x_2, ...\}$ be a set of variables. A signature Σ is a collection of function symbols $f \notin V$ equipped with a function $ar : \Sigma \to N$. The set $\mathbb{T}(\Sigma)$ of terms over a signature Σ is defined recursively by: 1) $V \subseteq \mathbb{T}(\Sigma)$; 2) if $f \in \Sigma$ and $t_1, ..., t_{ar(f)} \in \mathbb{T}(\Sigma)$, then $f(t_1, ..., t_{ar(f)}) \in \mathbb{T}(\Sigma)$.

A term $c()$ is abbreviated as c. For $t \in \mathbb{T}(\Sigma)$, $var(t)$ denotes the set of variables that occur in t. $\mathbb{T}(\Sigma)$ is the set of closed terms over Σ, i.e., the terms $p \in \mathbb{T}(\Sigma)$ with $var(p) = \emptyset$. A Σ substitution ζ is a mapping from V to $\mathbb{T}(\Sigma)$.

In the paper, we will use $p, q, ...$ to range over the closed terms, and call them processes.

Definition 2.2 A positive Σ-literal is an expression $t \xrightarrow{\alpha} t'$ and a negative Σ-literal is an expression $t \xrightarrow{\alpha} \not\rightarrow$ with $t, t' \in \mathbb{T}(\Sigma)$ and $\alpha \in Act \cup \{\tau\}$. A transition rule over Σ is an expression of the form $\frac{H}{C}$ with H a set of Σ literals (the premises of the rule) and C a positive Σ-literal (the conclusion). The left- and right-hand side of C are called the source and the target of the rule, respectively. Moreover, if $r = \frac{H}{t \xrightarrow{\alpha} t'}$ then define $ante(r) = H$, $cons(r) = \{t \xrightarrow{\alpha} t'\}$, and the output of r as α.

A TSS, written as (Σ, Ψ), consists of a signature Σ and a set Ψ of transition rules over Σ. A TSS is positive if the premises of its rules are positive. In the paper, we often use language as an alias of the TSS.

Definition 2.3 Let Σ be a signature. A context C of n holes over Σ is simply a term in $\mathbb{T}(\Sigma)$ in which n variables occur, each variable only once. If $t_1, ..., t_n$ are terms over Σ, then $C(t_1, ..., t_n)$ denotes the term obtained by substituting t_1 for the first variable occurring in C, t_2 for the second variable occurring, etc. Thus, if $x_1, ..., x_n$ are all different variables, then $C(x_1, ..., x_n)$ denotes a context of n holes in which x_i is the ith occurring variable.

Then, we can give the definition on the congruence of an equivalence in a language.

Definition 2.4 Let $\mathcal{L} = (\Sigma, \Psi)$ be a language. An equivalence relation $\sim$ is congruent on language $\mathcal{L}$ iff $\forall i \in \{1, ..., n\} : p_i \sim q_i \Longrightarrow C(p_1, ..., p_n) \sim C(q_1, ..., q_n)$ for any context C of n holes in language $\mathcal{L}$, where p_i and q_i are closed terms, i.e., processes, over Σ.

Definition 2.5 Let Σ be a signature. A transition relation over Σ is a relation $\mathrm{Tr} \subseteq \mathrm{T}(\Sigma) \times Act \cup \{\tau\} \times \mathrm{T}(\Sigma)$. Element (p, α, p') of a transition relation is written as $p \xrightarrow{\alpha} p'$.

Thus a transition relation over Σ can be regarded as a set of closed positive Σ-literals(transitions).

Furthermore, for an action sequence $\delta = \alpha_1 ... \alpha_n$, if there exist $p_1, ..., p_n \in \mathrm{T}(\Sigma)$ such that $p \xrightarrow{\alpha_1} p_1 \xrightarrow{\alpha_2} ... \xrightarrow{\alpha_n} p_n$, then we call δ a trace of p, denoted as $p \xrightarrow{\delta}$ or $p \xrightarrow{\alpha_1} ... \xrightarrow{\alpha_n}$.

In weak semantics, the weak transition relations and the weak traces are also needed to be defined. Let p be a process, we write $p \overset{a}{\Longrightarrow}$ iff $p \xrightarrow{\tau^*} \xrightarrow{a} \xrightarrow{\tau^*}$, where τ^* denotes any number of internal transitions. Hence, for an observable action sequence $\delta = a_1 ... a_n$, $p \overset{\delta}{\Longrightarrow}$ iff $p \overset{a_1}{\Longrightarrow} ... \overset{a_n}{\Longrightarrow}$.

By imposing some syntactic constraints on TSS's, we will obtain the so-called rule formats with some properties on their induced operational semantics. Within these properties, it is specially important that whether a behavioral equivalence can be preserved in the languages with this format. Some rule formats have been proposed to meet the numerous behavioral equivalences, such as GSOS format, de Simone format, ntyft/nxyft format, etc. The readers are referred to Mousavi, Reniers and Groote (Groote, 2007) for a latest review on the rule formats.

The de Simone language will be employed as our starting point in retrieving the rule formats for the weak parametric failure equivalences.

Definition 2.6 (Simone, 1985) Let Σ be a signature. A transition rule r is in de Simone format if it has the form $\dfrac{\{x_i \xrightarrow{a_i} y_i\}_{i \in I}}{f(x_1, ..., x_{ar(f)}) \xrightarrow{a} t}$, where $I \subseteq \{1, ..., ar(f)\}$ and the variables x_i and y_i are all distinct and the only variables occuring in r. Moreover, the target $t \in \mathbb{T}(\Sigma)$ does not contain variable x_i for $i \in I$ and has no multiple occurrence of variables.

Below, two special classes of rules are defined. They will be discussed in the paper. The first class is the patience rules, and the second class is the rules with τ-conclusion.

Definition 2.7 (Aceto, 2001; Groote, 2007) Let $\mathcal{L} = (\Sigma, \Psi)$ be a de Simone language, and f be a function symbol in Σ. A rule of the form $\dfrac{x_i \xrightarrow{\tau} x_i'}{f(x_1, ..., x_i, ..., x_n) \xrightarrow{\tau} f(x_1, ..., x_i', ..., x_n)}$ with $1 \leq i \leq n$ is called a patience rule of the ith argument of f.

In the following, a rule is called a plain rule if it is not a patience rule.

Definition 2.8 (van Glabbeek, 2005) Let $\mathcal{L} = (\Sigma, \Psi)$ be a de Simone language, and f be a function symbol in Σ. An argument $i \in N$ of an operator f is active if f has a rule in which x_i appears as left-hand side of a premise. A variable x occurring in a term t is receiving in t if t is the target of a rule in which x is the right-hand side of a premise. An argument $i \in N$ of an operator f is receiving if a variable x is receiving in a term t that has a subterm $f(t_1, ..., t_n)$ with x occurring in t_i.

Then, the set of all arguments Arg of an operator can be divided into three classes: active arguments Arg_a, receiving arguments Arg_r and others Arg_o, which is inspired by van Glabbeek (van Glabbeek, 2005). Therefore, $Arg = Arg_a + Arg_r + Arg_o$.

Similarly, patience rules of an operator can be divided into three classes. It should be noted that an argument may be both an active argument and a receiving argument, i.e., $Arg_a \cap Arg_r \neq \phi$. However for clarity, from now on, if we say that an argument is a receiving argument, then it should not be an active argument , i.e., receiving arguments below are only those receiving arguments which are not active arguments simultaneously. Therefore, Arg_a, Arg_r and Arg_o will be disjoint.

Definition 2.9 Let $\mathcal{L} = (\Sigma, \Psi)$ be a de Simone language, and f be a function symbol in Σ. A rule of the form $\dfrac{H}{f(x_1, ..., x_n) \xrightarrow{\tau} t}$ is called a rule with τ-conclusion, if it is not a patience rule and there exists at least one positive Σ literal in H.

An notable example of the rules with τ-conclusion, which will be used in Section 4.2, is the first transition rule of the hiding operator in CSP as follows.

$$p/A : \frac{p \xrightarrow{\alpha} p'}{p/A \xrightarrow{\tau} p'/A}\ \alpha \in A \qquad \frac{p \xrightarrow{\alpha} p'}{p/A \xrightarrow{\alpha} p'/A}\ \alpha \notin A$$

Like the definition of a rule with τ-conclusion, a transition rule is a rule with τ-premise iff there exists a positive Σ literal like $t \xrightarrow{\tau} t'$ in its premises. It is trivial that patience rules are rules with τ-premise.

Before concluding this section, we will presume a small set of operators with default operational semantics: *nil* : means the successful termination.

$$a \cdot X : a \cdot X \xrightarrow{a} X$$

$$X \boxplus Y : \frac{X \xrightarrow{a} X'}{X \boxplus Y \xrightarrow{a} X'} \quad \frac{Y \xrightarrow{a} Y'}{X \boxplus Y \xrightarrow{a} Y'} \quad \frac{X \xrightarrow{\tau} X'}{X \boxplus Y \xrightarrow{\tau} X' \boxplus Y} \quad \frac{Y \xrightarrow{\tau} Y'}{X \boxplus Y \xrightarrow{\tau} X \boxplus Y'}$$

$$X \oplus Y : X \oplus Y \xrightarrow{\tau} X \qquad X \oplus Y \xrightarrow{\tau} Y$$

$$X \rhd Y : \frac{X \xrightarrow{a} X'}{X \rhd Y \xrightarrow{a} X'} \quad \frac{X \xrightarrow{\tau} X'}{X \rhd Y \xrightarrow{\tau} X' \rhd Y} \qquad X \rhd Y \xrightarrow{\tau} Y$$

where $a \in Act$. Operators $\boxplus, \oplus, \rhd$ are used to substitute the + operator and the prefixing with τ, because many weak equivalences may not be preserved under the + operator of CCS. We call this language **B** (Ulidowski, 2000).

Using these operators, $ap + bq$, $\tau ap + \tau bp$ and $ap + \tau bq$ can be represented by $ap \boxplus bq$, $ap \oplus bp$ and $ap \rhd bq$, respectively.

3 Weak Parametric Failure Equivalences

Before presenting the formal definitions of the weak parametric failure equivalences, two canonical equivalences, i.e., the weak failure equivalence and the weak impossible future equivalence, will be introduced. As we will see, they both are the special cases of the weak parametric failure equivalences.

Definition 3.1 $(\sigma, A) \in Act^* \times \mathcal{P}(Act)$ is a weak failure pair of process p iff there exists some p' such that $p \overset{\sigma}{\Longrightarrow} p' \wedge A \cap \mathcal{S}(p') = \emptyset$, where $\mathcal{S}(p') = \{a \in Act \mid p' \overset{a}{\Longrightarrow}\}$. The set of all weak failure pairs of process p is called the weak failure of p, denoted by $\mathcal{F}(p)$.

Weak Failure Equivalence $\sim_f$: for any two processes p and q, $p \sim_f q$ iff $\mathcal{F}(p) = \mathcal{F}(q)$.

Definition 3.2 $(\sigma, \Phi) \in Act^* \times \mathcal{P}(Act^*)$ is a weak impossible future pair of process p iff there exists some p' such that $p \overset{\sigma}{\Longrightarrow} p' \wedge \Phi \cap \mathcal{T}(p') = \emptyset$, where $\mathcal{T}(p') = \{\delta \in Act^* \mid p' \overset{\delta}{\Longrightarrow}\}$. The set of all weak impossible future pairs

of process p is called the weak impossible future of p, denoted by $\mathcal{IF}(p)$.

Weak Impossible Future Equivalence $\sim_{if}$: p and q are two processes, $p \sim_{if} q$ iff $\mathcal{IF}(p) = \mathcal{IF}(q)$.

As can be seen in the above definitions, the difference between the weak failure pair and the weak impossible future pair exists on their second parameters. The second parameter of the weak failure pair is a set of actions which cannot be enabled by p'. On the other hand, the second parameter of the weak impossible future pair is a set of action sequences which cannot be enabled by p'.

By the above observation, we put forward the definition on the weak parametric failure equivalences:

Definition 3.3 $(\sigma, \Phi) \in Act^* \times \mathcal{P}(Act^*)$ is a weak i-failure pair of process p iff there exists some p' such that $p \stackrel{\sigma}{\Longrightarrow} p' \wedge \Phi \cap \mathcal{T}(p', i) = \emptyset$, where $\mathcal{T}(p', i) = \{\delta \in Act^* \mid p' \stackrel{\delta}{\Longrightarrow} \wedge |\delta| \leqslant i\}$. The set of all weak i-failure pair of process p is called the weak i-failure of p, denoted by $\mathcal{F}(p, i)$.

Weak Parametric Failure Equivalences $\sim_f^i$: for any two processes p and q, $p \sim_f^i q$ iff $\mathcal{F}(p, i) = \mathcal{F}(q, i)$.

Also, we will often say that p and q are weak i-failure equivalent if $p \sim_f^i q$.

It is trivial that, in this framework, the weak failure equivalence is the weak 1-failure equivalence and the weak impossible future equivalence is the weak ω-failure equivalence. In fact, $\mathcal{S}(p') = \mathcal{T}(p', 1)$ and $\mathcal{T}(p') = \mathcal{T}(p', \omega)$.

The proposition below says that if p and q are weak j-failure equivalent with $1 \leq j \leq \omega$, then they are also weak i-failure equivalent for $i < j$.

Proposition 3.4 Let $1 \leq i < j \leq \omega$, p and q are two processes. If $p \sim_f^j q$ then $p \sim_f^i q$.

Proof By the definition of weak parametric failure equivalences, $p \sim_f^j q$ iff $\mathcal{F}(p, j) = \mathcal{F}(q, j)$. Then, $\mathcal{F}(p, i) = \mathcal{F}(q, i)$ can be obtained from Definition 3.3, $\mathcal{F}(p, j) = \mathcal{F}(q, j)$ and $1 \leq i < j \leq \omega$. Therefore, $p \sim_f^i q$. □

Before concluding this section, an alternative characterization of the weak parametric failure equivalences are to be presented. This alternative characterization will be useful in obtaining the rule formats.

Proposition 3.5 Let p, q be two processes. For $1 \leq i \leq \omega$, $p \sim_f^i q$ iff

1) for any $\sigma \in \mathcal{T}(p, \omega)$ and p' with $p \stackrel{\sigma}{\Longrightarrow} p'$, there exists q' such that $q \stackrel{\sigma}{\Longrightarrow} q'$ and $\mathcal{T}(q', i) \subseteq \mathcal{T}(p', i)$, and

2) for any $\sigma \in \mathcal{T}(q, \omega)$ and q' with $q \stackrel{\sigma}{\Longrightarrow} q'$, there exists p' such that $p \stackrel{\sigma}{\Longrightarrow} p'$ and $\mathcal{T}(p', i) \subseteq \mathcal{T}(q', i)$.

Proof ($\Longleftarrow$) It is enough to prove that $\mathcal{F}(p, i) = \mathcal{F}(q, i)$. If it is not true, then, without loss of generality, suppose that there exists some $(\sigma, \Phi) \in (Act^* \times \mathcal{P}(Act^*))$ such that $(\sigma, \Phi) \in \mathcal{F}(p, i)$ but $(\sigma, \Phi) \notin \mathcal{F}(q, i)$.

By $(\sigma, \Phi) \in \mathcal{F}(p, i)$ and the definition of weak i-failure pair in Definition 3.3, there must exists some p' such that $p \stackrel{\sigma}{\Longrightarrow} p' \wedge \Phi \cap \mathcal{T}(p', i) = \emptyset$.

By $p \stackrel{\sigma}{\Longrightarrow} p'$ and the hypothesis, there exists q' such that $q \stackrel{\sigma}{\Longrightarrow} q'$ and $\mathcal{T}(q', i) \subseteq \mathcal{T}(p', i)$. Then, from $\Phi \cap \mathcal{T}(p', i) = \emptyset$, we have $\Phi \cap \mathcal{T}(q', i) = \emptyset$.

Therefore, there exists q' such that $q \stackrel{\sigma}{\Longrightarrow} q'$ and $\Phi \cap \mathcal{T}(q', i) = \emptyset$, which contradicts with $(\sigma, \Phi) \notin \mathcal{F}(q, i)$.

($\Longrightarrow$) By the symmetry, we need only prove the first point. Suppose that σ is any trace in $\mathcal{T}(p, \omega)$ and p' is a process such that $p \stackrel{\sigma}{\Longrightarrow} p'$. Let $\Phi = I^i_{all} - \mathcal{T}(p', i)$ with I^i_{all} is the set of all action sequences whose lengthes are not exceed number i. Then, we have $(\sigma, \Phi) \in \mathcal{F}(p, i)$.

By Definition 3.3, $(\sigma, \Phi) \in \mathcal{F}(q, i)$. Hence, there exists some q' such that $q \stackrel{\sigma}{\Longrightarrow} q'$ and $\Phi \cap \mathcal{T}(q', i) = \emptyset$. By $\Phi = I^i_{all} - \mathcal{T}(p', i)$, we have $(I^i_{all} - \mathcal{T}(p', i)) \cap \mathcal{T}(q', i) = \emptyset$. Therefore, $\mathcal{T}(q', i) \subseteq \mathcal{T}(p', i)$. □

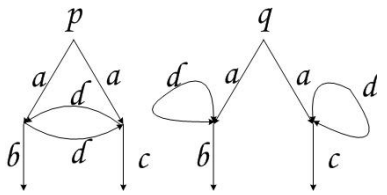

Figure 1: p and q are weak 1-failure equivalent, but $p/\{d\}$ and $q/\{d\}$ are not weak 1-failure equivalent.

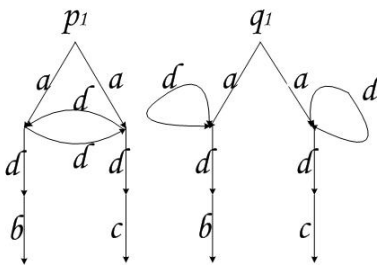

Figure 2: p_1 and q_1 are weak 2-failure equivalent, but $p_1/\{d\}$ and $q_1/\{d\}$ are not weak 2-failure equivalent.

4 Intuitive Motivations on Rule Formats

This section gives several representative examples to show some intuitive motivations on the rule formats of the weak parametric failure equivalences. However, we do not want to discuss them from the scratch, only the two critical points sketched in the introduction are to be mentioned: the first subsection is to observe the feasibility of adding rules with τ-conclusion; the second subsection is to inspect the necessity of the patience rules for receiving arguments.

It should be noted that, in this section, we mainly concern the intuitive motivations. The results retrieved in this section will be formally defined and proved in the next section. Also, as the starting point, we assume the basic language **B** which has been introduced in section 2.

4.1 On Rules with τ-conclusion

Let's see an example in Figure 1 and Figure 2. Firstly, the two graphs in Figure 1, i.e., p and q, are weak 1-failure equivalent. However, after hiding d actions, $p/\{d\}$ is not weak 1-failure equivalent to $q/\{d\}$, which can be seen from the weak 1-failure pair $(a, \{c\}) \in \mathcal{F}(q/\{d\}, 1)$ but $(a, \{c\}) \notin \mathcal{F}(p/\{d\}, 1)$. If we take the weak 2-failure equivalence into consideration, we may find that p and q are not yet weak 2-failure equivalent, because $(a, \{b, db\}) \in \mathcal{F}(q, 1)$ but $(a, \{b, db\}) \notin \mathcal{F}(p, 1)$.

As for the weak 2-failure equivalence, p_1 and q_1, the two graphs in Figure 2, are weak 2-failure equivalent. However, this equivalence also cannot be preserved after hiding d actions. In fact, for any weak i-failure equivalence with $i < \omega$, a similar counterexample exists. On the contrary, the weak ω-failure equivalence can be preserved under the hiding operator.

Generalizing to any rules with τ-conclusion, a common characterization of these rules is that they all consume the observable actions of the subprocesses and produce τ transitions at the same time. Therefore, we conjecture that any weak i-failure equivalence with $i < \omega$ will probably be broken under the rules with τ-conclusion, but the weak ω-failure equivalence will be preserved.

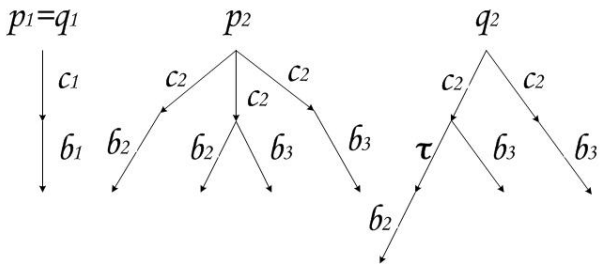

Figure 3: p_1 and q_1, p_2 and q_2 are weak i-failure equivalent for $i \in \mathbb{N} \cup \{\omega\}$.

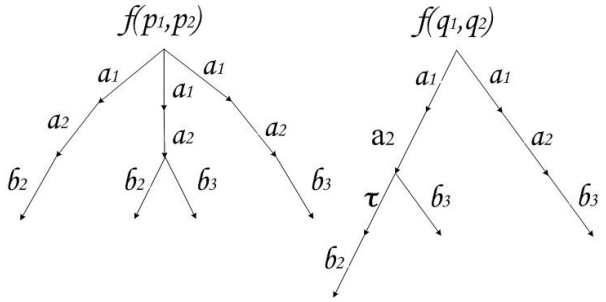

Figure 4: $f(p_1, p_2)$ and $f(q_1, q_2)$ are weak 1-failure equivalent but not weak 2-failure equivalent.

4.2 Patience Rules for Receiving Arguments

Similarly, see an example in Figure 3, Figure 4 and Figure 5. Consider adding the rules $r_1 = \frac{x_1 \xrightarrow{c_1} x_1', x_2 \xrightarrow{c_2} x_2'}{f(x_1, x_2) \xrightarrow{a_1} g(x_1', x_2')}$, $r_2 = \frac{x_1 \xrightarrow{b_1} x_1'}{g(x_1, x_2) \xrightarrow{a_2} h(x_2)}$, $r_3 = \frac{x_1 \xrightarrow{b_3} x_1'}{h(x_1) \xrightarrow{b_3} nil}$, $r_4 = \frac{x_1 \xrightarrow{b_2} x_1'}{h(x_1) \xrightarrow{b_2} nil}$ and their associated patience rules for active arguments into language **B**. Note that $g(x_1, x_2)$ has its second argument as a receiving argument.

Let p_1, p_2, q_1, q_2 be the processes shown in Figure 3. It can be easily verified that $p_1 \sim_f^1 q_1$ and $p_2 \sim_f^1 q_2$. According to the above rules, we have $f(p_1, p_2)$ and $f(q_1, q_2)$ shown in the two graphs of Figure 4. Now, $f(p_1, p_2)$ and $f(q_1, q_2)$ are also weak 1-failure equivalent, i.e., $f(p_1, p_2) \sim_f^1 f(q_1, q_2)$. Therefore, It seems that patience rules for receiving arguments are not prerequisite for the weak 1-failure equivalence.

When it turns to the weak 2-failure equivalence, we also have $p_1 \sim_f^2 q_1$ and $p_2 \sim_f^2 q_2$. However, $f(p_1, p_2)$ and $f(q_1, q_2)$ are not weak 2-failure equivalent yet, because $(a_1, \{a_2 b_3\})$ is a weak 2-failure pair of $f(p_1, p_2)$ but not a weak 2-failure pair of $f(q_1, q_2)$. Hence, the weak 2-failure equivalence is not preserved under the above rules. However, if we further add patience rule for the second argument of $g(x_1, x_2)$ into language **B**, then $f(p_1, p_2)$ and $f(q_1, q_2)$ is the two graphs in Figure 5. Now, it can be easily verified that $f(p_1, p_2) \sim_f^2 f(q_1, q_2)$. Therefore, adding the patience rules for receiving arguments may preserve the weak 2-failure equivalence.

In fact, we will assert, in the next section, that the patience rules for receiving arguments are prerequisite for the weak i-failure equivalence with $i > 1$, but, they are not necessary for the weak 1-failure equivalence.

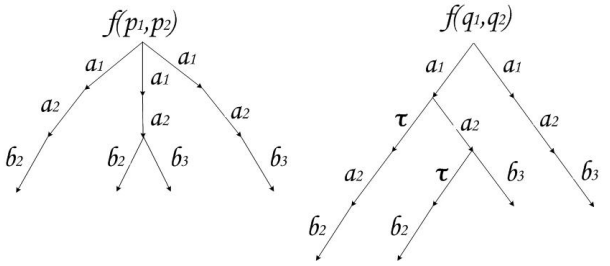

Figure 5: $f(p_1, q_1)$ and $f'(p_2, q_2)$ are weak 2-failure equivalent.

5 Rule Formats for Weak Parametric failure Equivalence

After the intuitive observations in the preceding section, we will, in this section, present formally the rule formats for the weak parametric failure equivalences. In fact, as stated in the introduction, we will present three different rule formats: weak 1-failure format for the weak 1-failure equivalence, weak finite failure format for the weak i-failure equivalence with $1 < i < \omega$ and weak ω-failure format for the weak ω-failure equivalence.

5.1 Formal Definitions on the Rule Formats

The de Simone language is employed as our starting point in retrieving the rule formats for the weak parametric failure equivalences.

Definition 5.1.1 A de Simone language $\mathcal{L}$ is in weak 1-failure format if

1) patience rules are the only rules with τ-premises,

2) patience rules for active arguments are prerequisite,

3) rules with τ-conclusion are not permitted.

Following it, the weak finite failure format is:

Definition 5.1.2 A de Simone language $\mathcal{L}$ is in weak finite failure format if

1) patience rules are the only rules with τ-premises,

2) patience rules for active arguments and receiving arguments are all prerequisite,

3) rules with τ-conclusion are not permitted.

Then, the weak ω-failure format for the weak ω-failure equivalence is to be presented.

Definition 5.1.3 A de Simone language $\mathcal{L}$ is in weak ω-failure format if

1) patience rules are the only rules with τ-premises, and

2) patience rules for active arguments and receiving arguments are all prerequisite.

It should be pointed out that the exclusion of rules with τ-conclusion and the allowance of dropping the patience rules for receiving arguments are not two separated restrictions. In fact, to obtain the effect of the allowance of dropping the patience rules for receiving arguments in the weak 1-failure format, the exclusion of rules with τ-conclusion is a precondition. We will prove this conclusion in Lemma 5.3.5.

Below, we will study the relations between the above three rule formats. To fulfil this purpose, we need to define the 'tighter than' relation between rule formats.

Definition 5.1.4 Let A and B be two rule formats. A is tighter than B iff, for any languages $\mathcal{L} = (\Sigma, \Psi)$ in format A, all transition rules in Ψ are also in format B. Moreover, A is strictly tighter than B iff A is tighter than B and there exists some languages $\mathcal{L} = (\Sigma, \Psi)$ in format B such that at least one of the transition rules in Ψ are not in format A.

Theorem 5.1.5 The weak 1-failure format is strictly tighter than the weak finite failure format and the weak finite failure format is strictly tighter than the weak ω-failure format.

Proof Comparing Definition 5.1.1 and Definition 5.1.2, patience rules for receiving arguments are not yet necessary for weak 1-failure format. Therefore, for any languages $\mathcal{L}$ in weak 1-failure format, its transition rules will also be in weak finite failure format. Likewise, Definition 5.1.2 and Definition 5.1.3 are only different on the rules with τ-conclusion. Therefore, after refusing all rules with τ-conclusion, any languages $\mathcal{L}$ in weak finite failure format will have its transition rules in the weak ω-failure format.

The strictness between weak 1-failure format and weak finite failure format can be witnessed by the languages in Section 4.2. introducing the patience rules for receiving arguments, it is a weak finite failure language. However, patience rules for receiving arguments are not in weak 1-failure format. The strictness between weak finite failure format and weak ω-failure format can be witnessed by introducing the hiding operator of CSP into any weak ω-failure language. The obtained languages are still weak ω-failure languages. However, one of transition rules of hiding operator is not in weak finite failure format. □

5.2 Ruloids And Ruloid Theorems On The Two Formats

Ruloids and the ruloid theorem originated from the works of Bloom (Bloom, 1995, 1990) for the GSOS format. In this section, we will introduce the ruloids and the ruloid theorem for the weak ω-failure format. And the ruloids and the ruloid theorem for the other two formats can be retrieved in the same way. The ruloid theorems will be useful for the proving the congruence theorems in the next three subsections.

For a language $\mathcal{L} = (\Sigma, \Psi)$ in the weak ω-failure format, the ruloids $\mathscr{R}(C, \alpha)$, for a context C of n holes and an action α, are a set of expressions like the transition rules:

$$\frac{\{x_i \xrightarrow{\alpha_i} x_i'\}_{i\in I}}{C(x_1, ..., x_n) \xrightarrow{\alpha} D(y_1, ..., y_n)} \quad (1)$$

such that $y_i = x_i'$ for $i \in I$ and $y_i = x_i$ for $i \notin I$, where $I \subseteq \{1, 2, ..., n\}$. These expressions characterize all possible behaviors of the context C in the language.

It should be noted that context D does not need to have exactly n holes. In fact, after leaving out the copying operation in the de Simone format (the weak ω-failure format is a subformat of the de Simone format), the number of the holes of D should be less than or equivalent to n. But for convenience, in form (1), we still write it as $D(y_1, ..., y_n)$.

Furthermore, two properties are needed to be imposed on $\mathscr{R}(C, \alpha)$, we call them soundness property and completeness property, by a little abusing the terminologies.

Definition 5.2.1 Let $\mathcal{L} = (\Sigma, \Psi)$ be a language in the weak ω-failure format, and $C(x_1, ..., x_n)$ be any context of n holes in $\mathcal{L}$. A set $\mathscr{R}(C, \alpha)$ of ruloids of form (1) are ruloids of context C and action α, with $\alpha \in Act \cup \{\tau\}$, iff

1) Soundness. Let $r \in \mathscr{R}(C, \alpha)$ be a ruloid of form (1). If ζ is a closed Σ substitution such that $\zeta(x_i) \xrightarrow{\alpha_i} \zeta(x_i')$ for all $i \in I$, then there must exist a context D such that $\zeta(C(x_1, ..., x_n)) \xrightarrow{\alpha} \zeta(D(y_1, ..., y_n))$.

2) Completeness. Let ζ be any closed Σ substitution. If $\zeta(C(x_1, ..., x_n)) \xrightarrow{\alpha}$, then there must exist a ruloid r of form (1) in ruloids $\mathscr{R}(C, \alpha)$, and $\zeta(x_i) \xrightarrow{\alpha_i}$ for all $i \in I$.

Below, we will present a strategy to retrieve the ruloids of context C and action α, and then prove that the obtained ruloids satisfy the above two properties, which form the ruloid theorem.

Strategy 5.2.2 Let $\mathcal{L} = (\Sigma, \Psi)$ be a language in the weak ω-failure format. $C(x_1, ..., x_n)$ is any context of n holes in $\mathcal{L}$ and $\alpha \in Act \cup \{\tau\}$ is an action.

1) If $C \in V$, i.e., C is a variable, then $\mathscr{R}(C, \alpha) = \{\frac{x \xrightarrow{\alpha} x'}{x \xrightarrow{\alpha} x'}\}$;

2) If $C = f(x_1, ..., x_n)$ with $f \in \Sigma$ and $ar(f) = n$, then $\mathscr{R}(C, \alpha) = (f, \alpha)$, where (f, α) denotes the set of all rules in Ψ whose source is $f(x_1, ..., x_n)$ and output is α.

3) If C is any context. We can rewrite $C(x_1, ..., x_n)$ as $f(C_1[X_1], ..., C_m[X_m])$, where $f \in \Sigma$ and $ar(f) = m$. Note that $X_i \cap X_j = \emptyset$ with $1 \le i, j \le m$ and $i \ne j$. Without loss of generality, we may suppose that $X_i = x_{i1}x_{i2}...x_{im_i}$ for C_i is a context of m_i holes. Now, let r be any ruloid of form (1) in (f, α) and $\mathscr{R}(C_i, \alpha_i)$ be ruloids of context C_i and action α_i retrieved by induction on this strategy. Then, any ruloids in $\mathscr{R}(C, \alpha)$ can be obtained by the following steps:

i) pick out randomly from $\mathscr{R}(C_i, \alpha_i)$ a rule r_i, for all $i \in I$;

ii) substitute the variables x_j in r_i with x_{ij}, for all $1 \le j \le m_i$;

iii) substitute $x_i \xrightarrow{\alpha_i} x_i'$ in the premise of r with $ante(r_i)$, for all $i \in I$.

4) $\mathscr{R}(C, \alpha)$ is the set of all possible ruloids that can be retrieved from step 3). □

Theorem 5.2.3 Let $\mathcal{L} = (\Sigma, \Psi)$ be a language in the weak ω-failure format, and $C(x_1, ..., x_n)$ be any context of n holes in $\mathcal{L}$. The set of ruloids $\mathscr{R}(C, \alpha)$ obtained from the Strategy 5.2.2 are ruloids of context C and action α with $\alpha \in Act \cup \{\tau\}$.

Proof First, the obtained ruloids $\mathscr{R}(C, \alpha)$ of context C and action α are all in form (1), which can be easily retrieved from the construction procedure in the Strategy 5.2.2.

Second, the obtained ruloids $\mathscr{R}(C, \alpha)$ of context C and action α satisfy the soundness property. Let $r \in \mathscr{R}(C, \alpha)$ be a ruloid of form (1), where C is a context of n holes and $\alpha \in Act \cup \{\tau\}$ is an action. ζ is a closed Σ substitution such that $\zeta(x_i) \xrightarrow{\alpha_i} \zeta(x_i')$ for all $i \in I$. Make an induction on the context C.

i) if $C \in V$, then, without loss of generality, suppose $C = x$. The soundness property is trivial from $\mathscr{R}(C, \alpha) = \{\frac{x \xrightarrow{\alpha} x'}{x \xrightarrow{\alpha} x'}\}$;

ii) if $C = f(x_1, ..., x_n)$, then $\mathscr{R}(C, \alpha) = (f, \alpha)$. Therefore, the soundness property is guaranteed by the transition rules;

iii) if C is any context of n holes, then, from the strategy, there exist contexts $C_1[X_1], ..., C_m[X_m]$ such that, after substitution, $ante(r_i)$ is a part of the premise of r for $1 \le i \le m$, where $r_i \in \mathscr{R}(C_i, \alpha_i)$. Now, by the hypothesis, ζ is a closed Σ substitution making all premises of r enable. Hence, $cons(r_i)$ is enabled, which means that $C_i[X_i] \xrightarrow{\alpha_i} D_i[Y_i]$ for all $1 \le i \le m$. Furthermore, C is rewritten as $f(C_1[X_1], ..., C_m[X_m])$. Therefore, the transition rules in (f, α) guarantee the enableness of $C(x_1, ..., x_n) \xrightarrow{\alpha}$.

Third, the obtained ruloids $\mathscr{R}(C, \alpha)$ of context C and action α satisfy the completeness property, which can also be easily retrieved from the construction procedure of the Strategy 5.2.2. □

As we can see that, for a ruloid of form (1), its premises need not include all x_i for $1 \le i \le n$. However, we can add $x_i \xrightarrow{\epsilon} x_i'$, for $i \in \{1, ..., n\} \backslash I$, into the premises, as in the form (2). And form (1) and form (2) are equivalent when any closed Σ substitution ζ is applied on them. In this case, $\zeta(x_i) \xrightarrow{\epsilon} \zeta(x_i')$ denotes that subprocess $\zeta(x_i)$ executes no transition.

$$\frac{\{x_i \xrightarrow{\alpha_i} x_i'\}_{i\in I}\{x_i \xrightarrow{\epsilon} x_i'\}_{i\in\{1,...,n\}\backslash I}}{C(x_1, ..., x_n) \xrightarrow{\alpha} D(y_1, ..., y_n)} \quad (2)$$

Like the definitions on the transition rules, we can also define the patience ruloids and ruloids with τ-conclusion.

Theorem 5.2.3 Let $\mathcal{L} = (\Sigma, \Psi)$ be a language in the weak ω-failure format, and $C(x_1, ..., x_n)$ be any context of n holes in $\mathcal{L}$. The set of ruloids $\mathscr{R}(C, \alpha)$ obtained from the Strategy 5.2.2 is the ruloids of context C and action α satisfies $\alpha \in Act \cup \{\tau\}$.

Proof Firstly, the obtained ruloids $\mathscr{R}(C, \alpha)$ of context C and action α are all in form (1), which can be easily retrieved from the construction procedure in the Strategy 5.2.2.

Secondly, the obtained ruloids $\mathscr{R}(C, \alpha)$ of context C and action α satisfy the soundness property. Let $r \in \mathscr{R}(C, \alpha)$ be a ruloid of form (1), where C is a context of n holes and $\alpha \in Act \cup \{\tau\}$ is an action. ζ is a closed Σ substitution such that $\zeta(x_i) \xrightarrow{\alpha_i} \zeta(x_i')$ for all $i \in I$.

i) if $C \in V$, then, without loss of generality, suppose $C = x$. The soundness property is trivial from $\mathscr{R}(C, \alpha) = \{\frac{x \xrightarrow{\alpha} x'}{x \xrightarrow{\alpha} x'}\}$;

ii) if $C = f(x_1, ..., x_n)$, then $\mathscr{R}(C, \alpha) = (f, \alpha)$. Therefore, the soundness property is guaranteed by the transition rules;

iii) if C is any context of n holes, then by Strategy 5.2.2, $C(x_1, ..., x_n)$ can be rewritten as $f(C_1(X_1), ..., C_m(X_m))$ for some operator $f \in \Sigma$ and $ar(f) = m$, and $ante(r)$ consist of $anti(r_1), ..., ante(r_m)$, where $r_i \in \mathscr{R}(C_i, \alpha_i)$ for all $1 \le i \le m$. By the assumption of the soundness property that $ante(r)$ is enabled in closed Σ substitution ζ. Therefore, by the induction hypothesis, $cons(r_1), ..., cons(r_m)$ are all enabled in ζ. This means that $\zeta(C_i(X_i)) \xrightarrow{\alpha_i} \zeta(D_i(Y_i))$ for all $1 \le i \le m$. In fact, $cons(r_1), ..., cons(r_m)$ constitute $ante(f)$. Still by the induction hypothesis on operator f, the transition rules in (f, α) guarantee the enableness of $C(x_1, ..., x_n) \xrightarrow{\alpha}$.

Last, the obtained ruloids $\mathscr{R}(C, \alpha)$ of context C and action α satisfy the completeness property, which can also be easily retrieved from the construction procedure of the Strategy 5.2.2. □

As we can see that, for a ruloid of form (1), its premises need not include all x_i for $1 \le i \le n$. However, we can add $x_i \xrightarrow{\epsilon} x_i'$, for $i \in \{1, ..., n\} \backslash I$, into the premises, as in the form (2). In this case, $\zeta(x_i) \xrightarrow{\epsilon} \zeta(x_i')$ denotes that subprocess $\zeta(x_i)$ executes no transition.

$$\frac{\{x_i \xrightarrow{\alpha_i} x_i'\}_{i \in I} \{x_i \xrightarrow{\epsilon} x_i'\}_{i \in \{1,...,n\} - I}}{C(x_1, ..., x_n) \xrightarrow{\alpha} D(y_1, ..., y_n)} \quad (3)$$

The ϵ transition will not be added to the TSS. In fact, a TSS is a pair (Σ, Ψ), where Σ is a set of function symbols and Ψ is a set of transition rules assigned to the function symbols. Therefore, even no ruloids are in the TSS.

The introducing of ϵ transition is to substitute the ruloids of form (1) with the ruloids of form (2), since these two forms are equivalent when any closed Σ substitution ζ is applied. In fact, we want to express a viewpoint that, for any ruloid r, it should have two different but equivalent forms, i.e., form (1) and form (2).

For the equivalence between form (1) and form (2), we want to take an example to show it. Let $x_j \xrightarrow{\epsilon} x_j'$ be any ϵ-premise in some ruloid r. In fact, it denotes that, when ruloid r is applied in some Σ substitution ζ, subprocess $\zeta(x_j)$ is not fired at all. Also, if the form (1) of r is applied, the same results are retrieved.

The introducing of ϵ transitions and thus form (2) will make Lemma 5.3.1 and its proof prone to be comprehended. In Lemma 5.3.1, we will see that, in the weak ω-failure languages, when process $C(p_1, ..., p_n)$ evolves into $C'(p_1, ..., p_n')$ by applying a ruloid and produce a transition (observable action or τ transition), each subprocess p_i will also evolve into p_i' and produce a transition (observable action, τ transition or ϵ transition).

Based on the ruloids and the ruloid theorem, we may restate several classes of rules, which have been defined previously, with the notion of ruloids. And, they will be more intuitive and prone to be used in the following.

The first class of rules which we concern is the patience rules. As their counterparts, the definition of patience ruloids is as follows.

Definition 5.2.4 A ruloid of the form $\frac{x_i \xrightarrow{\tau} x_i'}{C(x_1, ..., x_i, ..., x_n) \xrightarrow{\tau} C(x_1, ..., x_i', ..., x_n)}$ with $1 \le i \le n$ is called a patience ruloid of the ith argument of the context C.

In the following, a ruloid is called a plain ruloid if it is not a patience ruloid. Similar to the division in the patience rules, we also need to divide the patience ruloids into three classes, i.e., patience ruloids for active arguments, patience ruloids for receiving arguments and patience ruloids for other arguments.

In fact, Strategy 5.2.2 has already provided a canonical way to retrieve this division. Let $\mathcal{L}$ be a de Simone language and C be any context of n holes in it.

1) If only adding the patience rules for active arguments into the language, then, after using Strategy 5.2.2, the patience ruloids in $\mathscr{R}(C, \tau)$ are patience ruloids for active arguments.

2) If further adding the patience rules for receiving arguments into the language, then, after using Strategy 5.2.2, the patience ruloids in $\mathscr{R}(C, \tau)$ are patience ruloids for active arguments and receiving arguments. Therefore, getting rid of the patience ruloids for active arguments, we can obtain the patience ruloids for receiving arguments.

This division is obtained indirectly from Strategy 5.2.2 and patience rules, and thus it is hard to be used in the following. Here, we will propose another division which is directly based on the arguments of a context.

Definition 5.2.5 Let $\mathcal{L} = (\Sigma, \Psi)$ be a weak ω-failure language, and C be any context of n holes. The ith argument of the context C is active if there exists a plain ruloid r of form (1) in $\mathscr{R}(C, \tau)$ such that x_i appears as left-hand side of a premise. The ith argument of the context C is receiving if it is not active and there exist another context D and a plain ruloid r of form (1) in $\mathscr{R}(D)$ such that $C(x_1', ..., x_n')$ appears as the target of r and x_i' appears as right-hand side of a premise.

Below, we will prove that these two divisions are indeed equivalent, i.e., a patience ruloid of some context C is a patience ruloid for active (resp. receiving, other) argument obtained from Strategy 5.2.2 and patience rules iff it is a patience ruloid for active (resp. receiving, other) argument defined by Definition 5.2.5.

Proposition 5.2.6 The division defined by Definition 5.2.5 is equivalent to the division obtained from Strategy 5.2.2 and patience rules.

Proof ($\Longleftarrow$) Let $\mathcal{L} = (\Sigma, \Psi)$ be a de Simone language, and C be any context of n holes.

If only adding the patience rules for active arguments into the language, we need to prove that each active argument of the context C defined by Definition 5.2.5 has a patience ruloid. We will prove by making an induction on the context C and Strategy 5.2.2.

1) If $C \in V$ or $C \in \Sigma$, then it can be easily obtained from Strategy 5.2.2 and Definition 2.8.

2) If C is any context, then it can be rewritten as $f(C_1(X_1), ..., C_m(X_m))$. Assume that contexts $C_1, ..., C_m$ satisfy that each active argument has a patience ruloid.

3) We need to prove that each active argument of C defined by Definition 5.2.5 has a patience ruloid. Suppose that the ith argument of C is an active argument. Then, by Definition 5.2.5, there exists a plain ruloid r of form (1) in $\mathscr{R}(C, \tau)$ such that x_i appears as left-hand side of a premise. By Strategy 5.2.2, x_i must appear as left-hand

side of a premise of some context. Without loss of generality, assume that x_i is the kth argument of the C_j. By the induction hypothesis, the kth argument of C_j is active and thus has a patience ruloid. Also by Strategy 5.2.2, the jth argument of functor f is active and thus has a patience ruloid. Therefore, we have that the ith argument of C has a patience ruloid by Strategy 5.2.2 and the above two patience ruloids for C_j and f, respectively.

If further adding the patience rules for receiving arguments into the language, we need to prove that each receiving argument of the context C defined by Definition 5.2.5 has a patience ruloid. Assume that the ith argument of context C is receiving. Then, by Definition 5.2.5, there exist another context D and a plain ruloid r of form (1) in $\mathscr{R}(D)$ such that $C(x'_1, ..., x'_n)$ appears as the target of r and x'_i appears as right-hand side of a premise. We will prove by making an induction on context C and Strategy 5.2.2.

1) If $C \in V$ or $C \in \Sigma$, then, by Definition 2.8, the ith argument of C is receiving. Therefore, it should have a patience rule by the hypothesis. By Strategy 5.2.2, each patience rule is also a patience ruloid.

2) If C is any context, then it can be rewritten as $f(C_1(X_1), ..., C_m(X_m))$. Assume that contexts $C_1, ..., C_m$ satisfy that each receiving argument has a patience ruloid.

3) We need to prove that the ith argument of C defined by Definition 5.2.5 has a patience ruloid. By Strategy 5.2.2, x_i must appear as right-hand side of a premise of some context. Without loss of generality, assume that x_i is the kth argument of the C_j. By the induction hypothesis, the kth argument of C_j is receiving or active and thus has a patience ruloid. Also by Strategy 5.2.2, the jth argument of functor f is receiving or active and thus has a patience ruloid. Therefore, we have that the ith argument of C has a patience ruloid by Strategy 5.2.2 and the above two patience ruloids for C_j and f, respectively.

($\Longrightarrow$) It is trivially true since, according to Strategy 5.2.2, each rule is also a ruloid. That is to say, we may first obtain the division on patience rules from the division on patience ruloids in Definition 5.2.5, and then using Strategy 5.2.2 to obtain the division from Strategy 5.2.2 and patience rules. □

The second class of rules is the rules with τ conclusion. Likewise, we may define the ruloids with τ conclusion as their counterparts.

Definition 5.2.7 A ruloid of the form $\frac{H}{C(x_1, ..., x_n) \xrightarrow{\tau} D(y_1, ..., y_n)}$ is called a ruloid with τ-conclusion, if it is not a patience ruloid and there exists at least one positive Σ literal in H.

Also, we want to show that the exclusion of rules with τ-conclusion is equivalent to the exclusion of ruloids with τ-conclusion.

Proposition 5.2.8 Let $\mathcal{L}$ be a weak ω-failure language, and $C(x_1, ..., x_n)$ be any context of n holes. If no rule with τ-conclusion is allowed, then no ruloid with τ-conclusion can be in $\mathscr{R}(C, \tau)$, and vice versa.

Proof This can be easily obtained from a fact that, by Strategy 5.2.2, the output of any ruloid is in fact the output of some rule. □

5.3 Weak 1-Failure Format for Weak 1-Failure Equivalence

In this subsection, several necessary lemmas are to be presented and the usage of them to prove the congruence theorems in the following three subsections has been listed in Table 1. The symbol $\surd$ in the table denotes that some lemma is to be used in the proof of the congruence theorem for the corresponding format. For example, the congruence theorem for the weak ω-failure format needs the first three lemmas, i.e., Lemma 5.3.1, Lemma 5.3.2 and Lemma 5.3.3.

The following lemma states that, in the weak ω-failure languages, any weak trace of a composite process may be decomposed into weak traces of its subprocesses. Besides, this lemma also holds in weak finite failure languages and weak 1-failure languages.

Table 1: The Usage of Lemmas in Section 5.3 in Proving the Congruence Theorems

format\ Lemma	5.3.1	5.3.2	5.3.3	5.3.4	5.3.5
1-failure format	$\surd$	$\surd$		$\surd$	$\surd$
finite failure format	$\surd$	$\surd$	$\surd$	$\surd$	
ω-failure format	$\surd$	$\surd$	$\surd$		

Lemma 5.3.1 Let $\mathcal{L} = (\Sigma, \Psi)$ be a weak ω-failure language, and $C(x_1, ..., x_n)$ be any context of n holes. Suppose that ζ is any closed Σ substitution mapping x_i into p_i. If σ is a trace in $\mathcal{T}(C(p_1, ..., p_n), \omega)$, then, for all $1 \leq i \leq n$, there is a trace σ_i in $\mathcal{T}(p_i, \omega)$ such that, when $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p'_1, ..., p'_n)$, we have $p_i \stackrel{\sigma_i}{\Longrightarrow} p'_i$.

Proof Since $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p'_1, ..., p'_n)$, we have $C(p_1, ..., p_n) = C_0(p_{10}, ..., p_{n0}) \xrightarrow{\alpha_1} C_1(p_{11}, ..., p_{n1}) \xrightarrow{\alpha_2} ... \xrightarrow{\alpha_m} C_m(p_{1m}, ..., p_{nm}) = C'(p'_1, ..., p'_n)$, where $\forall 1 \leq j \leq m : \alpha_j \in Act \cup \{\tau\}$ and $\sigma' = \alpha_1 ... \alpha_m$ is equivalent to σ if all its τ transitions are omitted.

We will prove this lemma by making an induction on the length of σ'.

1) $|\sigma'| = 1$. Let $\sigma' = \alpha$. By the completeness property of the ruloids, there should be a ruloid of form (1) in $\mathscr{R}(C, \alpha)$, and $p_i \xrightarrow{\alpha_i}$ for all $i \in I$. As is shown before that, we have a ruloid of form (2) corresponding with form (1). Therefore, there exist $p_i \xrightarrow{\alpha_i} p'_i$ for all $i \in I$ and $p_i \xrightarrow{\epsilon} p'_i$ for all $i \in \{1, ..., n\} - I$, i.e., when $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p'_1, ..., p'_n)$, we have $p_i \stackrel{\alpha_i}{\Longrightarrow} p'_i$ for all $i \in I$ and $p_i \xrightarrow{\tau^*} p'_i$ for all $i \in \{1, ..., n\} - I$.

2) Assume that, when $|\sigma'| = m - 1$ with $m \geq 1$, if σ is a trace in $\mathcal{T}(C(p_1, ..., p_n), \omega)$ then, for all $1 \leq i \leq n$, there should be a trace σ_i in $\mathcal{T}(p_i, \omega)$ such that, when $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p'_1, ..., p'_n)$, we have $p_i \stackrel{\sigma_i}{\Longrightarrow} p'_i$.

3) For $|\sigma'| = m$, suppose that $C(p_1, ..., p_n) = C_0(p_{10}, ..., p_{n0}) \xrightarrow{\alpha_1} C_1(p_{11}, ..., p_{n1}) \xrightarrow{\alpha_2} ... \xrightarrow{\alpha_m} C_m(p_{1m}, ..., p_{nm}) = C'(p'_1, ..., p'_n)$. By the induction hypothesis, when $C(p_1, ..., p_n) = C_0(p_{10}, ..., p_{n0}) \xrightarrow{\alpha_1} C_1(p_{11}, ..., p_{n1}) \xrightarrow{\alpha_2} ... \xrightarrow{\alpha_{m-1}} C_{m-1}(p_{1(m-1)}, ..., p_{n(m-1)}) = C''(p''_1, ..., p''_n)$, we have $p_i \stackrel{\sigma''_i}{\Longrightarrow} p''_i$. Now, when $C_{m-1}(p_{1(m-1)}, ..., p_{n(m-1)}) = C''(p''_1, ..., p''_n) \xrightarrow{\alpha_m} C_m(p_{1m}, ..., p_{nm}) = C'(p'_1, ..., p'_n)$, we have $p''_i \stackrel{\alpha'_m}{\Longrightarrow} p'_i$. Therefore, when $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p'_1, ..., p'_n)$, we have $p_i \stackrel{\sigma''_i \alpha'_m}{\Longrightarrow} p'_i$. □

The following lemma states that, in the weak ω-failure languages, the weak trace equivalence will be preserved and the composite processes can reach the same contexts after same weak traces. The definition of weak trace equivalence is that: two processes p and q are weak trace equivalent, denoted as $p \sim_t q$, iff they have the same set of weak traces, i.e., $p \sim_t q$ iff $\mathcal{T}(p, \omega) = \mathcal{T}(q, \omega)$. This lemma also holds in weak finite failure languages and weak 1-failure languages.

Before that, we need one more definition on delay processes. Suppose that $\sigma \in \mathcal{T}(p, \omega)$ for some process p, then delay processes of $p \stackrel{\sigma}{\Longrightarrow}$ are those satisfying that 1) if $|\sigma| = 0$, then p itself is the delay process, and 2) if

$|\sigma| \geq 1$, then let $\sigma = \sigma' a$ and delay processes are those processes p' such that $p \stackrel{\sigma}{\Longrightarrow} \stackrel{a}{\longrightarrow} p'$.

Lemma 5.3.2 Let $\mathcal{L} = (\Sigma, \Psi)$ be a weak ω-failure language, and $C(x_1, ..., x_n)$ be any context of n holes. Suppose that ζ and ξ are any two closed Σ substitution mapping x_i into p_i and q_i, respectively. If for all $1 \leq i \leq n$, $p_i \sim_t q_i$, then

1) for any trace $\sigma \in \mathcal{T}(C(p_1, ..., p_n), \omega)$ and some context C' such that $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p'_1, ..., p'_n)$, there exist $q'_1, ..., q'_n$ such that $C(q_1, ..., q_n) \stackrel{\sigma}{\Longrightarrow} C'(q'_1, ..., q'_n)$, and

2) if there exists a patience ruloid for the ith argument of context C' then q'_i can be any process such that $q_i \stackrel{\sigma_i}{\Longrightarrow} q'_i$, and if there does not exist a patience ruloid for the ith argument of context C' then q'_i can be any delay processes of $q_i \stackrel{\sigma_i}{\Longrightarrow}$, where σ_i is obtained by decomposing σ into the weak traces of subprocess p_i.

Proof Suppose that σ is a trace of $C(p_1, ..., p_n)$, and $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p'_1, ..., p'_n)$. By Lemma 5.3.1, when $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p'_1, ..., p'_n)$, we have $p_i \stackrel{\sigma_i}{\Longrightarrow} p'_i$ for all $1 \leq i \leq n$. Then, by $p_i \sim_t q_i$, we have $q_i \stackrel{\sigma_i}{\Longrightarrow}$ for all $1 \leq i \leq n$.

For $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p'_1, ..., p'_n)$, we have $C(p_1, ..., p_n) = C_0(p_{10}, ..., p_{n0}) \stackrel{\alpha_1}{\longrightarrow} C_1(p_{11}, ..., p_{n1}) \stackrel{\alpha_2}{\longrightarrow} ... \stackrel{\alpha_m}{\longrightarrow} C_m(p_{1m}, ..., p_{nm}) = C'(p'_1, ..., p'_n)$, where $\forall 1 \leq j \leq m : \alpha_j \in Act \cup \{\tau\}$ and $\sigma' = \alpha_1 ... \alpha_m$ is equivalent to σ if all its τ transitions are omitted.

Suppose that the sequence of plain ruloids applied in the above procedure is $r_1 r_2 ... r_k$. It is enough to show that $C(q_1, ..., q_n)$ can also apply ruloids $r_1 r_2 ... r_k$ in the same order, and $C(q_1, ..., q_n) \stackrel{\sigma}{\Longrightarrow} C'(q'_1, ..., q'_n)$ for some $q'_1, ..., q'_n$. Furthermore, if there exists a patience ruloid for the ith argument of context C' then q'_i can be any process such that $q_i \stackrel{\sigma_i}{\Longrightarrow} q'_i$, and if there does not exist a patience ruloid for the ith argument of context C' then q'_i can be any delay processes of $q_i \stackrel{\sigma_i}{\Longrightarrow}$.

We will prove it by making an induction on k.

1) $k = 0$. Then, $C = C'$ and only patience ruloids are applied when $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p'_1, ..., p'_n)$ and thus $\sigma = \tau^*$. By Lemma 5.3.1, $\sigma_i = \tau^*$ for all $1 \leq i \leq n$. Therefore, there must exist $q'_i, ..., q'_n$ such that $C(q_1, ..., q_n) \stackrel{\tau^*}{\Longrightarrow} C'(q'_1, ..., q'_n)$ since an extreme possibility is that $q_i \equiv q'_i$ for all $1 \leq i \leq n$. Now, if there exists a patience ruloid for the ith argument of context C' then q'_i can be any process such that $q_i \stackrel{\tau^*}{\longrightarrow} q'_i$ by the soundness property of ruloids and the definition of patience ruloids. On the other hand, if there does not exist a patience ruloid for the ith argument of context C' then q'_i can be q_i.

2) Assume that, when $k = m - 1$ with $m \geq 1$, the above statement holds.

3) For $k = m$, suppose that, $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C''(p''_1, ..., p''_n)$ and $C''(p''_1, ..., p''_n) \stackrel{\delta}{\Longrightarrow} C'(p'_1, ..., p'_n)$, where the first $k - 1$ plain ruloids of $r_1 r_2 ... r_k$ are applied when $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C''(p''_1, ..., p''_n)$ and the kth plain ruloid are applied when $C''(p''_1, ..., p''_n) \stackrel{\delta}{\Longrightarrow} C'(p'_1, ..., p'_n)$.

By Lemma 5.3.1, there exist σ_i, δ_i for all $1 \leq i \leq n$ such that $p_i \stackrel{\sigma_i}{\Longrightarrow} p''_i$ and $p''_i \stackrel{\delta_i}{\Longrightarrow} p'_i$. By $p_i \sim_t q_i$, we have $q_i \stackrel{\sigma_i \delta_i}{\Longrightarrow}$.

Then, by the induction hypothesis, $C(q_1, ..., q_n)$ can also apply the first $k-1$ ruloids and reaches $C''(q''_1, ..., q''_n)$. Moreover, if there exists a patience ruloid for the ith argument of context C'' then q''_i can be any process such that $q_i \stackrel{\sigma_i}{\Longrightarrow} q''_i$, and if there does not exist a patience ruloid for the ith argument of context C'' then q''_i can be any delay process of $q_i \stackrel{\sigma_i}{\Longrightarrow}$.

Furthermore, for all $1 \leq i \leq n$, let q''_i be any delay process of $q_i \stackrel{\sigma_i}{\Longrightarrow}$ such that $q_i \stackrel{\sigma_i}{\Longrightarrow} q''_i \stackrel{\delta_i}{\Longrightarrow}$. There always exists a such q''_i since $q_i \stackrel{\sigma_i \delta_i}{\Longrightarrow}$.

Suppose that the kth ruloid r_k is in form (1). Then, by the definition of the weak ω-failure format, all arguments in I have corresponding patience ruloids since they are all active arguments of C'' by Definition 5.2.5. Therefore, by the soundness property of the ruloids, we may apply the patience ruloids for the arguments in I and obtain $C''(q''_1, ..., q''_n) \stackrel{\tau^*}{\Longrightarrow} C''(q'''_1, ..., q'''_n)$, such that $q'''_i \equiv q''_i$ if $i \notin I$ and $q'''_i \stackrel{\delta_i}{\longrightarrow}$ if $i \in I$. Then, also by the soundness property of the ruloids, ruloid r_k will be applied and $C''(q'''_1, ..., q'''_n) \stackrel{\delta}{\longrightarrow} C'(q''''_1, ..., q''''_n)$, where $q''''_i \equiv q''_i$ if $i \notin I$ and q''''_i is any process satisfying $q'''_i \stackrel{\delta_i}{\longrightarrow} q''''_i$ if $i \in I$.

Now, we can see that q''''_i is indeed a delay process of $q_i \stackrel{\sigma_i \delta_i}{\Longrightarrow}$.

Finally, if there exists a patience ruloid for the ith argument of context C' then q''''_i may evolve into any process q'_i such that $q''''_i \stackrel{\tau^*}{\Longrightarrow} q'_i$ and thus q'_i may be any process such that $q_i \stackrel{\sigma_i \delta_i}{\Longrightarrow} q'_i$. On the other hand, if there does not exist a patience ruloid for the ith argument of context C' then let q'_i be q''''_i, and thus q'_i is any delay process of $q_i \stackrel{\sigma_i \delta_i}{\Longrightarrow}$. □

As a strengthened results of the above lemma, Lemma 5.3.3 below will show that, in weak finite failure languages and weak ω-failure languages, if the ith argument is neither an active argument nor a receiving argument, i.e., is an other argument, then q'_i can be q_i or any process such that $q_i \stackrel{\sigma_i}{\Longrightarrow} q'_i$. Though we only prove this lemma in weak ω-failure languages, it also holds in weak finite failure languages.

Lemma 5.3.3 Let $\mathcal{L} = (\Sigma, \Psi)$ be a weak ω-failure language, and $C(x_1, ..., x_n)$ be any context of n holes. Suppose that ζ and ξ are any two closed Σ substitution mapping x_i into p_i and q_i, respectively. For all $1 \leq i \leq n$, $p_i \sim_t q_i$, and thus for any trace $\sigma \in \mathcal{T}(C(p_1, ..., p_n), \omega)$ and some context C' with $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p'_1, ..., p'_n)$, there exist $q'_1, ..., q'_n$ such that $C(q_1, ..., q_n) \stackrel{\sigma}{\Longrightarrow} C'(q'_1, ..., q'_n)$. Now, if the ith argument of C' is an other argument, then q'_i can be q_i or any process such that $q_i \stackrel{\sigma_i}{\Longrightarrow} q'_i$.

Proof Like the proof of Lemma 5.3.2, suppose that the sequence of plain ruloids applied in the procedure $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p'_1, ..., p'_n)$ is $r_1 r_2 ... r_k$.

We will prove it by making an induction on k.

1) $k = 0$. Then, $C = C'$ and only patience ruloids are applied when $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p'_1, ..., p'_n)$ and thus $\sigma = \tau^*$. In this case, we can let q'_i be q_i if the ith argument of C' is an other argument.

2) Assume that, when $k = m-1$ with $m \geq 1$, the lemma holds.

3) For $k = m$, suppose that, $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C''(p''_1, ..., p''_n)$ and $C''(p''_1, ..., p''_n) \stackrel{\delta}{\Longrightarrow} C'(p'_1, ..., p'_n)$, where the first $k - 1$ plain ruloids of $r_1 r_2 ... r_k$ are applied when $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C''(p''_1, ..., p''_n)$ and the kth plain ruloid are applied when $C''(p''_1, ..., p''_n) \stackrel{\delta}{\Longrightarrow} C'(p'_1, ..., p'_n)$.

However, observe that the ith argument of C'' cannot be in set I of ruloid r_k. Or else, the ith argument of C' will

be at least a receiving argument by Definition 5.2.5. We separate it into two cases:

i) If the ith argument of C'' is an active argument or a receiving argument, then it has a patience ruloid since the language $\mathcal{L}$ is a weak ω-failure language. Therefore, by Lemma 5.3.2, q_i' can be any process such that $q_i \stackrel{\sigma_i\delta_i}{\Longrightarrow} q_i'$ with $\delta_i = \tau^*$.

ii) If the ith argument of C'' is an other argument, then, by the induction hypothesis, q_i'' can be q_i or any process such that $q_i \stackrel{\sigma_i}{\Longrightarrow} q_i''$. Now, since $\mathcal{L}$ is a weak ω-failure language, no patience ruloid for the ith argument of C'' and the ith argument of C'. Therefore, q_i' is just q_i'', and thus q_i' can be q_i or any process such that $q_i \stackrel{\sigma_i}{\Longrightarrow} q_i''$. Then, by $\delta_i = \tau^*$, q_i' can be q_i or any process such that $q_i \stackrel{\sigma_i\delta_i}{\Longrightarrow} q_i'$. □

Note that, the above lemma does not hold in weak 1-failure languages since patience ruloids for receiving arguments are needed when proving it. Therefore, it will not be used when proving the congruence theorem for the wean 1-failure format.

The following lemma shows that, in a weak finite failure language, if a process executes an action sequence (weak trace) with length k, then, at the same time, all the lengthes of the action sequences executed by its subprocesses may not exceed k. This lemma also holds in weak 1-failure languages.

Lemma 5.3.4 Let $\mathcal{L}$ be a weak finite failure language. $C(x_1, ..., x_n)$ is any context of n holes in $\mathcal{L}$. Suppose that ζ is any closed Σ substitution mapping x_i into p_i. If $C(p_1, ..., p_n)$ is a process and σ is a weak trace of $C(p_1, ..., p_n)$, then each p_i will execute a weak trace σ_i at the same time, for $1 \leq i \leq n$. We can conclude that $\forall 1 \leq i \leq n : |\sigma_i| \leq k$ when $|\sigma| = k$.

Proof By Lemma 5.3.1, if $C(p_1, ..., p_n)$ is a process and σ is a weak trace of $C(p_1, ..., p_n)$, then each p_i will execute a weak trace σ_i at the same time. We also need to prove that $|\sigma_i| \leq k$ when $|\sigma| = k$.

We will prove it by making an induction on $|\sigma| = k$.

1) $k = 0$. Then, $C(p_1, ..., p_n) \stackrel{\tau^*}{\Longrightarrow} C'(p_1', ..., p_n')$. By the definition of the weak ω-failure format, the ruloids applied in this procedure can only be patience ruloids or ruloids with τ conclusion. By the hypothesis, no rules with τ conclusion are present in $\mathcal{L}$, and thus, by Proposition 5.2.8, no ruloids with τ conclusion are present in $\mathscr{R}(C, \tau)$. Therefore, when $C(p_1, ..., p_n) \stackrel{\tau^*}{\Longrightarrow} C'(p_1', ..., p_n')$, only patience ruloids are applied. However, from the definition of patience ruloids and its corresponding ruloids in form (2), $p_i \stackrel{\tau}{\longrightarrow} p_i'$ or $p_i \stackrel{\epsilon}{\longrightarrow} p_i'$ for all $1 \leq i \leq n$. And $|\tau^*| = |\epsilon| = 0$.

2) Assume that, when $k = m - 1$ with $m \geq 1$, we have $|\sigma_i| \leq k$ when $|\sigma| = k$.

3) For $k = m$, let $\sigma = \sigma'\alpha$. Then, we have $C(p_1, ..., p_n) \stackrel{\sigma'}{\Longrightarrow} C''(p_1'', ..., p_n'') \stackrel{\alpha}{\Longrightarrow} C'(p_1', ..., p_n')$. Extending it, we obtain that $C(p_1, ..., p_n) \stackrel{\sigma'}{\Longrightarrow} C''(p_1'', ..., p_n'') \stackrel{\tau^*}{\longrightarrow} C^1(p_1^1, ..., p_n^1) \stackrel{\alpha}{\longrightarrow} C^2(p_1^2, ..., p_n^2) \stackrel{\tau^*}{\longrightarrow} C'(p_1', ..., p_n')$.

By Lemma 5.3.1, for all $1 \leq i \leq n$, there exist $p_i \stackrel{\sigma_i'}{\Longrightarrow} p_i'' \stackrel{\sigma_i^1}{\Longrightarrow} p_i^1 \stackrel{\alpha_i}{\Longrightarrow} p_i^2 \stackrel{\sigma_i^2}{\Longrightarrow} p_i'$.

It is trivial that $|\sigma'| = m - 1$. Therefore, by the induction hypothesis, we have $|\sigma_i'| \leq m - 1$. Also, when $C''(p_1'', ..., p_n'') \stackrel{\tau^*}{\longrightarrow} C^1(p_1^1, ..., p_n^1)$ and $C^2(p_1^2, ..., p_n^2) \stackrel{\tau^*}{\longrightarrow} C'(p_1', ..., p_n')$, $|\tau^*| = 0$. Therefore, by the induction base, we have $|\sigma_i^1| = |\sigma_i^2| = 0$. Furthermore, $|\alpha_i| \leq 1$ can be obtained by the ruloids of form (2).

In all, $|\sigma_i| \leq k$ when $|\sigma| = k$. □

The following lemma shows that, in weak 1-failure languages, process $C(p_1, ..., p_i, ..., p_n)$ and $C(p_1, ..., p_i', ..., p_n)$ have the same sets of next observable actions if $p_i \stackrel{\tau^*}{\longrightarrow} p_i'$ and the ith argument is not an active argument.

Lemma 5.3.5 Let $\mathcal{L} = (\Sigma, \Psi)$ be a weak 1-failure language, and $C(x_1, ..., x_n)$ be any context of n holes. Suppose that ζ is any closed Σ substitution mapping x_i into p_i. If the ith argument is not an active argument of $C(x_1, ..., x_n)$ and $p_i \stackrel{\tau^*}{\longrightarrow} p_i'$, then $\mathcal{T}(C(p_1, ..., p_i, ..., p_n), 1) = \mathcal{T}(C(p_1, ..., p_i', ..., p_n), 1)$.

Proof Without loss of generality, suppose that $p = C(p_1, ...p_i, ..., p_n)$ and $q = C(p_1, ..., p_i', ..., p_n)$, where C is any context of n holes in the language $\mathcal{L}$. Let $A_1 = \{a \in Act | p \stackrel{a}{\Longrightarrow}\}$ and $A_2 = \{a \in Act | p \stackrel{a}{\Longrightarrow}\}$. We need to prove $A_1 = A_2$. Consider the next ruloid which will be applied.

1) If the next ruloid is a patience ruloid, then it should be a patience ruloid for active argument, since $\mathcal{L}$ is a weak 1-failure language. However, applying the patience ruloid will not produce observable actions for $C(p_1, ...p_i, ..., p_n)$ and $C(p_1, ..., p_i', ..., p_n)$. Because the ith argument is not an active argument, $C(p_1, ...p_i, ..., p_j, ..., p_n) \stackrel{\tau}{\longrightarrow} C(p_1, ...p_i, ..., p_j', ..., p_n)$ and $C(p_1, ...p_i', ..., p_j, ..., p_n) \stackrel{\tau}{\longrightarrow} C(p_1, ...p_i', ..., p_j', ..., p_n)$ when the jth argument of context C is an active argument and $p_j \stackrel{\tau}{\longrightarrow} p_j'$. Now, it is enough to consider the set of next observable actions of $C(p_1, ...p_i, ..., p_j', ..., p_n)$ and $C(p_1, ...p_i', ..., p_j', ..., p_n)$.

2) If the next ruloid is a plain ruloid, then it should not be a ruloid with τ conclusion, since $\mathcal{L}$ is a weak 1-failure language. Suppose that the applied ruloid r is in form (1), then the ith argument is not in I since it is not an active argument. Therefore, by the soundness property of the ruloids, the p_i will not be fired when applying the ruloid r. Furthermore, since p and q are only different in p_i and p_i', we have $A_1 = A_2$. □

5.4 Weak 1-Failure Format for Weak 1-Failure Equivalence

Now, we will prove the congruence theorem for the weak 1-failure format.

Theorem 5.4.1 The weak 1-failure format is a congruence format for the weak 1-failure equivalence.

Proof It is enough to prove that if $\forall 1 \leq j \leq n : p_j \sim_f^1 q_j$ then $C(p_1, ..., p_n) \sim_f^1 C(q_1, ..., q_n)$, where C is any context of n holes in a weak 1-failure language $\mathcal{L}$. By the symmetry of the alternative characterization of weak 1-failure equivalence in Proposition 3.5, we only need to prove that if $\forall 1 \leq j \leq n : p_j \sim_f^1 q_j$, then, for any $\sigma \in \mathcal{T}(C(p_1, ..., p_n), \omega)$ and $C'(p_1', ..., p_n')$ with $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p_1', ..., p_n')$, there exists $C'(q_1', ..., q_n')$ such that $C(q_1, ..., q_n) \stackrel{\sigma}{\Longrightarrow} C'(q_1', ..., q_n')$ and $\mathcal{T}(C'(q_1', ..., q_n'), 1) \subseteq \mathcal{T}(C'(p_1', ..., p_n'), 1)$. Observe that, though it needs only there exists some $C''(q_1'', ..., q_n'')$ such that $C(q_1, ..., q_n) \stackrel{\sigma}{\Longrightarrow} C''(q_1'', ..., q_n'')$ and $\mathcal{T}(C''(q_1'', ..., q_n''), 1) \subseteq \mathcal{T}(C'(p_1', ..., p_n'), 1)$, we will prove in the following that we can safely let C'' be C', and thus we write $C''(q_1'', ..., q_n'')$ as $C'(q_1', ..., q_n')$.

By Lemma 5.3.1, when $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p_1', ..., p_n')$, there exists $p_j \stackrel{\sigma_j}{\Longrightarrow} p_j'$ for all subprocess p_j with $1 \leq j \leq n$. Similarly, for all $a \in \mathcal{T}(C'(p_1', ..., p_n'), 1)$, when $C'(p_1', ..., p_n') \stackrel{a}{\Longrightarrow}$, we have $p_j' \stackrel{\delta_j}{\Longrightarrow}$ for all subprocess p_j with $1 \leq j \leq n$. Let A_j' be the set of all δ_j. Note that, for some $a \in A$, there may exist several δ_j corresponding with

it. And we should add all of them into the set A'_j.

Then, by Lemma 5.3.4, the exclusion of the rules with τ-conclusion will make the length of δ_j not exceed 1, i.e., $\forall \delta_j \in A'_j : |\delta_j| \leq 1$. Therefore, for all $1 \leq j \leq n$, we have $A'_j \subseteq \mathcal{T}(p'_j, 1)$.

Now, by $p_j \sim_f^1 q_j$ and Proposition 3.5, there exists some q'_j such that $q_j \stackrel{\sigma_j}{\Longrightarrow} q'_j$ and $\mathcal{T}(q'_j, 1) \subseteq \mathcal{T}(p'_j, 1)$.

By Lemma 5.3.2, σ is also a trace of $C(q_1, ..., q_n)$ and $C(q_1, ..., q_n) \stackrel{\sigma}{\Longrightarrow} C'(q''_1, ..., q''_n)$. Observe that, it is possible that q''_j is not equivalent to q'_j. The reason is that, from Lemma 5.3.2, we can only obtain that, there exist $q''_1, ..., q''_n$ such that $C(q_1, ..., q_n) \stackrel{\sigma}{\Longrightarrow} C'(q''_1, ..., q''_n)$, but not the very $q'_1, ..., q'_n$ which are obtained from $p_j \sim_f^1 q_j$ and Proposition 3.5.

By Lemma 5.3.2, if the jth argument of C' is an active argument, then q''_j can be any process such that $q_j \stackrel{\sigma_j}{\Longrightarrow} q''_j$ and thus we may let q''_j be q'_j safely since $q_j \stackrel{\sigma_j}{\Longrightarrow} q'_j$. On the other hand, if the jth argument of C' is not an active argument, then no patience ruloids are present in the language. Therefore, q''_j can be any delay process of $q_i \stackrel{\sigma_j}{\Longrightarrow}$. Note that, for q'_j, there must exist some delay process q''_j such that $q_j \stackrel{\sigma_j}{\Longrightarrow} q''_j \stackrel{\tau^*}{\longrightarrow} q'_j$.

Now, by Lemma 5.3.5 and $q''_j \stackrel{\tau^*}{\longrightarrow} q'_j$, we assert that $\mathcal{T}(C'(q''_1, ..., q''_n), 1) = \mathcal{T}(C'(q'_1, ..., q'_n), 1)$. Note that, there may exist several arguments of C' such that they are not active arguments. However, we can finally obtain $\mathcal{T}(C'(q''_1, ..., q''_n), 1) = \mathcal{T}(C'(q'_1, ..., q'_n), 1)$ by applying Lemma 5.3.5 for several times.

Moreover, by $\mathcal{T}(q'_j, 1) \subseteq \mathcal{T}(p'_j, 1)$, we have $\mathcal{T}(C'(q'_1, ..., q'_n), 1) \subseteq \mathcal{T}(C'(p'_1, ..., p'_n), 1)$.

Finally, we obtain that $\mathcal{T}(C'(q''_1, ..., q''_n), 1) \subseteq \mathcal{T}(C'(p'_1, ..., p'_n), 1)$. □

5.5 Weak Finite Failure Format for Weak i-Failure Equivalence

The following is the congruence theorem for the weak finite failure format.

Theorem 5.5.1 The weak finite failure format is a congruence format for the weak i-failure equivalence with $1 < i < \omega$.

Proof Similar with Theorem 5.4.1, it is enough to prove that if $\forall 1 \leq j \leq n : p_j \sim_f^i q_j$ then $C(p_1, ..., p_n) \sim_f^i C(q_1, ..., q_n)$, where C is any context of n holes in a weak 1-failure language $\mathcal{L}$. By the symmetry of the alternative characterization of weak i-failure equivalence in Proposition 3.5, we only need to prove that if $\forall 1 \leq j \leq n : p_j \sim_f^i q_j$, then, for any $\sigma \in \mathcal{T}(C(p_1, ..., p_n), \omega)$ and $C'(p'_1, ..., p'_n)$ with $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p'_1, ..., p'_n)$, there exists $C'(q'_1, ..., q'_n)$ such that $C(q_1, ..., q_n) \stackrel{\sigma}{\Longrightarrow} C'(q'_1, ..., q'_n)$ and $\mathcal{T}(C'(q'_1, ..., q'_n), i) \subseteq \mathcal{T}(C'(p'_1, ..., p'_n), i)$.

By Lemma 5.3.1, when $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p'_1, ..., p'_n)$, there exists $p_j \stackrel{\sigma_j}{\Longrightarrow} p'_j$ for all subprocess p_j with $1 \leq j \leq n$. Similarly, for all $\delta \in \Phi$, when $C'(p'_1, ..., p'_n) \stackrel{\delta}{\Longrightarrow}$, we have $p'_j \stackrel{\delta_j}{\Longrightarrow}$ for all subprocess p_j with $1 \leq j \leq n$. Let Φ'_j be the set of all δ_j. Note that, for some $\delta \in \Phi$, there may exist several δ_j corresponding with it. And we should add all of them into the set Φ'_j.

By Lemma 5.3.4, the exclusion of the rules with τ-conclusion will make the length of δ_j not exceed i, i.e., $\forall \delta_j \in \Phi'_j : |\delta_j| \leq i$. Therefore, for all $1 \leq j \leq n$, we have $\Phi'_j \subseteq \mathcal{T}(p'_j, i)$.

Now, by $p_j \sim_f^i q_j$, there exists some q'_j such that $q_j \stackrel{\sigma_j}{\Longrightarrow} q'_j$ and $\mathcal{T}(q'_j, i) \subseteq \mathcal{T}(p'_j, i)$.

By Lemma 5.3.2, σ is also a trace of $C(q_1, ..., q_n)$ and $C(q_1, ..., q_n) \stackrel{\sigma}{\Longrightarrow} C'(q''_1, ..., q''_n)$. Moreover,

1) if the jth argument of C' is a receiving argument or an active argument, then it has a patience ruloid since the language is a weak finite failure language. Therefore, by Lemma 5.3.2, we can let q''_j be q'_j since q''_j can be any process such that $q_j \stackrel{\sigma_i}{\Longrightarrow} q''_j$, and

2) if the jth argument of C' is an other argument, then, by Lemma 5.3.3, we can let q''_j be q_j or any process such that $q_j \stackrel{\sigma_j}{\Longrightarrow} q''_j$. We want to separate it into two cases:

i) if q''_j is any process such that $q_j \stackrel{\sigma_j}{\Longrightarrow} q''_j$, then we can also let q''_j be q'_j.

ii) if q''_j is q_j, then q'_j and q''_j are both delay processes since the jth argument of C' is an other argument and thus no patience ruloid for it. Therefore, we can obtain that $q''_j \equiv q'_j \equiv q_j$.

In all, we can always let q''_j be q'_j, i.e., $C(q_1, ..., q_n) \stackrel{\sigma}{\Longrightarrow} C'(q'_1, ..., q'_n)$.

Moreover, by $\mathcal{T}(q'_j, i) \subseteq \mathcal{T}(p'_j, i)$, we have $\mathcal{T}(C'(q'_1, ..., q'_n), i) \subseteq \mathcal{T}(C'(p'_1, ..., p'_n), i)$,. □

5.6 Weak ω-Failure Format for Weak ω-Failure Equivalence

The congruence theorem for the weak ω-failure format is as follows.

Theorem 5.6.1 The weak ω-failure format is a congruence format for the weak ω-failure equivalence.

Proof Similar with Theorem 5.4.1, it is enough to prove that if $\forall 1 \leq j \leq n : p_j \sim_f^\omega q_j$ then $C(p_1, ..., p_n) \sim_f^\omega C(q_1, ..., q_n)$, where C is any context of n holes in a weak 1-failure language $\mathcal{L}$. By the symmetry of the alternative characterization of weak ω-failure equivalence in Proposition 3.5, we only need to prove that if $\forall 1 \leq j \leq n : p_j \sim_f^\omega q_j$, then, for any $\sigma \in \mathcal{T}(C(p_1, ..., p_n), \omega)$ and $C'(p'_1, ..., p'_n)$ with $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p'_1, ..., p'_n)$, there exists $C'(q'_1, ..., q'_n)$ such that $C(q_1, ..., q_n) \stackrel{\sigma}{\Longrightarrow} C'(q'_1, ..., q'_n)$ and $\mathcal{T}(C'(q'_1, ..., q'_n), \omega) \subseteq \mathcal{T}(C'(p'_1, ..., p'_n), \omega)$.

By Lemma 5.3.1, when $C(p_1, ..., p_n) \stackrel{\sigma}{\Longrightarrow} C'(p'_1, ..., p'_n)$, there exists $p_j \stackrel{\sigma_j}{\Longrightarrow} p'_j$ for all subprocess p_j with $1 \leq j \leq n$. Similarly, for all $\delta \in \Phi$, when $C'(p'_1, ..., p'_n) \stackrel{\delta}{\Longrightarrow}$, we have $p'_j \stackrel{\delta_j}{\Longrightarrow}$ for all subprocess p_j with $1 \leq j \leq n$. Let Φ'_j be the set of all δ_j. Note that, for some $\delta \in \Phi$, there may exist several δ_j corresponding with it. And we should add all of them into the set Φ'_j.

Therefore, for all $1 \leq j \leq n$, we have $\Phi'_j \subseteq \mathcal{T}(p'_j, \omega)$.

Now, by $p_j \sim_f^\omega q_j$, there exists some q'_j such that $q_j \stackrel{\sigma_j}{\Longrightarrow} q'_j$ and $\mathcal{T}(q'_j, \omega) \subseteq \mathcal{T}(p'_j, \omega)$.

By Lemma 5.3.2, σ is also a trace of $C(q_1, ..., q_n)$ and $C(q_1, ..., q_n) \stackrel{\sigma}{\Longrightarrow} C'(q''_1, ..., q''_n)$. Moreover,

1) if the jth argument of C' is a receiving argument or an active argument, then it has a patience ruloid since the language is a weak ω-failure language. Therefore, by Lemma 5.3.2, we can let q''_j be q'_j since q''_j can be any process such that $q_j \stackrel{\sigma_i}{\Longrightarrow} q''_j$, and

2) if the jth argument of C' is an other argument, then, by Lemma 5.3.3, we can let q''_j be q_j or any process such that $q_j \stackrel{\sigma_j}{\Longrightarrow} q''_j$. We want to separate it into two cases:

i) if q''_j is any process such that $q_j \stackrel{\sigma_j}{\Longrightarrow} q''_j$, then we can also let q''_j be q'_j.

ii) if q''_j is q_j, then q'_j and q''_j are both delay processes since the jth argument of C' is an other argument and thus no patience ruloid for it. Therefore, we can obtain that $q''_j \equiv q'_j \equiv q_j$.

In all, we can always let q''_j be q'_j, i.e., $C(q_1, ..., q_n) \stackrel{\sigma}{\Longrightarrow} C'(q'_1, ..., q'_n)$.

Moreover, by $\mathcal{T}(q'_j, \omega) \subseteq \mathcal{T}(p'_j, \omega)$, we have $\mathcal{T}(C'(q'_1, ..., q'_n), \omega) \subseteq \mathcal{T}(C'(p'_1, ..., p'_n), \omega)$. □

6 Conclusions

In the paper, we first introduce a series of behavioral equivalences, named weak parametric failure equivalences, which take the weak failure equivalence and the weak impossible future equivalence as their special cases. Then, based on the idea of Structural Operational Semantics, rule formats are proposed to meet these behavioral equivalences. By the intuitive opinions and formal proofs, we have shown that these rule formats are all congruence formats of their corresponding behavioral equivalences.

An advantage of these rule formats is that we can easily decide whether a behavioral equivalence is congruent under a given operator. In fact, for any behavioral equivalences, one of the most frequently-asked problems is whether or not it can be preserved under some frequently-used operators, such as prefixing, choice, parallel composition, etc., in classical process algebraic languages like CCS (Milner, 1989), CSP (Hoare, 1985) and ACP (Baeten, 1990). Generally, there exist two ways to deal with this problem: The first one is to prove the congruence properties of these operators one by one. It is a straightforward and intuitive way, but may be somewhat clumsy. The second one is to pursue a rule format for this specified behavioral equivalence. And the given behavioral equivalence can be preserved under any operators in this format.

However, we have noticed that equivalences in strong notion, such as strong bisimulation and decorated trace semantics, were paid more attentions to than equivalences in weak notion, such as weak bisimulation and testing theory. In fact, almost all classical strong equivalences have found their corresponding rule formats, but much less works have been done on the rule formats of weak equivalences, especially on the rule formats of the equivalences in testing theoretical notions. And more specifically, no rule formats have been presented to be congruence formats for the weak failure equivalence or the weak impossible future equivalence. The difference may exist in the increasing complexity after introducing τ transitions by weak equivalences. The aim of our paper is to make a progress along this direction.

References

R.J. van Glabbeek. The Linear Time - Branching Time Spectrum I: The Semantics of Concrete, Sequential Processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, Handbook of Process Algebra, chapter 1, pages 3-100. Elsevier, 2001.

R.J. van Glabbeek. The Linear Time - Branching Time Spectrum II: The semantics of sequential systems with silent moves. In E. Best, editor, Concur'93, LNCS 715, pages 66-81. Springer-Verlag, 1993.

R. Milner. Communication and Concurrency. Prentice-Hall, 1989.

M.R. Mousavi, M.A. Reniers, J.F. Groote (2007). SOS formats and meta-theory: 20 years after. Theoretical Computer Science 373, pages 238-272.

L. Aceto, W.J. Fokkink and C. Verhoef. Structural Operational Semantics. In J.A. Bergstra, A. Ponse and S.A. Smolka, editors, Handbook of Process Algebra, Chapter 3, pages 197-292. Elsevier, 2001.

G.D. Plotkin. A Structural Approach to Operational Semantics. The Journal of Logic and Algebraic Programming 60-61, 17-139, 2004.

J.C.M. Baeten and W.P. Weijland. Process Algebra. volume 18 of Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1990.

J.F. Groote and F.W. Waandrager. Structural Operational Semantics and Bisimulation as a Congruence. Information and Computation 100(2), 202-260, 1992.

R.D. Simone. Higher-level synchronising devices in Meiji-SCCS. Theoretical Computer Science 37, 245-267, 1985.

J.F. Groote. Transition System Specifications with Negative Premises. Theoretical Computer Science 118, 263-299, 1993.

B. Bloom, S. Istrail and A. R. Meyer. Bisimualtion can't be Traced. Journal of the ACM 42(1), 232-268, 1995.

A. Rensink, W. Vogler. Fair testing. Information and Computation, Volume 205, Issue 2, February 2007, Pages 125-198.

R.J. van Glabbeek, On Cool Congruence Formats for Weak Bisimulations. In D.V. Hung and M. wirsing, editors, Proceedings International Colloquium on Theoretical Aspects of Computing, LNCS 3722, page 331-346. Springer, 2005.

I. Ulidowski, Finite Axiom Systems for Testing Preorder and De Simone Process Languages. Theoretical computer Science, 239(1):97-139, 2000.

C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall, Englewood Cliffs, NJ, 1985.

R.J. van Glabbeek, The Meaning of Negative Premises in Transition System Specification II. The Journal of Logic and Algebraic Programming 60-61, pages 229-258, 2004.

B. Bloom. Structural operational semantics for weak bisimulations. Theoretical Computer Science 146, pages 27-68, 1995.

B. Bloom. Ready Simulation, Bisimulation, and the Semantics of CCS-Like Languages. PhD thesis, MIT, 1990.

R. Milner. Communicating and Mobile Systems: the π-Calculus. Cambridge University Press, 1990.

Modelling for Lazy Clause Generation

Olga Ohrimenko[2] and Peter J. Stuckey[1,2]

[1] NICTA Victoria Research Lab
[2] Department of Comp. Sci. and Soft. Eng,
University of Melbourne, Victoria 3010, Australia,
Email: {olgao,pjs}@csse.unimelb.edu.au

Abstract

Lazy clause generation is a hybrid SAT and finite domain propagation solver that tries to combine the advantages of both: succinct modelling using finite domains and powerful nogoods and backjumping search using SAT technology. It has been shown that it can solve hard scheduling problems significantly faster than SAT or standard finite domain propagation alone. This new hybrid opens up many choices in modelling problems because of its dual representation of problems as both finite domain and SAT variables. In this paper we investigate some of those choices. Arising out of the modelling choices comes a novel combination of bounds representation and domain propagation which creates a form of propagation of disjunctions. We show this novel modelling approach can outperform more standard approaches on some problems.

1 Introduction

We consider the problem of solving *Constraint Satisfaction Problems* (CSPs) defined in the sense of [7], which can be stated briefly as follows:

> We are given a set of variables, a domain of possible values for each variable, and a set (read as a conjunction) of constraints. Each constraint is a relation defined over a subset of the variables, limiting the combination of values that the variables in this subset can take. The goal is to find a *consistent* assignment of values to the variables so that all the constraints are satisfied simultaneously.

Finite domain propagation systems solve CSPs using elaborate search strategies working in tandem with propagation to reduce the search space by removing inconsistent assignments as early as possible. There has been a significant amount of research on how to solve CSPs by encoding them in a Boolean clausal representation and then using Boolean satisfiability (SAT) solver to find a solution. Although this approach is quite successful for some problem classes, on other problems it turns out that the brute-force translation of the problem is too big to be handled effectively.

Finite domain propagation solvers effectively represent the possible values of variables by a set of choices which can be naturally modelled as Boolean variables. Recently [11] we described how we can mimic a finite domain propagation engine, by mapping propagators into clauses in a SAT solver. This immediately results in strong nogoods for finite domain propagation. We showed how we can convert propagators to lazy clause generators for a SAT solver. The resulting system can solve scheduling problems significantly faster than generating the clauses from scratch, or using Satisfiability Modulo Theories [10] solvers with difference logic. The resulting hybrid [11] combines the advantages of SAT solving, in particular powerful and efficient nogood learning and backjumping, with the advantages of finite domain propagation, simple and powerful modelling and specialized and efficient propagation of information.

In this paper we extend our previous work by exploiting the possibilities that the new system offers.

We show that this approach allows independence between the Boolean representation of integer variables and the propagators that act upon them. This representation independence leads to a new type of propagation: mixing bounds representation and domain propagators. The new propagator results in disjunctive propagation, where new information is created by propagation which is disjunctive in nature, even though the propagator was not a disjunctive at the start. Since the underlying SAT representation of propagation can represent disjunctive information efficiently, it allows us to create new "disjunctive propagators" from scratch.

The next section introduces notations and the lazy clause generation solving approach. We then explore modelling choices that arise in lazy clause generation solving, in particular we show that the choice of propagator can be independent of the choice of Boolean variable representation. In Section 4 we discuss the implementation of lazy clause generation and how it has to be extended to sup-

port new features of the modelling. We give experimental results in Section 5, and then conclude.

2 Lazy Clause Generation

2.1 Finite Domain Propagation

We consider a set of integer variables $\mathcal{V}$. A *domain* D is a complete mapping from $\mathcal{V}$ to finite sets of integers. We can understand a domain D as a formula $\wedge_{v\in\mathcal{V}}(v \in D(v))$ stating for each variable v that its value is in its domain.

Let D_1 and D_2 be domains and $V \subseteq \mathcal{V}$. We say that D_1 is *stronger* than D_2, written $D_1 \sqsubseteq D_2$, if $D_1(v) \subseteq D_2(v)$ for all $v \in \mathcal{V}$ and that D_1 and D_2 are *equivalent modulo* V, written $D_1 =_V D_2$, if $D_1(v) = D_2(v)$ for all $v \in V$. The *intersection* of D_1 and D_2, denoted $D_1 \sqcap D_2$, is defined by the domain $D_1(v) \cap D_2(v)$ for all $v \in \mathcal{V}$.

We use *range* notation: $[\,l\,..\,u\,]$ denotes the set of integers $\{d \mid l \leqslant d \leqslant u, d \in \mathbb{Z}\}$. We assume an *initial domain* D_{init} such that all domains D that occur will be stronger i.e. $D \sqsubseteq D_{init}$.

A *valuation* θ is a mapping of variables to values, written $\{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$. We extend the valuation θ to map expressions or constraints involving the variables in the natural way. Let *vars* be the function that returns the set of variables appearing in an expression, constraint or valuation. In an abuse of notation, we define a valuation θ to be an element of a domain D, written $\theta \in D$, if $\theta(v) \in D(v)$ for all $v \in vars(\theta)$.

A constraint is a restriction placed on the allowable values for a set of variables. We define the *solutions* of a constraint c to be the set of valuations θ that make that constraint true, i.e. $solns(c) = \{\theta \mid (vars(\theta) = vars(c)) \ \wedge \ (\models \theta(c))\}$

We associate with every constraint c a set of *propagators*. A propagator f for c is a monotonically decreasing function on domains such that for all domains $D \sqsubseteq D_{init}$: $f(D) \sqsubseteq D$ and $\{\theta \in D \mid \theta \in solns(c)\} = \{\theta \in f(D) \mid \theta \in solns(c)\}$. This is a weak restriction since, for example, the identity mapping is a propagator for any constraint.

Example 1 A common propagator f_d for the constraint $x \neq y$ is

$$\begin{array}{lcll} f(D)(x) & = & D(x) - \{d\}, & \text{if } D(y) = \{d\} \\ f(D)(x) & = & D(x), & \text{otherwise} \\ f(D)(y) & = & D(y) - \{d\}, & \text{if } D(x) = \{d\} \\ f(D)(y) & = & D(y), & \text{otherwise} \\ f(D)(v) & = & D(v), & v \notin \{x, y\} \end{array}$$

Let $D_1(x) = \{3,4,5,6\}$ and $D_1(y) = \{5\}$, then $f(D_1)(x_1) = \{3,4,6\}$ and $f(D_1)(y) = \{5\}$. □

A *propagation solver* for a set of propagators F and current domain D, $solv(F, D)$, repeatedly applies all the propagators in F starting from domain D until there is no further change in resulting domain. $solv(F, D)$ is the weakest domain $D' \sqsubseteq D$ which is a fixpoint (i.e. $f(D') = D'$) for all $f \in F$. In other words, $solv(F, D)$ returns a new domain defined by

$$\begin{array}{lcl} solv(F, D) & = & \text{gfp}(\lambda d.iter(F, d))(D) \\ iter(F, D) & = & \sqcap_{f\in F} f(D). \end{array}$$

where gfp denotes the greatest fixpoint w.r.t $\sqsubseteq$ lifted to functions.

2.2 Atomic Constraints and Propagation Rules

In order to convert propagation to clauses we need to extract the "pointwise" behavior of a propagator. To do so we use atomic constraints and propagation rules.

An *atomic constraint* represents the basic changes in domain that occur during propagation. For integer variables, the atomic constraints represent the elimination of values from an integer domain, i.e. $x \leqslant d$, $x \geqslant d$, $x \neq d$ or $x = d$ where $x \in \mathcal{V}$ and d is an integer. Note these correspond to events in a propagation engine: upper bound change, lower bound change, domain change and fixing the variable. We also consider the atomic constraint *false* which indicates that unsatisfiability is the direct consequence of propagation.

Define a *propagation rule* as $C \rightarrowtail c$ where C is a conjunction of *atomic constraints*, and c is a single atomic constraint such that $\not\models C \rightarrow c$. A propagation rule $C \rightarrowtail c$ defines a propagator (for which we use the same notation) in the obvious way

$$(C \rightarrowtail c)(D)(v) = \begin{cases} \{\theta(v) \mid \theta \in D \cap solns(c)\} \\ \quad vars(c) = \{v\} \wedge \models D \rightarrow C \\ D(v) \quad \text{otherwise.} \end{cases}$$

In another words, $C \rightarrowtail c$ defines a propagator that removes values from D based on c only when D implies C.

A propagator f *implements* a propagation rule $C \rightarrowtail c$ iff $\models D \rightarrow C$ implies $\models f(D) \rightarrow c$ for all $D \sqsubseteq D_{init}$.

Example 2 The propagator f_d of Example 1 implements the following propagation rules (among many others) for $D_{init}(x) = D_{init}(y) = [\,l\,..\,u\,]$.

$$\begin{array}{lcll} x = d & \rightarrowtail & y \neq d, & l \leqslant d \leqslant u \\ y = d & \rightarrowtail & x \neq d, & l \leqslant d \leqslant u \end{array}$$ □

A set of propagation rules $F \subseteq rules(f)$ *implements* f iff $solv(F, D) = f(D)$, for all $D \sqsubseteq D_{init}$.

In order to translate a propagator f to Boolean clauses we want to have a concise representation in terms of propagation rules, $rep(f)$, such that $rep(f)$ implements f.

Example 3 Consider the reified difference inequality $c \equiv b \Leftrightarrow x+c \leqslant y$ where $D_{init}(b) = \{0,1\}$, $D_{init}(x) = [\,l\,..\,u\,]$, $D_{init}(y) = [\,l\,..\,u\,]$. Then a set of propagation rules $rep(f)$ implementing the domain propagator f for c is

$$\begin{array}{rcl} b \geqslant 1 \wedge x \geqslant d & \rightarrowtail & y \geqslant d + c \\ b \geqslant 1 \wedge y \leqslant d & \rightarrowtail & x \leqslant d - c \\ b \leqslant 0 \wedge x \leqslant d & \rightarrowtail & y \leqslant d + c - 1 \\ b \leqslant 0 \wedge y \geqslant d & \rightarrowtail & x \geqslant d - c + 1 \\ x \geqslant d - c + 1 \wedge y \leqslant d & \rightarrowtail & b \leqslant 0 \\ x \leqslant d \wedge y \geqslant d + c & \rightarrowtail & b \geqslant 1 \end{array}$$

where $l \leqslant d \leqslant u$, except for the last two where $l - c \leqslant d \leqslant u + c$. □

A *bound propagation rule* only makes use of atomic constraints of the form $x \leqslant d$, $x \geqslant d$ and *false*. We can classify a propagator f as a *bounds propagator* if it has a representation $rep(f)$ which only makes use of bounds propagation rules.

Example 4 The propagator in Example 3 is clearly a bounds propagator. A bounds propagator f_b for the constraint $x \neq y$ is defined by the propagation rules for $D_{init}(x) = D_{init}(y) = [\, l \,..\, u \,]$ where $l \leqslant d \leqslant u$:

$$\begin{array}{lcl} x \leqslant d \wedge x \geqslant d \wedge y \leqslant d & \rightarrowtail & y \leqslant d-1 \\ x \leqslant d \wedge x \geqslant d \wedge y \geqslant d & \rightarrowtail & y \geqslant d+1 \\ y \leqslant d \wedge y \geqslant d \wedge x \leqslant d & \rightarrowtail & x \leqslant d-1 \\ y \leqslant d \wedge y \geqslant d \wedge x \geqslant d & \rightarrowtail & x \geqslant d+1. \end{array}$$ □

2.3 SAT and Unit Propagation

A *proposition* p is a Boolean variable from a universe of Boolean variables, $\mathcal{P}$. A *literal* l is either: a proposition p, its negation $\neg p$, the false literal $\bot$, or the true literal $\top$. The *complement* of a literal l, $\neg l$ is $\neg p$ if $l = p$ or p if $l = \neg p$, while $\neg\bot = \top$ and $\neg\top = \bot$. A *clause* C is a disjunction of literals. An *assignment* is either a set of literals A excluding $\bot$ such that $\forall p \in \mathcal{P}.\{p, \neg p\} \not\subseteq A$, or the failed assignment $\{\!\{\bot\}\!\}$. We define $A \sqsubseteq \{\!\{\bot\}\!\}$, and $A \sqcup A' = A \cup A$ unless the union contains $\bot$ or $\{p, \neg p\}$ for some literal p in which case $A \sqcup A' = \{\!\{\bot\}\!\}$.

An assignment A *satisfies* a clause C if one of the literals in C appears in A. A *theory* T is a set of clauses. An assignment is a *solution* to theory T if it satisfies each $C \in T$.

A SAT solver takes a theory T and determines if it has a solution. Complete SAT solvers typically involve some form of the DPLL algorithm which combines search and propagation by recursively fixing the value of a proposition to either $\top$ (true) or $\bot$ (false) and using unit propagation to determine the logical consequences of each decision made so far. The unit propagation algorithm finds all unit resolutions of an assignment A with the theory T. It can be defined as follows where C denotes a clause:

$$\begin{array}{rcl} up(A, C) & = & \begin{cases} \{\!\{\bot\}\!\} & \forall l \in C. \neg l \in A \\ A \sqcup \{l\} & \exists l \in C, , \neg l \notin A, \\ & \quad \forall l' \in (C \setminus \{l\}). \neg l' \in A \\ A & \text{otherwise} \end{cases} \\ UP(A, T) & = & \text{lfp}.(\lambda a. \bigsqcup_{C \in T} up(a, C))(A) \end{array}$$

Example 5 Given the theory $T = \{\, \neg p_1 \vee p_2 \vee p_3 \vee \neg p_4 \vee \neg p_5,\, p_1 \vee p_2,\, p_4 \vee \neg p_5 \}$ and the assignment $A_1 = \{\neg p_2, p_5\}$ unit propagation on $p_1 \vee p_2$ adds p_1, and on $p_4 \vee \neg p_5$ adds p_4, then unit propagation with the first clause adds p_3. Hence $UP(A_1, T) = \{p_1, \neg p_2, p_3, p_4, p_5\}$. □

2.4 Lazy Clause Generation

The lazy clause generation hybrid solver defined in [11] works as follows. We execute a SAT solver using a Boolean representation of the integer variables of the problem. When the SAT solver reaches an assignment A on these Boolean variables we calculate a corresponding domain D to A, and execute the propagators $f \in F$ on D. Any propagation rule r in $rep(f)$ that creates new information (that is $r(D) \not\equiv D$) is converted to a clause and added to the SAT solver. Unit propagation on this new clause will cause the assignment A to change to agree with $r(D)$.

We represent an integer variable x with domain $D_{init}(x) = [\, l \,..\, u \,]$ using the Boolean variables $[\![x = l]\!], \ldots, [\![x = u]\!]$ and $[\![x \leqslant l]\!], \ldots, [\![x \leqslant u-1]\!]$. The variable $[\![x = d]\!]$ is true if x takes the value d, and false if x takes a value different from d. Similarly the variable $[\![x \leqslant d]\!]$ is true if x takes a value less than or equal to d and false if x takes a value greater than d.

Not every assignment of Boolean variables is consistent with the integer variable x, for example $\{[\![x = 3]\!], [\![x \leqslant 2]\!]\}$ requires that x is both 3 and $\leqslant 2$. In order to ensure that assignments represent a consistent set of possibilities for the integer variable x we add the clauses $DOM(x)$ to the SAT solver

$$\begin{array}{rl} \neg[\![x \leqslant d]\!] \vee [\![x \leqslant d+1]\!] & l \leqslant d < u-1 \\ \neg[\![x = d]\!] \vee [\![x \leqslant d]\!] & l \leqslant d < u \\ \neg[\![x = d]\!] \vee \neg[\![x \leqslant d-1]\!] & l < d \leqslant u \\ [\![x = l]\!] \vee \neg[\![x \leqslant l]\!] & \\ [\![x = d]\!] \vee \neg[\![x \leqslant d]\!] \vee [\![x \leqslant d-1]\!] & l < d < u \\ [\![x = u]\!] \vee [\![x \leqslant u-1]\!] & \end{array}$$

These clauses encode $[\![x \leqslant d]\!] \rightarrow [\![x \leqslant d+1]\!]$ and $[\![x = d]\!] \leftrightarrow ([\![x \leqslant d]\!] \wedge \neg[\![x \leqslant d-1]\!])$. We let $DOM = \cup\{DOM(v) \mid v \in \mathcal{V}\}$.

Any unit fixpoint A of $DOM(x)$ can be converted to a domain for variable x:

$$\begin{array}{rl} domain(A)(x) = \{ & d \in D_{init}(x) \mid \forall [\![c]\!] \in A. \\ & vars(l) = \{x\} \Rightarrow x = d \models c\} \end{array}$$

that is the domain of all values for x that are consistent with all the Boolean variables related to x.

Example 6 For example the assignment $A = \{[\![x_1 \leqslant 10]\!],\ \neg[\![x_1 \leqslant 5]\!],\ \neg[\![x_1 = 7]\!],\ \neg[\![x_1 = 8]\!],\ [\![x_2 \leqslant 11]\!],\ \neg[\![x_2 \leqslant 5]\!],\ [\![x_3 \leqslant 10]\!],\ \neg[\![x_3 \leqslant -2]\!]\}$ is consistent with $x_1 = 6, x_1 = 9$ and $x_1 = 10$. hence $domain(A)(x_1) = \{6, 9, 10\}$. For the remaining variables $domain(A)(x_2) = [\, 6 \,..\, 11 \,]$ and $domain(A)(x_3) = [\, -1 \,..\, 10 \,]$. Note that for brevity A is not a fixpoint of $DOM(x_1)$ since we are missing many implied literals such as $\neg[\![x_1 = 5]\!]$, $\neg[\![x_1 = 12]\!]$, etc. □

The propagators F are run on the created domain, and each propagation rule that creates new information is converted to a Boolean clause. This is straightforward since we can map atomic constraints to Boolean literals. The mapping *lit* is

defined as: (where $D_{init}(x) = [\,l\,..\,u\,]$)

$$
\begin{array}{rcl}
lit(false) & = & \bot \\
lit(x = d) & = & \begin{cases} [\![x = d]\!] & l \leqslant d \leqslant u \\ \bot & \text{otherwise} \end{cases} \\
lit(x \neq d) & = & \begin{cases} \neg[\![x = d]\!] & l \leqslant d \leqslant u \\ \top & \text{otherwise} \end{cases} \\
lit(x \leqslant d) & = & \begin{cases} \top & d \geqslant u \\ \bot & d < l \\ [\![x \leqslant d]\!] & \text{otherwise} \end{cases} \\
lit(x \geqslant d) & = & \begin{cases} \top & d \leqslant l \\ \bot & d > u \\ \neg[\![x \leqslant d-1]\!] & \text{otherwise} \end{cases}
\end{array}
$$

We can transform a propagation rule r to a clause $cl(r)$ by:

$$cl(C \rightarrowtail c) = (\bigvee_{c' \in C} \neg lit(c')) \vee lit(c)$$

Example 7 Given the domain D corresponding to assignment A from Example 6, imagine a propagator f fires the propagation rule

$$x_1 \leqslant 10 \wedge x_2 \geqslant 6 \rightarrowtail x_3 \leqslant 1$$

This is transformed into the clause

$$\neg[\![x_1 \leqslant 10]\!] \vee [\![x_2 \leqslant 5]\!] \vee [\![x_3 \leqslant 1]\!]$$

This clause is added to the SAT solver. Unit propagation using the assignment A and the clause above adds the new information $[\![x_3 \leqslant 1]\!]$ to get assignment A'. □

Just as we can convert an assignment A to a domain D, we can convert a domain D to an assignment

$$
\begin{array}{rcl}
assign(D, x) & = & \{lit(c) \mid x \in D(x) \models c, \\
 & & \quad x \in vars(x)\} \\
assign(D) & = & \begin{cases} \{\!\{\bot\}\!\} & \exists v \in \mathcal{V}.D(v) = \varnothing \\ \bigcup_{v \in \mathcal{V}} assign(D, v) & \text{otherwise} \end{cases}
\end{array}
$$

Using the lazy clause generation we can show that the SAT solver maintains an assignment which is equivalent to the domains. In particular if we have clauses representing all the propagators F then unit propagation is guaranteed to be at least as strong as finite domain propagation.

Theorem 1 ([11]) *Let $rep(f)$ be a set of propagation rules implementing propagator f. Let $A = UP(assign(D), DOM \cup \bigcup\{cl(r) \mid f \in F, r \in rep(f)\})$. Then $A = \{\!\{\bot\}\!\}$ or $A \supseteq assign(solv(F, D))$.* □

3 Modelling Choices

Lazy clause generation proved to be a powerful approach to tackling finite domain problems with large amounts of search. In [11] we show that it can solve hard open shop scheduling problems more efficiently than pure SAT approaches and other finite domain solvers using the same model (Laborie [6] shows how to tackle hard scheduling problems using finite domains solvers by using complex resource constraints and specialized searching methods).

In this paper we explore some of the modelling possibilities that the novel solving technology of lazy clause generation allows.

3.1 Laziness and Eagerness

An important choice in the lazy clause generation approach is whether to implement a propagator lazily (which is the default) or eagerly. The eager representation of a propagator f simply adds the clauses $cl(r)$ for all $r \in rep(f)$ into the SAT solver before beginning the search. This clearly can improve search, since more information is known apriori, but the size of the clausal representation may make it inefficient.

Example 8 The representation of the domain propagator for disequality $x \neq y$ where $D_{init}(x) = D_{init}(y) = [\,l\,..\,u\,]$ requires $2(u - l + 1)$ binary clauses. Hence it is possible to model eagerly.

The representation of the bounds propagator for $x_1 + \cdots + x_n \leqslant k$ where $D_{init}(x_1) = \cdots = D_{init}(x_n) = [\,0\,..\,1\,]$ has ${}^nC_k = n!/((n-k)!k!)$ propagation rules. Clearly it is impossible to represent this eagerly for large n and k. □

In practice eager representation is useful for constraints that have very small representations.

3.2 Variable representation

The lazy clause generation approach represents variables domains of possible values in dual manner: a Boolean assignment and a domain D on integer variables. There are a number of choices of how we can represent integer variables in terms of Boolean variables. The default choice (full integer representation) is described in the previous section and was used in [11]. We present new choices below.

3.2.1 Non-continuous variables

We can represent an integer variable where $D_{init}(x) = \{d_1, \ldots, d_n\}$ where $d_i < d_{i+1}, 1 \leqslant i \leqslant n$, and the values are noncontinuous. This requires fewer Boolean variables, and fewer domain constraints then representing the domain $[\,d_1\,..\,d_n\,]$. The Boolean representation uses variables $[\![x = d_i]\!], 1 \leqslant i \leqslant n$ and $[\![x \leqslant d_i]\!], 1 \leqslant i < n$.

The clauses $DOM(x)$ required to maintain consistency of the Boolean assignment are:

$$
\begin{array}{rl}
\neg[\![x \leqslant d_i]\!] \vee [\![x \leqslant d_{i+1}]\!] & 1 \leqslant i < n-1 \\
\neg[\![x = d_i]\!] \vee [\![x \leqslant d_i]\!] & 1 \leqslant i < n \\
\neg[\![x = d_i]\!] \vee \neg[\![x \leqslant d_{i-1}]\!] & 1 < i \leqslant n \\
[\![x = d_1]\!] \vee \neg[\![x \leqslant d_1]\!] & \\
[\![x = d_i]\!] \vee \neg[\![x \leqslant d_i]\!] \vee [\![x \leqslant d_{i-1}]\!] & 1 < i < n \\
[\![x = d_n]\!] \vee [\![x \leqslant d_{n-1}]\!] &
\end{array}
$$

3.2.2 Bounds variables

We can represent an integer variable only using the bounds variables $[\![x \leqslant d]\!], l \leqslant d < u$ where $D_{init}(x) = [\,l\,..\,u\,]$. While this means we cannot represent all possible subsets of $[\,l\,..\,u\,]$, it has the advantage of requiring fewer Boolean variables, and the domain representation requires only the clauses:

$$\neg[\![x \leqslant d]\!] \vee [\![x \leqslant d+1]\!] \quad l \leqslant d < u-1$$

3.2.3 Non-continuous bounds variables

We can clearly restrict the representation of non-continuous variables to bounds only analogously, just using the Boolean variables $[\![x \leqslant d_i]\!]$

3.3 Propagator and variable representation independence

In a usual finite domain solver we are restricted so that if we use bounds variables, they must be restricted to only occur in bounds propagators. Indeed in [11] we use this observation to avoid using full integer variables for variables that only occur in bounds propagators. In the lazy clause generation solver we can separate the variable representation from the propagator type. To do so we make use of the more flexible clausal representation of propagators of the lazy clause generation solver.

With this separation the propagation engine can work without knowing whether integer variable x is a full integer, non-continuous, or bounds variable, since the translation of assignments to domains, and from propagation rules to clauses, completely captures the relationship between the Boolean representation and the integer variable.

Because of this separation we can independently choose which propagator we will use to represent a problem, without considering the variable representation. Hence for an individual constraint we can choose any of the propagators for that constraint.

3.3.1 Non-continuous variables

We extend the translation of atomic constraints *lit* to map atomic constraints involving non-continuous variable x where $D_{init}(x) = \{d_1, \ldots, d_n\}$ as follows:

$$lit(x = d) = \begin{cases} \bot & d \notin \{d_1, \ldots, d_n\} \\ [\![x = d_i]\!] & d = d_i \end{cases}$$

$$lit(x \neq d) = \begin{cases} \top & d \notin \{d_1, \ldots, d_n\} \\ \neg[\![x = d_i]\!] & d = d_i \end{cases}$$

$$lit(x \leqslant d) = \begin{cases} \top & d >= d_n \\ \bot & d < d_1 \\ [\![x \leqslant d_i]\!] & d_i < d \leqslant d_{i+1} \end{cases}$$

$$lit(x \geqslant d) = \begin{cases} \top & d \leqslant d_1 \\ \bot & d > d_n \\ \neg[\![x \leqslant d_i]\!] & d_i < d \leqslant d_{i+1} \end{cases}$$

Note that each atomic constraint is translated as a single literal.

Example 9 Consider the translation of the propagation rules $x = 3 \rightarrowtail y \neq 3$ and $x \neq 3 \rightarrowtail y = 3$, where $D_{init}(x) = \{0, 3, 5\}$ and $D_{init}(y) = \{1, 2, 4\}$. The resulting clauses are $\neg[\![x = 3]\!] \vee \top$ or $\top$ (the always true clause) and $[\![x = 3]\!] \vee \bot$ or equivalently $[\![x = 3]\!]$. □

3.3.2 Bounds variables

We extend the translation of atomic constraints *lit* to map atomic constraints involving bounds variable x where $D_{init}(x) = [\,l\,..\,u\,]$ as follows:

$$lit(x = d) = \begin{cases} [\![x \leqslant d]\!] & d = l \\ [\![x \leqslant d]\!] \wedge \neg[\![x \leqslant d-1]\!], & l < d < u \\ \neg[\![x \leqslant u-1]\!] & d = u \\ \bot & \text{otherwise} \end{cases}$$

$$lit(x \neq d) = \begin{cases} \neg[\![x \leqslant d]\!] & d = l \\ \neg[\![x \leqslant d]\!] \vee [\![x \leqslant d-1]\!], & l < d < u \\ [\![x \leqslant u-1]\!] & d = u \\ \top & \text{otherwise} \end{cases}$$

The translations of $x \leqslant d$ and $x \geqslant d$ are as for full integer variables. Note that these translations now no longer guarantee to return a single literal.

Clearly "Boolean integer" variables x where $D_{init}(x) = [\,0\,..\,1\,]$ can be represented as bounds only variables without loss of expressiveness since $x \leqslant 0 \leftrightarrow x = 0 \leftrightarrow \neg(x = 1)$.

We can translate any propagation rule to a conjunction of clauses by simply applying *lit* as before. This creates (a possibly non-clausal) Boolean formulae which can be transformed to conjunctive normal form.

Example 10 Consider the translation of the propagation rule $x = 3 \rightarrowtail y \neq 3$, where x and y are bounds only variables. The resulting formula is $\neg[\![x \leqslant 3]\!] \vee [\![x \leqslant 2]\!] \vee [\![y \leqslant 2]\!] \vee \neg[\![y \leqslant 3]\!]$, which is a clause already.

Consider the translation of the propagation rule $x \neq 3 \rightarrowtail y = 3$. The resulting formula is $\neg([\![x \leqslant 2]\!] \vee \neg[\![x \leqslant 3]\!]) \vee ([\![y \leqslant 3]\!] \wedge \neg[\![y \leqslant 2]\!])$. The conjunctive normal form is

$$\begin{array}{l} \neg[\![x \leqslant 2]\!] \vee [\![y \leqslant 3]\!] \\ [\![x \leqslant 3]\!] \vee [\![y \leqslant 3]\!] \\ \neg[\![x \leqslant 2]\!] \vee \neg[\![y \leqslant 2]\!] \\ [\![x \leqslant 3]\!] \vee \neg[\![y \leqslant 2]\!] \end{array}$$

It would appear that the conversion of propagation rules including bounds variables could lead to an exponential explosion in the number of clauses required to represent them. By restricting the conversion of the rules to clauses which may actually be able to cause unit propagation, in fact we can represent them with at most 2 clauses.

Lemma 1 *If domain $D = domain(A)$ is such that $D(x) \models x \neq d$ where x is a bounds only variable, then $D(x) \models x \geqslant d+1$ or $D(x) \models x \leqslant d-1$.*

Proof: Now A can only include literals $[\![x \leqslant d']\!]$ or $\neg[\![x \leqslant d']\!]$ for some d'. Hence $domain(A)(x)$ is a range domain. If $D(x) \models x \neq d$ then either $D(x) \models x \geqslant d+1$ or $D(x) \models x \leqslant d-1$. □

Define the bounds simplification $bs(r)$ of a propagation rule $r \equiv C \rightarrowtail c$, for domain $D = domain(A)$ for some assignment A which fires the rule, as follows. Replace each atomic constraint $x \neq d$ appearing in C where x is a bounds only variable by either $x \leqslant d-1$ or $x \geqslant d+1$, whichever holds in D. The resulting propagation rule can create at most 2 clauses.

Theorem 2 *The conjunctive normal form of the clausal representation of $bs(r)$ involves at most 2 clauses.*

Proof: Each atomic constraint appearing in the left hand side of $bs(r)$ is translated as a single Boolean literal. The only conjunction that can occur in the translation is if the right hand side is an atomic constraint $x = d$ and x is a bounds variable. The resulting CNF has two clauses. □

Example 11 Consider the translation of the propagation rule $r \equiv x \neq 3 \rightarrowtail y = 3$ where x and y are bounds variables ranging over $[\,0\,..\,10\,]$. Suppose the domain that causes it to fire is $D = domain(A)$ where $A = \{[\![x \leqslant 1]\!]\}$. Then $D(x) = [\,0\,..\,1\,]$ and $D(x) \models x \leqslant 2$ and $bs(r) \equiv x \leqslant 2 \rightarrowtail y = 3$. The translation to Booleans is the formula $\neg[\![x \leqslant 2]\!] \vee ([\![y \leqslant 3]\!] \wedge \neg[\![y \leqslant 2]\!])$, which in CNF is $(\neg[\![x \leqslant 2]\!] \vee [\![y \leqslant 3]\!]) \wedge (\neg[\![x \leqslant 2]\!] \vee \neg[\![y \leqslant 2]\!])$. Note that the two clauses from Example 10 that are missing could not fire in A. □

There is an important new behaviour that arises when we consider using domain propagators on bounds variables. The result of propagation is always a clause of a form

$$cl(C \rightarrowtail c) = \vee_{c' \in C}(\neg\, lit(c')) \vee lit(c),$$

where $\neg\, lit(c')$ are all false in the current assignment and $lit(c)$ is either undefined or false in the current assignment. Previously $lit(c)$ was always a single literal, hence we could guarantee unit propagation would apply, and set $lit(c)$ to true. Now there is a possibility that $lit(c)$ is itself a disjunction and unit propagation will not apply.

Example 12 Consider the execution of the domain propagation for $x \neq y$ (Example 1) where x and y are bounds variables on the assignment $A = \{[\![x \leqslant 3]\!], \neg[\![x \leqslant 2]\!]\}$. Then in the corresponding $domain(A)(x) = \{3\}$ and the propagation rule $x = 3 \rightarrowtail y \neq 3$ fires. The resulting clause is $\neg[\![x \leqslant 3]\!] \vee [\![x \leqslant 2]\!] \vee \neg[\![y \leqslant 3]\!] \vee [\![y \leqslant 2]\!]$. No unit propagation is possible using A and this new clause.

In fact the domain propagator for $x \neq y$ applied to bounds variables x and y generates exactly the same clauses as the bounds propagator, but it generates them earlier! □

3.3.3 Disjunctive propagators

The discussion at the end of the last subsection motivates examining a new possibility. Propagation rules are designed so that the result of the propagation is a single atomic constraint, which can then be represented immediately as a change in domain. Given that we will convert the propagation rules to clauses in any case we can extend them to allow disjunction on the right hand side. A *disjunctive propagation rule* has the form $c_1 \wedge \cdots \wedge c_n \rightarrowtail c_{n+1} \vee \cdots \vee c_m$. The translation to clauses is clear $cl(c_1 \wedge \cdots \wedge c_n \rightarrowtail c_{n+1} \vee \cdots \vee c_m) = \neg\, lit(c_1) \vee \cdots \vee \neg\, lit(c_n) \vee lit(c_{n+1}) \vee \cdots \vee lit(c_{n+m})$. Presently we restrict our implementation to only support disjunctive propagation rules with at most two literals on the right hand side.

Example 13 Consider the constraint $|x - y| \geqslant k$ for constant $k > 0$. The bounds propagator for this constraint has representation given by the propagation rules: (where $l + k > u - k$)

$$\begin{array}{rcl} x \geqslant l \wedge x \leqslant u \wedge y \leqslant l + k - 1 & \rightarrowtail & y \leqslant u - k \\ x \geqslant l \wedge x \leqslant u \wedge y \geqslant u - k + 1 & \rightarrowtail & y \geqslant l + k \\ y \geqslant l \wedge y \leqslant u \wedge x \leqslant l + k - 1 & \rightarrowtail & x \leqslant u - k \\ y \geqslant l \wedge y \leqslant u \wedge x \geqslant u - k + 1 & \rightarrowtail & x \geqslant l + k \end{array}$$

A more eager version of this propagator fires when the range on one variable is small enough to guarantee some (disjunctive) constraints on the other variable. It is defined by the disjunctive propagation rules: (where $l + k > u - k$)

$$\begin{array}{rcl} x \geqslant l \wedge x \leqslant u & \rightarrowtail & y \geqslant l + k \vee y \leqslant u - k \\ y \geqslant l \wedge y \leqslant u & \rightarrowtail & x \geqslant l + k \vee x \leqslant u - k \end{array}$$

Disjunctive propagators can be seen as a more eager form of lazy clause generation.

4 Implementation

The creation of a practical lazy clause generation solver involves many more considerations than were addressed in Section 2.4. To build the system we add a cut down propagation engine into a SAT solver and modify it as a lazy clause generator. We first describe this process as defined in [11] and then describe the extensions required.

The SAT solver applies unit propagation, and when it reaches a fixpoint it calls the propagation engine. The new literals set by the SAT solver are converted into domain changes in the propagation solver, and these "events" queue up propagators for execution.

The first propagator in the queue is then executed. If it causes propagation, then the clausal representation of the first propagation rule that fires is added to the SAT solver and unit propagation is applied. When the SAT solver finishes we re-execute the same propagator (which is still at the head of the queue) to search for another firing propagation rule. When there are no more firing rules the propagator is removed from the queue and the next propagator considered. The reason we add clauses as soon as possible is to detect failure as soon as possible. Unit propagations may schedule (or re-schedule) propagators. The process continues until the propagation queue is empty and unit propagation is at fixpoint. At this point the SAT solver makes a decision about a literal to set *true* and search continues.

On failure the propagation queue is cleared, and the SAT solver backtracks up the trail of decided and inferred literals. For each canceled literal we undo the domain change on the corresponding integer variable in the propagation engine.

A subtle point we have not addressed is why we do not worry about a propagator creating duplicates of clauses corresponding to its propagation rules, particularly since we can execute the propagator repeatedly simply to create all the propagation rules that fire for one domain. The reason is that since a propagator f is only run at domain $D = \mathit{domain}(A)$ for an assignment A which is a unit propagation fixpoint, then if $cl(r)$ is already in the SAT solver then r cannot fire on domain D (it has no new information).

Example 14 Consider the propagation of the constraint $x = y$ with $D_{init}(x) = D_{init}(y) = [0..4]$. After the SAT solver sets $\neg[\![x = 2]\!]$ and $\neg[\![y = 3]\!]$ the first propagation rule that fires is $x \neq 2 \rightarrowtail y \neq 2$. This is added as the clause $[\![x = 2]\!] \vee \neg[\![y = 2]\!]$ and propagated to set $\neg[\![y = 2]\!]$. Returning to the propagation engine the, the propagator for $x = y$ is still head of the queue. The original propagation rule no longer fires since $y \neq 2$ is not new information. Hence the next propagation rule $y \neq 3 \rightarrowtail x \neq 3$ is considered. □

The extensions of lazy clause generation we consider in this paper require modifications to the implementation. The reason is that using domain propagators on bounds variables, or more generally disjunctive propagators means that we can not be sure that a newly added clause does not already exist (has not previously been added) since it may not cause unit propagation with the current assignment.

This requires two modifications to the approach. First disjunctive propagators at the head of the queue must store an index of propagation rule processed last, and clear this index every time the propagator queue is cleared. This is to avoid them regenerating the same propagation rule when they are still the head of the queue. Secondly, before adding a clause corresponding to a disjunctive propagation rule we need to check that it is not already in the SAT solver.

We could build a separate data structure to record which clauses have been sent to the solver. To avoid the complexity and space required to do this we re-use existing data structures in the SAT solver. Suppose a propagation rule $C \rightarrowtail c_1 \vee c_2$ already has its corresponding clause Cl in the SAT solver. All literals in the clause except $\mathit{lit}(c_1)$ and $\mathit{lit}(c_2)$ must be false in the current assignment, otherwise the propagation rule would not fire. The SAT solver keeps track of at least two literals in each clause which are not false, the so-called *watched literals*, in order to detect unit propagations. Hence $\mathit{lit}(c_1)$ and $\mathit{lit}(c_2)$ must be the watched literals for Cl. To check if Cl appears in the SAT solver already, we check all clauses where $\mathit{lit}(c_1)$ is a watched literal (the SAT solver provides this data structure), and see if one is identical to Cl.

This check is reasonably expensive, but much cheaper than looking at all clauses involving $\mathit{lit}(c_1)$ since it will be the watched literal in few of them.

5 Experimental results

All experiments are performed on a 3.4GHZ Intel Pentium D with 4Gb RAM running on Debian Linux 4. The lazy clause generation system is built using MiniSat [9] version 2.0 beta. We compare our results against a highly optimized propagation solver Gecode 1.3.1 [3]. Eager models are run on MiniSat version 2.0 beta.

5.1 `alldifferent` propagators

The disequality `alldifferent`$([x_1, \ldots, x_n])$ constraint requires that $\forall 1 \leqslant i < j \leqslant n, x_i \neq x_j$.

In the lazy clause generation solver we can represent the disequality constraint $x \neq y$ in a number of ways: (a) using the domain propagator f_d from Example 1, (b) using the bounds propagator f_b from Example 4, and (c) using (bounds) propagators F_r for the reified set of constraints $b_1 \vee b_2$, $b_1 \Leftrightarrow x+1 \leqslant y$, $b_2 \Leftrightarrow y+1 \leqslant x$. In fact the last two representation have exactly the same propagation behaviour

Lemma 2 *Let* $D(b_1) = D(b_2) = [0..1]$, *then* $\mathit{solv}(F_r, D) =_{\{x,y\}} \mathit{solv}(\{f_b\}, D)$.

Proof: Suppose a propagation rule for f_b fires in D. Assume it has the form $x \geqslant d \wedge x \leqslant d \wedge y \geqslant d \rightarrowtail y \geqslant d+1$, reasoning for the other rules is analogous. Then the propagation rule $y \geqslant d \wedge x \leqslant d \rightarrowtail b_2 \leqslant 0$ from $b_2 \Leftrightarrow y + 1 \leqslant x$ fires. Hence the propagation rule $b_2 \leqslant 0 \rightarrowtail b_1 \geqslant 1$ from $b_1 \vee b_2$ fires, and hence the rule $b_1 \geqslant 1 \wedge x \geqslant d \rightarrowtail y \geqslant d+1$ from $b_1 \Leftrightarrow x + 1 \leqslant y$ fires.

In the reverse direction suppose a propagation rule for F_r fires in D modifying x or y. Assume it is of the form $b_1 \geqslant 1 \wedge x \geqslant d \rightarrowtail y \geqslant d+1$, reasoning for other rules is analogous. Then since $b_1 \geqslant 1$ is true, and is not true in D, either a rule $x \leqslant d' \wedge y \geqslant d'+1 \rightarrowtail b_1 \geqslant 1$ fires or $b_2 \leqslant 0 \rightarrowtail b_1 \geqslant 1$ fires.

Suppose a rule of the first kind fired. Now $d' \geqslant d$ since $x \geqslant d$ and $x \leqslant d'$ both hold and $d+1 > d'+1$ otherwise $y \geqslant d+1$ is not new information. This is a contradiction

Hence the second rule must fire. Since $b_2 \leqslant 0$ is now true, a rule of the form $y \geqslant d'' \wedge x \leqslant d'' \rightarrowtail b_2 \leqslant 0$ must have fired for some d''. Since the first rule creates new information $y \geqslant d+1$ is stronger that $y \geqslant d''$ hence $d \geqslant d''$. But since D ensures both $x \geqslant d$ and $x \leqslant d''$ we have that $d \geqslant d'' \geqslant x \geqslant d$, so $d = d''$. Hence D ensure that $x \geqslant d$, $x \leqslant d$ and $y \geqslant d$ and hence f_b fires the rule $x \geqslant d \wedge x \leqslant d \wedge y \geqslant d \rightarrowtail y \geqslant d+1$, causing $y \geqslant d+1$. □

There are more complex propagators for `alldifferent`$([x_1, \ldots, x_n])$ (see the survey [15]) that implement more complex rules based on Hall sets [4]. A *hall set* H is a subset of $\{x_1, \ldots x_n\}$

such that $|H| \geqslant |S|$ where $S = \cup_{v \in H} D(v)|$. If $|H| > |S|$ the propagation rule is

$$\wedge_{v \in H} \wedge_{d \in D_{init}(v) - S} v \neq d \rightarrowtail \mathit{false}$$

If $|H| = |S|$ the propagation rules are for each $v' \in \{x_1, \ldots, x_n\} - H$ and $d' \in S$

$$\wedge_{v \in H} \wedge_{d \in D_{init}(v) - S} v \neq d \rightarrowtail v' \neq d'$$

The domain propagation of Regin [12] implements all propagation rules for all possible Hall sets. Given there are exponentially many Hall sets, these stronger propagators do not necessarily lead to advantage in lazy clause generation.

5.2 Quasigroup Completion Problems

A $n \times n$ *latin square* is a square of values $x_{ij}, 1 \leqslant i, j \leqslant n$ where each number $[\,1\,..\,n\,]$ appears exactly once in each row and column. It is represented by constraints

$$\begin{array}{ll} \texttt{alldifferent}([x_{i1}, \ldots, x_{in}], & 1 \leqslant i \leqslant n \\ \texttt{alldifferent}([x_{1j}, \ldots, x_{nj}], & 1 \leqslant j \leqslant n \end{array}$$

The quasigroup completion problem (QCP) is a latin square problem where some of the x_{ij} are given. These are challenging problems which exhibit phase transition behaviour. We use examples from the 2006 Constraint Satisfaction Solver Competition [2].

Table 1 compares the user time for finding the first solution of quasigroup completion problems of size 15×15 for various modelling possibilities. The choices are 3 letter codes: **e**ager or **l**azy modelling, **b**ounds or **f**ull integer representation, and **b**ounds (f_b), **d**omain (f_d) or **r**eified (F_r) propagators for representing disequality. Note that for the eager approach with bound variable representation the clauses for the bound and domain propagator are exactly the same, and thus we write **eb(bd)** to denote **ebb** and **ebd**. We also compare against Gecode [3]. For eager modelling the time for constructing the clausal representation is included, it is either 0.01 or 0.02 seconds. The benchmarks 0–9 are satisfiable while 10–14 are unsatisfiable. We omit **lbr** from the tables, since they are not competitive for these benchmarks.

The eager approaches are best for these examples, while the **lfd** combination is the best lazy approach. This is interesting as the bounds representation is quite poor for the lazy approach, but better than the domain representation for the eager approach. The larger the search required the poorer Gecode performs in comparison to the SAT/hybrid approaches.

Table 2 shows the results on 25×25 QCP problems in order to see the trend for modelling choices as size increases. These problems are hard for Gecode, taking hours to complete. In 6 out of 15 instances **lfd** improves upon the eager approach **efd**, and overall it solves the whole suite faster. Even though QCP problems are small (the cost of eager clause generation is less than 0.10 seconds) the lazy approach avoids the overhead of examining many useless clauses, and hence starts outperforming the eager approach as the problem size grows. Interestingly **eb(bd)** is still better than the lazy approach **lfd** for these problems, even though the lazy bounds representations are poor. Examining the novel combination **lbd** we see that for 2 instances it gives the best results, and it suffers significantly greater overhead because it has to check for duplicate clauses. With a dedicated systems for duplicate clause checking it could be improved further.

Table 3 shows the search space for each approach. While **lfd** has the overhead of propagator execution compared to **efd** and **eb(bd)** it usually requires less search, since only the used clauses are counted for the search heuristic. Clearly there is an overhead for the full integer representation. When **lbd** leads to around the same search space as **lfd** it is twice as fast.

5.3 Magic Squares Problems

A $n \times n$ *magic square* is a square of values $x_{ij}, 1 \leqslant i, j \leqslant n$ where each number in $\left[\,1\,..\,n^2\,\right]$ occurs exactly once and each row, column and major diagonal adds to the same number ($s = n(n^2+1)/2$). It is represented by one `alldifferent` constraint, and $2n + 2$ linear equations.

In Table 4 we compare various modelling choices for magic square problems, for finding the first solution (F) and all solutions (A)(for small problems). The $*$ entries arise since the eager approach **eb(bd)** could not search for all solutions (A) since this required modifying the SAT solver.

For these problems, the first fail search strategy of Gecode is clearly much better than the VSIDS search used by our hybrid. The eager modelling approach quickly fails since just the generation of the clauses for $\sum_{i=1}^{n} x_{ij} = s$ requires more than 400 seconds. The additional variables $[\![x = d]\!]$ in the full integer variable representation cause too much overhead for these example, the bounds representations are clearly superior. Of these the hybrid disjunctive propagator performs well. Interestingly **lbr** which has the same propagation strength as **lbb** is superior on the harder problems. This may be because nogoods can make use of the Boolean reification variables to record more pertinent information about failures.

5.4 CELAR Radio Link Frequency Assignment Problems

The CELAR Radio Link Frequency Assignment Problems [1] consist of a set of radio frequencies and a set of radio links to assign a frequency to each radio link. Some pairs of radio links must be an exact distance apart in frequency, while other should be at least some distance apart. We use the first 5 problems (where all constraints are mutually satisfiable) while minimizing the maximum frequency used. The set of possible frequencies F is non-continuous:

$$\begin{array}{c} \{2 + 14i | 1 \leqslant i \leqslant 11\} \cup \{2 + 14i | 18 \leqslant i \leqslant 28\} \\ \cup \{8 + 14i | 29 \leqslant i \leqslant 30\} \cup \{8 + 14i | 46 \leqslant i \leqslant 56\}, \end{array}$$

Table 1: QCP 15×15 instances: user time

Benchmark	Time(sec)					
	efd	eb(bd)	lfd	lbd	lbb	gecode
qcp-15-120-0_ext	0.05	0.02	0.03	0.14	0.64	0.02
qcp-15-120-1_ext	0.04	0.04	0.06	0.22	0.74	0.08
qcp-15-120-2_ext	0.08	0.02	0.05	0.16	0.74	454.53
qcp-15-120-3_ext	0.05	0.04	0.14	0.26	0.84	0.19
qcp-15-120-4_ext	0.20	0.02	0.02	0.33	0.65	5.50
qcp-15-120-5_ext	0.15	0.09	0.21	0.62	2.52	117.08
qcp-15-120-6_ext	0.04	0.02	0.01	0.17	1.01	38.01
qcp-15-120-7_ext	0.11	0.13	0.29	0.24	0.97	1.28
qcp-15-120-8_ext	0.05	0.10	0.04	0.18	0.76	6.70
qcp-15-120-9_ext	0.08	0.14	0.24	0.27	0.78	1685.44
qcp-15-120-10_ext	0.06	0.04	0.04	0.20	0.55	1044.80
qcp-15-120-11_ext	0.03	0.05	0.01	0.32	0.41	47.64
qcp-15-120-12_ext	0.03	0.01	0.02	0.04	0.41	862.29
qcp-15-120-13_ext	0.16	0.30	0.17	0.21	1.57	179.18
qcp-15-120-14_ext	0.02	0.01	0.01	0.01	0.62	2034.72
Arith mean	0.08	0.07	0.09	0.22	0.88	431.83
Geom mean	0.06	0.04	0.05	0.17	0.78	24.67

Table 2: QCP 25×25: user time

Benchmark	Time(sec)				
	efd	eb(bd)	lfd	lbd	lbb
qcp-25-264-0_ext	114.07	65.56	149.88	85.89	242.73
qcp-25-264-1_ext	832.31	108.37	99.84	374.77	1346.06
qcp-25-264-2_ext	15.40	44.40	12.25	47.34	144.92
qcp-25-264-3_ext	542.61	273.36	442.57	532.47	1655.22
qcp-25-264-4_ext	265.00	268.84	24.87	418.33	1136.17
qcp-25-264-5_ext	108.60	146.36	341.25	158.62	4810.77
qcp-25-264-6_ext	255.60	185.53	130.06	127.91	871.80
qcp-25-264-7_ext	35.36	1.52	34.07	78.26	269.61
qcp-25-264-8_ext	9.52	48.36	81.10	171.35	998.53
qcp-25-264-9_ext	27.80	153.52	286.20	710.96	1043.52
qcp-25-264-10_ext	30.92	125.67	165.77	346.78	631.13
qcp-25-264-11_ext	0.14	0.06	0.10	0.17	7.16
qcp-25-264-12_ext	0.23	0.21	0.24	0.32	11.90
qcp-25-264-13_ext	0.36	0.29	0.34	0.34	9.83
qcp-25-264-14_ext	107.82	131.88	175.01	176.97	901.36
Arith mean	156.38	103.60	129.57	215.37	938.71
Geom mean	26.40	23.75	30.31	53.07	326.03

using only 44 values in the range $[\,16\,..\,792\,]$ of 777 possible values. We model the problem using bounds propagators for $|x - y| \geqslant k$ (see Example 13), and model $|x - y| = k$ using the bounds propagators for $|x-y| \geqslant k \wedge x-y \leqslant k \wedge y-x \leqslant k$.

We compare the **f**ull integer representation, **n**on-continuous representation, **b**ounds representation, and n**o**n-continuous bounds representation. For the full integer representation we statically add constraints $\neg[\![x = d]\!], d \in [\,16\,..\,792\,] - F$ to the SAT solver, while for the bounds representation we statically add the constraints $\neg[\![x \leqslant d_i]\!] \vee [\![x \leqslant d_{i+1}]\!]$ where d_i and d_{i+1} are consecutive values in F. We also compare with Gecode using reified constraints to represent $|x - y| \geqslant k$ as $x - y \geqslant k \vee y - x \geqslant k$.

The results for the various modelling choices are shown for: user time in Table 6, failures in Table 7, and unit propagation executed in Table 8. Clearly the non-continuous representations are significantly better than the continuous representations, they involve around $20\times$ fewer variables. The failure results show that it is not the results of a better search because there are fewer Boolean variables to branch on, instead it is simply the overhead of more unit propagations to deal with the larger number of variables.

Table 6: CELAR problems: user time

Prob	User Time(sec)				
	lfb	lnb	lbb	lob	gecode
scen01	285.22	13.67	104.65	9.37	> 400
scen02	2.03	0.16	0.86	0.11	> 400
scen03	39.90	3.16	20.06	2.19	> 400
scen04	2.17	0.16	0.88	0.10	0.46
scen05	2.25	0.17	0.96	0.10	0.34

This clearly shows the benefit of separation of propagator implementation from variable representation. The propagator is highly effective on

Table 3: QCP 25×25: conflicts (000s)

Benchmark	Conflicts/Failures				
	efd	eb(bd)	lfd	lbd	lbb
qcp-25-264-0_ext	212	117	159	174	588
qcp-25-264-1_ext	1037	178	119	626	2498
qcp-25-264-2_ext	44	99	29	125	463
qcp-25-264-3_ext	814	393	399	892	3284
qcp-25-264-4_ext	417	405	42	760	2424
qcp-25-264-5_ext	210	256	325	345	7890
qcp-25-264-6_ext	397	282	161	273	1701
qcp-25-264-7_ext	84	9.6	60	178	631
qcp-25-264-8_ext	30	96	102	352	2142
qcp-25-264-9_ext	70	261	291	1301	2307
qcp-25-264-10_ext	76	226	182	709	1503
qcp-25-264-11_ext	0.2	0.2	0.3	0.7	11
qcp-25-264-12_ext	1.6	3.1	2.1	2.8	48
qcp-25-264-13_ext	4.1	4.1	3.9	3.1	31
qcp-25-264-14_ext	192	208	170	352	1832
Arith mean	239	169	136	406	1824
Geom mean	64	61	53	137	736

Table 4: Magic squares: user time

nT	User Time(sec)					
	eb(bd)	lfd	lbd	lbb	lbr	gecode
3F	0.16	0.00	0.00	0.00	0.00	0.00
3A	*	0.00	0.00	0.00	0.00	0.00
4F	8.92	0.04	0.04	0.01	0.16	0.01
4A	*	866.38	745.87	810.84	803.09	2.26
5F	> 400	307.15	1.04	1.19	0.79	0.81
6F	> 400	31.87	0.39	99.92	17.50	0.00
7F	> 400	> 400	> 400	> 400	> 400	5.25

Table 7: CELAR problems: Conflicts/Failures

Prob	Conflicts/Failures				
	lfb	lnb	lbb	lob	gecode
scen01	5036	4542	4160	4247	—
scen02	202	127	180	261	—
scen03	3039	2380	2667	2553	—
scen04	7	6	2	1	31
scen05	17	22	36	24	74

Table 8: CELAR problems: unit propagations

Prob	Unit Propagations			
	lfb	lnb	lbb	lob
scen01	177561515	13081789	133403108	7133763
scen02	1969516	183084	1732660	112612
scen03	43087573	3608960	38598246	1918102
scen04	628192	36289	304949	17368
scen05	901257	65516	1375927	47145

the non-continuous Boolean representations without being modified.

Interestingly for these problems the disjunctive propagator explained in Example 13 does not improve upon the bounds propagator.

6 Related Work and Conclusion

The motivating earlier work for the lazy clause generation approach was twofold.

The paper [5] described a hybrid binary decision diagram (BDD) and SAT solver for solving problems involving set variables, which used the SAT solver as nogood engine for a BDD propagation solver. The hybrid leaves control of search to the BDD solver, and does not include integer variables. Lazy clause generation imbeds the propagation engine in the SAT solver and puts the SAT solver in charge of search. Set variables have only a single possible Boolean representation so the modelling choices we explore here do not arise.

The paper [14] explained how to statically encode linear arithmetic constraints into CNF (hence eager modelling) using the propositions $[\![x \leqslant d]\!]$. The approach is manifestly impractical when the linear constraint involves a significant number of variables (as illustrated by e.g. magic squares 5). The lazy clause generation approach makes the encoding of linear arithmetic possible for large linear constraints, and allows encoding of arbitrary propagators.

There are propagation solvers which allow different representation of integers, in particular Minion [8] and Gecode [3]. All representations either support all atomic constraints or are restricted in the propagators they can be used. The views ap-

Table 5: Magic squares: conflicts

nT	Conflicts/Failure					
	eb(bd)	**lfd**	**lbd**	**lbb**	**lbr**	**gecode**
3F	15	3	9	14	12	6
3A	*	31	34	43	34	36
4F	51	569	560	160	1326	892
4A	*	1000776	898347	1050572	869813	235545
5F	—	201531	7578	8149	3212	72227
6F	—	53332	2991	137545	21644	27
7F	—	—	—	—	—	481301

proach of Gecode [13] allows variables defined by simple constraints to be seen as mappings from atomic constraint to atomic constraints, and hence has some similarity with the mapping idea of this paper. For example a variable $y = x+3$ effectively rewrites atomic constraint like $x \geqslant 4$ to $y \geqslant 6$ and vice versa. It would be useful to include views in the lazy clause generation solver, since it reduces the number of Boolean variables required.

In this paper we examine the modelling choices that arise when using the lazy clause generation hybrid solving approach devised in [11]. We find that the separation of choice of propagator from Boolean variable representation leads to an increased number of modelling choices. The direct representation of non-continuous variables is clearly advantageous, and there is some evidence that the use of disjunctive propagators (domain propagators for bounds variables) can improve upon other modelling approaches.

References

[1] B. Cabon, S. de Givrey, L. Lobjois, T. Schiex, and L.P. Warners. Radio link frequency assignment. *Constraints*, 4(1):78–89, 1999.

[2] CSP competition 2006. `http://cpai.ucc.ie/06/Competition.html`. [Jun07].

[3] GECODE. `www.gecode.org`. [Feb07].

[4] P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 10:26–30, 1935.

[5] P. Hawkins and P.J. Stuckey. A hybrid BDD and SAT finite domain constraint solver. In P. Van Hentenryck, editor, *Proceedings of the Practical Applications of Declarative Programming*, number 3819 in LNCS, pages 103–117. Springer-Verlag, 2006.

[6] P. Laborie. Complete MCS-based search: Application to resource constrained project scheduling. In *Proceedings IJCAI 2005*, pages 181–186, 2005.

[7] Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[8] Minion. `minion.sourceforge.net`. [Feb07].

[9] MiniSat. `www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/`. [Dec06].

[10] R. Niewenhuis, A. Oliveras, and C. Tinelli. Abstract DPLL and abstract DPLL modulo theories. In *LPAR'04*, volume 3452 of *LNAI*, pages 36–50, 2004.

[11] O. Ohrimenko, P.J. Stuckey, and M. Codish. Propagation = lazy clause generation. In C. Bessiere, editor, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, LNCS, page to appear. Springer-Verlag, 2007.

[12] J-C. Regin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, volume 1, pages 362–367, Seattle, WA, USA, 1994. AAAI Press.

[13] Guido Tack, Christian Schulte, and Gert Smolka. Generating propagators for finite set constraints. In Fréderic Benhamou, editor, *12th International Conference on Principles and Practice of Constraint Programming*, volume 4204 of *Lecture Notes in Computer Science*, pages 575–589. Springer, 2006.

[14] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara. Compiling finite linear CSP to SAT. In *Proceedings of CP-2006*, volume 4204 of *LNCS*, pages 590–603, 2006.

[15] W.J. van Hoeve. The alldifferent constraint: a survey. `http://arxiv.org/abs/cs/0105015`, 2001.

The Core Concept for 0/1 Integer Programming

Sam Huston[2] **Jakob Puchinger**[1,2] **Peter Stuckey**[1,2]

[1] NICTA Victoria Laboratory
[2] Department of Computer Science and Software Engineering,
University of Melbourne, Victoria 3010, Australia,
Email: {shuston,jakobp,pjs}@csse.unimelb.edu.au

Abstract

In this paper we examine an extension of the core concept for the 0/1 Multidimensional Knapsack Problem (MKP) towards general 0/1 Integer Programming (IP) by allowing negative profits, weights and capacities. The core concept provides opportunities for heuristically solving the MKP, achieving higher quality solutions and shorter runtimes than general IP methods. We provide the theoretical foundations of the extended core concept and further provide computational experiments showing that we can achieve similar computational behavior for extended MKP instances with negative weights, profits and capacities.

1 Introduction

The core concept for the 0/1 Multidimensional Knapsack Problem (MKP) (Puchinger et al. 2006, 2007) has been shown to be very effective in providing opportunities for heuristically solving the MKP, achieving higher quality solutions and shorter run-times than general IP methods. In this paper we will examine the possibilities of extending the core concept towards general 0/1 Integer Programming (IP).

The Multidimensional Knapsack Problem (MKP) is defined as:

$$\text{maximize} \quad z = \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_{ij} x_j \leq c_i, \quad i = 1, \ldots, m \tag{2}$$

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, n., \tag{3}$$

where the profits p_j, the weights w_{ij}, and the capacities c_i are all positive. Allowing negative values for those parameters results in general 0/1 Integer Problems. This is because it is possible to transform any 0/1 IP into this format. (Bertsimas & Tsitsiklis 1997)

The aim of the core concept is to reduce the original problem to a *core* of items for which it is hard to decide whether or not they will occur in an optimal solution. All variables corresponding to items outside the core are fixed to their optimal values.

The *core* of a given MKP is defined with respect to some ordering of the variables in the problem. The ordering results in variables that are expected to be in the knapsack (set to one) occur before those which are not expected to be in the knapsack (set to zero). Given the optimal solution of the MKP the exact core is defined as the set of variables from the first variable that takes the value zero to the last variable that takes the value one. In order to devise an exact core for a given MKP, the optimal solution has to be known. However, being able to heuristically obtain good approximations to cores and solve those smaller problems, may lead to high quality solutions in short computational times.

The underlying concept of such a heuristic is to order the items of the MKP according to a specific efficiency measure. This ordering will allow to partition the items into three sections. The first section which contains items which are included in the knapsack (variables set to one). The second section, named the *approximate core*, containing the items which may or may not be included in the knapsack (variables set to either one or zero). Finally the third section contains the items which are not included in the knapsack (variables set to zero). The aim is to have the approximate core closely mimic the exact core.

The following example illustrates the core concept for a small 2-dimensional knapsack problem. The variables are ordered by an efficiency measure described later. The first line shows the optimal integer solution, while the second line shows the optimal solution to the LP-relaxation of the problem. The exact core is shown in bold in the first line, while an approximate core (adding 2 variables around the non 0-1 LP solution values) is shown in bold in the second line.

IP	1	1	**0**	**0**	**1**	**0**	**1**	0	0	0
LP	1	1	1	**1**	**1**	**0.96**	**0.23**	**0**	**0**	0

In the remainder of the paper, we first introduce the core concept in the context of its previous uses. We then extend the efficiency measure used for MKPs to general 0/1 Integer Programs, and prove that the efficiency measure is tightly related to the optimal solution. In Section 4 we show the result of experiments illustrating the effectiveness of the approximate core computations, the loss of precision that arises from restricting the problem to an approximate core, and the improvement in best solutions found if we use approximate cores.

2 Background

2.1 The Multidimensional Knapsack Problem

A comprehensive overview of practical and theoretical results for the MKP can be found in the monograph on knapsack problems by Kellerer et al. (Kellerer et al. 2004). Solving the MKP with heuristic methods seems to be the method of choice for the bigger instances described in the literature, since no exact method is known for solving these instances to optimality. Besides exact techniques for solving small to moderately sized instances, based on dynamic programming (Gilmore & Gomory 1966, Weingartner & Ness 1967) and branch-and-bound (Shih 1979, Gavish & Pirkul 1985), many kinds of meta-heuristics have been applied to the MKP (Glover & Kochenberger 1996, Chu & Beasley 1998). See (Raidl & Gottlieb 2005) for a recent survey and comparison of evolutionary algorithms for the MKP. The hybrid tabu-search methods presented in (Vasquez & Hao 2001, Vasquez & Vimont 2005) are currently yielding the best known results for the commonly used benchmark instances.

2.2 The core concept for KP and MKP

The core concept was first presented for the classical 0/1-Knapsack Problem (KP) (Balas & Zemel 1980) and led to very successful knapsack algorithms (Martello & Toth 1988, Pisinger 1995, 1997). These ideas were also studied in the context of bi-criteria knapsack problems in (Gomes da Silva et al. 2005). The core concept was successfully extended to the MKP (Puchinger et al. 2006, 2007), leading to highly competitive heuristic algorithms.

It should be noted here that the KP core concept is not effective for producing good heuristic solutions for strongly correlated problem instances. Pisinger (Pisinger 1995) discusses why these problems are difficult to solve using the core concept. We would expect similar results for strongly correlated general 0/1 IP problems.

The classical greedy heuristic for KP packs the items into the knapsack in decreasing order of their *efficiencies* $\frac{p_j}{w_j}$ as long as the knapsack constraint is not violated. The same ordering also defines the solution structure of the LP-relaxation, which consists of three parts: The first part contains all variables set to one, the second part consists of at most one *split item* (s), whose corresponding LP-value is fractional, and finally the remaining variables, which are always set to zero, form the third part.

The precise definition of the core of KP introduced by (Balas & Zemel 1980) requires the knowledge of an optimal integer solution x^*. Assume that the items are sorted according to decreasing efficiencies and let

$$a := \min\{j \mid x_j^* = 0\}, \qquad b := \max\{j \mid x_j^* = 1\}. \tag{4}$$

The core is given by the items in the interval $C = \{a, \ldots, b\}$. It is obvious that the split item is always part of the core. If the split item would not be part of the core, the core would either start after the split item or end before it. The first case is impossible, since this would break the capacity constraint. The second case would contradict the optimality of the solution, because we would still be able to add more items to the knapsack without violating the capacity constraint.

These ideas have been expanded to MKP without major difficulties (Puchinger et al. 2006, 2007). The main difference is that the choice of the efficiency measure is not obvious for the MKP any more. The efficiency measure:

$$e_j = \frac{p_j}{s_j},$$

where $s_j = \sum_{i=1}^{m} u_i w_{ij}$, and u_i are the dual variable values of the LP-relaxation of the MKP, provided the best theoretical and practical results.

Let x^* be an optimal solution and assume that the items are sorted according to the decreasing efficiency measure e, then define

$$a := \min\{j \mid x_j^* = 0\} \quad \text{and} \quad b := \max\{j \mid x_j^* = 1\}. \tag{5}$$

The core is given by the items in the interval $C := \{a, \ldots, b\}$, and the core problem (MKPC) is defined as

$$\text{maximize} \quad z = \sum_{j \in C} p_j x_j + \tilde{p} \tag{6}$$

$$\text{subject to} \quad \sum_{j \in C} w_{ij} x_j \leq c_i - \tilde{w}_i, \; i = 1, \ldots, m \tag{7}$$

$$x_j \in \{0, 1\}, \quad j \in C, \tag{8}$$

with $\tilde{p} = \sum_{j=1}^{a-1} p_j$ and $\tilde{w}_i = \sum_{j=1}^{a-1} w_{ij}$, $i = 1, \ldots, m$.

In contrast to KP, the solution of the LP-relaxation of MKP does not consist of a single fractional split item, but its up to m fractional values give rise to a whole *split interval* $S := \{s, \ldots, t\}$, where s and t are the first and the last index of variables with fractional values after sorting by efficiency e.

The split interval S_e has been precisely characterized. Let x^{LP} be the optimal solution of the LP-relaxation of MKP.

Theorem 1 ((Puchinger et al. 2006, 2007)) *For efficiency values e_j we have:*

$$x_j^{\mathrm{LP}} = \begin{cases} 1 & \text{if } e_j > 1\,, \\ \in [0,1] & \text{if } e_j = 1\,, \\ 0 & \text{if } e_j < 1\,. \end{cases} \tag{9}$$

3 Extending the core concept

As mentioned above, the main goal of this paper is to extend the core concept to general 0/1 Integer Programs. We show how the efficiency measure for the classical MKP problem can be adapted in such a way that the ordering of the variables according to this measure remains valuable for devising good approximate cores. We further provide a characterization of the structure of the LP relaxation of the 0/1 Integer Program.

We take all 0/1 IP problems to be transformable into the same structure as MKP, see equation 1.

Such a transformation is possible for any 0/1 IP, see any linear programming text book, (e.g. (Bertsimas & Tsitsiklis 1997)). This means that the only difference between MKP and this formulation of 0/1 IP problems is that the coefficients are permitted to take either positive or negative values.

We extend the previously defined efficiency measure e_j to create a tuple based measure. We introduce ordering variables, o_j which take values representing a section of the ordering. Variables x_j are sorted in decreasing (lexicographic) order of efficiency (o_j, e_j), in other words they are first sorted by section variable, o_j, and then efficiency value e_j.

These extensions are implemented as indicated in equation 10:

$$(o_j, e_j) = \begin{cases} (7, \frac{p_j}{s_j}) & \text{if } p_j > 0 \wedge s_j < 0 \\ (6, p_j) & \text{if } p_j > 0 \wedge s_j = 0 \\ (5, \frac{1}{s_j}) & \text{if } p_j = 0 \wedge s_j < 0 \\ (4, \frac{p_j}{s_j}) & \text{if } p_j > 0 \wedge s_j > 0 \\ (4, \frac{s_j}{p_j}) & \text{if } p_j < 0 \wedge s_j < 0 \\ (4, 1) & \text{if } p_j = 0 \wedge s_j = 0 \\ (3, \frac{1}{s_j}) & \text{if } p_j = 0 \wedge s_j > 0 \\ (2, p_j) & \text{if } p_j < 0 \wedge s_j = 0 \\ (1, \frac{p_j}{s_j}) & \text{if } p_j < 0 \wedge s_j > 0 \end{cases} \tag{10}$$

The ordering is designed to minimize the size of the split interval, which is completely contained in section $o_j = 4$. Our experiments in Section 4 show that the center of the core and the center of the split interval are close for our benchmark instances.

The sections $o_j \in \{7, 6, 5\}$ contain items that are purely beneficial: they either increase profit or "on average" remove weight from the optimal solution without decreasing profit. They are ordered to maximize profit and removal of weight. Similarly the sections $o_j \in \{1, 2, 3\}$ contain items that are purely detrimental to the problem: they either decrease profit or add weight. Again they are ordered to maximize profit and removal of weight.

Using this efficiency measure the nature of the split interval can be characterized as follows. Let x^{LP} be the optimal solution of the LP-relaxation of general 0/1 IP.

Theorem 2

$$x_j^{LP} = \begin{cases} 1 & \text{if } e_j > 1 \text{ or } o_j > 4, \\ \in [0,1] & \text{if } e_j = 1 \text{ and } o_j = 4, \\ 0 & \text{if } 0 \le e_j < 1 \text{ or } o_j < 4. \end{cases} \tag{11}$$

Proof The dual LP (DLP) associated with the LP-relaxation of the general 0/1 IP formulation is given by:

$$\text{minimise} \sum_{i=1}^{m} c_i u_i + \sum_{j=1}^{n} v_j \tag{12}$$

$$\text{subject to} \sum_{i=1}^{m} w_{ij} u_i + v_j \ge p_j, j = 1, ..., n \tag{13}$$

$$u_i, v_j \ge 0, i = 1, ..., m, j = 1, ..., n, \tag{14}$$

where u_i are the dual variables corresponding to the problem's constraints and each v_j corresponds to the inequality $x_j \le 1$. For the optimal primal and dual solutions the following complementary slackness conditions hold for $j = 1, ..., n$. .

$$x_j \left(\sum_{i=1}^{m} w_{ij} u_i + v_j - p_j \right) = 0 \tag{15}$$

$$v_j \left(x_j - 1 \right) = 0 \tag{16}$$

(For more details on linear programming duality and complementary slackness conditions, refer to any textbook on linear programming, e.g.(Bertsimas & Tsitsiklis 1997).)

We illustrate the result for each section in the definitions of the ordered tuple (o_j, e_j).

Consider the top 3 sections ($o_j \in \{7, 6, 5\}$). Clearly in each case we have that $p_j > s_j$. Hence satisfying equation (13) requires that $v_j > 0$. Therefore equation (16) implies that $x_j = 1$.

Consider the bottom 3 sections ($o_j \in \{1, 2, 3\}$). Clearly in each case $p_j < s_j$. Hence the expression $\sum_{i=1}^{m} w_{ij} u_i + v_j - p_j$ or equivalently $s_j + v_j - p_j$ is greater than 0, since $v_j \ge 0$. In order to satisfy equation 15 $x_j = 0$.

For section $(4, \frac{p_j}{s_j})$ we consider two cases. If $e_j = \frac{p_j}{s_j} > 1$ then $p_j > s_j$ since $s_j > 0$ and the same reasoning as for cases $o_j \in \{7, 6, 5\}$ applies, while if $e_j < 1$ then $p_j < s_j$ and the same reasoning as cases $o_j \in \{1, 2, 3\}$ applies.

For section $(4, \frac{s_j}{p_j})$ we consider two cases. If $e_j = \frac{s_j}{p_j} > 1$ then $p_j > s_j$ since $p_j < 0$ and the same reasoning as for cases $o_j \in \{7, 6, 5\}$ applies,

while if $e_j < 1$ then $p_j < s_j$ and the same reasoning as cases $o_j \in \{1, 2, 3\}$ applies.

For the remaining case, $e_j = 1$ and $o_j = 4$, there is nothing to prove.

□

The ordering within the top and bottom groups $o_j \in \{1, 2, 3\}$, and $o_j \in \{5, 6, 7\}$ is not set by the proof, these orderings are based upon maximizing profit. Other possible orderings could be considered for these groups without changing our characterization of the split interval.

An illustration of the ordering (o_j, e_j) is given in Table 1. This example shows some of the sections described above. As predicted by the theorem, the split interval exists entirely within section $o_j = 4$.

4 Computational experiments

4.1 Benchmark Problems

All of the following computational experiments were performed on a 3GHz Intel Pentium D with 4 Gb RAM, using the programming language Mercury and the commercial mixed integer programming solver CPLEX 10.0.

In order to study the core concept on 0/1 IP we generated example problems using the Chu and Beasley (Chu & Beasley 1998) benchmark instances for the MKP, as the starting point.

These benchmark problems consist of classes of randomly created instances for each combination of $n \in \{100, 250, 500\}$ items, $m \in \{5, 10, 30\}$ constraints and tightness ratios $0.25, 0.50.75$. The tightness ratio refers to the ratio between the constraint value and the sum of the corresponding weights.

$$\alpha = \frac{c_i}{\sum_{j=1}^{n} w_{ij}} \in \{0.25, 0.5, 0.75\} \qquad (17)$$

Weights are integers between 0 and 1000. Profits are generated by the equation:

$$p_j = \sum_{i=1}^{m} \frac{w_{ij}}{m} + [500 r_i] \in \{0.25, 0.5, 0.75\}$$

where r_i is a random number generated from $(0, 1]$. For each permutation of n, m and α, 10 instances are provided.

In order to generate 0/1 Integer Programs with negative values we multiply a random percentage of the weights and profits of a given problem by -1. The percentages used are $5\%, 10\%, 20\%$. Larger percentages of negative values were tried, however the problems quickly became optimally solvable in short run-times.

The set of profits and the set of all weights are operated on separately. This ensures that there is a fixed percentage of negative profits and a fixed percentage of negative weights. Combinations of different percentages of negative weights and profits were tried, however almost invariably this made the problems easier and thus faster to solve.

This process will change the tightness ratio. In order to maintain the tightness ratio the capacities have to be adjusted:

$$\hat{c}_i = \frac{c_i * \sum_{j=1}^{n} \hat{w}_{i,j}}{\sum_{j=1}^{n} w_{i,j}},$$

$w_{i,j}$ represents the original weights, $\hat{w}_{i,j}$ represents the adjusted weights, and $\hat{c}_i$ represents the new constraint value. Since the sum of the adjusted weights may become negative, it is possible that the new capacity $\hat{c}_i$ will also be negative.

It can be seen that the generated problems are general 0/1 IP problems. There are nine classes of generated problems for each combination of n and m, corresponding to three different tightness ratios, and the three percentages of negative coefficients.

4.2 0/1 IP Core Analysis

We provide empirical results supporting our adaptation of the efficiency measure e, (see Table 2). This table shows information about actual cores when the above efficiency function is utilized. The problems shown in these tables are based upon the smaller instances in Chu and Beasley's benchmark library (Chu & Beasley 1998). Specifically these problems use $n = 100, m \in \{5, 10\}$, and $n = 250, m = 5$. These problems were chosen because they are solvable in reasonable runtime. This means that the optimal solutions can be found, and the size of the core can be determined.

The tables show the averaged values over 10 problem instances. Average values listed include size of the split interval ($|S_e|$), size of the exact core ($|C_e|$), percentage that the split interval covers the exact core (ScC), percentage that the exact core covers the split interval (ScC), and the distance between the center of the split interval and the center of the exact core ($|C_{dist}|$) as a percentage of the number of items in the problem.

The table entries for 0% negative coefficients shows that the newly defined efficiency value provides equivalent results for standard MKP problems as those reported in (Puchinger et al. 2006, 2007). As expected from Theorem 2, negative values do not increase the size of the split interval. The size of the split interval and the core actually decreases as the number of negative weights increases. The center of the core remains close to the center of the split interval. These results show that the chosen ordering, based on the optimal dual variable values of the LP-relaxation, is a good indicator of the actual location of the core.

4.3 Approximate core algorithm

In order to evaluate the influence of negative values on solution quality and run-times an approximate core algorithm was implemented. This algorithm is similar to the algorithm implemented by (Puchinger et al. 2006, 2007). The approximate core is generated by adding δ items to either side of the center of the split interval. The values of

$Weight_1$	$Weight_2$	Profit	s_j	(o_j, e_j)	LP	IP
-1	-1	1	-0.89	(7,0.89)	1.00	1
-9	12	7	-5.17	(7,0.74)	1.00	1
0	0	8	0.00	(6,8.00)	1.00	1
-2	13	12	0.24	(4,50)	1.00	1
5	0	19	3.77	(4,5.04)	1.00	1
-5	-4	-3	-4.31	(4,1.44)	1.00	1
16	21	18	14.88	(4,1.21)	1.00	1
13	15	14	11.81	(4,1.19)	1.00	**0**
-6	-10	-5	-5.87	(4,1.17)	1.00	**1**
20	-8	14	14.00	(4,1.00)	**0.71**	**1**
-6	-11	-6	-6.00	(4,1.00)	**0.85**	0
20	16	15	17.22	(4,0.87)	0.00	0
10	8	6	8.61	(4,0.70)	0.00	0
-1	-14	-4	-2.63	(4,0.66)	0.00	0
14	9	5	11.76	(4,0.43)	0.00	0
16	-3	3	11.66	(4,0.26)	0.00	0
-7	23	-9	-2.19	(4,0.24)	0.00	0
-1	10	0	0.59	(3,1.69)	0.00	0
7	0	-5	5.28	(1,0.95)	0.00	0
24	10	-9	19.43	(1,0.46)	0.00	0

Table 1: Example 2-dimensional 0/1 IP problem. The 3 sections are separated based upon the IP solution. This example shows an exact core using the efficiency measure defined in Theorem 2.

			e - 0% negative weights 0% negative profits					e - 5% negative weights 5% negative profits				
n	m	α	$\lvert S_e \rvert$	$\lvert C_e \rvert$	ScC	CcS	C_{dist}	$\lvert S_e \rvert$	$\lvert C_e \rvert$	ScC	CcS	C_{dist}
100	5	0.25	5.00	20.20	28.12	100.00	3.30	5.00	20.00	31.58	100.00	2.90
		0.5	5.00	22.10	27.49	100.00	3.45	5.00	15.90	28.33	86.00	2.65
		0.75	5.00	20.00	26.32	100.00	3.40	5.00	14.80	35.91	98.00	3.50
250	5	0.25	2.00	12.68	18.16	100.00	2.46	2.00	13.36	17.35	100.00	3.12
		0.5	2.00	12.20	18.45	100.00	1.38	2.00	9.60	21.47	100.00	1.20
		0.75	2.00	10.40	20.18	100.00	1.56	2.00	10.96	21.04	100.00	1.92
100	10	0.25	10.00	23.20	46.57	100.00	2.90	9.90	25.80	42.74	96.67	3.45
		0.5	9.80	25.80	48.17	96.00	3.10	9.70	23.70	44.06	100.00	3.00
		0.75	9.70	18.30	54.36	94.00	3.00	9.20	16.90	60.09	93.19	2.45
Average			5.61	18.32	31.98	98.89	2.73	5.53	16.75	33.62	97.10	2.69
			e - 10% negative weights 10% negative profits					e - 20% negative weights 20% negative profits				
n	m	α	$\lvert S_e \rvert$	$\lvert C_e \rvert$	ScC	CcS	C_{dist}	$\lvert S_e \rvert$	$\lvert C_e \rvert$	ScC	CcS	C_{dist}
100	5	0.25	5.00	23.60	22.31	100.00	4.30	5.00	18.60	29.55	100.00	2.70
		0.5	5.00	19.40	27.11	100.00	3.00	4.60	9.80	57.68	95.50	1.20
		0.75	4.80	12.40	47.28	98.00	1.60	2.50	16.90	30.34	86.67	7.00
250	5	0.25	2.00	10.36	20.27	100.00	1.22	2.00	10.52	20.22	98.00	2.02
		0.5	2.00	11.72	18.79	100.00	2.22	2.00	8.84	24.31	100.00	1.94
		0.75	2.00	7.28	30.13	98.00	1.24	1.56	6.08	40.80	100.00	1.74
100	10	0.25	9.70	24.20	42.19	97.89	4.05	9.70	28.00	37.91	100.00	4.15
		0.5	9.50	20.10	49.20	97.00	3.20	8.40	20.10	47.86	98.89	3.35
		0.75	8.80	14.30	65.99	92.70	2.05	4.30	13.80	35.51	92.50	4.05
Average			5.42	15.92	35.92	98.18	2.54	4.45	14.74	36.02	96.84	3.13

Table 2: Split intervals, core sizes, mutual coverage of the split interval and cores, distances of the centers for various percentages of negative values. (Values are averaged over 10 instances of each problem.)

δ were chosen to approximately reflect the size of the actual cores detected in the previous section: $\delta \in 0.1n, 0.15n, 0.2n, 0.1n + 2m, 0.2n + 2m$.

The problems shown in these tables are based upon the smaller instances in Chu and Beasley's benchmark library (Chu & Beasley 1998). They are the same set of problems used to investigate the actual core sizes in the previous section.

The results of this experiment are shown in Table 3. It shows the average values over 10 problems with the same tightness ratio. Values shown for the original problem include the average optimal IP solution for the problem ($\bar{z}$), then the average amount of CPU-time taken to produce the optimal IP solution in seconds (t[s]). Values shown for each core include the average percentage difference between the optimal IP solution(z^*) and the IP solution produced by the core problem(z), ($\%_{opt} = 100 * (z^* - z)/z^*$), the number of times the optimal solution was reached (#), and the average CPU-time taken to solve the core IP, as a percentage of the CPU-time taken to solve the original IP problem, $\%t = 100 * (t_{core}/t_{original})$.

The solution to the approximate cores are (on average over 10 problem instances) always within 0.7 % of the optimal solution. The results shown in Table 3 show that smaller approximate core sizes produce a significant increase in speed. However they are less likely to produce the optimal solution, and on average produce solutions of lesser quality than the larger cores. As the percentage of negative values increases the problems become faster to solve. Larger negative percentages were examined, however run-times were too small to see any benefit from the core concept.

4.4 Larger 0/1 IP with Fixed Time Runs

We now investigate fixed-time runs over larger problem instances. These tests are performed over instances which are currently very hard or not at all solvable to optimality. The instances used are based on the hardest benchmarks provided by Chu and Beasley (Chu & Beasley 1998), $n = 500,\ m \in 5, 10, 30$

Again these problems were adjusted to contain negative values in a manner similar to the problems above. All of the results shown here are performed over problems with 10% negative weights, and 10% negative profits. The constraints are also adjusted accordingly.

Table 4 shows the best feasible solution for the original problem and the core problems as a percentage of the LP solution, ($\%_{LP} = 100 * (LP - IP)/LP$). These values are averaged over 10 instances of similar problems. Standard deviations are provided as subscripts. The smallest values for each row are highlighted in bold. This table also shows the number of times a particular core size has lead to the best solution for a problem, (#). The final column for each core size is the average number of nodes explored in the branch and cut tree used by CPLEX.

The experiments show that for the considered time limits the results obtained on the core problems are, on average, better that the results obtained from the original problem. There is also a inverse relationship between the size of the core and the number of nodes explored. As the size of the core decreases the number of nodes explored increases. The best average results for a time limit of 500 seconds is $\delta = 0.2$. It can be seen that smaller time limits provide best results with smaller approximate core sizes.

5 Related Work

The most closely related work to this paper is the application of the core concept to the MKP (Puchinger et al. 2006, 2007). We extend the results therein to general 0/1 Integer Programs, and show that the core concept continues to be valuable in the more general case.

Recently, very interesting results have been achieved with heuristics for 0/1 Mixed Integer Programming Problems with the goal of devising better feasible solutions earlier in the optimization process. Local Branching (Fischetti & Lodi 2003) combines local search and general branch-and-bound by introducing local branching constraints forcing the search to explore the neighborhoods of current feasible solutions first.

In Relaxation Induced Neighborhood Search (RINS) (Danna et al. 2005) subproblems for finding better feasible solutions are solved at some nodes of the branch-and-bound tree. The subproblems are obtained by fixing the variables having identical values in the current best feasible solution and in the current solution of the LP-relaxation, leaving the remaining variables free.

RINS and local branching are local-search based ideas, reducing the subproblems to certain neighborhoods around a currently feasible solution. Our approach requires an LP solution only, and does not make use of feasible solutions at all.

6 Conclusions

We have extended the core concept, previously successfully used for finding better solutions to Multiple Knapsack Problems to general 0/1 Integer Programs. We provided an ordering of the variables using dual information, which results in a compact split interval just as for the standard MKP. This ordering is used to reduce the size of the tackled instances and obtain near-optimal solutions in shorter run-times. Our computational experiments show, that for challenging 0/1 Integer Programs with a large number of variables compared to the number of constraints, the core concept provides better solutions than directly solving the original problem using a commercial solver. In the future we plan to test our approach on other widely used large 0/1 IP benchmarks.

Acknowledgements

National ICT Australia is funded by the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

5 % negative weights, 5 % negative profits																		
			no core		$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$			$\delta = 0.1n + 2m$			$\delta = 0.2n + 2m$		
n	m	α	$\bar{z}$	t[s]	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$
100	5	0.25	26603	1.97	0.113	3	5	0.014	8	30	0.004	9	51	0.004	9	51	0.000	10	82
		0.50	44666	1.71	0.072	5	6	0.001	9	21	0.000	10	46	0.000	10	46	0.000	10	73
		0.75	60387	0.51	0.025	7	14	0.013	8	43	0.011	9	60	0.011	9	60	0.000	10	66
250	5	0.25	68598	56.11	0.007	7	39	0.004	8	70	0.003	9	119	0.004	8	70	0.003	9	130
		0.50	113794	93.78	0.000	10	22	0.000	10	47	0.000	10	68	0.000	10	42	0.000	10	66
		0.75	152330	42.46	0.002	7	39	0.000	10	62	0.000	10	65	0.000	10	45	0.000	10	70
100	10	0.25	24211	16.94	0.614	1	0	0.133	6	3	0.026	7	16	0.000	10	57	0.000	10	74
		0.50	43587	21.50	0.287	1	0	0.068	6	3	0.011	9	18	0.000	10	69	0.000	10	94
		0.75	59130	4.49	0.081	6	2	0.018	7	12	0.000	10	35	0.000	10	62	0.000	10	72
Average			65923	26.61	0.133	5.2	14	0.028	8.0	32	0.006	9.2	53	0.002	9.6	56	0.000	9.9	81

10 % negative weights, 10 % negative profits																		
			no core		$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$			$\delta = 0.1n + 2m$			$\delta = 0.2n + 2m$		
n	m	α	$\bar{z}$	t[s]	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$
100	5	0.25	28582	2.04	0.136	2	6	0.067	6	38	0.000	10	57	0.000	10	57	0.000	10	75
		0.50	45222	1.92	0.051	4	5	0.004	8	30	0.000	10	47	0.000	10	47	0.000	10	74
		0.75	59180	0.16	0.003	8	23	0.000	10	47	0.000	10	66	0.000	10	66	0.000	10	81
250	5	0.25	74521	44.80	0.000	10	30	0.000	10	46	0.000	10	68	0.000	10	39	0.000	10	71
		0.50	116296	36.00	0.001	9	33	0.000	10	56	0.000	10	66	0.000	10	43	0.000	10	73
		0.75	150236	6.49	0.000	10	45	0.000	10	62	0.000	10	71	0.000	10	61	0.000	10	75
100	10	0.25	25581	28.60	0.673	1	0	0.201	5	3	0.038	8	20	0.000	10	62	0.000	10	95
		0.50	44562	35.82	0.194	3	0	0.003	8	2	0.000	10	16	0.000	10	56	0.000	10	79
		0.75	58563	0.77	0.031	8	9	0.027	9	27	0.002	9	41	0.000	10	64	0.000	10	79
Average			66971	17.40	0.121	6.1	17	0.034	8.4	35	0.004	9.7	50	0.000	10.0	55	0.000	10.0	78

20 % negative weights, 20 % negative profits																		
			no core		$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$			$\delta = 0.1n + 2m$			$\delta = 0.2n + 2m$		
n	m	α	$\bar{z}$	t[s]	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$	$\%^{-}_{opt}$	#	$\%t$
100	5	0.25	34071	0.70	0.117	4	13	0.008	9	42	0.000	10	67	0.000	10	67	0.000	10	93
		0.50	46970	0.14	0.008	9	24	0.000	10	48	0.000	10	58	0.000	10	58	0.000	10	77
		0.75	56538	0.03	0.029	7	40	0.006	8	63	0.006	8	60	0.006	8	60	0.006	8	77
250	5	0.25	85206	33.14	0.005	8	30	0.000	10	48	0.000	10	70	0.000	10	45	0.000	10	81
		0.50	118236	8.60	0.000	10	38	0.000	10	55	0.000	10	72	0.000	10	52	0.000	10	86
		0.75	142169	0.21	0.000	10	39	0.000	10	61	0.000	10	64	0.000	10	60	0.000	10	75
100	10	0.25	29090	13.36	0.604	1	0	0.190	5	5	0.048	7	18	0.001	9	64	0.000	10	77
		0.50	45267	2.68	0.203	6	2	0.000	10	13	0.000	10	40	0.000	10	76	0.000	10	97
		0.75	55821	0.06	0.033	7	24	0.021	8	48	0.000	10	57	0.000	10	67	0.000	10	83
Average			68152	6.55	0.111	6.9	23	0.025	8.9	43	0.006	9.4	56	0.001	9.7	61	0.001	9.8	83

Table 3: Solving different sized cores for various percentages of negative values to optimality. (All values shown are averaged over 10 problem instances)

Time Limit = 5 Seconds													
			original problem			$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$	
n	m	α	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes
500	5	0.25	$0.120_{0.021}$	2	16421	$\mathbf{0.114}_{0.025}$	4	31847	$0.116_{0.022}$	3	26955	$0.117_{0.014}$	4	24337
		0.50	$0.067_{0.011}$	2	15976	$\mathbf{0.051}_{0.012}$	9	31901	$0.061_{0.011}$	4	27587	$0.061_{0.017}$	3	23958
		0.75	$0.041_{0.004}$	5	18598	$\mathbf{0.041}_{0.004}$	8	32791	$0.042_{0.006}$	6	29194	$0.042_{0.005}$	6	27454
500	10	0.25	$0.438_{0.024}$	0	8061	$\mathbf{0.352}_{0.048}$	6	23208	$0.383_{0.051}$	3	15496	$0.364_{0.037}$	6	13707
		0.50	$0.171_{0.029}$	2	8262	$\mathbf{0.163}_{0.025}$	4	23548	$0.165_{0.023}$	5	15865	$0.175_{0.036}$	3	13743
		0.75	$0.097_{0.018}$	3	9765	$0.093_{0.011}$	6	22537	$0.094_{0.013}$	4	17021	$\mathbf{0.092}_{0.012}$	5	15421
500	30	0.25	$1.220_{0.115}$	2	2977	$\mathbf{1.191}_{0.105}$	3	10262	$1.230_{0.044}$	3	7881	$1.214_{0.113}$	2	6014
		0.50	$0.562_{0.040}$	0	3088	$0.507_{0.042}$	6	10649	$0.536_{0.018}$	1	8181	$\mathbf{0.495}_{0.047}$	5	6101
		0.75	$0.285_{0.021}$	2	3627	$0.282_{0.036}$	2	11431	$0.283_{0.015}$	2	8715	$\mathbf{0.273}_{0.021}$	4	6593
Average			$0.333_{0.031}$	2.0	9642	$\mathbf{0.310}_{0.034}$	5.3	22019	$0.323_{0.023}$	3.4	17433	$0.315_{0.034}$	4.2	15259
Time Limit = 50 Seconds													
			original problem			$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$	
n	m	α	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes
500	5	0.25	$0.103_{0.016}$	3	172471	$0.100_{0.015}$	5	330280	$0.100_{0.015}$	6	272642	$\mathbf{0.099}_{0.015}$	7	249998
		0.50	$0.049_{0.011}$	6	181177	$\mathbf{0.046}_{0.008}$	9	328066	$0.048_{0.011}$	8	287341	$0.047_{0.009}$	8	260145
		0.75	$0.038_{0.004}$	6	188492	$0.038_{0.004}$	7	322998	$\mathbf{0.038}_{0.004}$	8	289610	$0.038_{0.004}$	9	275409
500	10	0.25	$0.331_{0.026}$	2	85187	$0.301_{0.019}$	4	231031	$\mathbf{0.296}_{0.024}$	6	150726	$0.312_{0.031}$	4	132856
		0.50	$0.144_{0.019}$	3	88829	$0.133_{0.014}$	4	235044	$0.135_{0.014}$	5	156361	$\mathbf{0.131}_{0.016}$	4	136481
		0.75	$0.082_{0.010}$	4	107407	$\mathbf{0.077}_{0.010}$	8	233931	$0.078_{0.012}$	5	171427	$0.079_{0.012}$	5	157293
500	30	0.25	$1.108_{0.076}$	1	33855	$\mathbf{1.045}_{0.066}$	6	102129	$1.098_{0.077}$	2	78250	$1.094_{0.080}$	2	62556
		0.50	$0.477_{0.030}$	2	34741	$\mathbf{0.458}_{0.034}$	5	104797	$0.467_{0.026}$	2	80025	$0.470_{0.035}$	3	62627
		0.75	$0.270_{0.019}$	0	38640	$0.254_{0.024}$	3	111507	$\mathbf{0.254}_{0.027}$	6	87269	$0.255_{0.024}$	1	65981
Average			$0.289_{0.023}$	3.0	103422	$\mathbf{0.272}_{0.022}$	5.7	222198	$0.279_{0.023}$	5.3	174850	$0.281_{0.025}$	4.8	155927
Time Limit = 500 Seconds													
			original problem			$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$	
n	m	α	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes	$\%_{LP}$	#	Nnodes
500	5	0.25	$\mathbf{0.092}_{0.011}$	10	1468306	$\mathbf{0.092}_{0.011}$	10	2156859	$\mathbf{0.092}_{0.011}$	10	2038741	$0.092_{0.011}$	9	1844037
		0.50	$0.045_{0.008}$	9	1474390	$\mathbf{0.044}_{0.007}$	10	2282445	$\mathbf{0.044}_{0.007}$	10	2184819	$0.045_{0.008}$	9	1968420
		0.75	$\mathbf{0.038}_{0.004}$	10	1051111	$\mathbf{0.038}_{0.004}$	10	1170739	$\mathbf{0.038}_{0.004}$	10	1193311	$\mathbf{0.038}_{0.004}$	10	1133267
500	10	0.25	$0.291_{0.022}$	1	752103	$\mathbf{0.266}_{0.027}$	4	1973083	$0.269_{0.024}$	5	1190391	$0.274_{0.018}$	4	1070741
		0.50	$0.125_{0.018}$	3	804076	$0.120_{0.011}$	5	2064695	$\mathbf{0.119}_{0.015}$	6	1247341	$0.124_{0.014}$	3	1131538
		0.75	$0.077_{0.011}$	7	1013516	$0.076_{0.011}$	8	2359838	$0.075_{0.010}$	7	1530181	$\mathbf{0.075}_{0.010}$	9	1409720
500	30	0.25	$0.982_{0.088}$	3	312874	$0.982_{0.026}$	3	990696	$0.974_{0.025}$	2	759324	$\mathbf{0.952}_{0.073}$	3	610611
		0.50	$0.431_{0.013}$	3	318599	$0.428_{0.031}$	2	1024538	$\mathbf{0.427}_{0.017}$	3	772248	$0.429_{0.021}$	3	620238
		0.75	$0.238_{0.017}$	2	368143	$\mathbf{0.228}_{0.018}$	5	1081771	$0.235_{0.020}$	2	858274	$0.233_{0.021}$	3	670913
Average			$0.258_{0.021}$	5.3	840346	$0.253_{0.016}$	6.3	1678296	$0.253_{0.015}$	6.1	1308292	$\mathbf{0.251}_{0.020}$	5.9	1162165

Table 4: Fixed time runs of larger benchmark instances. Various core sizes are shown for 10% negative coefficients. All values shown are averaged over 10 problem instances.

References

Balas, E. & Zemel, E. (1980), 'An algorithm for large zero-one knapsack problems', *Operations Research* **28**(5), 1130–1154.

Bertsimas, D. & Tsitsiklis, J. N. (1997), *Introduction to Linear Optimization*, Athena Scientific.

Chu, P. & Beasley, J. (1998), 'A genetic algorithm for the multidimensional knapsack problem', *Journal of Heuristics* **4**(1), 63–86.

Danna, E., Rothberg, E. & Le Pape, C. (2005), 'Exploring relaxation induced neighborhoods to improve MIP solutions', *Mathematical Programming, Series A* **102**, 71–90.

Fischetti, M. & Lodi, A. (2003), 'Local Branching', *Math. Programming Series B* **98**, 23–47.

Gavish, B. & Pirkul, H. (1985), 'Efficient algorithms for solving the multiconstraint zero-one knapsack problem to optimality', *Mathematical Programming* **31**, 78–105.

Gilmore, P. & Gomory, R. (1966), 'The theory and computation of knapsack functions', *Operations Research* **14**, 1045–1074.

Glover, F. & Kochenberger, G. (1996), Critical event tabu search for multidimensional knapsack problems, *in* I. Osman & J. Kelly, eds, 'Metaheuristics: Theory and Applications', Kluwer Academic Publishers, pp. 407–427.

Gomes da Silva, C., Clímaco, J. & Figueira, J. (2005), Core problems in bi-criteria 0,1-knapsack: new developments, Technical Report 12/2005, INESC-Coimbra.

Kellerer, H., Pferschy, U. & Pisinger, D. (2004), *Knapsack Problems*, Springer.

Martello, S. & Toth, P. (1988), 'A new algorithm for the 0–1 knapsack problem', *Management Science* **34**, 633–644.

Pisinger, D. (1995), 'An expanding-core algorithm for the exact 0-1 knapsack problem', *European Journal of Operational Research* **87**(1), 175–187.

Pisinger, D. (1997), 'A minimal algorithm for the 0-1 knapsack problem', *Operations Research* **45**(5), 758–767.

Puchinger, J., Raidl, G. & Pferschy, U. (2006), The core concept for the multidimensional knapsack problem, *in* 'Evolutionary Computation in Combinatorial Optimization - EvoCOP 2006', Vol. 3906 of *LNCS*, Springer, pp. 195–208.

Puchinger, J., Raidl, G. & Pferschy, U. (2007), The multidimensional knapsack problem: Structure and algorithms, Technical Report 006149, National ICT Australia, Melbourne, Australia. submitted for publication.

Raidl, G. & Gottlieb, J. (2005), 'Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem', *Evolutionary Computation* **13**(4), 441–475.

Shih, W. (1979), 'A branch and bound method for the multiconstraint zero-one knapsack problem', *Journal of the Operational Research Society* **30**, 369–378.

Vasquez, M. & Hao, J. (2001), 'A hybrid approach for the 0–1 multidimensional knapsack problem', *Proceedings of the 17th International Joint Conference on Artificial Intelligence* pp. 328–333.

Vasquez, M. & Vimont, Y. (2005), 'Improved results on the 0-1 multidimensional knapsack problem', *European Journal of Operational Research* **165**(1), 70–81.

Weingartner, H. M. & Ness, D. N. (1967), 'Methods for the solution of the multidimensional 0/1 knapsack problem', *Operations Research* **15**, 83–103.

An ILP for the metro-line crossing problem

Matthew Asquith[1,2] **Joachim Gudmundsson**[3] **Damian Merrick**[2,3]

[1] Sophos Pty Ltd, North Sydney, Australia

[2]School of Information Technologies, University of Sydney, Australia.

[3] NICTA*, Sydney, Australia.
Email: {joachim.gudmundsson,damian.merrick}@nicta.com.au

Abstract

In this paper we consider a problem that occurs when drawing public transportation networks. Given an embedded graph $G = (V, E)$ (e.g. the railroad network) and a set H of paths in G (e.g. the train lines), we want to draw the paths along the edges of G such that they cross each other as few times as possible. For aesthetic reasons we insist that the relative order of the paths that traverse a vertex does not change within the area occupied by the vertex. We prove that the problem, which is known to be NP-hard, can be rewritten as an integer linear program that finds the optimal solution for the problem.

In the case when the order of the endpoints of the paths is fixed we prove that the problem can be solved in polynomial time. This improves a recent result by Bekos et al. (2007).

1 Introduction

In 1931, graphic designer Harry Beck first proposed that passengers would be more interested in how train lines connect rather than the true geographical layout of stations in a city (Garland 1994). His concept of the metro map was so successful that it has been adopted by virtually every subway company in the world. Similar diagrams have been used to visualise wiring layouts (Benkert et al. 2006) and more abstract connected information, such as website networks (Sandvad et al. 2001). This has motivated recent research on how to automate their construction through the use of computer algorithms.

The construction of metro maps can be broken down into a sequence of steps. First, one finds an embedding of the network that balances true geographical positioning with diagrammatic simplicity. Second, the individual lines representing the train routes are embedded into this graph. Any routes that share stations in the same sequence need to be given a line ordering and the positions of any crossings are assigned. Finally, a labelling of the stations and important features is added to the drawing.

Previous research has primarily focused on the first and third step. Hong et al. (2006) present force-directed graph drawing approaches to metro map layout, which are also used to produce metro map layouts of non-geographical networks. Slower but more geographically accurate optimisation-based methods are detailed by Stott & Rodgers (2004), and more recently by Nöllenburg & Wolff (2005). They approached the layout problem using a mixed-integer program (MIP) approach, a well-established method to solve linear equations. Merrick & Gudmundsson (2006a,b) propose an alternative method to solve the metro map layout problem. Their method simplifies polygonal chains by allowing them a threshold from which they can move from their original locations.

The final step of the above construction is labelling, i.e. writing the names of stations on the diagram, preferably in such a way that both the labels and the rest of the diagram are clearly readable. Labelling maps in general is a computationally hard problem (Formann & Wagner 1991). Many variations of the map labelling problem have been investigated; Wolff & Strijk (2007) maintain an extensive bibliography on the topic. Within the context of metro maps, only a limited amount of work has been done on labelling. Hong et al. (2006) proposed using a simulated annealing algorithm and a greedy heuristic for labelling, but noted that the results required manual editing to be acceptable. Nöllenburg (2005) produced fully-automatic labelling by incorporating additional constraints into the MIP approach (Nöllenburg & Wolff 2005), which was demonstrated to be feasible on the S-Bahn RheinNeckar system, a network of 108 vertices and 111 edges. Nöllenburg presents this as an initial attempt at automatic labelling; there are some label overlaps in the solutions produced. Generating metro map labelling solutions for large networks remains an open problem.

However, hardly any research has been done on the problem of embedding the individual lines in a map. Note that even when the geometric embedding of the network is given, finding a good ordering of the individual lines along the network is far from simple. In this paper we will focus on placing the lines such that the number of crossings is minimised. This criterion has been noted as one of the most important characteristics in ensuring diagrams are easy to comprehend (Purchase et al. 1996). The first research devoted to the drawing of metro lines was published by Benkert et al. (2006). They introduced the following problem:

Problem 1 *Benkert et al. (2006) Given an embedded graph $G = (V, E)$ and a set H of paths in G, draw the paths along the edges of G such that they cross each other as few times as possible.*

Their research was dedicated solely to looking at a very restricted case where the crossings can only be minimized along a single edge. They devise a dynamic programming algorithm that can solve any instance

(*) National ICT Australia is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

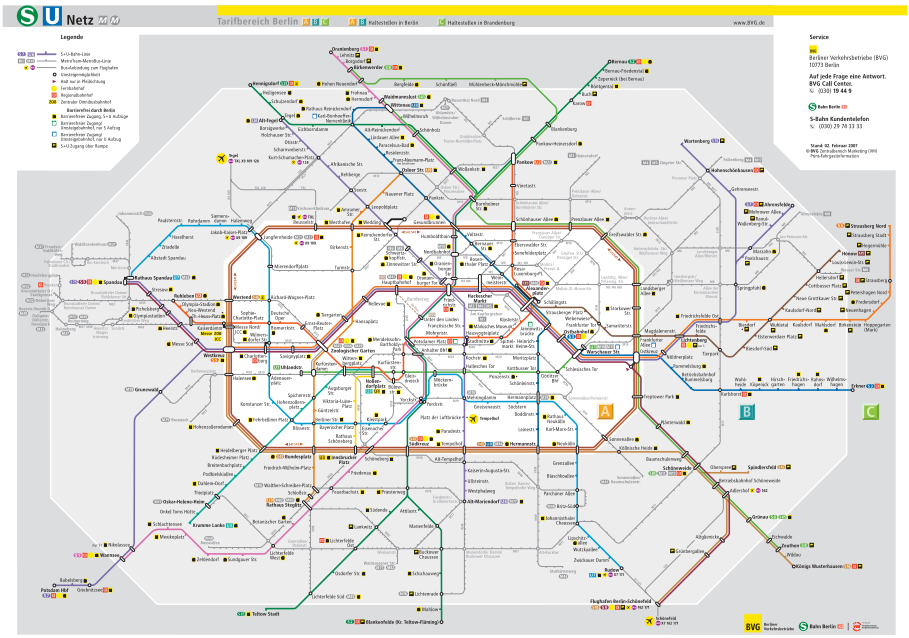

Figure 1: A map illustrating the tram, subway and train lines in Berlin.

in $O(n^2)$ time. Their discussion for extending their algorithm for use in more general cases concludes that interactions between terminating lines would hinder the use of dynamic programming.

Recently Bekos et al. (2007) proved that the problem is NP-hard even in the case when the underlying graph G is a path. They also considered a special case when the position of the terminators[1]are fixed (terminators are the start and endpoint of a path in H), and proved that in this case the problem can be solved in polynomial time if the underlying graph is a path or a tree. They call this problem the *Metro-line crossing minimization problem with terminals at fixed station ends* (MLCM-FixedSE).

In this paper we prove that the MLCM-FixedSE problem can be computed in polynomial time for any underlying graph G (Corollary 2). In the case when the terminators are not fixed then the problem can be solved using integer linear programming.

1.1 Definitions and preliminaries

A *metro map* is a connected graph $G = (V, E)$ representing the layout of, for example, a transportation system. Let H be a set of paths of G representing the individual routes traversed through the network, such as the subset of stations a train passes. In Fig. 2a, G is the underlying network and H is the set of three different train lines (one with thick solid lines, one with thin solid lines and one with dashed lines).

A (metro) *line* is an individual paths's depiction in the network. In metro maps, each line is usually drawn separately in a unique colour. This means each line will have an ordering at each vertex with respect to any other lines that share a common edge. If the relative order of two lines changes between two incident vertices then the two lines must intersect between the vertices. We define this as an *edge crossing*. This is illustrated in Fig. 2a, where the order of the two lines changes between u and v. As opposed to an edge crossing, a *vertex crossing* occurs inside a vertex. This often results in the corresponding diagrams becoming harder to understand and therefore it is only used for crossings that cannot be placed along an edge. This can only occur if two subgraphs g and h in H share a common path consisting of a single vertex v and both have degree two at v, see Fig. 2b-c. A *terminator* of a subgraph $h \in H$ is a vertex v where $\deg_h(v) = 1$, i.e. h *terminates* at v if $\deg_h(v) = 1$, see Fig. 3b.

In most metro maps, all lines terminate on the outside of the other lines travelling in parallel, as shown in Fig. 3a. This is to emphasise that a line terminates at that station. We call this the *periphery condition*. The problem we will study in this paper can now be formulated as follows:

Problem 2 *Given a graph $G = (V, E)$ and a set H of paths of G, find an order of lines at every vertex $v \in V$ that minimises the total number of edge crossings between the paths in H and fulfils the periphery condition.*

Recall that this problem is denoted the *MLCM problem with terminals at station ends* (MLCM-SE) in (Bekos et al. 2007).

We say that a crossing between two subgraphs g and h in H is *forced* if changing the positions of any of the terminators of g or h cannot prevent the crossing from occurring, as illustrated in Fig. 3b. A feature of an optimal solution is that there cannot be any *redundant* crossings between any pair of subgraphs g and $h \in H$, thus there will be at most one crossing between g and h along a common subpath between them. This will be proven in the next section.

2 An ILP approach

Recall that Bekos et al. (2007) showed that the problem is NP-hard even in the case when the underlying graph is a path. In this section we prove that the problem can be solved using ILP. Our approach works in four steps:

1. For each pair of lines $g, h \in H$ compute all (maximal) common subpaths $\delta_1(g, h), \ldots, \delta_m(g, h)$.
2. Each common subpath $\delta(g, h)$ is converted into a set of crossing rules C encoding the relations between the terminators of g and h.

[1]Terminator is sometimes called a terminal in the literature.

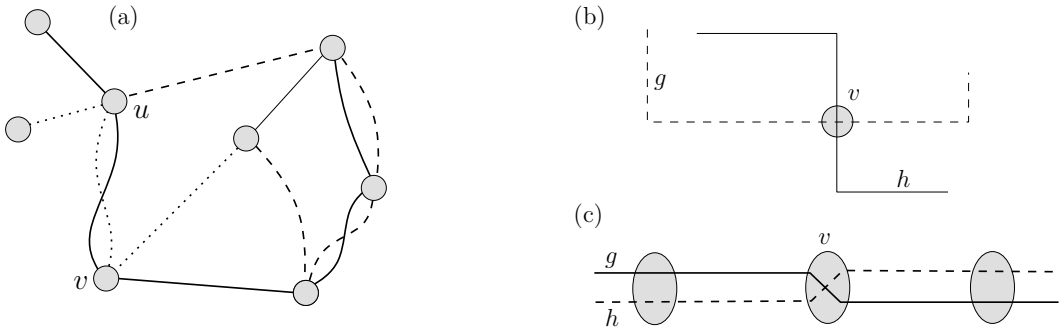

Figure 2: (a) A graph representation of a metro map. (b) A valid vertex crossing at v between g and h since their common subpath is v. (c) An invalid vertex crossing at v between g and h since v is a subset of their common subpath.

3. The positions of these terminators and the number of crossings in G is determined using an integer linear program.
4. The actual line ordering at each vertex is decided.

Note that the only step that cannot be performed in polynomial time is step 3. If we assume that the terminators are fixed (MLCM-FixedSE problem) then Bekos et al. (2007) showed that the problem can be solved in polynomial time for the very restricted case when the underlying graph G is a tree. Below we will prove that the problem can be solved in polynomial time for any underlying graph G.

We start with some important properties of an optimal solution. First we show that any optimal solution cannot contain any redundant crossings, i.e. where two lines cross more than once in a single common subpath.

Theorem 1 *In an optimal solution every pair of lines in H will cross at most once along any common subpath.*

Proof. The proof is done by contradiction. Consider an optimal solution S and assume that there exists a pair of lines g and h in H that cross twice along a common subpath $P = \langle p_1, p_2, \ldots, p_k \rangle$ in G. To simplify the description it is assumed that P is a horizontal path with p_i to the left of p_{i+1}, for $1 \leq i < k$. If there is more than one pair of subpaths along P that cross twice then assume that g and h is the topmost pair along P. Let c_g and c_h denote the number of crossings along P between the subpaths in H and g, and between the subpaths in H and h, respectively. Without loss of generality it is assumed that $c_h \leq c_g$, and that g is positioned above h at p_1 and p_k as illustrated in Fig. 4a.

Consider the following modification to S, denoted S', where g is moved such that it lies above h along P. That is, the part of g below h is moved such that g lies immediately above h in this interval, see Fig 4b. We claim that the number of crossings in S' is less than the number of crossings in S. We will have two cases:

(i) If g and h is the only pair of subpaths along P that cross twice then the number of crossing in S' will have decreased with two. Because of the periphery condition no subpath f could terminate in between the two crossings between g and h. As a result a subpath f intersecting g and h after the modification must have intersected g and h h before the modification. Thus, no more crossings have been introduced.

(ii) In the case when there is more than one pair of subpaths along P that cross twice then we consider a subpath $f \in H$. It is not hard to see that the only configuration that may increase the number of crossings is if f crosses h, but not g, twice in S. In this case the number of crossings in S' would increase by two, since f would cross both g and h twice in S'. However, this contradicts the above assumption that g and h are the topmost pair of subpaths along P that cross twice.

In both cases we get a contradiction, thus the theorem follows. ∎

The next observation follows from the above theorem.

Observation 1 *Consider two lines g and h in H, and let $t(h)$ be a terminator of h at a vertex v of G. Changing the order of h at v can only affect the number of crossings between g and h if v belongs to a common subpath of g and h.*

Proof. Consider a common subpath without any terminators of g and h. The observation follows trivially from the fact that the interval order at the endpoints of their common subpaths is fixed and, according to Theorem 1, g and h cannot intersect twice. ∎

A somewhat stronger result can be obtained using exactly the same arguments.

Corollary 1 *Let $t(g)$ be a terminator of $g \in H$ at vertex v and let $h_1, h_2 \in H$. Changing the order of $t(g)$ at v can only affect the number of crossings between h_1 and h_2, along a common subpath $\delta(h_1, h_2)$, if v belongs to $\delta(h_1, h_2)$.*

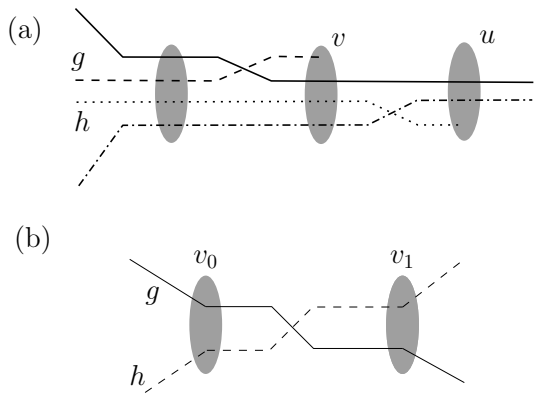

Figure 3: (a) Terminators for g at v and h at u fulfilling the periphery condition. (b) Illustrating a forced edge intersection between g and h.

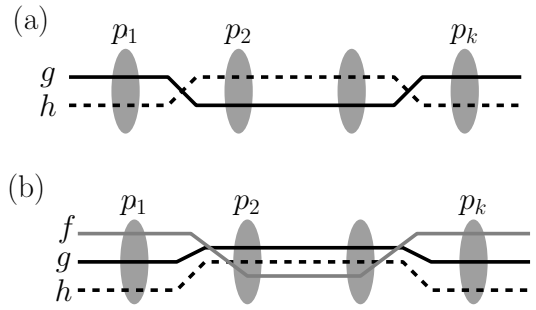

Figure 4: (a) Relative terminator position directions. (b) Relative terminator position directions.

Theorem 2 *Once the terminator positions are fixed, we can decide in $O(|H|^2 \cdot |E|)$ time if there will be 0 or 1 crossings between each pair of subpaths g and $h \in H$ along any common subpath $P = \langle v_0, v_1, \ldots, v_m \rangle$.*

Proof. The most time consuming step is to compute a list of all the maximal common subpaths. These can easily be computed in $O(|H|^2 \cdot |E|)$ time by testing every pair of paths in H.

Consider a common subpath P of g and h, where g and h do not both share a terminator at v_0 or v_m. Now, both end vertices of P must either have unique subsequent vertices or contain a defined terminator position for both g and h. In either case, g and h have a well defined relative line ordering at each end vertex of P (the periphery condition guarantees this in the latter case). By Theorem 1, whether the relative order of g and h is different between v_0 and v_m determines whether there will be one or no forced crossing.

In the case where g and h both have degree 1 at a terminator $v_t \in \{v_0, v_m\}$ (share a terminator), if both g and h are positioned on the same side of v_t, their ordering (and thus the number of crossings) can no longer be uniquely determined. Here, we can always choose an ordering which ensures no crossing. ∎

From the above proofs, we can conclude that once the terminating positions are fixed, we can compare all common subpaths between all lines to determine the total crossing number of the graph. Note, that the crossing number is no longer related to the edge at which any fixed crossings occur inside the common subpaths and the actual ordering at each vertex can be decided at a later stage. Therefore the number of crossings is determined entirely by the positions of terminators.

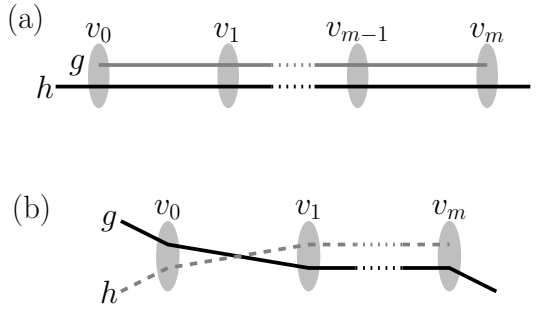

Figure 5: (a) A maximal common subpath $\delta(g, h)$ of the lines g and h. (b) A possible common subpath with interactions of terminators between g and h.

2.1 Converting common subpaths to crossing rules

A list of all the maximal common subpaths can be obtained in $O(|H|^2 \cdot |E|)$ time by testing every pair of lines. We will now examine the conditions which can lead to an edge crossing between two lines along a common subpath.

Once a list of all common subpaths has been retrieved from G, we can represent their potential crossings as a set of rules dependent solely on the positions of the terminators. Clearly, a crossing between two lines g and h can only occur along a common subpath. Consider one such common subpath $\delta(g, h) = \langle v_0, v_1, \ldots, v_m \rangle$. Without loss of generality, assume that the vertices of $\delta(g, h)$ lie on a horizontal line and that v_i lies to the left of v_{i+1}, for $0 \leq i < m$, as shown in Fig. 5a. To determine whether g and h will intersect along $\delta(g, h)$ we examine the relative ordering of g and h at the end vertices v_0 and v_m of $\delta(g, h)$. If the ordering changes, there must be exactly one edge crossing placed on an edge along $\delta(g, h)$, according to Theorem 1 (Fig. 5b shows an example). If at least one of the lines has a terminator at v_0 or v_m then changing the terminator positions will switch the relative ordering and thus decide whether there will be a crossing or not. Since the ordering must change along the common subpath, a crossing will always result from one of two possible cases:

Case 1: (g is above h at v_0) AND (g is below h at v_m), or

Case 2: (h is above g at v_0) AND (h is below g at v_m).

We will label each case as a *crossing rule* c and let C be the set of all crossing rules. Each individual ordering restriction (e.g. "g is above h at v") at a vertex $v \in P$ is called a *condition*. We can rewrite each condition as a boolean variable, and each case as a boolean expression joining the conditions. For example, the first case above can be written $g_h^{v_0} \wedge \neg g_h^{v_m}$, where g_h^v denotes a boolean variable that is true if g is ordered above h at vertex v, or false otherwise. Since h_g^v is equivalent to $\neg g_h^v$, we can write the second case above as $\neg g_h^{v_0} \wedge g_h^{v_m}$.

If the conditions of a crossing rule c are all true, then c will be given a value of 1. Alternatively, if any condition is false, c will be set to 0. In this way, the crossing rules act as counters for the number of crossings between lines, for a given line ordering. The condition for whether a line g is above a line h at a vertex $v \in P$ is dependent on the degree of both g and h at v. Note that we are only interested in the cases $v = v_0$ and $v = v_m$, as these are the endpoints of the common subpath at which the ordering of two lines may differ.

Using the above list of conditions, each common subpath will now be represented as two distinct crossing rules that can never be simultaneously true. For example, in the common subpath $\langle v_0, v_1, \ldots, v_m \rangle$ between g and h in Fig. 5a, examination of the end vertices shows the two crossing rules to be the following:

$$g_h^{v_0} \wedge \neg g_h^{v_m} \quad \text{and} \quad \neg g_h^{v_0} \wedge g_h^{v_m}.$$

Alternatively, Fig. 5b contains a common subpath with a fixed ordering at v_0. The corresponding conditions are no longer dependent on the position of a terminator but are instead Boolean statements of whether the ordering is present:

$$(\text{true}) \wedge \neg g_h^{v_m} \quad \text{and} \quad (\text{false}) \wedge g_h^{v_m}.$$

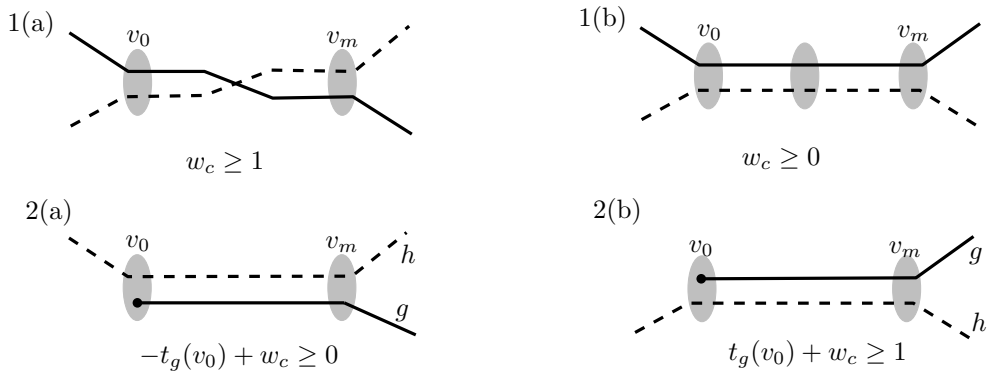

Figure 6: Conversion to constraints – cases 1 and 2.

Since the second crossing rule can never have all its conditions true, its corresponding c will always be 0, and we do not need to add it to the set C.

The number of crossing rules in C is dependent on the total number of common paths between each pair of subpaths in H. If every common subpath starts and ends with at least one terminator, i.e. there are no fixed crossings, each subpath can at most form two common subpaths with each other subpath. Therefore, $|C| = O(|H|^2)$ in this case. With fixed crossings allowed, there can be a large number of common subpaths between each pair of subpaths in H. Hence, in the worst case, $|C| = O(|H|^2 \cdot |E|)$.

With each potential crossing of H in G represented as a crossing rule $c \in C$, the crossing number is the minimum sum of all crossing rules $c \in C$ over all valid line orderings. In the following sections, we will present two different methods to compute this crossing number.

2.2 Transformation of the rules into an ILP

Integer Linear Programming (ILP) is an established method of finding solutions to optimization problems that can be expressed in terms of an objective function subject to a series of constraints. ILP solvers (such as CPLEX) use highly optimised branch and cut techniques in order to provide an optimal solution. This has been a highly effective method for finding solutions to more general crossing minimisation problems (Buchheim et al. 2005).

In our ILP formulation, every terminator contained in at least one crossing rule is represented as a unique binary variable $t_h(v)$, which is true if the terminator of line h at v is ordered "above" all continuing lines, or false if it is ordered "below". To simplify the description of the ILP, we take the same assumption as in Section 2.1, that the vertices lie on a horizontal line, ordered from left to right ("above" and "below" are then defined intuitively). For each crossing rule c, an additional unique binary variable w_c is added to the objective function. This allows us to represent each rule as a constraint that forces its w_c to be set to 1 if all its conditions are true, i.e. if there is a crossing. Therefore, the objective function is a summation of all potential crossings.

Consider a maximal common subpath $\delta(g,h) = \langle v_0, \ldots, v_m \rangle$ of g and h. Let $T \subseteq \{t_g(v_0), t_g(v_m), t_h(v_0), t_h(v_m)\}$ be the binary variables corresponding to the set of terminators (if any) of g and h within the subpath $\delta(g,h)$. We can now incorporate each crossing rule into the ILP by adding one or more constraints, according to the following rules (see Fig. 6-7):

1. If $|T| = 0$ (i.e. there are no terminators in $\delta(g,h)$:
   (a) If $(g_h^{v_0} \wedge \neg g_h^{v_m}) \vee (\neg g_h^{v_0} \wedge g_h^{v_m})$ is always true then the constraint $w_c \geq 1$ is added.
   (b) If $(g_h^{v_0} \wedge \neg g_h^{v_m}) \vee (\neg g_h^{v_0} \wedge g_h^{v_m})$ is always false then the constraint $w_c \geq 0$ is added.
2. If $|T| = 1$, then there is one terminator $t \in T$, and we look at which position of t makes $(g_h^{v_0} \wedge \neg g_h^{v_m}) \vee (\neg g_h^{v_0} \wedge g_h^{v_m})$ true (i.e. generates a crossing):
   (a) If t, then the constraint $-t + w_c \geq 0$ is added.
   (b) If $\neg t$, then the constraint $t + w_c \geq 1$ is added.
3. If $|T| = 2$, then $(g_h^{v_0} \wedge \neg g_h^{v_m}) \vee (\neg g_h^{v_0} \wedge g_h^{v_m})$ is dependent on the positions of two separate terminators $t_1, t_2 \in T$.
   (a) If t_1, t_2 are on the same line (either g or h), then we add the two constraints $-t_1 + t_2 + w_c \geq 0$ and $t_1 - t_2 + w_c \geq 0$.
   (b) If t_1, t_2 are on different lines, but both terminate at the same vertex (either v_0 or v_m), then we consider which conditions cause $(g_h^{v_0} \wedge \neg g_h^{v_m}) \vee (\neg g_h^{v_0} \wedge g_h^{v_m})$ to become true:
      i. If $t_1 \wedge \neg t_2$, then the constraint $-t_1 + t_2 + w_c \geq 0$ is added.
      ii. If $\neg t_1 \wedge t_2$, then the constraint $t_1 - t_2 + w_c \geq 0$ is added.
   (c) If t_1, t_2 are on different lines and terminate at different vertices (one at v_0 and one at v_m), then we add the two constraints $t_1 + t_2 + w_c \geq 1$ and $-t_1 - t_2 + w_c \geq -1$.
4. If $|T| = 3$, then constraints are added according to case 3b above, applied to the two terminators that share a vertex – the third terminator does not influence the number of crossings.
5. If $|T| = 4$, then we add the two constraints $t_g(v_0) - t_g(v_m) - t_h(v_0) + t_h(v_m) + w_c \geq -1$ and $-t_g(v_0) + t_g(v_m) + t_h(v_0) - t_h(v_m) + w_c \geq -1$.

We now give a small example. Consider the graph and the lines in Fig. 8a. Using the above rules the instance can be converted to an ILP. Consider the four common subpaths. We get:

(i and h): case 1a $\implies w_1 \geq 1$

(i and j): case 2a $\implies -t_j(v_2) + w_2 \geq 0$

(g and h): case 3b-ii $\implies t_g(v_0) - t_h(v_0) + w_3 \geq 0$.

(h and j): case 3c $\implies t_h(v_4) + t_j(v_2) + w_4 \geq 1$ and $-t_h(v_4) - t_j(v_2) + w_4 \geq -1$.

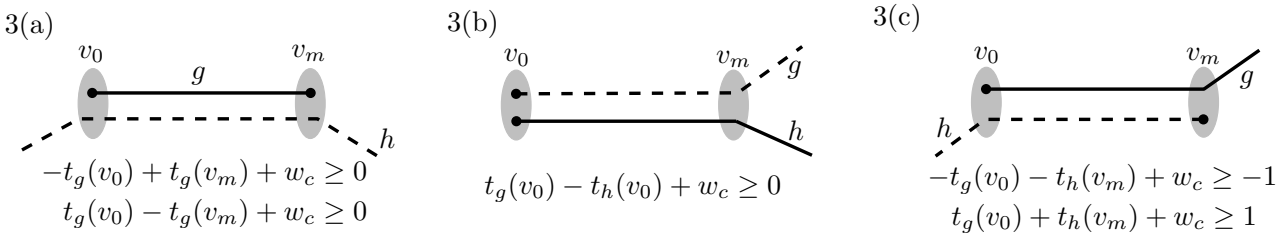

Figure 7: Conversion to constraints – cases 3a, 3b and 3c.

As a result we have:

$$\begin{aligned} \text{Minimize} \quad & w_1 + w_2 + w_3 + w_4 \\ \text{subject to} \quad & \\ & w_1 \geq 1 \\ & -t_j(v_2) + w_2 \geq 0 \\ & t_g(v_0) - t_h(v_0) + w_3 \geq 0 \\ & t_h(v_4) + t_j(v_2) + w_4 \geq 1 \\ & -t_h(v_4) - t_j(v_2) + w_4 \geq -1 \end{aligned}$$

An optimal solution of the ILP will clearly fix the order of the terminators while minimising the number of intersections.

Theorem 3 *An instance of the line ordering problem with the periphery condition can be transformed into an ILP in $O(|H|^2 \cdot |E|)$ time, where H is the set of metro lines and V is the set of vertices in the input network.*

3 A heuristic

Above we showed how the problem can be rewritten as an ILP. However, even though there are fast tools for solving ILPs in practice, they still require exponential time in the worst case. In this section we consider a heuristic algorithm for the problem. We show that if a certain condition is fulfilled by the input, the heuristic produces an optimal solution.

There are situations where nested relations between terminators impede a local approach. In such cases, a wrong decision may lead to a solution that differs significantly from the global optimum. However, there are cases where the optimal terminator position can be chosen locally. For other cases, we can estimate the likelihood of a given terminator position being the optimal position. The general idea of the heuristic is to greedily choose a position for the terminator whose estimated likelihood of optimality is the highest. When the position of this terminator is fixed, it will force a line ordering at the end of any other common subpaths that contain it, and we can continue iteratively. We call any terminators whose positions have yet to be fixed *unresolved* terminators.

Consider a crossing rule $c \in C$ induced by two lines g and h along the maximal common subpath $\delta(g,h) = \langle v_0, \ldots, v_m \rangle$. If c depends on an unresolved terminator t at v_0 then it can be grouped into two categories depending on the ordering at v_m:

- Fixed: If the ordering at v_m has already been fixed, then the position of t directly determines whether a crossing is introduced or not.
- Relation: If the ordering at v_m has not been fixed then we cannot immediately decide which position of t will cause an extra crossing. For each pair of terminators with a relation, there will be two of these crossing rules as there are two possible pairs of terminator positions that can cause a crossing.

The above categories are defined symmetrically for an unresolved terminator at v_m. These categories will help us to calculate a certainty value, denoted $u(t)$, for each unresolved terminator t. To calculate $u(t)$ we need to define the following three values:

- $f_T(t)$ and $f_B(t)$: How many crossings are *unavoidable* if one position of t is chosen, i.e. the number of fixed crossing rules involving t. They are calculated as the number of crossings if t were to be positioned at the top ($f_T(t)$) or at the bottom ($f_B(t)$).
- $r(t)$: An upper bound on the number of crossings that could *potentially* occur if one position of t is chosen over the other. It is calculated as the number of unresolved terminators with a relation with t.

Once all crossing rules have been checked, our certainty value for an unresolved terminator t is calculated as $u(t) = |f_T(t) - f_B(t)| - r(t)$. The algorithm is straight forward. Let T be the set of all terminators, and iteratively perform the following steps until T is empty:

1. Compute the value $u(t)$ for every terminator $t \in T$.
2. Select the terminator $t \in T$ with highest $u(t)$ and fix it in the position that locally minimises the number of crossings.
3. Remove t from T.

Since we have $O(|H|^2 \cdot |E|)$ crossing rules and $O(|H|)$ terminators, this algorithm runs in $O(|H|^3 \cdot |E|)$ time in the worst case. Interestingly, it can be shown that under certain conditions, the algorithm is guaranteed to find an optimal solution.

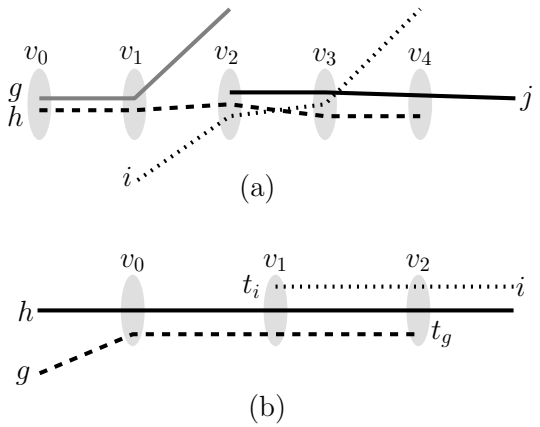

Figure 8: (a) Conversion of a graph to an ILP. (b) A situation where terminator positions can be decided locally.

Theorem 4 *The algorithm finds an optimal solution if every terminator $t \in T$ is sequentially resolved with $u(t) \geq 0$.*

Proof. We will prove that the first terminator t can always be locally chosen if $u(t) \geq 0$. Suppose $f_T(t) = f_B(t)$. In this case, $r(t) = 0$ for $u(t) \geq 0$, which implies that the terminator has no relations with any other unresolved terminators and it is easy to see that either position of t can be chosen.

Now suppose $f_B(t) > f_T(t)$. This means that at most, the number of crossings prevented by choosing the bottom position for t will be $r(t) - |f_T(t) - f_B(t)|$. However, if $u(t) \geq 0$, then $r(t) - |f_T(t) - f_B(t)| \leq 0$, meaning it is not possible to locally save any crossings by choosing the bottom position. When the terminator position is fixed, up to $r(t)$ additional crossings may be introduced to the other terminators with relations to t. But we also saved a minimum of $|f_T(t) - f_B(t)|$ fixed crossings at t, and hence, $|f_T(t) - f_B(t)| \geq r(t)$. Therefore, locally choosing t cannot lead to a worse solution if $u(t) \geq 0$. The same can be shown symmetrically for $f_T(t) > f_B(t)$. Once t is fixed, we can treat G as a new graph with a fixed position of t. Thus, the above process can be repeated until all terminators are resolved. ∎

In most metro maps and wiring diagrams, the majority of lines branch off into different paths. Consequently, the relations between the terminators are generally sparse and rarely nested. Therefore, they will in many cases be solvable in polynomial time – on the condition that $u(t) \geq 0$ for every terminator greedily selected. The heuristic can be applied to the general case but without any approximation bound; preliminary experiments have shown that the obtained solution may be far from the optimum.

4 Line Ordering

With the positions of the terminators fixed and the crossing number determined, we still need to order the lines along each edge. This includes the placement of any inevitable crossings and, importantly, ensuring the chosen ordering does not introduce any redundant crossings. In this section, we describe how to perform this ordering process by transforming to an existing problem in circuit layout.

In the previous sections we focussed on the case when H is a set of paths. In this section we allow H to be a set of binary trees.

Crossing minimisation problems have already been extensively researched in both the graph drawing community (Buchheim et al. 2005, Eades et al. 1986) and in the circuit design community (Groenveld 1989, Marek-Sadowska & Sarrafzadeh 1995). Groenveld (1989) defined the problem of finding a configuration of wires (or nets) on a circuit board that minimises the number of times they cross. Note that the terminals of the wires are fixed. This became known as the *Constrained Crossing Minimization Problem* (CCMP). The problem was investigated by Marek-Sadowska & Sarrafzadeh (1995) who added an additional constraint to more accurately reflect circuit boards. Any unavoidable crossings must be distributed in separate regions due to the fact that wire crossings take up physical space. This added a level of complexity to the CCMP and the new problem was named the *Crossing Distribution Problem* (CDP). Here wires travel through a planar layout of *regions*. The ends of the wires are referred to as *terminals* and the position of a terminal is fixed on the perimeter of a region. The problem is to find an ordering of the wires at the boundaries of each region that both minimises the total number of crossings and distributes any inevitable crossings amongst the regions according to a predetermined quota (Marek-Sadowska & Sarrafzadeh 1995).

A transformation of a metro map into an instance of the CDP would allow us to use a solution to the CDP to find a line ordering at each vertex, once the terminator positions have been decided. Since the periphery condition ensures all terminating lines finish on the outside, terminators are the equivalent of terminals lying on the perimeter of a circuit board region - the chosen side of the terminator merely affects its position along this perimeter. Marek-Sadowska & Sarrafzadeh (1995) presented an $O(m \cdot \xi^{3/2})$ time algorithm for the CDP, where ξ is the number of crossings and m is the number of regions. In our setting we have $m = O(|E|)$ and $\xi = O(|H|^2 \cdot |E|)$.

Suppose we have chosen the terminator positions for a graph G (Fig. 9a). Let the degree of a vertex v in G be denoted by $\deg(v)$. In the first step, we find all distinct maximal paths $P = \langle v_0, v_1, \ldots, v_{m-1}, v_m \rangle$ where $\deg(v_0) \neq 2$, $\deg(v_m) \neq 2$ and every intermediate vertex has degree 2. Each such path P will become a region r_P in the circuit board, as shown in Fig. 9b.

Additionally, each vertex $v \in V$ with $\deg(v) \geq 3$ will be represented by its own region r_v. Since we wish to minimis the number of crossings that are placed inside vertices, we set the crossing quota to be the number of forced vertex crossings. In the case when H is a set of paths, this will always be 0. Also, any path P containing v as an end vertex will define a region boundary between r_v and r_P. The ordering returned by the following algorithm for any region boundary between r_v and r_P will define the ordering at r_v with respect to the edge incident to v in P.

If a path P contains any terminators between the vertices $\langle v_1, \ldots, v_{m-1} \rangle$, place them on the perimeter of r_P as terminals. The ordering of the terminals is determined directly according to the positions of the corresponding terminators. For any vertex shared by two terminators, we order the terminators around the perimeter such that no crossings between the corresponding subpaths are introduced. Now the CDP algorithm will return the line orderings at each vertex at the end of each path (see Fig. 9c).

For the orderings of the vertices inside each path P, we will represent P as a separate rectangle, with the ordering at each end corresponding to the output from the previous step. Any terminators are placed along on their corresponding sides in the same way as their circuit board equivalent in the previous step. Now, straight lines are drawn to connect the start and end of the corresponding subpaths on the rectangle of the path P. The ordering of these lines at the ends of the rectangle induces the ordering of the corresponding subpaths. From the above transformation, the main theorem of this section follows.

Theorem 5 *Given positions for the terminators of the subgraphs in H, a placement of H in G that does not introduce any redundant crossings can be computed in $O(|H|^3 \cdot |E|^{\frac{5}{2}})$ time.*

By simply combining Theorem 2 and Theorem 5 we obtain:

Corollary 2 *The MLCM-FixedSE problem can be solved in $O(|H|^3 \cdot |E|^{\frac{5}{2}})$ time.*

5 Concluding remarks and acknowledgements

We believe that all the results in this paper can be generalised to the case when H is a set of binary trees. However, the description of this case is very long and

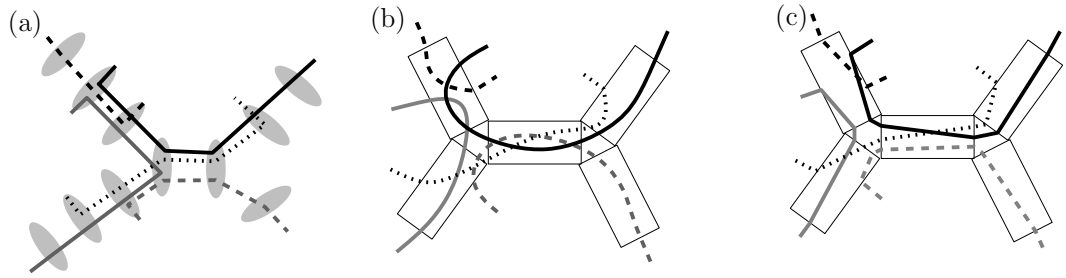

Figure 9: (a) A graph with terminator positions chosen. (b) Transformation into a circuit board instance. (c) Output from the CDP algorithm.

somewhat tedious, and is therefore omitted. The interested reader can find more details in (Asquith 2007).

The transformation described in Section 2 has been implemented, but so far only very preliminary experiments have been done.

We would like to thank Michael Forster and Marc Benkert for interesting discussions during the initial stages of this work. Finally, we would like to thank the reviewers for their diligent work.

References

M. Asquith (2007), The Metro map line ordering problem, Honours thesis, School of IT, University of Sydney, 2007.

M. Bekos, M. Kaufmann, K. Potika & A. Symvonis (2007), Line Crossing Minimization on Metro Maps, *in* 'Proceedings of the 15th International Symposium on Graph Drawing', 2007.

M. Benkert, M. Nöllenburg, T. Uno & A. Wolff (2006), Minimizing Intra-Edge Crossings in Wiring Diagrams and Public Transportation Maps, *in* 'Proceedings of the 14th International Symposium on Graph Drawing', 2006.

C. Buchheim, D. Ebner, M. Jünger, G. Klau, P. Mutzel and R. Weiskircher (2005), Exact Crossing Minimization, *in* 'Proceedings of the 13th International Symposium on Graph Drawing', 2005.

P. Eades, B. McKay & N. Wormald (1986), On an edge crossing problem, *in* 'Proceedings of the 9th Australian Computer Science Conference', 1986.

M. Formann & F. Wagner(1991), A Packing Problem with Applications to Lettering of Maps, *in* 'Proceedings of the 7th Annual ACM Symposium on Computational Geometry', 1991.

K. Garland (1993), Mr Beck's Underground Map, Capital Transport Publishing, 1994.

G. Groenveld (1989), On global wire ordering for macro-cell routing, *in* 'Proceedings of the 26th ACM/IEEE Conference on Design Automation', 1989.

S.-H. Hong, D. Merrick & H. A. D. do Nascimento (2006), Automatic visualisation of metro maps. Journal of Visual Languages & Computing, 17(3): 203–224, 2006.

M. Marek-Sadowska & M. Sarrafzadeh(1995), The Crossing Distribution Problem, IEEE Transactions on Computer-Aided Design, 14:423–433, 1995.

D. Merrick & J. Gudmundsson (2006), Increasing the Readability of Graph Drawings with Centrality-Based Scaling, *in* 'Proceedings of the Asia-Pacific Symposium on Information Visualisation', 2006.

D. Merrick & J. Gudmundsson (2006), C-Directed Path Simplification for Metro Map Layout, *in* 'Proceedings of the 14th International Symposium on Graph Drawing', 2006.

M. Nöllenburg (2005), Automated Drawing of Metro Maps, Master Thesis, Fakultät für Informatik, Universität Karlsruhe, Germany, 2005.

M. Nöllenburg & A. Wolff (2005), A Mixed-Integer Program for Drawing High-Quality Metro Maps, *in* 'Proceedings of the 13th International Symposium on Graph Drawing', 2005.

H. Purchase, R. Cohen & M. James (1996), Validating Graph Drawing Aesthetics, *in* 'Proceedings of the 3rd International Symposium on Graph Drawing', 1996.

E. Sandvad, K. Grønbæk, L. Sloth & J. Lindskov Knudsen(2001), A metro map metaphor for guided tours on the Web: the Webvise guided tour system, *in* 'Proceedings of the 10th International ACM Conference on World Wide Web', 2001.

J. Stott & P. Rodgers (2004), Metro map layout using multicriteria optimization, *in* 'Proceedings of the 8th International Conference on Information Visualisation', 2004.

A. Wolff & T. Strijk (2007), The Map-Labeling Bibliography, Accessed on 24 May 2007, `http://i11www.ira.uka.de/map-labeling/bibliography/`.

A Multidimensional Bisection Method for Unconstrained Minimization Problem

E.Y. Morozova

Applied Mathematics Department
Herzen State Pedagogical University of Russia
48, Moika Emb., St.-Petersburg, 191186, Russia

melena65@mail.ru

Abstract

An extension of a new multidimensional bisection method for minimizing function over simplex is proposed for solving nonlinear unconstrained minimization problem. The method does not require a differentiability of function, and is guaranteed to converge to the minimizer for the class of strictly unimodal functions. The computational results demonstrating an effectiveness of algorithm for minimizing nonsmooth functions are presented.

Keywords: Convex set, *n*-dimensional simplex, strictly unimodal function, direct search methods, nonlinear unconstrained optimization.

1 Introduction

The problem considered here is an unconstrained minimization problem, which has the general form:

$$f(x) \to \min,\ x \in R^n, \qquad \text{(P)}$$

where $f : R^n \to R$ is a bounded below continuous strictly unimodal function.

We use the following definition of strict unimodality.

Definition. Let D be a bounded closed convex set in R^n. Function $f : D \to R$ is *strictly unimodal over set D* iff for any segment $\Delta \subset D$ $\# Arg \min\{f(x) | x \in \Delta\} = 1$, where «$\# A$» is the cardinality of set A.

The multidimensional bisection method (Baushev and Morozova, 2007) allows to solve constrained minimization problem when the feasible region is n-dimensional simplex. This method generalizes a one-dimensional bisection method for the case n>1 using a recursive procedure. This paper will present an extension of the multidimensional bisection method for solving problem (P). This method does not require a differentiability of function f, and is guaranteed to converge to the minimizer for the class of strictly unimodal functions.

It is known a class of methods that do not explicitly use derivatives - direct search methods for unconstrained optimization. Recent researches have shown the global convergence of pattern search algorithms (a class of direct search methods) for the case when function f is continuously differentiable (Aude & Dennis, 2003, Torczon, 1997). The advantage of the multidimensional bisection method presenting in this paper is that it convergence does not require an assumption about differentiability of function f and method allows to find the minimizer of nonsmooth functions.

In point 2 we describe the multidimensional bisection method (MBM) for minimizing function over simplex. In point 3, the details of the extension of MBM for solving the unconstrained minimization problem are presented. In point 4 some numerical results illustrate the robustness of the method.

2 The Multidimensional Bisection Method

The problem considered is

$$f(x) \to \min,\quad x \in S, \qquad (1)$$

where S - *a n-dimensional* simplex in R^n, and f – a continuous function.

1. Case *n*=1.

The one-dimensional bisection algorithm solves the problem

$$f(x) \to \min,\quad x \in [a,b], \qquad (2)$$

where f is a strictly unimodal function over segment $[a,b]$.

Let $bis(f,a,b,\varepsilon)$ denote the recursive one-dimensional bisection procedure. The inputs for this procedure are: the procedure for calculation values of f, the segment $[a,b]$ and the accuracy ε. The outputs are the estimations x_m for the minimizer x^* and f_m for the value of the minimum of the function f over the segment $[a,b]$.

The iteration of the recursive procedure includes the following steps.

Step 0. If $b - a \ge \varepsilon$, go to step 1, otherwise stop.

Step 1.

$$c = \frac{a+b}{2},\ a' = \frac{a+c}{2},\ b' = \frac{b+c}{2},\ f(c),\ f(a'),\ f(b').$$

Step 2.

If $f(a') \le f(c) \le f(b')$, set $b = b'$.

If $f(a') \ge f(c) \ge f(b')$, set $a = a'$.

If $f(c) \le \min\{f(a'), f(b')\}$, set $a = a'$, $b = b'$.

Step 3. Execute $bis(f, a, b, \varepsilon)$ with new inputs.

2. Case *n*>1.

Let S be a n-dimensional simplex. Let fix the vertex V^0 and denote by $V^1, ..., V^n$ the opposite vertices. Set for each $t \in [0,1]$

$$S_t = conv\{V^0 + t(V^1 - V^0), ..., V^0 + t(V^n - V^0)\}. \quad (3)$$

The set S_t is the $n-1$-dimensional simplex for $0 < t \le 1$.

Set $x_t \equiv \arg\min\{f(x) \mid x \in S_t\}$. Each simplex S_t for $0 < t < 1$ part the initial simplex S in two sets: $conv\{V^0, S_t\}$ and $conv\{S_1, S_t\}$. Fig. 1 illustrate an example of the partition of the simplex S for the case $n = 3$.

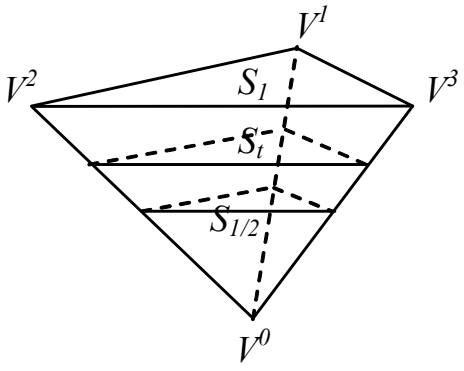

Figure 1: The partition of the simplex *S*

Let $bissimpl(f, SS_1, SS_2, d, \varepsilon)$ denote the recursive procedure in case $n > 1$. The inputs for this procedure are: the procedure for calculation values of *f*, boundary simplices SS_1 and SS_2, the current dimension *d* and the accuracy *ε*. The outputs are the estimations x_m for the minimizer x^* and f_m for the value of the minimum of the function f over the set $conv\{SS_1, SS_2\}$. Originally *d* is equal *n*, then this parameter varies depending on the dimension of the simplex where the point of a minimum is searched. Actually this parameter at first decreases to value 1, and then increases to value *d*=*n*. Three circles of such calculations we consider as the iteration with number *k*. Denote by f^k the estimation of a minimum of the function f and by x^k the estimation of the point x^*. The parameter *d* and the outputs must be declared as global variables and its initial values must be defined before starting procedure $bissimpl(f, SS_1, SS_2, d, \varepsilon)$. More concretely the preliminary step includes the following destinations:

$SS_1 = S_0 = V^0$, $SS_2 = S_1 = conv\{V^1, \dots V^n\}$ according to (7), *d*=*n*;

x^0, x^1, f^0, f^1 we define in a such way that the condition

$$\max\left\{\left|f^k - f^{k-1}\right|, \left\|x^k - x^{k-1}\right\|\right\} < \varepsilon \quad (4)$$

be failed.

Step 1.

If (4) is hold, stop. Otherwise set $\sigma_1 = SS_1$, $\sigma_2 = SS_2$ and go to step 2.

Step 2.

If *d*=1, execute $bis(f, a, b, \varepsilon)$ with $a = SS_1$, $b = SS_2$ Otherwise, go to step 3.

Step 3.

Two cases are possible.

1) SS_1 and SS_2 are *d*-dimensional simplices. Let $V^0_{SS_1}, V^1_{SS_1}, ..., V^d_{SS_1}$ and $V^0_{SS_2}, V^1_{SS_2}, ..., V^d_{SS_2}$ be vertices of simplices SS_1 and SS_2 accordingly. Then we define $S_{\frac{1}{2}}$ $S_{\frac{1}{4}}$ $S_{\frac{3}{4}}$ by

$$S_t = conv\{V^0_{SS_1} + t(V^0_{SS_2} - V^0_{SS_1}), V^1_{SS_1} + t(V^1_{SS_2} - V^1_{SS_1}), \dots, V^d_{SS_1} + t(V^d_{SS_2} - V^d_{SS_1})\}. \quad (5)$$

2) One of the sets SS_1, SS_2 is a vertex, another is *d*-dimensional simplex. In this case $S_{\frac{1}{2}}$ $S_{\frac{1}{4}}$ $S_{\frac{3}{4}}$ are defined by (3). Set $d = d - 1$.

Step 4.

For each of simplices $S_{\frac{1}{2}}$ $S_{\frac{1}{4}}$ $S_{\frac{3}{4}}$ the following actions must be done:

1) Fix a vertex V^0 in the simplex S_t and let $V^1, ..., V^n$ be an opposite vertices.

2) Execute $bissimpl(f, SS_1, SS_2, d, \varepsilon)$ with new values $SS_1 = V^0$ and $SS_2 = conv\{V^1, ..., V^n\}$.

Step 5.

Let x^1_m x^2_m, x^3_m and f^1_m f^2_m, f^3_m be results of the previous step (for $S_{\frac{1}{2}}$ $S_{\frac{1}{4}}$ $S_{\frac{3}{4}}$ accordingly).

If $f^2_m \le f^1_m \le f^3_m$, set $SS_1 = \sigma_1$ $SS_2 = S_{\frac{3}{4}}$.

If $f^2_m \ge f^1_m \ge f^3_m$, set $SS_1 = S_{\frac{1}{4}}$ $SS_2 = \sigma_2$.

If $f^1_m \le \min\{f^2_m, f^3_m\}$, set $SS_1 = S_{\frac{1}{4}}$, $SS_2 = S_{\frac{3}{4}}$.

Set $d = d + 1$.

Step 6.

Execute $bissimpl(f, SS_1, SS_2, d, \varepsilon)$ with current inputs.

The following theorem presents the convergence result.

Theorem. Let $x(\varepsilon)$ be the final estimation of the minimizer x^* for the function *f* where f is a continuous strictly unimodal function over *n-dimensional* simplex *S* then $\lim_{\varepsilon \to 0} x(\varepsilon) = x^*$.

3 The main algorithm

Consider problem (P). The algorithm consecutively solves the following constrained optimization problems:

$$f(x) \to \min,\ x \in S^k, \qquad (6)$$

where S^k is *a n-dimensional* simplex in R^n, k is a number of iteration, k=0, 1, 2,…. Each of problems (6) is solved by the multidimensional bisection method described above. Let $x^k = \{\arg\min f(x) \mid x \in S^k\}$. The algorithm constructs simplexes S^k using two basic operations of reflection and shift, so that $x^{k-1} \in \text{int}S^k$, and generates a sequence of points $\{x^k\}$ with decreasing vales of f:

$$f(x^k) \le f(x^{k-1}), \quad k \in N.$$

This iterative process stops when point $x^k \in \text{int}S^k$. A more formal description of the algorithm is as follows:

Step 1.
Choose $S^0 = conv\{v^{0,i} \mid i = 0,...,n\}$ - an arbitrary initial simplex. Set k=0.

Step 2.
Call the procedure *bissimpl* which solves problem (1).

Step 3.
Finding barycentric coordinates

$$\lambda_i \ge 0,\ i = 0,...,n,\ \sum_{i=0}^{n} \lambda_i = 1 \qquad (7)$$

of the point $x^k \in S^k = conv\{v^{k,i} \mid i = 0,...,n\}$.

Step 4.
If $x^k \in \text{Int}S^k$, then set $x^* = x^k$, stop; otherwise go to step 5.

Step 5.
We have $x^k \in \partial S^k$. Construction of the simplex S^{k+1} by the procedure $reflect(v^0,...,v^n,x^k,\lambda_0,...,\lambda_n,\theta)$ (we shall describe *reflect* below).

Step 6.
Set k=k+1. Go to step 2.

Now consider step 5 in details and describe the procedure $reflect(v^0,...,v^n,x^k,\lambda_0,...,\lambda_n,\theta)$.
The inputs for this procedure are: the vertices $\{v^i \mid i = 0,...,n\}$ of the simplex S^k, the point $x^k = \{\arg\min f(x) \mid x \in S^k\}$, barycentric coordinates (7), small positive number θ. The outputs are the vertices $\{v^{k+1,i} \mid i = 0,...,n\}$ of new simplex S^{k+1}. Let

$$I_0(v) = \{i \mid \lambda_i(v) = 0\},\ i = 0,...,n,$$

$$I_1(v) = \{j \mid \lambda_j(v) \ne 0\},\ j = 0,...,n.$$

The procedure *reflect* includes two following operations (at iteration with number k):

1. *Reflection.* This move reflects the points $v^{k,i}$ for all $i \in I_0(v)$ through the point x^k:

$$\tilde{v}^{k+1,i} = v^{k,i} + 2(x^k - v^{k,i}) \text{ for all } i \in I_0(v).$$

2. *Shift.* Parallel displacement of the vertices $\tilde{v}^{k+1,i}, v^{k,j}$ for all $i \in I_1(v), j \in I_1(v)$ of the simplex $\tilde{S}^{k+1}$ along vector $(x^k - x^c)$, where x^c - centroid of $\tilde{S}^{k+1}$:

$$v^{k+1,i} = \tilde{v}^{k+1,i} + \theta(x^k - x^c) \text{ for all } i \in I_0(v),$$

$$v^{k+1,j} = v^{k,j} + \theta(x^k - x^c) \text{ for all } j \in I_1(v),$$

where θ - some small positive number.

Then $S^{k+1} = conv(v^{k+1,i} \mid i = 0,...,n)$ and we get $x^k \in \text{Int}S^{k+1}$ (figure 2).

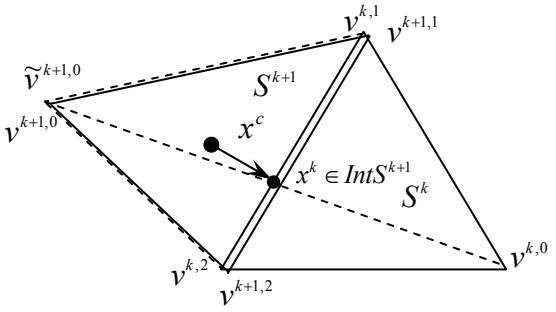

Figure 2: Construction of the simplex S^{k+1} in space R^2

Remark. Note that we get an ε-approximate solution to the original minimization problem (P), where ε is an input for procedure *bissimpl*.
The following lemma is needed to prove the convergence of our algorithm.

Lemma. Let f be continuous strictly unimodal function on the set D and let segment $[x^1,x^2] \subset D$. If $x^3 \in \text{Int}[x^1,x^2]$ and $f(x^1) < f(x^3)$, then $f(x^3) \le f(x^2)$.

Proof. Assume that $f(x^3) > f(x^2)$. Then there is $x^* \in \text{Int}[x^1,x^2] = \arg\max\{f(x) \mid x \in [x^1,x^2]\}$ and there are positive numbers δ_1, δ_2 such that function f increases over segment $[x^* - \delta_1, x^*]$ and decreases over segment $[x^*, x^* + \delta_2]$. Then $f(x^* - \delta_1) = f(x^* + \delta_2)$ by virtue of continuity of the function f, i.e. the points $x^* - \delta_1$ and $x^* + \delta_2$ are minimizers of the function f over segment $[x^* - \delta_1, x^* + \delta_2]$ that contradicts to definition of strict unimodality. □
The following theorem presents the convergence result.
Theorem. If $f : R^n \to R$ is a bounded below continuous strictly unimodal function, S^0 is an arbitrary simplex, $\{x^k\}$, $k = 0, 1, 2,...$ and $S^1, S^2, ..., S^k, ...$ are found by the

above described algorithm then there is number k^* such that $x^* = \{\arg\min f(x) \mid x \in R^n\} \in \operatorname{int} S^{k^*}$.

Sketch of proof. The algorithm of this paper generates the sequence $\{x^k\}$ and $x^k = \{\arg\min f(x) \mid x \in S^k\}$. If $x^k \in \operatorname{int} S^k$, then x^k solves problem (1) by virtue of strict unimodality of function f. If $x^k \in \partial S^k$, then $x^k \in \operatorname{int} S^{k+1}$ according to the rule of construction of simplex S^{k+1}. Let Γ^k be the nearest to the point x^* $n-1$–dimensional face of simplex S^k. We shall show that $x^k \in \Gamma^k$. Assume that $x^k \in \tilde{\Gamma}^k$. Consider segment $[x^k, x^*]$. Let y^k be the point of intersection of face Γ^k with segment $[x^k, x^*]$. Then $f(x^k) < f(y^k) > f(x^*)$, that contradicts to lemma. So, $x^k \in \Gamma^k$. Let $\rho_k = \rho(x^*, \Gamma^k) = \min\{\rho(x^*, y) \mid y \in \Gamma^k\}$. We shall show that $\liminf \rho_k = 0$. Let z^k be the point of emergence of ray $x^k + t(x^* - x^K)$ from simplex S^k and $r_k = \rho(z^k, x^*)$. Sequence r_k is monotonically decreasing to zero, $\lim r_k = 0$. So, $\liminf \rho_k = 0$. Thus, there is number k^* such that $x^* = \{\arg\min f(x) \mid x \in R^n\} \in \operatorname{int} S^{k^*}$. □

4 The numerical results

We implemented the multidimensional bisection method discussed above in MATLAB. The program was tested for different examples of minimization of nonsmooth functions. Some of numerical results we present in this section, some other examples can be found in (Morozova, 2006).

Example 1. Minimization of Dennis-Wood function.
Consider the following variant of Dennis-Wood function (Dennis & Wood, 1987):

$$f(x) = \frac{1}{2}\max\left\{\|x - c_1\|^2, \|x - c_2\|^2\right\}, \qquad (8)$$

where $c_1 = (1, -1)$ $c_2 = -c_1$. This function is continuous and strictly convex, but its gradient is discontinuous everywhere on the line $x_1 = x_2$.

As shown in some works (Kolda and others, 2003, Torczon, 1991) such of direct search methods as compass search, multidirectional search algorithm can fail to converge to the minimizer of function (8).

We will illustrate the convergence of our algorithm to the minimizer of function (8).

The level sets of function (8) and the sequences of the simplexes S^k are shown in figure 3.

The sequence of the points x^k generated by our algorithm converges to the minimizer $x^5(0,0)$ as shown in figure 3. The regular simplex with centre $(1.5; -1)$ and the length of edge $l = 1$ was chosen as an initial simplex. The accuracy ε was chosen equal 10^{-6}. Figure 4 illustrates decreasing function values at the each of six iteration.

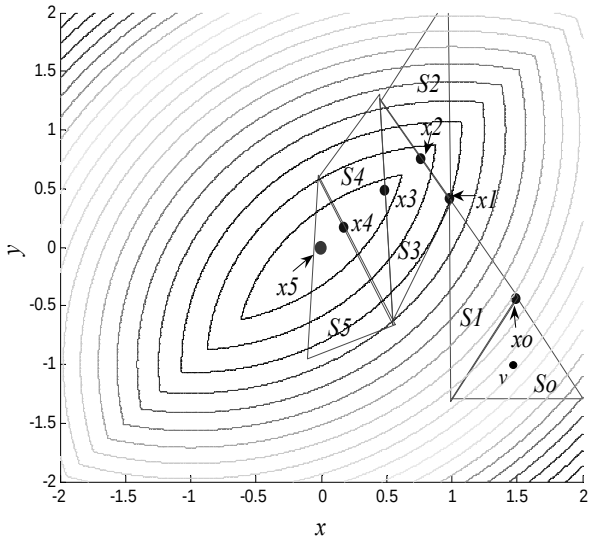

Figure 3: The level sets of the function (3), the sequences of the simplexes S^k and $\{x^k\}$

$x^*(0; 0)$, $f_{\min} = 1$

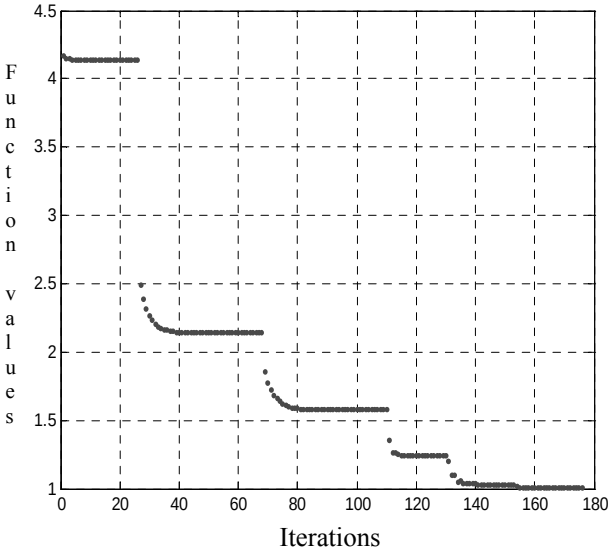

Figure 4: Decreasing function values at the each iteration

Table 1 shows the computational results for each of 6 iterations.

Iteration, k	Minimizer $x^k \subset S^k$	$f(x^k)$
0	$x^0(1.4910; -0.4382)$	4.1368
1	$x^1(0.9821; 0.4123)$	2.1370
2	$x^2(0.7575; 0.7575)$	1.5738
3	$x^3(0.4884; 0.4884)$	1.2385
4	$x^4(0.1717; 0.1717)$	1.0295
5	$x^5(0.0000; 0.0000)$	1.0000

Table 1: Iterative results

Example 2. Minimization of McKinnon function.
This example was chosen for comparing with the most popular direct search method – the Nelder-Mead algorithm (Nelder and Mead, 1965) which convergence is proved only for dimension 1 and some limited results for dimension 2 (Lagarias and others, 1998). At the same time there are examples of family of functions in R^2 (McKinnon, 1998) which demonstrate that the Nelder-Mead simplex algorithm can fail to converge to a stationary point of *f*. Consider the following function of this family:

$$f(x,y)=\begin{cases}360x^2+y+y^2, & x\le 0\\ 6x^2+y+y^2, & x\ge 0\end{cases}. \qquad (9)$$

Function (9) is strictly convex and has up to three continuous derivatives. As shown in (McKinnon, 1998) if the initial simplex is $S^0=conv\{v^0,v^1,v^2\}$,

$$v^0=(0,0),\ v^1=(\lambda_1,\lambda_2),\ v^2=(1,1),$$

$$\lambda_1=\frac{1+\sqrt{33}}{8},\ \lambda_2=\frac{1-\sqrt{33}}{8}, \qquad (10)$$

then all vertices in the Nelder-Mead method converge to a nonminimizing point.
We illustrate that our algorithm applied to the function (9) converges to the minimizer. The initial simplex was equal $S^0=conv\{v^0,v^1,v^2\}$, where vertices v^0,v^1,v^2, and values λ_1,λ_2 where chosen according to (10). The accuracy ε was chosen equal 10^{-6}. As shown in figure 5, point $x^0(0.0542,-0.0381)\in\partial S^0$, $f(x^0)=-0.0190$. After constructing new simplex S^1 and performing the first iteration of our algorithm we have point $x^1(0,-0.5)$ with function value $f(x^1)=-0.25$. Point $x^1\in \operatorname{int} S^1$ is the minimizer of function (9).

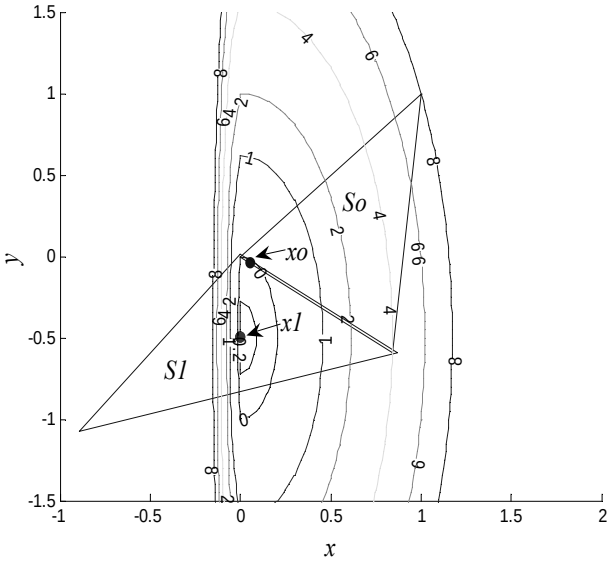

Figure 5: The level sets of the function (9), the sequences of the simplexes S^k and $\{x^k\}$

The level sets of function (9), the sequences of the simplexes S^k and the minimizers x^k are shown in figure 5. Figure 6 illustrates decreasing function values at the each of two iteration.

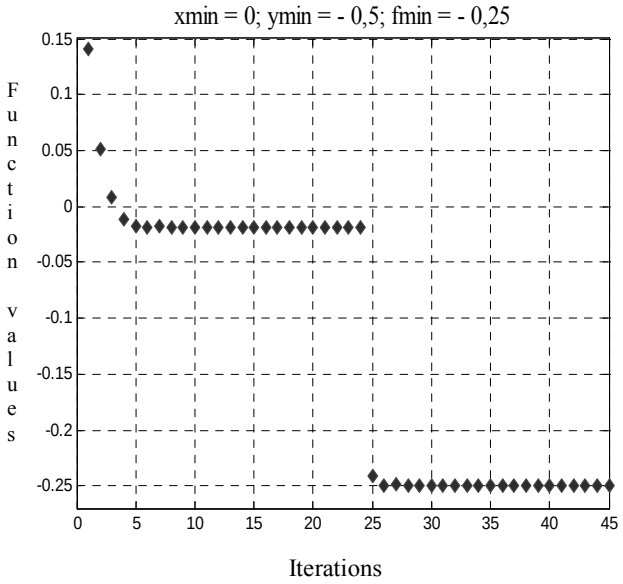

Figure 6: Decreasing function values at the each iteration

Example 3. Minimization of nonsmooth function.

The basic advantage of our algorithm is that it guarantees the convergence for a nonsmooth functions. Consider the following family of nonsmooth functions:

$$f(x,y)=\sum_{k=1}^{n}|x-a_k|^p+\sum_{k=1}^{n}|y-b_k|^p, \qquad (11)$$

where a_k and b_k are some pairwise different real numbers, *n* is an odd number, $0<p\le 1$. If we sort the numbers a_k and b_k in increasing order then the medium point minimizes function (11).
Figure 7 illustrates the convergence of our algorithm for this family of functions when $p=\frac{1}{2}, n=11$ and the numbers a_k and b_k were chosen from the uniform distribution on the segment $[0,1]$. Point $x^2\in \operatorname{int} S^2$ is the minimizer of function (11). Figure 8 illustrates decreasing function values at the each of three iterations.
Table 2 shows the computational results for each of 3 iterations.

Iteration, *k*	Minimizer $x^k\subset S^k$	$f(x^k)$
0	$x^0(17;\ 2)$	17.4847
1	$x^1(4.6858;\ 0.5711)$	8.2839
2	$x^2(0.6649;\ 0.5711)$	3.1503

Table 2: Iterative results

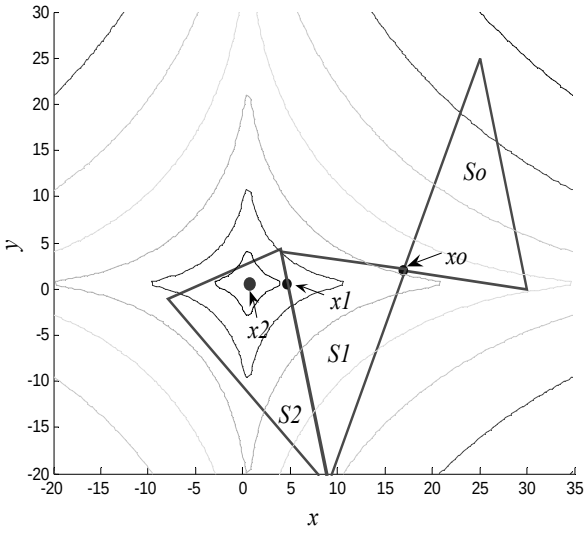

Figure 7: The level sets of the function (11), the sequences of the simplexes S^k and $\{x^k\}$

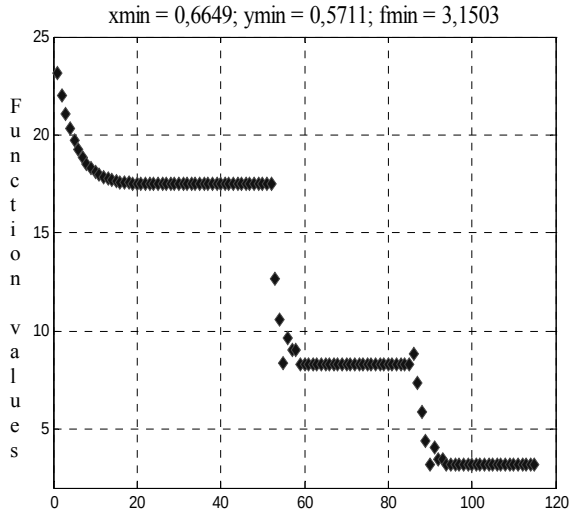

Figure 8: Decreasing function values at the each iteration

5. Conclusion

We have exposed our algorithm for the class of strictly unimodal functions only. However one can show that the algorithm can be applied for a wider class of functions, namely, we consider the class of functions Φ_S, where S - the n-dimensional simplex, defined as follows: $f \in \Phi_S$ iff for any segment $\Delta \subseteq S$ each local minimum of f over this segment is also a global minimum of the function f over this segment. The class Φ_S contains a subclass of strictly unimodal functions over set S. Function (11) considered in last example 3 is belong to the class Φ_S.

References

Baushev A.N., Morozova E.Y (2007): A multidimensional bisection method for minimizing function over simplex. *Lectures notes in engineering and computer science*, **2**:801-803.

Aude C. Dennis J.E. (2003): Analysis of Generalized Pattern Searches. *SIAM J. Optim,* **13**(3):889-903.

Torczon V. (1997): On the convergence of Pattern Search Algorithms. *SIAM J. Optim*, **7(**1):1-25.

Dennis, J. E., Woods, Jr. and Daniel, J. (1987): Optimization on microcomputers: The Nelder-Mead simplex algorithm, *in New Computing Environments: Microcomputers in Large-Scale Computing*, A. Wouk, ed., SIAM, Philadelphia, 116-122.

Morozova, E.Y. (2006): The direct search recursive algorithm for minimizing function of several variables. *The Review of applied and industrial mathematics*, **13**(5):783-796. (In Russian).

Kolda, T. J., Lewis, R. M., Torczon, V. (2003): Optimization by Direct Search: New Perspectives on Some Classical and Modern Methods. *SIAM Review*, **45**(3):385-482.

Torczon, V. (1991): On the convergence of the multidirectional search algorithm. *SIAM J. Optim.*, **1**:123-145.

Nelder J.A. and Mead R. (1965): A simplex method for function minimization. *Computer Journal*, **7**: 308-313.

Lagarias J.C., Reeds J.A., Wright M.H. and Wright P.E. (1998): Convergence properties of the Nelder Mead simplex algorithm in low dimensions. *SIAM J. Optim*, **9**: 112-147.

McKinnon, K.I.M. (1998): Convergence of the Nelder-Mead simplex method to a non-stationary point. *SIAM J. Optim*, **9**: 148-158.

Optimal Joint Vendor-Buyer Inventory Strategy for Deteriorating Items with Salvage Value

Nita H. Shah[1], Ajay S. Gor[2] and H. M. Wee[3*]

[1]Department of Mathematics, Gujarat University, Ahmedabad – 380 009
[2]Pramukh Swami Science & H. D. Patel Arts College, Kadi – 382 715
Gujarat, India
[3]Industrial Engineering Department, Chung Yuan Christian University,
Chungli 32023, Taiwan, ROC

Email: nita_sha_h@rediffmail.com, weehm@cycu.edu.tw

Abstract

This study develops a joint optimal inventory strategy for both the buyer and the vendor when the expired stocks have salvage value, and are subject to constant rate of deterioration. It is shown numerically that the joint approach results in a significant cost reduction when compared with an individual decision by the buyer. We also observed that although the joint total cost decreases, the buyer's cost increases due to larger order. To motivate the buyer to continue to replenish larger order quantity, a permissible delay in payments is offered by the vendor to the buyer. A negotiation factor is introduced to share the benefits of both the parties; the vendor and the buyer.

Keywords : Joint total cost, Deterioration, salvage value, permissible delay in payments.

1 Introduction

In the existing literature, most of the inventory models are derived from the buyer's point of view. This optimal decision policy may not be advantageous in economic terms for the vendor. Thus, there is need to derive a joint policy which turns out to be win-win strategy for both; the vendor and the buyer. Clark and Scarf (1970) studied the vendor-buyer integration for the first time. Banerjee (1986) extended Clark and Scarf's model by introducing finite replenishment rate. Goyal (1988) extended Banerjee's model by relaxing the assumptions of the lot-for-lot production.

The above stated models assumed that the units in inventory remain in utility for the period under review. However, blood components, fruits and vegetables, alcohol, medicines, fashion goods etc looses its utility with the passage of the time. The loss of utility, spoilage, decay or evaporation is categorized as deterioration. Raafat (1991), Shah and Shah (2000), Goyal and Giri (2001) gave up-to-date review of the research articles on deteriorating inventory.

In the preceding review articles, the models assumed that deteriorated units have no sale value. They are considered lost. However, in practice, vendor can offer to his buyer reduced unit cost for the deteriorated stocks. In this article, the joint vendor-buyer inventory system for deteriorating items with salvage value is developed. A negotiation factor is used to facilitate benefit sharing through offering permissible trade credit to the buyer; thus making cooperation relationship more realistic and mutually beneficial.

2 Mathematical Model

The mathematical model is developed on the basis of the following assumptions:

1. A system consists of single vendor and single buyer.
2. The demand rate is deterministic and known.
3. The replenishment rate is infinite.
4. Lead – time is zero or negligible.
5. Shortages are not allowed.
6. The deterioration rate is constant and proportional to on hand inventory.
7. There is no repair or replacement of the deteriorated units during the cycle time.
8. The permissible credit period is used to motivate the buyer to cooperate in the joint inventory system.

The following notations are used

θ : Deterioration rate ($0 < \theta \leq 1$)

d : Demand rate

C_v : Vendor's unit cost

C_b : Buyer's unit cost ($C_b > C_v$)

αC_v : Salvage value associated with deteriorate units for the vendor ($0 \leq \alpha \leq 1$)

αC_b : Salvage value associated with deteriorate units for the buyer

A_v : Vendor's ordering cost

*Corresponding author: Professor H. M. Wee is a faculty in the Industrial Engineering Department from Chung Yuan Christian University, Taiwan.

A_b : Buyer's ordering cost

h_v : Vendor's annual holding cost per time unit

h_b : Buyer's annual holding cost per time unit

T : Vendor's replenishment cycle time

T_b : Buyer's replenishment cycle time

n : Buyer's order times during [0, T]

$I_v(t)$: Vendor-buyer combined inventory level

$I_b(t)$: Buyer's inventory level

TC_v : Vendor's annual total cost per time unit

TC_b : Buyer's annual total cost per time unit

TC : Annual total cost for both the vendor and the buyer

The stocks on hand are depleted due to demand and deterioration. The instantaneous states of inventory for both the vendor and the buyer at any instant of time 't' can be represented by the following differential equations:

$$\frac{dI_b(t)}{dt} + \theta I_b(t) = -d,\ 0 \le t \le \frac{T}{n} \qquad (2.1)$$

and

$$\frac{dI_v(t)}{dt} + \theta I_v(t) = -d,\ 0 \le t \le T \qquad (2.2)$$

With the boundary conditions

$I_b\left(\frac{T}{n}\right) = 0$, $I_v(T) = 0$, $I_b(0) = I_{mb}$ (maximum inventory carried by the buyer) and $I_v(0) = I_{mv}$ (maximum inventory carried by the vendor), the solutions of the differential equations are

$$I_b(t) = \frac{d}{\theta}\left[e^{\theta\left(\frac{T}{n}-t\right)} - 1\right],\ 0 \le t \le \frac{T}{n} \qquad (2.3)$$

and

$$I_v(t) = \frac{d}{\theta}\left[e^{\theta(T-t)} - 1\right],\ 0 \le t \le T \qquad (2.4)$$

Hence, the maximum lot-sizes for the buyer and the vendor are

$$I_{mb}(t) = \frac{d}{\theta}\left[e^{\frac{\theta T}{n}} - 1\right] \qquad (2.5)$$

and

$$I_{mv}(t) = \frac{d}{\theta}\left[e^{\theta T} - 1\right] \qquad (2.6)$$

respectively.

During [0, T], the total inventory holding cost for the buyer is given by

$$IHC_b = nh_b \int_0^{T/n} I_b(t)dt$$

$$= nh_b \frac{d}{\theta^2}\left[e^{\frac{\theta T}{n}} - \frac{\theta T}{n} - 1\right] \qquad (2.7)$$

The actual vendor inventory level in the integrated two-echelon inventory model is the difference between the vendor-buyer combined average inventory level and the buyer average inventory level. Hence, the actual vendor's holding cost in the interval [0, T] is

$$IHC_v = h_v\left[\int_0^T I_v(t)dt - n\int_0^{T/n} I_b(t)dt\right]$$

$$= h_v \frac{d}{\theta^2}\left[e^{\theta T} - 1 - n\left(e^{\frac{\theta T}{n}} - 1\right)\right] \qquad (2.8)$$

In the time period, [0, T], the deterioration cost for the buyer is

$$CD_b = nC_b\left[I_{mb} - \frac{dT}{n}\right]$$

$$= nC_b \frac{d}{\theta}\left[e^{\frac{\theta T}{n}} - \frac{\theta T}{n} - 1\right] \qquad (2.9)$$

and for the vendor is

$$CD_v = C_v\left[I_{mv} - dT - n\left(I_{mb} - \frac{dT}{n}\right)\right]$$

$$= C_v \frac{d}{\theta}\left[e^{\theta T} - 1 - n\left(e^{\frac{\theta T}{n}} - 1\right)\right] \qquad (2.10)$$

The salvage value of the deteriorated units for the buyer during [0, T] is

$$SV_b = n\acute{a}C_b \frac{d}{\theta}\left[e^{\frac{\theta T}{n}} - \frac{\theta T}{n} - 1\right] \qquad (2.11)$$

and the salvage value of the deteriorated units for the vendor during [0, T] is

$$SV_v = \acute{a}C_v \frac{d}{\theta}\left[e^{\theta T} - 1 - n\left(e^{\frac{\theta T}{n}} - 1\right)\right] \qquad (2.12)$$

The ordering cost for the buyer is

$$OC_b = nA_b \qquad (2.13)$$

and for the vendor is

$$OC_v = A_v \qquad (2.14)$$

The buyer's total cost per time unit is given by

$$TC_b = \frac{1}{T}\,[IHC_b + CD_b + OC_b - SV_b]$$

$$= nh_b\,\frac{d}{\theta^2 T}\left[e^{\frac{\theta T}{n}} - \frac{\theta T}{n} - 1\right] + nC_b(1-á)\frac{d}{\theta T}\left[e^{\frac{\theta T}{n}} - \frac{\theta T}{n} - 1\right] + \frac{nA_b}{T} \qquad (2.15)$$

Similarly, the vendor's total cost per time unit is

$$TC_v = \frac{1}{T}\,[IHC_v + CD_v + OC_v - SV_v]$$

$$= h_v\,\frac{d}{\theta^2 T}\left[e^{\theta T} - 1 - n\left(e^{\frac{\theta T}{n}} - 1\right)\right] + C_v(1-á)\frac{d}{\theta T}\left[e^{\theta T} - 1 - n\left(e^{\frac{\theta T}{n}} - 1\right)\right] + \frac{A_v}{T} \qquad (2.16)$$

The annual joint total cost; TC is the sum of TC_b and TC_v. Since, $T_b = \frac{T}{n}$, TC is a function of continuous variable; T_b and discrete variable; n.

3 Computational Algorithm

There are two cases to be discussed.

Case 3.1 When the buyer and the vendor make strategic decision independently.

For the buyer to minimize TC_b, obtain T_b by setting $\frac{\partial TC_b}{\partial T_b} = 0.$

For the vendor, minimum TC_v can be obtain by putting $T = nT_b$ and also satisfying

$$TC_v(n-1) \geq TC_v(n) \leq TC_v(n+1) \qquad (3.1)$$

Then the total annual cost (say) TC_{NJ} without considering integration is

$$TC_{NJ} = \min_n \left\{\min_n\ TC_b) + TC_v\right\} \qquad (3.2)$$

Case 3.2 When the buyer and the vendor make decision jointly.

The optimum value of cycle time T_b and n can be obtained by following necessary conditions.

$$\frac{\partial TC}{\partial T_b} = 0 \qquad (3.3)$$

and

$$TC(n-1) \geq TC(n) \leq TC(n+1) \qquad (3.4)$$

The total cost considering joint decision (say) TC_J, is

$$TC_J = \min_{T_b, n}\,(TC_b + TC_v) \qquad (3.5)$$

Since TC_J is less than TC_{NJ}, there is total cost saving (say) $S_J = TC_{NJ} - TC_J$. Let the buyer's cost savings, S_b, be defined by $S_b = âS_J$ where â is the negotiation factor for benefit sharing. When â = 1, all total cost savings benefit the buyer only, for â = 0, total cost savings benefit the vendor only and when â = 0.5 the total cost savings are equally distributed between the buyer and the vendor. If r is the interest rate, the present value of the unit cost after time M is e^{-rM}. The length of the buyer's credit period M can be computed by solving the equation

$$dC_b\,(1 - e^{-rM}) = S_b$$

which gives

$$M = \frac{1}{r}\ln\left[\frac{dC_b}{dC_b - \beta S_J}\right] \qquad (3.6)$$

The percentage of the joint total cost reduction (PJCR) is defined as

$$PJCR = \frac{TC_{NJ} - TC_J}{TC_{NJ}} \times 100 \qquad (3.7)$$

4 Numerical Example and Sensitivity Analysis

Consider the following parametric values in proper units.

$[d, C_v, C_b, A_v, A_b, h_v, h_b, \alpha, \theta, r, \beta]$ =

[40000, 10, 12, 3000, 600, 1, 1.32, 0.2, 0.1, 0.03, 0.5]

Cases	Without joint decision	With joint decision
n	3	1
T_b	0.1145	0.2795
T	0.3435	0.2795
TC_b	10472.91	14959.58
TC_v	17047.68	10734.51
TC	27520.59	25694.09
PJCR	-	6.64 %
M (yr)	-	0.0635

Table 4.1: The optimal solution without and with joint decision

α	0.0	0.2	0.3	0.4
TC_{NJ}	28966	27520	26768	25995
TC_J	27002	25694	25014	24316
PJCR (in %)	6.78	6.63	6.55	6.46
M	0.0683	0.0634	0.0609	0.0584

Table 4.2: Sensitivity analysis of the proportion salvaged (α)

θ	0.05	0.10	0.15	0.20
TC_{NJ}	24323	27520	30378	32986
TC_J	23448	25694	28279	30644
PJCR (in %)	6.21	6.63	6.91	7.10
M	0.525	0.0634	0.0729	0.0814

Table 4.3: Sensitivity analysis of the deterioration rate (θ)

d	24000	32000	40000	48000
TC_{NJ}	21335	24623	27520	30139
TC_J	19918	22988	25694	29008
PJCR (in %)	6.645	6.640	6.63	6.634
M	0.0822	0.0710	0.0634	0.0579

Table 4.4: Sensitivity analysis of the demand rate (d)

h_b	0.792	1.056	1.32	1.584
TC_{NJ}	26330	26906	27520	28156
TC_J	22516	24158	25694	17142
PJCR (in %)	14.48	10.21	6.63	3.601
M	0.1327	0.0955	0.0634	0.0352

Table 4.5: Sensitivity analysis of the buyer's holding cost (h_b)

In Table 4.1, the comparative study of two cases without and with joint decision. For the joint decision with deterioration, it is observed that the buyer's total annual cost and cycle time increases. The vendor benefits \$ 6313 while the buyer looses \$ 4487 (when $\alpha = 0$, it is less than Yang and Wee (2005). Therefore, the buyer will be reluctant to go for joint strategy. To motivate the buyer to cooperate, the vendor offers the buyer a credit period of 23 days. The joint total cost is reduced by 6.64 %.

The sensitivity analysis of the proportion salvaged (α) is carried out in Table 4.2. It is found that the total annual cost and credit period for both the strategies decreases. Thus, increase in salvage value decreases permissible delay period. This is because instead of throwing away deteriorated units, the vendor is disposing them at a lower price and the buyer can reduce his total cost and so decreasing the permissible delay period is justified. For Table (4.3) and Table (4.4), i.e. increase in deterioration rate forces the buyer to buy more to fulfill his demand and hence to optimize his total cost, so delay period should be increased. The same applies for the case when there is a decrease in demand. Thus, for these two scenarios, the buyer-vendor should go for joint strategy. It is also shown that increasing the buyer's holding cost decreases the trade credit significantly (Table 4.5).

5 Conclusions

In this article, we develop a joint optimal vendor-buyer inventory strategy for deteriorating items with salvage value. It is observed that incorporating the salvage value results in a reduced joint total annual cost of the vendor and the buyer. However, the buyer's cost increases more when the joint decision is taken. To motivate the buyer's co-operation, trade credit offered to the buyer is incorporated in the model. With increasing holding cost, deterioration rate and decreasing salvage value, joint decision is especially beneficial to both parties.

References:

Banerjee, A. (1986), 'A joint economic lot-size model for purchaser and vendor', *Decision Sciences*, 17, 292 – 311.

Clark, A. J. and Scarf, H. (1960), 'Optimal policies for a multi-echelon inventory problem', *Management Science*, 6, 475 – 490.

Goyal, S. K. (1988), 'A joint economic lot-size model for purchaser and vendor: A comment', *Decision Sciences*, 19, 236 – 241.

Goyal, S. K. and Giri, B. C. (2001), 'Recent trends in modeling of deteriorated inventory', *European Journal of Operational Research*, 134, 1 – 16.

Raafat, F. (1991), 'Survey of literature on continuously deteriorating inventory model', *Journal of the Operational Research Society*, 42, 27 – 37.

Shah, Nita H. and Shah, Y. K. (2000), 'Literature survey on inventory models for deteriorating items', *Economic Annals*, XLIV, 221 – 237.

Yang, P. C. and Wee, H. M. (2005), 'A win-win strategy for an integrated vendor-buyer deteriorating inventory system', *Mathematical Modeling and Analysis, Proceeding of the 10-th International Conference MMA2005 & CMAM2*, Trakai, 541 – 546.

Tractable Cases of the Extended Global Cardinality Constraint*

Marko Samer **Stefan Szeider**

Department of Computer Science
Durham University, UK
Email: {marko.samer,stefan.szeider}@durham.ac.uk

Abstract

We study the consistency problem for extended global cardinality (EGC) constraints. An EGC constraint consists of a set X of variables, a set D of values, a domain $D(x) \subseteq D$ for each variable x, and a "cardinality set" $K(d)$ of non-negative integers for each value d. The problem is to instantiate each variable x with a value in $D(x)$ such that for each value d, the number of variables instantiated with d belongs to the cardinality set $K(d)$. It is known that this problem is NP-complete in general, but solvable in polynomial time if all cardinality sets are intervals.

First we pinpoint connections between EGC constraints and general factors in graphs. This allows us to extend the known polynomial-time case to certain non-interval cardinality sets.

Second we consider EGC constraints under restrictions in terms of the treewidth of the value graph (the bipartite graph representing variable-value pairs) and the cardinality-width (the largest integer occurring in the cardinality sets). We show that EGC constraints can be solved in polynomial time for instances of bounded treewidth, where the order of the polynomial depends on the treewidth. We show that (subject to the complexity theoretic assumption FPT $\neq$ W[1]) this dependency cannot be avoided without imposing additional restrictions. If, however, also the cardinality-width is bounded, this dependency gets removed and EGC constraints can be solved in linear time.

Keywords: Global constraints, general factor problem, bounded treewidth, parameterized complexity, dynamic programming, domain consistency

1 Introduction

Constraint satisfaction is a powerful formalism for encoding a wide range of combinatorial problems and is therefore attractive for both practitioners as well as theorists (Rossi et al. 2006). Special purpose constraints with non-constant arity, often referred to as *global constraints*, occur frequently in constraint modeling. Efficient propagation algorithms for such constraints are important for the performance of constraint solvers (van Hoeve & Katriel 2006). Currently the Global Constraint Catalog (Beldiceanu et al. 2005) lists 276 global constraints.

*Research supported by the EPSRC, project EP/E001394/1.

In the sequel we focus on *extended global cardinality constraints* (*EGC constraints*, for short), constraints that occur frequently in constraint modeling and are known as *global_cardinality* (Beldiceanu et al. 2005), *egcc* (Bessiére et al. 2004), *distribution* (Bourdais et al. 2003), and *card_var_gcc* (Régin & Gomes 2004). An EGC constraint is specified by a set D of values, a set X of variables where each variable $x \in X$ ranges over a set $D(x) \subseteq D$ of values, and sets $K(d)$ of non-negative integers associated with values $d \in D$; we refer to the sets $K(d)$ as *cardinality sets*. The EGC constraint requires that the number of variables in X instantiating to a value d must belong to the cardinality set $K(d)$. More specifically, an EGC constraint with variables X and domain D is *consistent* if there is a mapping $\alpha : X \to D$ such that

1. $\alpha(x) \in D(x)$ holds for all variables $x \in X$, and
2. $|\{\, x \in X : \alpha(x) = d \,\}| \in K(d)$ holds for all values $d \in D$.

The *value graph* provides a convenient visualization of the relationship between variables and values present in an EGC constraint; it is the bipartite graph with vertex sets X and D where an edge joins a variable x with a value d if and only if $d \in D(x)$. Figure 1 shows an EGC constraint and its value graph.

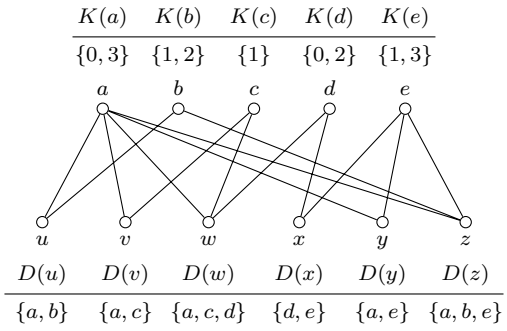

Figure 1: An EGC constraint and its value graph. The constraint is satisfied by $\alpha(u) = b$, $\alpha(v) = c$, $\alpha(w) = d$, $\alpha(x) = d$, $\alpha(y) = e$, and $\alpha(z) = b$.

We refer to the largest integer occurring in the cardinality sets of an EGC constraint as the *cardinality-width* of the constraint. For example, the constraint in Figure 1 has cardinality-width 3.

We consider the following decision problem:

EGCC-Consistency
Instance: An EGC constraint C.
Question: Is the constraint C consistent?

Quimper et al. (2004) have shown that EGCC-Consistency is NP-complete. However, as observed

by Régin (1996), EGCC-Consistency can be decided in polynomial time by network flow algorithms if all cardinality sets are intervals (such constraints are called *global cardinality constraints*). As we will see in Section 3, both results are special cases of an earlier result of Cornuéjols.

Contributions of this paper

We explore properties of EGC constraints that admit polynomial-time consistency checking even in the presence of non-interval cardinality sets.

We discuss connections between EGCC-Consistency and the *general factor problem* for graphs as introduced by Lovász (1970, 1972). This connection seems not to be known in the constraint satisfaction literature. In view of this connection, a general result of Cornuéjols (1988) for the general factor problem allows us to generalize Régin's polynomial-time result from intervals to "2-gap free" cardinality sets.

The main technical contributions of this paper are concerned with the complexity of EGCC-Consistency under structural restrictions. In particular, we consider instances whose value graphs have bounded treewidth. We present a dynamic programming algorithm that allows to decide EGCC-Consistency in polynomial time for instances of bounded treewidth. This algorithm can be easily extended to perform also *domain-filtering* efficiently, that is, to remove from the domains of the variables those values that do not participate in a solution of the constraint. Domain filtering is an important task in the context of constraint solving (van Hoeve & Katriel 2006).

The polynomial that bounds the running time of our dynamic programming algorithm depends on the treewidth of the instance. However, if we additionally bound the cardinality-width, this dependency is removed and the algorithm runs in linear time. The question arises whether this dependency can be avoided without bounding the cardinality-width. We answer this question negatively, subject to the complexity theoretic assumption FPT $\neq$ W[1] (see Section 6).

As a corollary, we obtain that Lovász's general factor problem, parameterized by the treewidth of the input graph, is W[1]-hard. This result may be of independent interest.

The remainder of the paper is organized as follows. In Section 2 we give basic definitions and background on constraints and treewidth. In Section 3 we discuss the connection between EGC constraints and general factors in graphs, and we give a tractability result applying general results of Cornuéjols and Courcelle. In Section 4 we present the dynamic programming algorithm for instances of bounded treewidth; in Section 5 we extend this algorithm to domain filtering. In Section 6 we prove the W[1]-hardness result.

2 Preliminaries

2.1 Constraint Satisfaction

A *constraint network* consists of a finite set X of *variables*, a finite set D of *values*, and a finite set of *constraints*. Each variable $x \in X$ ranges over a set $D(x) \subseteq D$ of values, the *domain* of x. Each constraint C specifies the allowed combinations of values for a set $\mathrm{var}(C) \subseteq X$ of variables, the *scope* of C; the *arity* of a constraint is the cardinality of its scope. An *assignment* is a mapping α that assigns to each variable $x \in X$ a value $\alpha(x) \in D(x)$. An assignment α *satisfies* a constraint C if α instantiates the variables in the scope of C such that an allowed combination of values is formed. An assignment that satisfies simultaneously all constraints is a *solution* of the constraint network. A constraint C is *consistent* if it is satisfied by at least one assignment, and it is *domain consistent* if for every variable $x \in \mathrm{var}(C)$ and every value $d \in D(x)$ there exists an assignment α that satisfies C and instantiates x with d. Given a constraint C, *domain filtering* is the task of removing values d from domains of variables $x \in \mathrm{var}(C)$ if there is no assignment that satisfies C and instantiates x with d. What we call domain filtering is sometimes called "complete" domain filtering in order to emphasize that *all* superfluous values are removed from domains, in contrast to weaker forms of domain filtering that only achieve "range consistency" or "bounds consistency" (van Hoeve & Katriel 2006).

For an EGC constraint C with set X of variables and set D of values we say that C is *over* (X, D). The input size of an EGC constraint C over (X, D) is of order $\|C\| = \sum_{x \in X}(|D(x)| + 1) + \sum_{d \in D}(|K(d)| + 1)$. Thus, for the value graph $G = (V, E)$ of C, we have $|V| + |E| \leq \|C\|$.

Note that we consider the EGC constraint as an *intensional* constraint. Typically, global constraints are considered as intensional constraints since for many global constraints an *extensional* representation (where all combinations of values that satisfy the constraint are explicitely listed) would require exponential space. Domain filtering and consequently testing for domain consistency or consistency is trivial for extensional constraints as these properties can be read off from the constraint relation. For intensional constraints, however, testing for domain consistency or consistency is nontrivial and known to be NP-complete for several classes of constraints (Bessiére et al. 2004, Quimper et al. 2004), in particular for the EGC constraint. Note that whenever domain filtering can be accomplished in polynomial time for a constraint, then its consistency can be checked in polynomial time as well (Bessiére et al. 2004), but the converse does not hold in general (Sellmann 2003).

2.2 Tree Decompositions

Treewidth is an important graph invariant which is a measure of "tree-likeness." For graphs of treewidth bounded by a constant many otherwise intractable problems become tractable, e.g., 3-colorability, Hamiltonicity, etc. It is generally believed that many practically relevant problems actually do have low treewidth (Bodlaender 1993).

The treewidth of a graph $G = (V, E)$ is defined via the following notion of decomposition: a *tree decomposition* of G is a pair (T, χ), where T is a tree and χ is a labeling function with $\chi(t) \subseteq V$ for every tree node t such that the following conditions hold:

1. Every vertex of G occurs in $\chi(t)$ for some tree node t.
2. For every edge uv of G there is a tree node t such that $u, v \in \chi(t)$.
3. For every vertex v of G, the tree nodes t with $v \in \chi(t)$ induce a connected subtree of T ("Connectedness Condition").

The *width* of a tree decomposition (T, χ) is the cardinality of a largest set $\chi(t)$ minus 1 among all nodes t of T. A tree decomposition of smallest width is *optimal*. The *treewidth* of a graph G is the width of an optimal tree decomposition of G.

Note that a graph $G = (V, E)$ of treewidth k has at most $k|V|$ edges (Kloks 1994). Thus if k is bounded by a constant, then $|E| = O(|V|)$.

In principle, one can compute in linear time an optimal tree decomposition of graphs with treewidth bounded by some constant k (Bodlaender 1996); however the running time of the known linear-time algorithm imposes a huge hidden constant. In practice one often prefers to obtain tree decompositions via heuristics. An important class of tree decomposition heuristics are based on finding an appropriate linear ordering of the vertices (Bodlaender 2005, Koster et al. 2001).

Once a tree decomposition of small width is found, one tries to solve the problem under consideration by dynamic programming via bottom-up traversal of the tree decomposition. For such an approach it is often convenient to consider tree decompositions in the following normal form (Kloks 1994): A triple (T, χ, r) is a *nice tree decomposition* of a graph G if (T, χ) is a tree decomposition of G, the tree T is rooted at node r, and each node of T is of one of the following four types:

1. a *leaf node*: a node having no children;
2. a *join node*: a node t having exactly two children t_1, t_2, and $\chi(t) = \chi(t_1) = \chi(t_2)$;
3. an *introduce node*: a node t having exactly one child t', and $\chi(t) = \chi(t') \cup \{v\}$ for a vertex v of G;
4. a *forget node*: a node t having exactly one child t', and $\chi(t) = \chi(t') \setminus \{v\}$ for a vertex v of G.

Note that for every vertex v of G that does not occur in $\chi(r)$ there is exactly one node t_v with parent t'_v such that $v \in \chi(t_v)$ and $v \notin \chi(t'_v)$ (t'_v is a forget node). If v occurs in $\chi(r)$ we set $t_v = r$. In both cases we say that t_v is the *final node* of v.

For every constant k, given a tree decomposition of a graph G of width k, one can effectively obtain in linear time a nice tree decomposition of G with $O(n)$ nodes and of width at most k (Kloks 1994).

3 EGC Constraints and General Factors in Graphs

3.1 Applying Cornuéjols's Theorem

Lovász (1970, 1972) introduced the following problem, known as the *general factor problem*:

> GenFactor
>
> *Instance:* A graph $G = (V, E)$ and a mapping K that assigns to each vertex $v \in V$ a set $K(v) \subseteq \{0, \ldots, d(v)\}$ of integers, where $d(v)$ denotes the degree of v in G.
>
> *Question:* Is there a subset $F \subset E$ such that for each vertex $v \in V$ the number of edges in F incident with v is an element of $K(v)$?

Clearly, EGCC-Consistency is the special case of GenFactor where G is bipartite with bipartition (X, D) and $K(v) = \{1\}$ for all $v \in X$. Thus, similar to EGC constraints, we call the sets $K(v)$ cardinality sets and we call the largest integer occurring in the cardinality sets of a GenFactor instance its cardinality-width.

Let $\mathcal{K}$ be a class of finite sets of non-negative integers. We denote by GenFactor($\mathcal{K}$) and EGCC-Consistency($\mathcal{K}$) the respective problems restricted to instances where all cardinality sets belong to the class $\mathcal{K}$. A set K of integers has an *s-gap* if there exists an integer i such that $\min(K) < i < \max(K)$ and $\{i, \ldots, i+s-1\} \cap K = \emptyset$. We denote by $\mathcal{I}_s$ the class of s-gap free sets of non-negative integers. Note that $\mathcal{I}_1$ is nothing but the class of non-negative intervals. We can state Régin's above mentioned result as follows.

Proposition 1 (Régin (1996))**.** EGCC-Consistency*($\mathcal{I}_1$) can be decided in polynomial time.*

The following dichotomy result fully classifies the problem GenFactor($\mathcal{K}$).

Theorem 2 (Cornuéjols (1988))**.** *If $\mathcal{K} \subseteq \mathcal{I}_2$, then* GenFactor*($\mathcal{K}$) can be decided in polynomial time. Otherwise,* GenFactor*($\mathcal{K}$) is* NP*-complete.*

Actually, the reduction given by Cornuéjols (1988) shows that the NP-hardness case of Theorem 2 even holds if the graph is bipartite and the vertices on one side have cardinality sets $\{1\}$, the vertices on the other side have cardinality sets drawn from the class $\mathcal{K}$. Hence, it follows that the dichotomy stated in Theorem 2 also holds for EGCC-Consistency($\mathcal{K}$).

Corollary 3. *If $\mathcal{K} \subseteq \mathcal{I}_2$, then* EGCC-Consistency *can be decided in polynomial time. Otherwise,* EGCC-Consistency*($\mathcal{K}$) is* NP*-complete.*

Thus EGCC-Consistency($\mathcal{I}_2$) can be decided in polynomial time, a proper generalization of Proposition 1. A further consequence of Corollary 3 is the NP-completeness of EGCC-Consistency($\{\{0, 3\}\}$), which gives the following.

Corollary 4. EGCC-Consistency *is* NP*-complete for instances of cardinality-width at least 3.*

3.2 Applying Courcelle's Theorem

Courcelle's Theorem (Courcelle 1987) provides a powerful tool for showing that certain graph properties can be checked in linear time for graphs of bounded treewidth. One only needs to define the considered property in terms of a certain formalism (Monadic Second Order Logic, MSO) where one is allowed to quantify over sets of vertices and sets of edges (see, e.g., Downey and Fellows' book (Downey & Fellows 1999) for further details and examples). In fact, with Courcelle's Theorem it is easy to establish the following.

Proposition 5. *The problems* GenFactor *and* EGCC-Consistency *can be decided in linear time for instances having both bounded treewidth and bounded cardinality-width.*

Let us sketch the proof. Note that we only need to consider GenFactor since it contains EGCC-Consistency as a special case. Let $G = (V, E)$ with cardinality sets $K(v)$, $v \in V$, be an instance of GenFactor with cardinality-width m. We may assume, w.l.o.g., that all vertices have degree at least two, as vertices of degree 0 or 1 can be easily eliminated. Let $K_1, \ldots, K_{2^{m+1}}$ be an enumeration of all subsets of $\{0, \ldots, m\}$. We assign to every vertex v of G an integer $i(v)$ such that $K(v) = K_{i(v)}$. Now we construct a graph G' from G by attaching to every vertex v of G new neighbors $v_1, \ldots, v_{i(v)}$ of degree 1. Since the added vertices are of degree 1, we can distinguish them from the old vertices. The new vertices allow us to reconstruct the cardinality sets $K(v)$. Since m is a constant, we can define predicates $P_1, \ldots, P_{2^{m+1}}$ such that $P_i(v)$ is true for a vertex of G' if and only if v is of degree at least 2 and has exactly i neighbors of degree 1 (equivalently, v belongs to G and $K(v) = K_{i(v)}$). It is now easy to state an MSO sentence that is true for G' if and only if G has a general

factor that meets the cardinality conditions imposed by the sets $K(v)$. Hence Proposition 5 follows from Courcelle's Theorem.

In the above construction it was essential that the cardinality-width is bounded, since otherwise we could not confine us to a finite number of predicates P_i. The question arises whether this limitation can be overcome by a different, more sophisticated approach. We will return to this question in Section 6, where we will provide a negative answer. Namely we will show that EGCC-Consistency, parameterized by the treewidth alone, is complete for the parameterized complexity class W[1]. Hence one cannot expect the existence of an algorithm that solves EGCC-Consistency (or GenFactor) for instances of bounded treewidth within a running time that is bounded by a polynomial of order independent of the treewidth.

Algorithms obtained via Courcelle's Theorem are impractical as the linear running time involves a huge hidden factor. Therefore we propose in the next section an efficient combinatorial algorithm based on dynamic programming.

4 Efficient Consistency Checking

In the following we consider an EGC constraint C over (X, D) together with a nice tree decomposition (T, χ, r) of the value graph of C. Let m denote the cardinality-width of C. For every node t of T let $\mathrm{var}(t)$ denote the set of variables in $\chi(t)$ and let $\mathrm{val}(t)$ denote the set of values in $\chi(t)$. We define $\mathrm{var}^*(t)$ as the union of $\mathrm{var}(t')$ for tree nodes t' that belong to the subtree rooted at t; $\mathrm{val}^*(t)$ is defined similarly. Thus, $\mathrm{var}^*(t) \setminus \mathrm{var}(t)$ and $\mathrm{val}^*(t) \setminus \mathrm{val}(t)$ are the sets of variables and values, respectively, that are already "forgotten" at tree node t; similarly $X \setminus \mathrm{var}^*(t)$ and $D \setminus \mathrm{val}^*(t)$ are the sets of variables and values, respectively, that are "not yet introduced" at tree node t.

With every tree node t we associate the set $A(t)$ of partial assignments $\alpha : X_\alpha \to \mathrm{val}^*(t)$ defined on sets of variables $X_\alpha \subseteq \mathrm{var}^*(t)$ such that

1. $\alpha(x) \in D(x)$ for all $x \in X_\alpha$ (α respects domains),
2. $\mathrm{var}^*(t) \setminus \mathrm{var}(t) \subseteq X_\alpha$ (α is defined for all forgotten variables), and
3. $|\{\, x \in \mathrm{var}^*(t) : \alpha(x) = d \,\}| \in K(d)$ for all $d \in \mathrm{val}^*(t) \setminus \mathrm{val}(t)$ (α respects cardinality sets for forgotten values).

A *record* of a tree node t is a mapping

$$R : \chi(t) \to \mathrm{val}(t) \cup \{\sqcup, \star\} \cup \{0, 1, 2, \ldots, m\}$$

such that the following two conditions are satisfied:

1. $R(x) \in (\mathrm{val}(t) \cap D(x)) \cup \{\sqcup, \star\}$ for $x \in \mathrm{var}(t)$;
2. $R(d) \in \{0, 1, 2, \ldots, \max(K(d))\}$ for $d \in \mathrm{val}(t)$.

We use records to represent the partial assignments α in the set $A(t)$: $R(x) = \sqcup$ means that α is not defined for x, and $R(x) = \star$ means that α maps x to a value that is already forgotten. More specifically, we say that a record R of a tree node t *represents* an assignment $\alpha \in A(t)$ if the following two conditions hold:

1. For all $x \in \mathrm{var}(t)$

$$R(x) = \begin{cases} \alpha(x) & \text{if } \alpha(x) \in \mathrm{val}(t), \\ \star & \text{if } \alpha(x) \in \mathrm{val}(t)^* \setminus \mathrm{val}(t), \\ \sqcup & \text{otherwise (i.e., if } x \in \mathrm{var}(t) \setminus X_\alpha); \end{cases}$$

2. for all $d \in \mathrm{val}(t)$

$$R(d) = |\{\, x \in \mathrm{var}^*(t) : \alpha(x) = d \,\}|.$$

Note that in general a record can represent an unbounded number of assignments, and every assignment in $A(t)$ is represented by exactly one record R of t. We say that a record R of t is *valid* if it represents some assignment $\alpha \in A(t)$.

The following lemma follows directly from the definitions.

Lemma 6. *C is consistent if and only if there is a valid record R of the root r such that $R(x) \neq \sqcup$ for all $x \in \mathrm{var}(r)$ and $R(d) \in K(d)$ for all $d \in \mathrm{val}(r)$.*

The next five lemmas will allow us to compute the valid records of a tree node from the valid records of its children.

Lemma 7 (Join nodes)**.** *Let t_1, t_2 be the children of t. A record R of t is valid if and only if there are valid records R_1 and R_2 of t_1 and t_2, respectively, such that*

1. *for all $x \in \mathrm{var}(t)$ one of the following holds*
   - *(a) $R(x) = R_1(x) = R_2(x) \in D \cup \{\sqcup\}$;*
   - *(b) $R(x) = R_1(x) = \star$ and $R_2(x) = \sqcup$;*
   - *(c) $R(x) = R_2(x) = \star$ and $R_1(x) = \sqcup$;*
2. *$R(d) = R_1(d) + R_2(d) - |\{\, x \in \mathrm{var}(t) : R(x) = d \,\}|$ for all $d \in \mathrm{val}(t)$.*

Proof. Let R be a valid record of t. By definition, R represents an assignment $\alpha \in A(t)$. For $i = 1, 2$ let $\alpha_i = \alpha|_{\mathrm{var}^*(t_i)}$ be the restriction of α to $\mathrm{var}^*(t_i)$. Since $\alpha \in A(t)$ it can be easily verified that also $\alpha_i \in A(t_i)$, $i = 1, 2$. Let R_i be the (valid) record of t_i that represents α_i, $i = 1, 2$. It remains to check that for R, R_1, and R_2 the two properties stated in the lemma hold. Let $x \in \mathrm{var}(t)$. If $R_1(x), R_2(x) \in D(x) \cup \{\sqcup\}$, then $R(x) = R_1(x) = R_2(x)$, since $\mathrm{var}(t) = \mathrm{var}(t_1) = \mathrm{var}(t_2)$ and $X_\alpha \cap \mathrm{var}(t) = X_{\alpha_1} \cap \mathrm{var}(t_1) = X_{\alpha_2} \cap \mathrm{var}(t_2)$. If $R_1(x) = \star$ and $R_2(x) = \sqcup$, then $\alpha_1(x) \in \mathrm{val}^*(t_1) \setminus \mathrm{val}(t_1)$ and $x \notin X_{\alpha_2}$; hence $\alpha(x) \in \mathrm{val}^*(t) \setminus \mathrm{val}(t)$, and so $R(x) = \star$. Symmetrically, if $R_1(x) = \sqcup$ and $R_2(x) = \star$, then $R(x) = \star$. Thus the first property of the lemma holds for R, R_1, and R_2. It is easy to see that $R(d) = R_1(d) + R_2(d) - |\{\, x \in \mathrm{var}(t) : R(x) = d \,\}|$ for all $d \in \mathrm{val}(t)$, hence the second property holds as well.

Conversely, let R be a record of t, and assume that there are valid records R_1 and R_2 of t_1 and t_2, respectively, such that the two properties stated in the lemma hold. Let $\alpha_1 \in A(t_1)$ and $\alpha_2 \in A(t_2)$ be assignments that are represented by R_1 and R_2, respectively. By the connectedness condition of a tree decomposition, it follows that the sets $\mathrm{var}^*(t_1) \setminus \mathrm{var}(t_1)$ and $\mathrm{var}^*(t_2) \setminus \mathrm{var}(t_2)$ are disjoint. Hence, we can combine α_1 and α_2 to an assignment $\alpha : X_\alpha \to D$, defined for the set $X_\alpha = X_{\alpha_1} \cup X_{\alpha_2}$, with $\alpha(x) \in D(x)$ for all $x \in X_\alpha$. It is easy to check that α corresponds to R, hence R is a valid record of t. □

The proofs of the following four lemmas are straightforward.

Lemma 8 (Introduce variable)**.** *Let t be an introduce node with child t' such that $\mathrm{var}(t) = \mathrm{var}(t') \cup \{x_0\}$ and $\mathrm{val}(t) = \mathrm{val}(t')$. A record R of t is valid if and only if there is a valid record R' of t' such that $R'(x) = R(x)$ for all $x \in \mathrm{var}(t')$, and one of the following prevails:*

1. *$R(x_0) = \sqcup$ and $R(d) = R'(d)$ for all $d \in \mathrm{val}(t)$,*

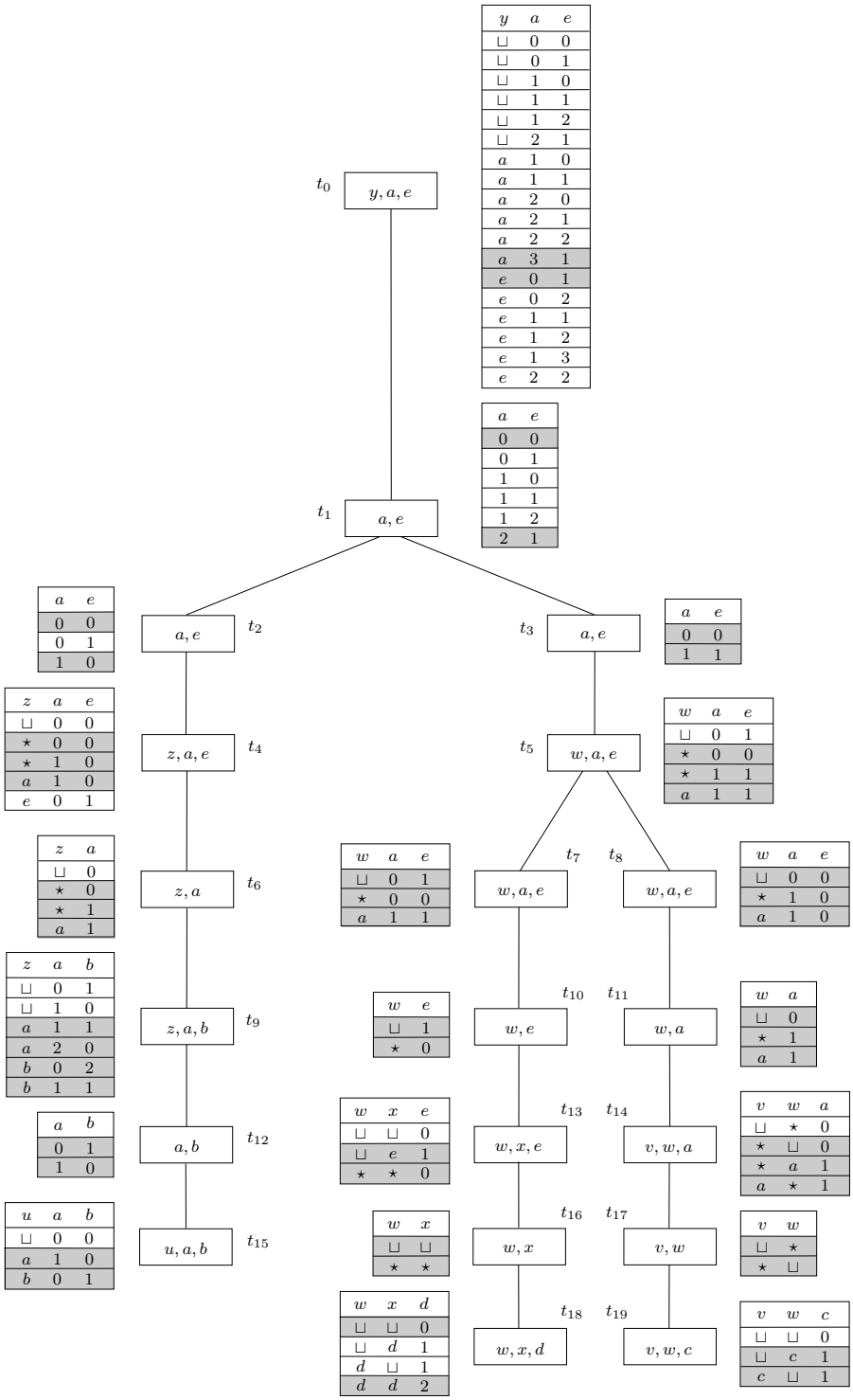

Figure 2: A nice tree decomposition of the value graph of the EGC constraint in Figure 1 with tables representing valid records. Rows with gray background indicate the result after domain filtering.

2. $R(x_0) = d_0$ *for some* $d_0 \in \mathrm{val}(t) \cap D(x_0)$ *such that* $R(d_0) = R'(d_0)+1$ *and* $R(d) = R'(d)$ *for all* $d \in \mathrm{val}(t) \setminus \{d_0\}$,

Lemma 9 (Introduce value)**.** *Let t be an introduce node with child t' such that* $\mathrm{val}(t) = \mathrm{val}(t') \cup \{d_0\}$ *and* $\mathrm{var}(t) = \mathrm{var}(t')$*. A record R of t is valid if and only if there is a valid record R' of t' such that the following conditions hold:*

1. *for all* $x \in \mathrm{var}(t)$

$$R(x) = \begin{cases} d \in \{\sqcup\} \cup (\{d_0\} \cap D(x)) & \text{if } R'(x) = \sqcup; \\ R'(x) & \text{otherwise;} \end{cases}$$

2. *for all* $d \in \mathrm{val}(t)$

$$R(d) = \begin{cases} |\{\, x \in \mathrm{var}(t) : R(x) = d_0 \,\}| & \text{if } d = d_0; \\ R'(d) & \text{otherwise;} \end{cases}$$

Lemma 10 (Forget variable)**.** *Let t be a forget node with child t' such that* $\mathrm{var}(t) = \mathrm{var}(t') \setminus \{x_0\}$ *and* $\mathrm{val}(t) = \mathrm{val}(t')$*. A record R of t is valid if and only if there is a valid record R' of t' such that $R'(x_0) \neq \sqcup$ and $R(z) = R'(z)$ for all $z \in \mathrm{var}(t) \cup \mathrm{val}(t)$.*

Lemma 11 (Forget value)**.** *Let t be a forget node with child t' such that* $\mathrm{val}(t) = \mathrm{val}(t') \setminus \{d_0\}$ *and* $\mathrm{var}(t) =$

$\mathrm{var}(t')$. *A record R of t is valid if and only if there is a valid record R' of t' such that the following conditions hold:*

1. $R'(d_0) \in K(d_0)$;
2. *for all* $x \in \mathrm{var}(t)$ *we have*

$$R(x) = \begin{cases} \star & \text{if } R'(x) = d_0; \\ R'(x) & \text{otherwise;} \end{cases}$$

3. $R(d) = R'(d)$ *for all* $d \in \mathrm{val}(t) = \mathrm{val}(t') \setminus \{d_0\}$.

Theorem 12. EGCC-Consistency *can be decided in linear time for instances having both bounded treewidth and bounded cardinality-width.*

Proof. Let $k, m \geq 0$ be arbitrary constants. We are given an EGC constraint C over (X, D) with treewidth and cardinality-width bounded by k and m, respectively. Let n denote the number of vertices of the value graph of C. We compute a nice tree decomposition (T, χ, r) of the value graph of C such that the width of the tree decomposition is at most k and T has $O(n)$ nodes. This can be accomplished in time $O(n)$ (see the discussion in Section 2.2).

With every tree node t of T we associate the set $M(t)$ of valid records of t. We can compute the sets $M(t)$ via a bottom-up traversal of T as follows. For a leaf node t we can compute $M(t)$ just by considering all possible rows R with $R(x) \in \mathrm{val}(t) \cup \{\sqcup\}$ for $x \in \mathrm{var}(t)$ and $R(d) = |\{\, x \in \mathrm{var}(t) : R(x) = d \,\}|$ for $d \in \mathrm{val}(t)$. The number of records of a node t is at most

$$|\mathrm{val}(t) \cup \{\sqcup, \star\}|^{|\mathrm{var}(t)|} \cdot m^{|\mathrm{val}(t)|} \leq \max(k+1, m)^{k+1},$$

i.e., bounded purely in terms of the constants k and m. Lemmas 7–11 ensure that for computing $M(t)$ of a non-leaf node t we only need to know the sets $M(t')$ of the children t' of t: For a join node t with children t_1 and t_2 we compute $M(t)$ by combining all pairs of records $R_1 \in M(t_1)$, $R_2 \in M(t_2)$, and by checking the conditions of Lemma 7. The time required for each pair is bounded in terms of the constants k and m. Hence, given $M(t_1)$ and $M(t_2)$, we can compute $M(t)$ in constant time. Computing the sets $M(t)$ for introduce and forget nodes t according to Lemmas 8–11 is even simpler. Hence we can compute the sets $M(t)$ for all $O(n)$ tree nodes t in time $O(n)$. According to Lemma 6 we can decide consistency of C by examining the records in $M(r)$ at the root r. □

Figure 2 shows a nice tree decomposition of the value graph of the constraint of Figure 1, together with the sets $M(t)$ as computed according to the proof of Theorem 12. Records are specified as table rows (the meaning of table rows with gray backgrounds will be discussed in the next section).

5 Efficient Domain Filtering

Consider an EGC constraint C over (X, D). For each pair $x \in X$ and $d \in D(x)$ let $C[x = d]$ denote the EGC constraint obtained from C by instantiating x with d (that is, x gets removed and $K(d)$ gets replaced with $K'(d) = \{\, j - 1 : j \in K(d) \setminus \{0\} \,\}$). Evidently, C is domain consistent (recall the definition in Section 2.1) if and only if all constraints $C[x = d]$ for $x \in X$ and $d \in D(x)$ are consistent. Hence, domain filtering for EGC constraints of bounded treewidth and bounded cardinality-width can be carried out in *quadratic time*, using the algorithm of Theorem 12 for each pair $x \in X$ and $d \in D(x)$. However, the following approach allows domain filtering in *linear time*.

As in the previous section, let C be an EGC constraint over (X, D). We assume that treewidth and cardinality-width of C are bounded by constants k and m, respectively. Let (T, χ, r) be a nice tree decomposition of the value graph of C such that the width of the tree decomposition is at most k and T has $O(n)$ nodes, $n = |X| + |D|$.

We call a record R of a tree node t *solution-valid* if R represents the restriction $\alpha|_{\mathrm{var}^*(t)}$ of a solution $\alpha : X \to D$ of C to $\mathrm{var}^*(t)$. Note that every solution-valid record is valid. The following lemma is a direct consequence of the definitions (recall from the end of Section 2.2 the notion of "final node").

Lemma 13. *C is domain consistent if and only if*

1. *for every pair x, d with $x \in X$ and $d \in D(x)$ there exists a solution-valid record R of some tree node t with $R(x) = d$, and*
2. *for every pair d, j with $d \in D$ and $j \in K(d)$ there exists a solution-valid record R at the final node of d such that $R(d) = j$.*

Hence, if we have computed all solution-valid records of all tree nodes, then we have solved the domain filtering task. With every tree node t of T we associate the set $M(t)$ of valid records and the set $M'(t) \subseteq M(t)$ of solution-valid records of t. The sets $M(t)$ are computed in linear time by the algorithm described in the proof of Theorem 12. Next we describe how we can compute the subsets $M'(t)$ by means of a top-down traversal of T. This process is illustrated in Figure 2 where solution-valid records are indicated as table rows with gray background.

For the root r we can easily compute $M'(r)$ from $M(r)$ since, according to Lemma 6, a valid record R at r is solution-valid if and only if $R(x) \neq \sqcup$ for all $x \in \mathrm{var}(r)$ and $R(d) \in K(d)$ for all $d \in \mathrm{val}(r)$.

Consider a join node t with children t_1, t_2, and assume that we have already computed the set $M'(t)$. It follows from Lemma 7 that a valid record R_1 of t_1 is solution-valid if and only if there is a solution-valid record R of t and a valid record R_2 of t_2 such that for the records R, R_1, R_2 the properties stated in Lemma 7 hold. Hence, we can compute the sets $M'(t_1)$ and $M'(t_2)$ from $M'(t)$ in time that only depends on the constants k and m. Similarly, for an introduce or forget node t with child t', a record R' of t' is solution-valid if and only if there is a solution-valid record R of t such that the properties stated in one of the Lemmas 8–11 hold. Thus if we know $M'(t)$ we can compute $M'(t')$ in time that only depends on the constants k and m.

Since T has $O(n)$ many nodes, we have shown the following result.

Theorem 14. *Domain filtering for extended global cardinality constraints can be carried out in linear time if both treewidth and cardinality-width are bounded.*

6 W[1]-Hardness for Parameter Treewidth

We return to the question raised at the end of Section 3 of whether bounding the cardinality-width is dispensable in Proposition 5. The framework of parameterized complexity (Downey & Fellows 1999) offers concepts and tools for answering this question. Let us briefly review the main concepts of this framework; for an in-depth treatment we refer to other sources (Downey & Fellows 1999, Flum & Grohe 2006, Niedermeier 2006).

In parameterized complexity one considers problems in two dimensions: one dimension is the usual size n of the instance and the second dimension is the parameter (usually a positive integer k). A parameterized problem is called *fixed-parameter tractable* if it can be solved in time $O(f(k)\cdot n^c)$ for some computable function f and constant c that is independent of the parameter. FPT denotes the class of fixed-parameter tractable decision problems. The parameterized complexity classes $\mathrm{W}[1] \subseteq \mathrm{W}[2] \subseteq \cdots \subseteq \mathrm{W}[\mathrm{P}]$ contain problems that are believed to be not fixed-parameter tractable (Downey & Fellows 1999); all inclusions are believed to be proper. There are different kinds of evidence for assuming that $\mathrm{W}[1] \neq \mathrm{FPT}$. For example, $\mathrm{W}[1] \neq \mathrm{FPT}$ would imply that the Exponential Time Hypothesis fails (cf. Flum & Grohe (2006)). A parameterized problem P *reduces* to a parameterized problem Q if we can transform an instance (x,k) of P into an instance $(x', g(k))$ of Q in time $O(f(k)\cdot|x|^c)$ (f, g are arbitrary computable functions, c is a constant) such that (x,k) is a yes-instance of P if and only if $(x', g(k))$ is a yes-instance of Q.

The Clique problem asks whether, given a graph G and an integer k, G contains a complete subgraph on k vertices. Clique (with parameter k) is a W[1]-complete problem (Downey & Fellows 1999). Below we shall use the special case of the problem where the vertex set of the given graph is partitioned into k independent sets. As observed by Fellows et al. (2007), Clique remains W[1]-complete under this restriction. This follows by the following reduction. Given $G = (V,E)$ and k, take disjoint copies $V_1, \ldots, V_k$ of V; let v_i denote the the copy of v in V_i. We construct the graph $G' = (V', E')$ with $V' = \bigcup_{i=1}^{k} V_i$ and $E' = \{\, u_i v_j : uv \in E, 1 \leq i < j \leq k \,\}$. Now it is easy to verify that G has a clique on k vertices if and only if G' has a clique on k vertices.

Theorem 15. EGCC-Consistency *parameterized by the treewidth of the value graph is* W[1]*-hard.*

Proof. We give a reduction from Clique. Consider an instance consisting of a graph $G = (V,E)$ and an integer k. As discussed above, we may assume that V is partitioned into independent sets $V_1, \ldots, V_k$. Let $V_i = \{v_i^1, \ldots, v_i^{n_i}\}$ and let $N = \max_{i=1}^{k} n_i + 1$.

We will construct an EGC constraint C such that C is consistent if and only if G has a clique on vertices $v_1^{j[1]}, \ldots, v_k^{j[k]}$ with $j[i] \in \{1, \ldots, n_i\}$. The general idea is that C consists of k parts $P_1, \ldots, P_k$ where the i-th part encodes the selection of $v_i^{j[i]}$ from V_i. Any two parts P_i and $P_{i'}$ are connected via a value $d_{\{i,i'\}}$. Assume w.l.o.g. $i < i'$. If P_i selects vertex $v_i^{j[i]}$, it instantiates $j[i]$ many variables with value $d_{\{i,i'\}}$; if $P_{i'}$ selects vertex $v_{i'}^{j[i']}$, it instantiates $N \cdot j[i']$ many variables with value $d_{\{i,i'\}}$. Now $K(d_{\{i,i'\}})$ is defined to contain exactly the integers $j[i] + N \cdot j[i']$ such that $v_i^{j[i]}$ and $v_{i'}^{j[i']}$ are adjacent in G. Since $j[i], j[i'] < N$, each integer in $K(d_{\{i,i'\}})$ corresponds uniquely to a certain pair $(j[i], j[i'])$.

More specifically, the constraint C is defined as follows. For every $1 \leq i \leq k$ we introduce a value d_i. For every $1 \leq i \leq k$ and $1 \leq j \leq n_i$ we introduce a variable x_i^j and a value d_i^j. For every $i, i' \in \{1, \ldots, k\}$, $i \neq i'$, and $1 \leq j \leq n_i$ we introduce a set $X_{i,i'}^j$ of variables such that

$$|X_{i,i'}^j| = \begin{cases} j & \text{if } i < i'; \\ N \cdot j & \text{otherwise.} \end{cases}$$

Finally, for every $1 \leq i < i' \leq k$ we introduce a value $d_{\{i,i'\}}$. Domains and cardinality sets are defined as follows:

$$\begin{aligned} D(x_i^j) &= \{d_i, d_i^j\} \\ D(x) &= \{d_i^j, d_{\{i,i'\}}\} \quad \text{for } x \in X_{i,i'}^j \\ K(d_i) &= \{1\} \\ K(d_i^j) &= \{0, 1 + \textstyle\sum_{i' \in \{1,\ldots,k\}\setminus\{i\}} |X_{i,i'}^j|\} \\ K(d_{\{i,i'\}}) &= \{\, |X_{i,i'}^j| + |X_{i',i}^{j'}| : v_i^j v_{i'}^{j'} \in E, \\ &\qquad 1 \leq j \leq n_i,\ 1 \leq j' \leq n_{i'} \,\}. \end{aligned}$$

This completes the construction of C; see Figure 3 for an illustration.

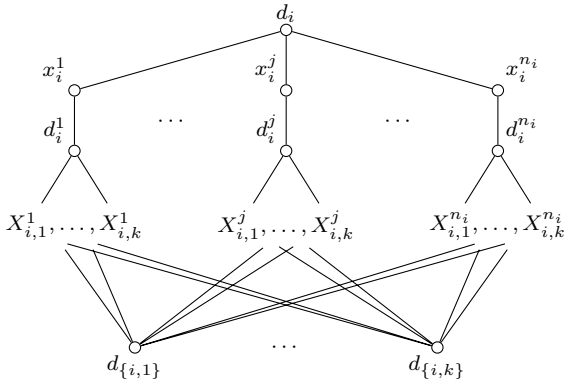

Figure 3: The i-th part of the value graph of the constraint constructed in the proof of Theorem 15.

Claim 1: C is consistent if and only if G has a clique of size k.

Assume that $S = \{v_1^{j[1]}, \ldots, v_k^{j[k]}\}$ induces a clique in G. We define an assignment α for C as follows. We put

$$\alpha(x_i^j) = \begin{cases} d_i & \text{if } j[i] = j; \\ d_i^j & \text{otherwise} \end{cases}$$

and for $x \in X_{i,i'}^j$ we put

$$\alpha(x) = \begin{cases} d_{\{i,i'\}} & \text{if } j[i] = j; \\ d_i^j & \text{otherwise.} \end{cases}$$

It can be easily checked that α satisfies C.

Conversely, let α be an assignment that satisfies C. For every $i \in \{1, \ldots, k\}$ there is exactly one $j \in \{1, \ldots, n_i\}$ with $\alpha(x_i^j) = d_i$, since $K(d_i) = \{1\}$. Let $S = \{v_1^{j[1]}, \ldots, v_k^{j[k]}\}$. We show that S induces a clique in G. To this aim, choose $1 \leq i < i' \leq k$ arbitrarily. It follows from the definition of C that the variables mapped to $d_{\{i,i'\}}$ are exactly the variables in the sets $X_{i,i'}^{j[i]}$ and $X_{i',i}^{j[i']}$. We have $|X_{i,i'}^{j[i]}| = j[i]$, $|X_{i',i}^{j[i']}| = N \cdot j[i']$, and $j[i] + N \cdot j[i'] \in K(d_{\{i,i'\}})$. Since $j[i], j[i'] < N$, $j[i] + N \cdot j[i'] \in K(d_{\{i,i'\}})$ implies that $v_i^{j[i]}$ and $v_{i'}^{j[i']}$ are adjacent in G. Since i and i' were chosen arbitrarily, it follows that all vertices in S are adjacent to each other, i.e., S induces a clique in G. Hence Claim 1 is shown.

Claim 2: The treewidth of the value graph of C is at most $\binom{k}{2} + 1$.

Let $W = \{ d_{\{i,i'\}} : 1 \leq i < i' \leq k \}$. If we delete all vertices in W from the value graph of C, then we are left with a collection of k disjoint trees. Hence without the vertices in W, the value graph admits a tree decomposition of width 1. Now adding W to all the bags of this tree decomposition yields a tree decomposition of the full value graph of G. The width of this tree decomposition is $|W|+1 = \binom{k}{2}+1$. Hence Claim 2 is shown.

Since the construction of C from G can certainly be carried out in polynomial time (polynomial in G and k), we have a reduction from the W[1]-complete CLIQUE problem to EGCC-CONSISTENCY with parameter treewidth. Hence the latter problem is W[1]-hard. □

Corollary 16. GENFACTOR *parameterized by treewidth is* W[1]*-hard.*

7 Conclusion

We have studied extended global cardinality constraints under structural restrictions. We have shown that (complete) domain filtering and consistency checking for these constraints can be carried out in linear time if the parameters treewidth and cardinality-width are both bounded by arbitrary constants. Furthermore we have shown that consistency checking is NP-hard if the cardinality-width is bounded alone and W[1]-hard if the treewidth is bounded alone. Furthermore we have pointed out the connection between extended global cardinality constraints and global factors of graphs. By means of this connection we could identify the largest class of cardinality sets that admits polynomial-time consistency checking. An empirical evaluation of our theoretical results is left for future research. We hope that our work stimulates further research on global constraints under structural restrictions as well as the development of fixed-parameter algorithms for other global constraints.

References

Beldiceanu, N., Carlsson, M. & Rampon, J.-X. (2005), 'Global constraint catalog', Technical Report T2005:08. Swedish Institute of Computer Science, Stockholm, Sweden.

Bessiére, C., Hebrard, E., Hnich, B. & Walsh, T. (2004), The tractability of global constraints, *in* 'Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP'04)', Vol. 3258 of *LNCS*, Springer-Verlag, pp. 716–720.

Bodlaender, H. L. (1993), 'A tourist guide through treewidth', *Acta Cybernetica* **11**(1-2), 1–22.

Bodlaender, H. L. (1996), 'A linear time algorithm for finding tree-decompositions of small treewidth', *SIAM Journal on Computing* **25**(6), 1305–1317.

Bodlaender, H. L. (2005), Discovering treewidth, *in* 'Proceedings of the 31st Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'05)', Vol. 3381 of *LNCS*, Springer-Verlag, pp. 1–16.

Bourdais, S., Galinier, P. & Pesant, G. (2003), HIBISCUS: A constraint programming application to staff scheduling in health care, *in* 'Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP'03)', Vol. 2833 of *LNCS*, Springer-Verlag, pp. 153–167.

Cornuéjols, G. (1988), 'General factors of graphs', *Journal of Combinatorial Theory, Series B* **45**(2), 185–198.

Courcelle, B. (1987), 'Recognizability and second-order definability for sets of finite graphs', Technical Report I-8634. Université de Bordeaux, Bordeaux, France.

Downey, R. G. & Fellows, M. R. (1999), *Parameterized Complexity*, Springer-Verlag.

Fellows, M. R., Hermelin, D. & Rosamond, F. (2007), 'On the fixed-parameter intractability and tractability of multiple-interval graph problems', Manuscript.

Flum, J. & Grohe, M. (2006), *Parameterized Complexity Theory*, Springer-Verlag.

van Hoeve, W.-J. & Katriel, I. (2006), Global constraints, *in* F. Rossi, P. van Beek & T. Walsh, eds, 'Handbook of Constraint Programming', Elsevier, chapter 6, pp. 169–208.

Kloks, T. (1994), *Treewidth: Computations and approximations*, Springer-Verlag.

Koster, A. M. C. A., Bodlaender, H. L. & van Hoesel, S. P. M. (2001), 'Treewidth: Computational experiments', *Electronic Notes in Discrete Mathematics* **8**.

Lovász, L. (1970), The factorization of graphs, *in* 'Combinatorial Structures and their Applications', Gordon and Breach, pp. 243–246.

Lovász, L. (1972), 'The factorization of graphs II', *Acta Mathematica Academiae Scientiarum Hungaricae* **23**, 223–246.

Niedermeier, R. (2006), *Invitation to Fixed-Parameter Algorithms*, Oxford University Press.

Quimper, C.-G., López-Ortiz, A., van Beek, P. & Golynski, A. (2004), Improved algorithms for the global cardinality constraint, *in* 'Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP'04)', Vol. 3258 of *LNCS*, Springer-Verlag, pp. 542–556.

Régin, J.-C. (1996), Generalized arc consistency for global cardinality constraint, *in* 'Proceedings of the 13th National Conference of the American Association for Artificial Intelligence (AAAI'96)', AAAI Press, pp. 209–215.

Régin, J.-C. & Gomes, C. P. (2004), The cardinality matrix constraint, *in* 'Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP'04)', Vol. 3258 of *LNCS*, Springer-Verlag, pp. 572–587.

Rossi, F., van Beek, P. & Walsh, T., eds (2006), *Handbook of Constraint Programming*, Elsevier.

Sellmann, M. (2003), Cost-based filtering for shorter path constraints, *in* 'Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP'03)', Vol. 2833 of *LNCS*, Springer-Verlag, pp. 694–708.

Parameterized Complexity of the Clique Partition Problem

Egbert Mujuni[*1] **Frances Rosamond**[2]

[1] Department of Mathematics
University of Dar-es-Salaam
Box 35062, Dar es Salaam
Tanzania
Email: emujuni@maths.udsm.ac.tz

[2] Parameterized Complexity Research Unit
University of Newcastle
Callaghan, Australia
Email: frances.rosamond@newcastle.edu.au

Abstract

The problem of deciding whether the edge-set of a given graph can be partitioned into at most k cliques is well known to be NP-complete. In this paper we investigate this problem from the point of view of parameterized complexity. We show that this problem is fixed parameter tractable if we choose the number of cliques as parameter. In particular, we show that in polynomial time, a kernel bounded by k^2 can be obtained, where k is the number of cliques. We also give an $\mathcal{O}(2^{((k+3)\log k)/2}n)$ algorithm for this problem in K_4-free graphs.

1 Introduction

The problem of finding a minimum set that covers or partitions the edge-set of a given graph arises in many applications (e.g., see (15)). The problem is defined as follows: Let G be a graph. A set $\mathcal{S} = \{G_1, G_2, \ldots, G_k\},\ k \geq 1$ of subgraphs of G is called a covering of G if $E(G) = \cup_{i=1}^{k} E(G_i)$. If each element of $\mathcal{S}$ is a clique, then $\mathcal{S}$ is called a *clique cover* of G. A *clique partition* is a clique cover $\mathcal{S}$ in which each edge belongs to exactly one member of $\mathcal{S}$; that is, for two distinct $C, C' \in \mathcal{S}$ it follows that $E(C) \cap E(C') = \emptyset$. The clique partition problem asks whether a given graph G has a clique partition of size at most k.

The clique partition problem is known to be NP-complete in general graphs (14). The problem remains NP-complete even for K_4-free graphs (16).

In this paper we investigate the parameterized complexity of this problem using the framework developed by Downey and Fellows (5). Here we give a quick review of parameterized complexity theory. For a detailed discussion we refer the reader to (5) or (13). In parameterized complexity theory, we consider the input of an instance of a parameterized problem as consisting of two parts; that is, a pair (I, k), where I is the main input and k (usually an integer) is a parameter. We say a problem of size n and parameter k is *fixed parameter tractable* if the problem can be solved in time $\mathcal{O}(f(k)n^c)$, where f denotes a computable function and c denotes a constant which is independent of the parameter k. Therefore, a parameterized algorithm may provide an efficient solution to a problem whose parameter is reasonably small.

Clustering problems have wide applicability (See for example, (2; 7; 8; 11)). The problem (EDGE) CLIQUE COVER in general graphs is an important NP-complete problem that has received considerable attention. Clique Cover in general graphs is hard to approximate in polynomial time and nothing better than a polynomial approximation factor is known (1). However Gramm *et al.* (See (9) and also (10)) show that CLIQUE COVER is fixed-parameter tractable with respect to the parameter k, the number of cliques, and has a kernel of size 2^k.

CLIQUE COVER
Instance : A graph $G = (V, E)$
Parameter : An integer k
Question : Is there a set of at most k cliques in G such that each edge in E has both its endpoints in at least one of the selected cliques?

Gramm *et al.* (9; 10) describe an exact algorithm based on search tree techniques. Combining their kernelization rules with a sophisticated search tree algorithm, they were able to obtain an FPT algorithm for CLIQUE COVER that can solve problem instances on graphs of several hundred vertices efficiently.

The key difference between CLIQUE COVER and our problem, CLIQUE PARTITION, is whether the cliques share edges or not. Although we have not implemented our kernelization rules, they are polynomial-time data reduction techniques similar to those of Gramm *et al.* that significantly shrink the input, and then for the reduced instances one can use search tree, exhaustive search or other algorithms to efficiently find optimal solutions in reasonable time.

More formally, in this paper we study following parameterized problem:

CLIQUE PARTITION
Instance : A graph $G = (V, E)$
Parameter : An integer k
Question : Is there a set of at most k cliques in G such that each edge in E has both its endpoints in exactly one of the selected cliques?

We develop a set of reduction rules that in polynomial time replace a given CLIQUE PARTITION instance (G, k) consisting of a graph G and a nonnegative integer k by a "simpler" instance (G', k') such

[*] The reseach has been supported by International Science Programme (ISP) of Sweden, under the project titled "The Eastern African Universities Mathematics Programme (EAUMP)".

that (G,k) has a solution iff (G',k') has a solution. An instance to which none of the reduction rules applies is called "reduced" with respect to these rules. A parameterized problem such as CLIQUE PARTITION (the parameter is k) is said to have a problem kernel if, after the application of the reduction rules, the reduced instance has size $f(k)$ for a function f depending only on k. It is a well-known result from parameterized complexity theory that the existence of a problem kernel implies fixed-parameter tractability for a parameterized problem (5; 13) and (10).

Main Results: In this paper we show that CLIQUE PARTITION has a kernel bounded by k^2, hence it is fixed parameter tractable. We also give an $\mathcal{O}(2^{((k+3)\log k)/2}n)$ algorithm for K_4-free graphs.

Notations: All graphs considered in this paper are undirected finite graphs without loops and multiple edges. Let $G=(V,E)$ be a graph. A *clique* is complete subgraph of G that is not necessarily maximal. The set of neighbours of a vertex v is denoted by $N(v)$, and we set $N[v]=N(v)\cup\{v\}$. For $T\subset V$, we set $N(T):=\bigcup_{v\in T}N(v)$. If $V'\subseteq V$, we denote by $G[V']$ the subgraph of G induced by V'. We refer the reader to (3) for graph theoretic terminology not defined in this paper.

2 Kernelization

In this section we present a set of reduction rules which leads to a problem kernel consisting of at most k^2 vertices. We show that if these rules are not applicable to an instance (G,k) of CLIQUE PARTITION and G has more than k^2 vertices then we conclude that G does not have a clique partition of size at most k.

Definition 1 *A kernelization for a parameterized problem $\mathcal{L}$ is a transformation which maps an instance (I,k) onto (I',k') (which is called a problem kernel) such that:*

1. *$k'\le k$ and $|I'|\le g(k)$ for some computable function g*
2. *The transformation from (I,k) onto (I',k') is computable in polynomial time.*
3. *(I,k) is a yes-instance of $\mathcal{L}$ if and only if (I',k') a yes-instance of $\mathcal{L}$.*

The function $g(k)$ is called the *size of a kernel* for $\mathcal{L}$. The following result is well known.

Lemma 2 *(6) A parameterized problem is fixed-parameter tractable if and only if it has a kernelization.*

We first present simple reduction rules that can be easily applied to simplify an instance of the CLIQUE PARTITION problem; trivially we may assume that $k>1$.

- **Rule 1:** Given an instance (G,k) of CLIQUE PARTITION and a vertex $v\in V(G)$ of degree 0, then the answer to (G,k) is yes if and only if $(G-v,k)$ is yes.
- **Rule 2:** Given an instance (G,k) of CLIQUE PARTITION and a vertex $v\in V(G)$ of degree 1, then the answer to (G,k) is yes if and only if $(G-v,k-1)$ is yes.
- **Rule 3:** Given an instance (G,k) of CLIQUE PARTITION and an edge $e=uv\in E(G)$ such that $N(u)\cap N(v)=\emptyset$, then the answer to (G,k) is yes if and only if $(G-\{e\},k-1)$ is yes.

Clearly, the following is true.

Lemma 3 *Rules 1-3 are correct and they can be executed in $\mathcal{O}(n^2)$ time, where n is the number of vertices of the input graph.*

Definition 4
- *A clique partition $\mathcal{S}$ of a complete graph G is said to be trivial $\mathcal{S}$ if it consists of a single clique.*
- *Let G be a complete graph. Denote by $\rho(G)$ the cardinality of a minimum non-trivial clique partition of G.*

Lemma 5 *Suppose G is a complete graph on n vertices. Then $\rho(G)=n$.*

Lemma 5 is just a corollary of the following result of de Bruijn and Erdos (1948), which was stated in terms of set theory.

Theorem 6 *(4) Suppose $A_1,\ldots,A_m$ are subsets of the set $A=\{a_1,\ldots,a_n\}$, and that $A_i\neq A$, $1\le i\le m$. If each pair $\{a_r,a_s\}$ occurs in one and only one A_i, then $m\ge n$, and equality holds if and only if either (1) $A_1=\{a_1,\ldots,a_{n-1}\}$, $A_2=\{a_1,a_n\},\ldots,A_n=\{a_{n-1},a_n\}$ or (2) n is of the form $n=k(k+1)+1$ and all the A_i's have precisely $k+1$ elements, and each a_j occurs in exactly $k+1$ of the A_i's, $1\le i\le m$, $1\le j\le n$.*

In the terminology of graph theory, Theorem 6 says that if $\mathcal{S}$ is a non-trivial clique partition of the edges of K_n, then $|\mathcal{S}|\ge n$ and equality holds if and only if $\mathcal{S}$ consists of one clique on $n-1$ vertices and $n-1$ copies of K_2 incident with a single vertex of K_n, or n is of the form $n=k^2+k+1$ and $\mathcal{S}$ consists of n copies of K_{k+1}, where each vertex of K_n belongs to exactly $k+1$ cliques of $\mathcal{S}$.

Lemma 7 *Let G be a graph. Let $\mathcal{S}$ be a clique partition of G of size k. If $G'\subseteq G$ is a complete subgraph of G on more than k vertices, then there is an element $C\in\mathcal{S}$ such that $G'\subseteq C$.*

Proof: If the edges of G' are covered by more than one clique of $\mathcal{S}$, then by Lemma 5, they must be covered by more than k cliques. This implies in turn that $|\mathcal{S}|>k$, which is a contradition to the hypothesis. ◇

With this lemma at hand, we can state the following reduction rule.

- **Rule 4:** Let (G,k) be an instance of CLIQUE PARTITION. Suppose that v is a vertex such that $|N[v]|>k$ and the graph G^* induced by $N[v]$ is a clique, then the answer to (G,k) is yes if and only if (G',k') is yes, where $G'=G-v-E(G^*)$ and $k'=k-1$.

Lemma 8 *Rule 4 is correct.*

Proof: First suppose that (G,k) is true. Let $\mathcal{S}=\{C_1,\ldots,C_m\}$, $m\le k$, be a clique partition of G, and let G^* be as defined in Rule 4. It follows that G^* is a maximal clique. Since G^* is a clique on more than k vertices, Lemma 7 implies that there is an index i such that $G^*\subseteq C_i$. However, since G^* is a maximal clique, it follows that $G^*=C_i$. Therefore, $\mathcal{S}-\{C_i\}$ is a clique partition of G' with $m-1\le k-1$ elements; i.e., (G',k') is true.

Now suppose that $(G',k-1)$ is true. Then (G,k) is true because G is the edge-disjoint union of G' and G^*, and G^* is a complete graph. ◇

We say that an instance (G,k) of CLIQUE PARTITION is reduced (with respect to Rules 1-4) if none of the reduction rules can be applied.

Theorem 9 *Suppose an instance (G, k) of CLIQUE PARTITION is reduced and that it does have a solution of size at most k. Then G has at most k^2 vertices.*

Proof: Suppose a reduced instance (G, k) of CLIQUE PARTITION has the answer yes. Let $\mathcal{S}$ be a clique partition of G of size at most k. We claim that each element of $\mathcal{S}$ has at most k vertices.

Suppose to the contrary that there is a clique $C \in \mathcal{S}$ with more than k vertices. Since Rule 4 is not applicable, each vertex of C has a neighbour in G which does not belong to C. This implies that each vertex of C belongs to another unique clique in $\mathcal{S}$. However, by the definition of the CLIQUE PARTITION problem, if C_x and C_y are cliques in $\mathcal{S} - \{C\}$ containing $x, y \in C$, respectively, $x \neq y$, then $C_x \neq C_y$. Thus $|\mathcal{S}| > k$, which is a contradition. We may now conclude that each element of $\mathcal{S}$ has at most k vertices.

Note that $\mathcal{S}$ covers the vertices of G, since Rule 1 is not applicable. Therefore

$$|V(G)| \leq \sum_{C \in \mathcal{S}} |V(C)| \leq k \times \max\{|C| : C \in \mathcal{S}\} \leq k^2$$

This completes the proof the theorem. ◇

Remark 10 As can be seen from the proof of Theorem 9, Rule 2 and Rule 3 have no impact there (i.e., Theorem 9 remains true if we restrict the reductions to applying Rules 1 and 4 only). However, we included Rule 2 and Rule 3 because they may be used to reduce the size of the input graph, and hence speedup the computations. For example, Gramm *et al.*(9) experimented with disabling one of their more complicated and expensive rules. They found that for larger cover sizes over 80, the rule nearly doubles the range of instances that can be solved smoothly and is clearly worthwhile.

As a consequence of Theorem 9, we have

Corollary 11 *CLIQUE PARTITION is fixed parameter tractable.*

3 Algorithm

In the previous section it was shown that CLIQUE PARTITION has a kernel of size k^2. We now describe an algorithm that decides CLIQUE PARTITION. The algorithm is based on bounded search tree. We proceed as follows. Let instance (G, k) be a reduced instance of CLIQUE PARTITION. We choose an edge uv such that $|N(u) \cap N(v)|$ is minimum, and then enumerate a set $\mathcal{S}$ of all cliques in the graph induced by $N(u) \cap N(v)$. Branch according to the elements of $K \in \mathcal{S}$ by adding the clique K' induced by $\{u, v\} \cup V(K)$ to the clique partition, and we set $G := G - E(K')$. The recursion stops as soon as a solution is found or k cliques are generated without finding a solution. This algorithm is presented Figure 1.

We analyze the algorithm in Figure 1 as follows. Let n and Δ be the number of vertices and maximum degree of G, respectively. $|N(u) \cap N(v)| < \Delta$. So $|\mathcal{S}| \in \mathcal{O}(2^{\Delta})$. Thus, each non-leaf node in the searching tree has at most $\mathcal{O}(2^{\Delta})$ children. Since the depth of tree is bounded by k and we can test each leaf in linear time, the algorithm computes the solution in $\mathcal{O}(2^{\Delta k} n)$. Note that, since G is reduced, $\Delta \leq k^2$.

```
C_Partition(Graph G, Set C, Integer k)
begin
  Reduce(G,k).
  if E(G) = ∅ Return TRUE.
  else if k = 0 Return FALSE.
  else
    choose uv ∈ E(G) such that
       |N(u) ∩ N(v)| is minimum.
    find a set S of all clique in
      G[N(u) ∩ N(v)].
    for each K ∈ S
        K' := V(K') ∪ {u, v}.
        C' := C ∪ {G[K']}.
        k' := k − 1.
        H := G − E(K').
        if C_Partition (H, C', k')
           Return TRUE.
    end for
  Return FALSE.
end.
```

Figure 1: Algorithm for CLIQUE PARTITION in general graphs.

4 K_4-Free Graphs

As mentioned in the introduction, the classical decision version of the CLIQUE PARTITION problem remains NP-hard even for K_4-free graphs (16). We now present a fixed-parameter algorithm for CLIQUE PARTITION in this class of graphs. First note that any non-trivial clique in this class of graphs must be K_2 or K_3.

Observation 12 *Let (G, k) be an instance of CLIQUE PARTITION, where G is K_4-free. If G contains an edge uv such that $|N(u) \cap N(v)| > \frac{k+1}{2}$, then G does not have a clique partition of G of size at most k.*

Theorem 13 *Let G be a K_4-free graph. Then the CLIQUE PARTITION problem can be solved in $\mathcal{O}(2^{((k+3)\log k)/2} n)$ time, where n is the number of vertices of G.*

Proof: We construct a bounded search tree T of height k and each node of T has at most $k+1$ children. Each node is associated with a set $\mathcal{C}$ of edge-disjoint cliques, a subgraph $H = G - \bigcup_{C \in \mathcal{C}} E(C)$ and $k' = k - |\mathcal{C}|$, where $\mathcal{C}$ is a partial clique cover constructed at each step of the algorithm. For the root, we have $H := G$, $\mathcal{C} := \emptyset$ and $k' := k$.

We recursively proceed as follows: At a node i, we choose an edge $uv \in E(H)$. If $|N_H(u) \cap N_H(v)| > \frac{k'+1}{2}$, we stop searching in this branch as we know H does not have a clique partition of size at most k', by Observation 12. Otherwise, we create a child for uv and one child for each vertex $w \in N_H(u) \cap N_H(v)$. Thus, this node has at most $\frac{k'+1}{2} + 1 = \frac{k'+3}{2}$ children.

Repeat this expansion for each child node, using the depth-searching strategy. Note that since we add one clique to the partial solution $\mathcal{C}$ at each expansion, the size of $\mathcal{C}$ at level l is also l. The recursion stops as soon as a solution is found or k cliques are generated without finding a solution. T has at most $k^{(k+3)/2} = 2^{(k+3)\log k/2}$ nodes. At each node we need $\mathcal{O}(n)$ time to compute the set $N(u) \cap N(v)$. Hence, the total

running time is $\mathcal{O}(2^{((k+3)\log k)/2}n)$.

◇

Proof of the above theorem yields a fixed parameter algorithm for the CLIQUE PARTITION problem in the class of K_4-free graphs. The algorithm is given in Figure 2.

```
C_Partition(K4-free G, Set C, Integer k)
begin
  if E(G) − E(C) = ∅ Return TRUE.
  else if k = 0 Return FALSE.
  else
    choose an edge uv ∈ E(G)
    if |N(u) ∩ N(v)| > (k+1)/2
       Return FALSE.
    else
       C' := C ∪ {{u, v}}.
       k' := k − 1.
       H := G − {uv}.
       if Clique_Partition (H, C', k')
          Return TRUE.
       else
         for each w ∈ N(u) ∩ N(v)
            C' := C ∪ {{u, v, w}}.
            k' := k − 1.
            H := G − {uv, uw, vw}.
            if C_Partition (H, C', k')
               Return TRUE.
         end for
  Return FALSE.
end.
```

Figure 2: Algorithm for CLIQUE PARTITION in K_4-free graphs.

5 Concluding Remarks

We have obtained the first fixed-parameter tractability result for the clique partition problem, when the number of cliques is the parameter. It would be interesting to improve our algorithm for clique partition. The parameterized complexity hierarchy:

$$P \subseteq Lin(\mathrm{k}) \subseteq Poly(\mathrm{k}) \subseteq FPT \subseteq W[1] \ldots$$

leads to the natural question of whether CLIQUE PARTITION is in Lin(k), or to show that a kernel of linear size in k is not possible. It is interesting that k-CLIQUE COVER is probably not in Poly(k) even though it is in FPT.

Acknowledgments

We are gratefully acknowledge the assistance of Mike Fellows, who originally suggested the problem and provided valuable discussions. We thank Hebert Fleischner for stimulating discussions and providing various references. This paper was done while the first author was enjoying the hospitality of the Vienna Technical University, Austria, which is gratefully acknowledged.

References

[1] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. Springer, 1999.

[2] M. Benson, L. Carlsson, G. Guillot, M. Jernis, M. A. Langston, M. Rudemo, and B. Andersson. A network-based analysis of allergen-challenged CD4+ T cells from patients with allergic rhinitis. *Genes and Immunity* 7 (2006) 514–521.

[3] G. Chartrand and L. Lesniak. *Graphs & Digraphs*. Chapman&Hall, third edition, 1991.

[4] N.G. de Bruijn and P. Erdos. On a combinatorial problem. *Indag. Math. 10*, pages 421–423, 1948.

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

[6] R. Downey, M. Fellows, and U. Stege. Parameterized complexity: A framework for systematically confronting computational intractability. *AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 49*, pages 49–99, 1999.

[7] J.D. Eblen, I.C. Gerling, A.M. Saxton, J. Wu, J.R. Snoddy, and M.A. Langston. Graph Algorithms for Integrated Biological Analysis, with Applications to Type 1 Diabetes Data. *Clustering Challenges in Biological Networks*, W. A. Chaovalitwongse, ed., World Scientific (2007).

[8] M.R. Fellows, M.A. Langston, F.A. Rosamond and P. Shaw. Efficient parameterized preprocessing for cluster editing. *Proc. FCT 2007*, Springer Verlag, *Lecture Notes in Computer Science* 4598 (2007) 312–321.

[9] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Data reduction, exact, and heuristic algorithms for Clique Cover. *Proc. 8th ACM-SIAM ALENEX*, ACM-SIAM (2006) 86–94. Long version to appear in *The ACM Journal of Experimental Algorithmics*.

[10] J. Guo and R. Niedermeier. Guest column: Invitation to data reduction and kernelization. *ACM SIGACT NEWS* 38 (March 2007) 31–45.

[11] M.A. Langston, A.D. Perkins, D.J. Beare, R.W. Gauldie, P.J. Kershaw, J.B. Reid, K. Winpenny, and A.J. Kenny. Combinatorial Algorithms and High Performance Implementations for Elucidating Complex Ecosystem Relationships from North Sea Historical Data. *Proceedings of the International Council for the Exploration of the Sea Annual Science Conference* (2006).

[12] W. Moon and L. Moser. On cliques in graphs. *Israel. J. Math. 3*, pages 23–28, 1965.

[13] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006.

[14] J. Orlin. Contentment in graph theory : Covering graphs with cliques. *Indagationes Math. 39*, pages 406–424, 1977.

[15] F.S. Roberts. Applications of edge coverings by cliques. *Discrete Appl. Math. 10*, pages 93–109, 1985.

[16] Ma Shaohan, W.D. Wallis, and Wu Ju Lin. The complexity of the clique partition number problem. *Nineteenth Southeastern Conference on Combinatorics, Graph Theory, and Computing (Baton Rouge, LA, 1988). Congr. Numer. 67*, pages 59–66, 1988.

The Parameterized Complexity of Regular Subgraph Problems and Generalizations

Luke Mathieson[1] **Stefan Szeider**[1]

[1]Department of Computer Science
University of Durham
South Road
Durham
DH1 3LE, UK
Email: {luke.mathieson,stefan.szeider}@durham.ac.uk

Abstract

We study variants and generalizations of the problem of finding an r-regular subgraph (where $r \geq 3$) in a given graph by deleting at most k vertices. Moser and Thilikos (2006) have shown that the problem is fixed-parameter tractable (FPT) if parameterized by (k, r). They asked whether the problem remains fixed-parameter tractable if parameterized by k alone. We answer this question negatively: we show that if parameterized by k alone the problem is $W[1]$-hard and therefore very unlikely fixed-parameter tractable. We also give $W[1]$-hardness results for variants of the problem where the parameter is the number of vertex and edge deletions allowed, and for a new generalized form of the problem where the obtained subgraph is not necessarily regular but its vertices have certain prescribed degrees. Following this we demonstrate fixed-parameter tractability for the considered problems if the parameter includes the regularity r or an upper bound on the prescribed degrees in the generalized form of the problem. These FPT results are obtained via kernelization, so also provide a practical approach to the problems presented.

Keywords: Parameterized Complexity, Regular Subgraphs

1 Introduction

The problem of deciding whether a graph contains a non-trivial (i.e., degree at least three) regular subgraph has a long history in the field of complexity theory. Chvátal et al. (1979) give an NP-completeness result for the Cubic Subgraph problem (i.e., the problem of deciding whether a given graph has a 3-regular subgraph). Plesník (1984) shows that the Cubic Subgraph problem remains NP-complete even when restricted to a planar bipartite graph with maximum degree 4, and that the r-Regular Subgraph problem with $r \geq 3$ is NP-complete even for bipartite graphs of degree at most $r + 1$. Cheah and Corneil (1990) extend this and show that the same result holds for general graphs. Stewart (1994, 1996, 1997) gives a series of results for further constraints.

From a parameterized complexity perspective (see Section 4 for a basic introduction) there are a few natural parameterizations, by either the size of the subgraph, by the number of vertices or edges to remove to obtain a regular subgraph, or by the regularity desired. Moser and Thilikos (2006) show that the problem of finding an r-regular induced subgraph on k vertices, parameterized by k is $W[1]$-hard. They also show that the Vertex Deletion to Regular Subgraph problem (which they call k-Almost r-Regular Graph) where the goal is to delete at most k vertices leaving an r-regular graph, is fixed-parameter tractable when parameterized by (k, r), with a problem kernel with $O(kr(r + k)^2)$ vertices. Stewart (2007) points out how the fixed-parameter tractability of Vertex Deletion to Regular Subgraph parameterized by (k, r) can be established by means of general logical methods. They also state that the complexity of Vertex Deletion to Regular Subgraph parameterized by k alone is an open problem.

In this paper we answer Moser and Thilikos's question, showing that Vertex Deletion to Regular Subgraph is $W[1]$-hard. We also explore several other variations of the problem, resulting in further hardness and tractability results.

The problems that we cover in this paper come in a few basic forms, centred around two general themes, whether the problem is parameterized by the number of deletion operations (deletion operations are explained in Section 2.2), k, or the number of deletion operations *and* the regularity of the graph, (k, r). This results in the following basic definition:

Deletion to Regular Subgraph
Instance: A graph $G = (V, E)$, two nonnegative integers k and r.
Question: Is there an r-regular subgraph of G obtainable by at most k deletions?

It is interesting to alter what deletion operations are available. If we restrict the operations to vertex deletion only, then we have Vertex Deletion to Regular Subgraph.

We can also further impose that we require *exactly* k operations be performed, giving Exact Deletion to Regular Subgraph.

It is also of interest to generalize both the desired degree and the cost of a deletion. To this end instead of aiming to have each remaining vertex be of degree r we introduce a degree function δ. The contribution of each edge to this total, and the cost of deleting an edge or vertex is described by a weight function ρ. This results in the following generalization:

Weighted Deletion to Chosen Degree Subgraph
Instance: A graph $G = (V, E)$, nonnegative integers k and r, a weight function $\rho : V \cup E \to \mathbb{N}^+$ and a degree function $\delta : V \to \{0, \ldots, r\}$.
Question: Is there a subgraph H of G obtainable by deletions of total cost at most k where for each

vertex v in $V(H)$, $\sum_{e \in E(v)} \rho(e) = \delta(v)$?

Of course we may also demand here that the cost be exact as well.

In this paper we show that in all examined cases parameterization by k alone gives $W[1]$-hardness, but parameterization by (k, r) gives fixed-parameter tractability. In fact if $r = 0$ the problem is equivalent to Vertex Cover, and is thus fixed-parameter tractable. We also give several hardness results for other problems that prove useful in completion of the result. Hardness is shown by reduction from Multi-Coloured Clique, a very useful problem, introduced by Fellows et al. (2007). Fixed parameter tractability is shown via kernelization. This is a particularly useful technique as it provides a polynomial time preprocessing algorithm (in the form of polynomial time reduction rules). This leaves a problem kernel which may then be solved by any chosen means, whether that be an exact method, approximation algorithm, or heuristic such as a genetic algorithm or simulated annealing. For a fuller treatment of kernelization in the context of parameterized complexity and preprocessing see the survey of Guo and Niedermeier (2007).

2 Preliminaries

2.1 Graph Theory and Notation

Throughout this paper we will refer only to simple, undirected graphs. Given a graph G the vertex (edge) set of G will be denoted $V(G)$ ($E(G)$) except specific labels are given. The edge between two vertices u and v will be denoted uv (or equivalently vu). The degree of a vertex u will be denoted $d(u)$.

As this paper focuses on graph modification problems, we also define the following operation for a graph G and a set S of vertices: $G - S = G[V(G) \setminus S]$, where $G[X]$ is the subgraph of G induced by vertex set X.

2.2 Graph Modification

There are two basic operations to modify a graph to obtain a subgraph, vertex deletion and edge deletion. These operations alter a graph $G = (V, E)$ into a new graph $G' = (V', E')$. Deleting an edge uv simply removes that edge from the graph (i.e., $E' = E \setminus \{uv\}$). Deleting a vertex u removes that vertex, and any incident edges (i.e., $V' = V \setminus \{u\}$, $E' = E \setminus \{\, uv \mid v \in V \,\}$).

In this paper we also use weighted versions of these operations, which are defined in the natural fashion. Given a weighted edge or vertex, the cost of deletion is simply that weight. Note particularly that when a vertex is deleted the cost is simply the weight of the vertex alone, not the weight of the vertex plus the weights of the incident edges, even though they are also removed (this is completely equivalent to the normal definition for unweighted graphs, where the cost of deleting a vertex is one operation, regardless of any incident edges).

2.3 Some Parameterized Complexity Theory

Here we will briefly introduce some relevant, key concepts of parameterized complexity. For a more in-depth introduction and study see the books of Downey & Fellows (1997), Flum & Grohe (2006) and Niedermeier (2006). For the sake of clarity any problem is understood to be a decision problem unless explicitly stated otherwise (and the parameterized complexity classes that are referenced are defined for decision problems).

Traditionally problems have been analyzed in one dimension, that of the size n of the input. The difficulty of solution of a problem with respect to this measure forms the fundamental basis of traditional complexity theory, and in particular the classes P and NP. Parameterized complexity adds a second measure, that of a parameter k, which is given as a special part of the input. Then, analogously to the definitions of P and NP, a series of complexity classes are defined with respect to their apparent difficulty of solution with respect to this two-dimensional measure. If a problem has an algorithm that runs in time $O(f(k)p(n))$, where p is a polynomial and f is any computable function of k, then the problem is *fixed-parameter tractable*, or in the class FPT. Naturally there are problems that are suspected not to be in FPT. These problems are members of various parameterized complexity classes, most commonly $W[t]$ for some fixed $t \geq 1$. Hardness (or completeness) in regards to such a class gives an analogous intuition to a problem being NP-hard in the classical structure. That is, it is not likely to be in FPT (i.e., not likely to have an algorithm that runs in time $O(f(k)p(n))$ as above).

Supporting this theory are many techniques for proof either of membership of FPT or of $W[t]$-hardness. Here we give a brief introduction to those techniques salient to this paper.

FPT Reductions

An FPT reduction is the parameterized complexity equivalent of a P-time many-one reduction in classical complexity theory. It is the primary method of demonstrating that two problems are of equivalent complexity, and that a particular problem is $W[t]$-hard. Given two parameterized problems Π_1 and Π_2, an FPT reduction $\Pi_1 \leq_{FPT} \Pi_2$ is a mapping from Π_1 to Π_2 that maps an instance (I, k) of Π_1 to an instance (I', k') of Π_2 such that

1. $k' = h(k)$ for some computable function h,
2. (I, k) is a Yes-instance of Π_1 if and only if (I', k') is a Yes-instance of Π_2 and
3. the mapping can be computed in time $O(f(k)p(|I|))$, where f is some computable function of the parameter k alone and p is a polynomial.

Then if Π_2 is in FPT, Π_1 is also in FPT and if Π_1 is $W[t]$-hard, Π_2 is also $W[t]$-hard. If two such mappings exist, one from Π_1 to Π_2 and another from Π_2 to Π_1, then the two problems are equivalent (with respect to FPT reductions) .

The classes $W[t]$, $t = 1, 2, \ldots$, are defined as equivalence classes of certain parameterized problems under FPT reductions. The classes form the chain FPT $\subseteq W[1] \subseteq W[2] \subseteq \ldots$, where all inclusions are believed to be strict.

Reduction Rules and Kernelization

One of the key techniques of parameterized complexity is that of reduction to problem kernel (kernelization). A problem is *kernelizable* if and only if given an instance (I, k) of the problem, where I is the (classical) input and k is the parameter, it is possible to produce in polynomial time an instance (I', k') where $|I'| \leq g(k')$ and $k' = h(k)$ for computable functions g and h, and (I, k) is a Yes-instance if and only if (I', k') is a Yes-instance. It can be shown that if a

problem is kernelizable in this sense, then it is fixed-parameter tractable (and vice versa). Kernelization is normally accomplished by the application of *reduction rules* to the instance. Estivill-Castro et al. (2005) give a recent example of the application of kernelization, along with more explanation of the theory.

2.4 A Useful Construction: The Fixing Gadget

Throughout the paper it will be useful to have a gadget that allows us to regularize any given graph. The following construction produces an almost r-regular graph, where all vertices have degree r except two with degree $r-1$. The first part of the construction consists of a vertex c, a set $L = \{l_1, \ldots, l_r\}$ of vertices, r edges cl_i, r further vertices $M = \{m_1, \ldots, m_r\}$, and edges such that each vertex $m_i \in M$ has an edge to each vertex $l_j \in L$ except when $i = j$. Then c has degree r, as does each vertex in L. Each vertex in M has degree $r-1$. Let C be the graph constructed so far, we then make a copy C', and add an edge between each vertex in $M \subseteq V(C)$ to its corresponding vertex in $M' \subseteq V(C')$, *except* between for m_r and m'_r. Thus each vertex now has degree r, except m_r and m'_r, which have degree $r-1$, and will be used as attachment points. We will refer to an instance of this construction as a *fixing gadget*. See Figure 1 for an example.

Note that it is also possible to use the following as an alternative in some cases: Take the complete graph K_{r+1} on $r+1$ vertices (so all vertices have degree r), then compute a matching (of size at most $(r+1)/2$). Each edge of the matching can then be broken as needed to provide two edges to join the clique to the rest of the graph. This second construction cannot be used in the hardness proofs however, as it introduces (non-trivial) cliques into the graph.

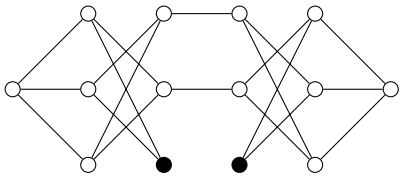

Figure 1: Fixing gadget for $r = 3$.

3 Hardness Results

The reduction for our hardness results will be from the Strongly Regular Multi-Coloured Clique problem, a variant of the Multi-Coloured Clique problem which was shown to be $W[1]$-hard by Fellows et al. (2007). The problem is defined as follows:

Multi-Coloured Clique
Instance: A graph $G = (V, E)$, vertex-coloured with k colours.
Parameter: k.
Question: Does G contain a properly coloured k-clique?

This problem may be alternately defined with the original graph being properly vertex-coloured, without changing its complexity. The Strongly Regular Multi-Coloured Clique problem is defined similarly, but with each vertex in the input graph having degree d to each colour class (so each vertex has degree kd), where d is an arbitrary integer.

Recall that the Clique problem asks if a given graph has a k-clique. Clique is $W[1]$-complete when parameterized by k. We then define the following special case of Clique:

Regular Clique
Instance: A regular graph $G = (V, E)$, an integer k.
Parameter: k.
Question: Does G contain a k-clique?

Before we proceed to the main result, we need first to prove some preliminary lemmas.

Lemma 3.1. Regular Clique *is* $W[1]$*-complete.*

Proof. Membership in $W[1]$ follows immediately as the problem is a special case of Clique. To prove hardness we reduce from Clique. Let (G, k) be an instance of Clique. We construct an instance (G', k) of Regular Clique by first taking G and modifying it. Let Δ be the maximum degree of G, then choose r to be Δ if Δ is even, or $\Delta+1$ otherwise (i.e., $r = \Delta + (\Delta \bmod 2)$). We will now demonstrate how to make the graph r-regular. We can now use the fixing gadget construction presented in Section 2.4 to increase the degree of each vertex as necessary by attaching as many fixing gadgets as necessary by the two attachment vertices. This attachment is made between a vertex v and an instance of the fixing gadget by adding the edges between each attachment vertex and v (or perhaps only one of these edges, as below). If the degree of the vertex is initially even, then this is an integral number of fixing gadgets. In the case where the degree of the vertex is initially odd, the vertex will reach degree $r-1$ by this method, and we will have to take another degree $r-1$ vertex and attach one fixing gadget attachment vertex to the first, and the other attachment vertex to the second. Note that there is an even number of vertices of odd degree in G (and G' initially, an immediate corollary of the basic theorem $\sum_{v \in V} d(v) = 2|E|$), and thus there is an even number of vertices requiring an odd increase of degree (i.e., where $r - d(v)$ is odd), as we have chosen r to be even. Thus there is always some pairing of such vertices as necessary. Let G' denote the constructed graph.

Now if there were a k-clique in G, there will certainly be a clique in G' on k vertices, since G is an induced subgraph of G'. Further note that the fixing gadgets added to create G' contain no cliques, and can introduce no non-trivial cliques (as the two attachment vertices in a fixing gadget are not adjacent), thus if there is a clique on k' vertices in G', it must be contained within the vertices that correspond to the vertices of G, thus G has a k-clique. Clearly the construction of the new instance can be done in polynomial time (and thus is a polynomial-time reduction, and subsequently an FPT reduction). □

Lemma 3.2. Strongly Regular Multi-Coloured Clique *is* $W[1]$*-complete.*

Proof. Again $W[1]$ membership follows as the problem is a special case of Clique.

It is useful to sketch the reduction from Clique to Multi-Coloured Cliqueas given by Fellows et al. (2007). Given an instance (G, k) of Clique, construct an instance of Multi-Coloured Clique (G', k') by taking k vertex disjoint copies $G_1, \ldots, G_k$ of G, assigning each G_i a different colour. Then for every pair of vertices u, v in G, if uv is an edge, add the edges $u_i v_j$, for all i, j, where a_i is the vertex in G_i corresponding to vertex a in G. Let $k' = k$. Then if there were a k-clique in the original instance, there

will be a properly coloured clique in the new instance, and vice versa.

We may use the same construction to reduce Regular Clique to Strongly Regular Multi-Coloured Clique. The result follows immediately. □

Theorem 3.3. Vertex Deletion to Regular Subgraph *and* Deletion to Regular Subgraph *are* $W[1]$*-hard for parameter* k*.*

Proof. Consider an instance (G, k), with $G = (V, E)$, of Strongly Regular Multi-Coloured Clique. Note that G is kd-regular and each vertex has exactly d neighbours in each colour class. We denote the set of vertices of colour i by V_i $(1 \leq i \leq k)$. Then $V = \bigcup_{i=1}^{k} V_i$ forms a partition of V. Observe also that each colour class is of the same size, denote this size as s (i.e., $|V_i| = s$ for all $1 \leq i \leq k$).

We construct an instance (G', k'), with $G' = (V', E')$, of Deletion to Regular Subgraph by first defining k sets V_i' $(1 \leq i \leq k)$ such that for each vertex $v \in V_i$ we add a vertex v' to V_i'. We add all possible edges between pairs of vertices in the same set V_i'. We will call each of these subgraphs a *colour class gadget* or *class gadget* for short.

For each edge uv in G where $u \in V_i$ and $v \in V_j$ with $i \neq j$, we add to G' two vertices $u'_{v'}$ and $v'_{u'}$, with the edges $u'u'_{v'}$, $u'_{v'}v'_{u'}$ and $v'_{u'}v'$. For each pair V_i' and V_j' (where $i \neq j$) of class gadgets, denote the set of these new vertices and edges as P_{ij}. We denote by P_{ij}^i the set of all vertices $u'_{v'} \in P_{ij}$ where $u' \in V_i'$. Furthermore, for each pair of vertices u_v and $u'_{v'}$ in the same P_{ij}^i we add the edge $u_v u'_{v'}$ to P_{ij}^i if u and u' belong to the same class gadget and $u \neq u'$. We call each such P_{ij} a *connection gadget*, and each P_{ij}^i a *side* of the connection gadget. There are $\binom{k}{2}$ connection gadgets in total. Figure 2 gives a sketch of the structure of a connection gadget.

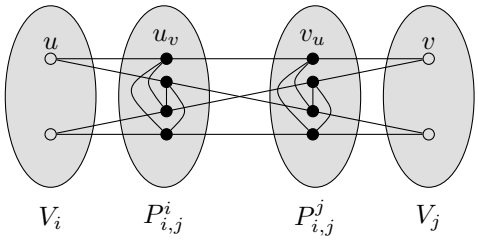

Figure 2: A sketch of illustrating the arrangement of the connection gadgets.

At this point we have k gadgets corresponding to the k colour classes in the original graph, each with s vertices of degree $(s-1)+d(k-1)$, and $\binom{k}{2}$ gadgets corresponding to the "inter-colour-class" edges, each with $2sd$ vertices of degree $2+(s-1)d$ (sd vertices in each half). Now we choose r for the instance such that $r \geq \max((s-1)+d(k-1), 2+(s-1)d)$, and $r \equiv s+1$ modulo 2 (i.e., r is of opposite parity to s). In particular we may choose the smallest r such that this is true.

Now we add for each class gadget V_i' a gadget V_i'' that contains $r+1-((s-1)+d(k-1))$ vertices with s edges per vertex, such that each vertex in V_i'' is adjacent to every vertex in the class gadget V_i'. We refer to V_i'' as a *degree gadget*. We then add a further set of fixing gadgets as before to complete the degree of each vertex in the degree gadget to $r+1$. Note that by choosing r to have opposite parity to s, we guarantee that this is possible (if s is odd, r will be even and each vertex will require $r+1-s$ additional edges, which is even, and thus achievable; if s is even, r will be odd, then $r+1-s$ is again even, and we can complete the construction). Thus each vertex in each class gadget and degree gadget has degree one too many, but the fixing gadgets attached to each degree gadget have the correct degree.

We similarly adjust the connection gadgets by adding two degree gadgets, each with $r+1-2+(s-1)d$ vertices, one for each side of the connection gadget. Every vertex in the degree gadget is connected to every vertex in its associated side of the connection gadget. Again we complete the degree of vertices in the degree gadgets to $r+1$ by adding fixing gadgets, and as before, by the choice of r we can guarantee that this can be done (if s is even, r is odd and $r+1-sd$ is even, if s is odd, r is even and $r+1-sd$ is even). Thus each vertex in the connection gadgets has degree $r+1$, as does each vertex in the degree gadgets. Each vertex in each fixing gadget has degree r.

Now we set $k' = k + 2\binom{k}{2}$.

Claim 3.1. The following statements are equivalent:

1. (G, k) is a Yes-instance of Strongly Regular Multi-Coloured Clique.
2. (G', k') is a Yes-instance of Vertex Deletion to Regular Subgraph.
3. (G', k') is a Yes-instance of Deletion to Regular Subgraph.

$(1 \Rightarrow 2)$ Assume that (G, k) is a Yes-instance of Strongly Regular Multi-Coloured Clique. Then there exist k vertices $v_1, \ldots, v_k$, one from each colour class, that form a properly coloured clique. Assume without loss of generality that $v_i \in V_i$. Then we can delete from G' the corresponding vertices v_i' from V_i', and the pairs of vertices $(v_i')_{v_j'}$ and $(v_j')_{v_i'}$ from P_{ij} that correspond to the edges in the clique. Then each remaining vertex in each class gadget has had precisely one incident edge removed from it, as have the vertices in each degree gadget associated with the class gadget. So the components corresponding to the colour classes and their immediate extension are now r-regular. Similarly each vertex in every connection gadget and their associated connetion gadgets has had exactly one incident edge removed, either by the vertex removed from the connection gadget, or from the parent vertex in the class gadget (but never both). Now each vertex in these gadgets has degree precisely r. We have chosen one vertex from each V_i', and two vertices from each P_{ij}, giving a total of $k' = k + 2\binom{k}{2}$ vertices, thus (G', k') is also a Yes-instance of Vertex Deletion to Regular Subgraph.

$(2 \Rightarrow 3)$ Assume that (G', k') is a Yes-instance of Vertex Deletion to Regular Subgraph. Then clearly it is also a Yes-instance of Deletion to Regular Subgraph.

$(3 \Rightarrow 1)$ Assume that (G', k') is a Yes-instance of Deletion to Regular Subgraph. Then there are $k + 2\binom{k}{2}$ deletions that can be made to make G' r-regular. Obviously we cannot delete any vertices from the fixing gadgets in the graph. Further we cannot delete any vertices from the degree gadgets, as this would reduce the degree of their attached fixing gadgets. Thus the deleted vertices must come from class and connection gadgets. Again there must be precisely one vertex from each such component, if there is less than one in such a component, the degree of at least some of the vertices in that component will

remain $r+1$, if there are more than one, the degree of some vertices in the component will drop below r. Also note that for each vertex u_v deleted from one side of a connection gadget, the vertex deleted in the other side must be the vertex v_u. If it were not, then at least one vertex in each side would have degree $r-1$. Also, the vertex deleted from each side of each connection gadget must be attached to the vertex deleted from the adjacent class gadget, otherwise the vertices attached to vertices deleted from the class gadget will have degree at most $r-1$. Thus we can see that if (G', k') is a YES-instance, the set of vertices to be deleted is very precise and restricted. In fact, if we are to use only the allotted budget of $k + 2\binom{k}{2}$, we must choose precisely one vertex from each class gadget, and two vertices from each connection gadget, where the vertices from the connection gadget component are connected to the vertices deleted from the two class gadgets it is associated with. Similarly, assume that some edge deletion is used, but then each edge deletion can only reduce the degree of two vertices, leaving us with too many edges to delete, or vertices of degree less than r. So clearly the only operation that can be used in this case is vertex deletion. Thus we may more precisely claim that if (G', k') is a YES-instance for DELETION TO REGULAR SUBGRAPH, it must be via vertex deletion alone. Thus it is clear that if (G', k') is a YES-instance for DELETION TO REGULAR SUBGRAPH, then (G, k) must be a YES-instance of STRONGLY REGULAR MULTI-COLOURED CLIQUE. The solution for (G, k) is the set of vertices $\{v_1, \ldots, v_k\}$, one from each colour class, corresponding to the k vertices chosen from the class gadgets. The edges of the clique correspond to the $2\binom{k}{2}$ vertices chosen from the connection gadgets.

We can construct G' from G in polynomial time, as we are adding only $(4r+3)(2r+2s-s-sd-dk+1)$ vertices, where $r, s, d \leq n$, thus it is also an FPT reduction, and we have the desired result. □

We also note that the above proof suffices if we also include the operation of edge addition, giving the following:

Corollary 3.4. EDIT TO REGULAR SUBGRAPH *parameterized by the number of* edit *operations k is $W[1]$-hard.*

We may also consider the similar problem of finding a regular subgraph of an unknown regularity (i.e., when r is not given):

Corollary 3.5. DELETION TO SOME REGULAR SUBGRAPH *parameterized by the number of edit operations k is $W[1]$-hard.*

Proof. Given an instance (G, k) of DELETION TO REGULAR SUBGRAPH, we construct an instance (G', k) of DELETION TO SOME REGULAR SUBGRAPH as follows:

We simply add one r-regular connected component with more than k vertices. This can be done by taking, for example, k fixing gadgets and connecting them in a ring. We clearly cannot alter this component within the budget, thus the only possible solution is the same as that for (G, k). Thus if (G', k) is a YES-instance of DELETION TO SOME REGULAR SUBGRAPH, (G, k) must be a YES instance of DELETION TO REGULAR SUBGRAPH. Naturally if (G, k) is a YES-instance of DELETION TO REGULAR SUBGRAPH, the same solution will result in a regular graph in (G', k), so (G', k) is a YES-instance of DELETION TO SOME REGULAR SUBGRAPH. □

Of course the same proof again suffices for the edit version of the problem.

We also obtain the following result.

Corollary 3.6. WEIGHTED DELETION TO CHOSEN DEGREE SUBGRAPH *parameterized by the number k of edit operations is $W[1]$-hard.*

Proof. Clearly DELETION TO REGULAR SUBGRAPH, is a restriction of WEIGHTED DELETION TO CHOSEN DEGREE SUBGRAPH, with $\rho(e) = 1$, $\rho(v) = 1$ and $\delta(v) = r$ for each edge e and vertex v. □

Once again we may make a similar claim for the edit version of the problem, WEIGHTED EDIT TO CHOSEN DEGREE SUBGRAPH.

4 Fixed Parameter Tractability

Moser and Thilikos (2006) give several tractability results for regular induced subgraph problems, and in doing so contribute several significant and natural ideas that are of use in the more general setting of this paper. Several of the reduction rules that we develop have direct analogs in their paper, and in particular we use their notion of a "clean region". We however exploit the structure available more fully, using annotation. In this case annotation proves a powerful tool for generalizing, and thus simplifying the problem. We are thus able to get more general results that include their results as special cases. In particular we avoid the complex clean region replacement that they undertake as the annotation allows a simpler representative replacement. Abu-Khzam & Fernau (2006) give a further examination of annotation with respect to kernelization.

4.1 Definitions

First we will define various terms that allow a more elegant treatment of the result.

Given a graph G, a function $\delta : V \to \{0, \ldots, r\}$ and a function $\rho : V \cup E \to \mathbb{N}^+$. We say a vertex $v \in V$ is *clean* if $\sum_{e \in E(v)} \rho(e) = \delta(v)$. Then a *clean region* is a set of *clean* vertices that form a connected subgraph. Note that not all edges incident on the vertices of the clean region need have both endpoints in the clean region. We can greedily calculate the collection of maximal clean regions in a graph in polynomial time. Note that these maximal clean regions are disjoint. In general when we refer to a clean region, we will mean a maximal clean region, though strictly the results are unaffected.

A clean region is *independent* if there are no edges from the clean region to any vertex outside the clean region.

Given a clean region C we call the set of vertices not in C adjacent to a vertex in C as the *boundary* of C.

It is also notationally convenient to define the degree of a vertex v restricted to a set X of vertices as $d_X(v)$. So $d_X(v)$ is the number of neighbours of v that are in the set X, and we extend this notation to sets of vertices, for example the degree of a boundary B restricted to its clean region C is $d_C(B) = \sum_{b \in B} d_C(b)$.

It is also useful to define a weighted degree function $d^\rho : V \to \mathbb{N}^+$ such that for each vertex v, $d^\rho(v) = \sum_{e \in E(v)} \rho(e)$. As above we denote the weighted degree of a vertex v restricted to a set of vertices X as $d^\rho_X(v)$ and extend it as before to sets.

4.2 Weighted Deletion to Chosen Degree Subgraph

In this section we consider the Weighted Deletion to Chosen Degree Subgraph problem as defined earlier, but parameterized by both the number of deletions k and the maximum desired degree r.

4.2.1 Reduction Rules

Let $(G,(k,r))$, with $G=(V,E)$, be an instance of Weighted Deletion to Chosen Degree Subgraph. The following reduction rules produce from $(G,(k,r))$ an equivalent instance $(G',(k',r'))$ of Deletion to Chosen Degree Subgraph. For all reduction rules $r'=r$.

Reduction Rule 1: If there exists a vertex $v\in V$ with $d^{\rho}(v)<\delta(v)$, then $G'=G-\{v\}$, $k'=k-\rho(v)$.

Reduction Rule 2: If there exists a vertex $v\in V$ with $d(v)>k+r$, then $G'=G-\{v\}$, $k'=k-\rho(v)$.

Reduction Rule 3: If there exists an independent clean region $C\subseteq V$, then $G'=G-C$, $k'=k$.

Reduction Rule 4: If there exists a clean region C with a vertex b in its boundary where $d^{\rho}_{C}(b)>\delta(b)$, then $G'=G-C$, $k'=k-\sum_{v\in C}\rho(v)$.

This also gives an algorithmically useful corollary.

Corollary 4.1. *If there exists a clean region C with a vertex b in its boundary such that $d^{\rho}_{C}(b)>\delta(b)$ and $\sum_{v\in C}\rho(v)>k$, then $(G,(k,r))$ is a* No*-instance.*

Reduction Rule 5: If there exists a clean region C with boundary B such that $\sum_{v\in C}\rho(v)>k$ and for each boundary vertex b we have $d^{\rho}_{C}(b)\leq\delta(b)$, then for each $b\in B$, set $\rho(b)=k+1$ and set $\delta(b)=\delta(b)-d^{\rho}_{C}(b)$, and $G'=G-C$, $k'=k$.

Reduction Rule 6: If there exists a clean region C, with boundary B such that Reduction Rules 4 and 5 do not apply, (i.e., there are no boundary vertices with excessive weighted degree into the clean region, and the weight of the clean region is not larger than k), then modify the instance as follows:

1. Add a new vertex v such that $\rho(v)=\sum_{c\in C}\rho(c)$, and $\delta(v)=d^{\rho}_{B}(C)$.
2. For each boundary vertex $b\in B$, add an edge bv such that $\rho(bv)=d^{\rho}_{C}(b)$.
3. Delete C.
4. Set $k'=k$.

Note that the vertex v added in Reduction Rule 6 is a special vertex in that we allow it to have $\rho(v)>r$. This does not affect the existence of a solution, and if desired, a less elegant, alternate reduction rule can be substituted where the region is replaced by a series of vertices each with ρ at most r. In the kernelization this increases the size of X (only) by a factor of at most k.

Lemma 4.2. *Reduction Rules 1–6 are sound. That is, each reduction rule takes an instance $(G,(k,r))$ of* Weighted Deletion to Chosen Degree Subgraph *and produces an instance $(G',(k',r'))$ of* Weighted Deletion to Chosen Degree Subgraph *such that $(G,(k,r))$ is a* Yes *instance if and only if $(G',(k',r'))$ is a* Yes *instance.*

Proof. Rule 1: Clearly v cannot remain in the final graph unmodified, but as we cannot add edges, there is no way of increasing the degree of v. Thus v must be deleted as part of any solution.

Rule 2: If v were to remain in the final graph, we must either delete more than k edges or neighbouring vertices, each with weight at least 1, which we cannot do. Thus the only possibility is to delete v.

Rule 3: Clearly an independent clean region needs no changing, thus we can safely ignore it, as it will play no role in the solution.

Rule 4: If there were such a b, then at least one of the edges from b into the clean region must be deleted, but then a vertex v of the clean region would now have weighted degree less than $\delta(v)$, and would have to be deleted (as per Reduction Rule 1). This would obviously cascade, resulting in the entire clean region being deleted. Thus the only possible option is to delete the clean region.

Rule 5: As with Reduction Rule 4, deletion of any vertex or edge in the clean region or between the clean region and the boundary would require the clean region to be deleted entirely. As the clean region is of total weight greater than k, it obviously cannot be deleted within a cost k solution. Thus it suffices to increase the weight of each vertex in the boundary, as these cannot be deleted either, and reduce their degree funtion appropriately.

Rule 6: As C is a clean region, deletion of any vertex or edge in the clean region or boundary will result in the entire clean region being deleted, thus it is sufficient to represent the clean region as one appropriately weighted clean vertex. □

4.2.2 Kernel Lemma

Lemma 4.3 (Kernel Lemma)**.** *If $(G=(V,E),(k,r))$ is reduced under Reduction Rules 1–6 and $|V|>k+k(k+r)+kr(k+r)$, then $(G,(k,r))$ is a* No*-instance for* Weighted Deletion to Chosen Degree Subgraph.

Proof. Assume that $(G=(V,E),(k,r))$ is a Yes-instance for k-Deletion r-Regular Subgraph. Further assume that the instance is reduced under Reduction Rules 1–6. Let S be the set of edges and vertices deleted as part of the solution, $|S|\leq k$ (more particularly $\rho(S)\leq k$). As any edge in the solution is adjacent to only two vertices, our worst case occurs when the solution is all vertices, so it suffices to only consider S. Further let H be the set of vertices consisting of the endpoints of any edges in S and the neighbours of any vertices in S, and $X=V\setminus\{H\cup S\}$. Note that H is a cut-set seperating S and X. Figure 3 gives an example of this partitioning for an example graph with $r=3$.

We make the following claims:

Claim 4.1. $|H|\leq k(k+r)$.

No vertex has degree greater than $(k+r)$, otherwise the graph is not reduced under Reduction Rule 2. Thus if S were all vertices, they could have at most $(k+r)$ neighbours each. As H is the entire neighbourhood of S, $|H|\leq|S|(k+r)=k(k+r)$.

Claim 4.2. $|X|\leq kr(k+r)$.

X must consist only of clean regions, otherwise S is not a solution. Each vertex h in H can have at most r neighours in X, otherwise S is not a solution. If h were adjacent to a clean region with total weight greater than k, this region would have been removed under Reduction Rule 5, thus it can only be adjacent to small clean regions. As the graph is reduced, each of these clean regions contains precisely one vertex,

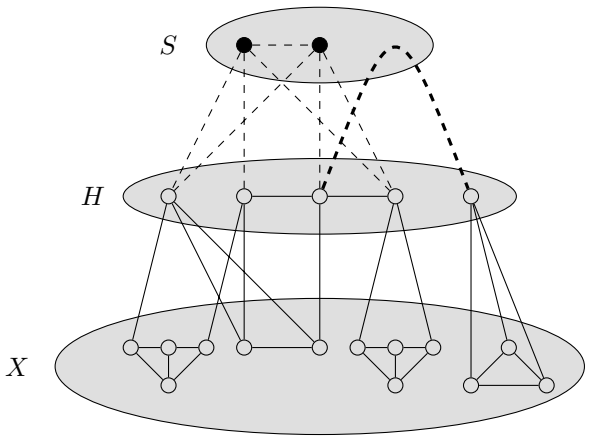

Figure 3: Example of the partitioning described in the Kernel Lemma. $r = 3$.

by Reduction Rule 6. Thus each h can have at most r neighbours. As there are $k(k + r)$ such vertices, $|X| \leq kr(k + r)$.

Claim 4.3. There are at most $k+k(k+r)+kr(k+r)$ vertices in G.

$|V| = |S| + |H| + |X|$. By Claims 4.2 and 4.1, $|H| \leq k(k + r)$, and $|X| \leq kr(k + r)$. There are no other vertices in the graph, otherwise the graph is not reduced under Reduction Rules 3 and 4. Thus as $|S| \leq k$, $|V| \leq k + k(k + r) + kr(k + r)$.

Then by Claim 4.3, if $(G = (V, E), (k, r))$ is a YES-instance for WEIGHTED DELETION TO CHOSEN DEGREE SUBGRAPH, then $|V| \leq k+k(k+r)+kr(k+r)$. Thus the Kernel Lemma holds. □

To complete the proof of FPT membership, we need to demonstrate that the Reduction Rules can be executed in polynomial time. Clearly Reduction Rules 1 and 2 can be carried out in linear time, and each can be applied at most k times. Thus Reduction Rules 1 and 2 contribute $O(kn)$ to the running time.

We can calculate the clean regions of the graph greedily in linear time, with the boundary calculated at that time. Thus an independent clean region can be identified in linear time, and deleted in linear time. Similarly any clean region with a vertex b in its boundary such that $d^{\rho}_{C}(b) > \delta(b)$, can be identified quickly. Similarly regions to which Reduction Rules 5 and 6 apply can be identified at this point, and replaced or removed as required. This can be done at most k times, thus Reduction Rules 3, 4, 5 and 6 contribute $O(kn)$ to the running time.

This leads immediately to the following theorem:

Theorem 4.4. WEIGHTED DELETION TO CHOSEN DEGREE SUBGRAPH *is fixed-parameter tractable for parameter (k,r).*

In particular an instance $(G, (k, r))$ of WEIGHTED DELETION TO CHOSEN DEGREE SUBGRAPH with n vertices and m edges can be solved in time $O(kn + f(k, r))$, where $f(k, r)$ is the running time of whatever algorithm or heuristic is applied to the kernel (whose size is bounded, so the running time is guaranteed to be a function of (k, r). A simple approach would be the application of a bounded search tree which branches on which problem vertex or edge to delete, which gives a running time of $O((k^3 + 2k^2r + kr^2)^k)$.

Note that if we have a graph where initially $\rho(v) = 1$, $\rho(e) = 1$ and $\delta(v) = r$ for each vertex v and edge e, then this is precisely the DELETION TO REGULAR SUBGRAPH problem, thus we also gain the following result:

Corollary 4.5. DELETION TO REGULAR SUBGRAPH *is fixed-parameter tractable for parameter (k,r).*

4.2.3 The Exact Case

The previous proof can be modified easily to demonstrate fixed-parameter tractability for the EXACT WEIGHTED DELETION TO CHOSEN DEGREE SUBGRAPH problem. In this case we are interested in deleting elements with a total weight of k. Thus we may be interested in deleting elements where the deletion does not fix the degree of some vertex, it simply adds to the total cost. However, the only areas where we may delete these from that are not already included in the kernel are independent clean regions of 'low' weight ($\leq k$).

Of course we need not retain all such independent clean regions on the chance that they may be needed. Obviously any independent clean region of weight greater than k can still be removed without consequence, it could never be part of any solution of cost k. So we need only concern ourselves with independent clean regions of weight less than or equal to k. Recall that a clean region is defined as a set of vertices, thus in particular, the weight of a clean region is the sum of weights of the vertices, not the edges.

Notice also that given a sufficient quantity of independent clean regions of a given weight (say i), we could never use all of them, and thus need only retain a small number. Thus if we replace Reduction Rule 3, we can adjust our kernel size appropriately:

Reduction Rule 3a: If there exist more than $\lfloor k/i \rfloor$ independent clean regions of weight $i \leq k$, delete all but $\lfloor k/i \rfloor$ of them, $k' = k$.

Then we may modify the Kernel Lemma as follows:

Lemma 4.6 (Exact Kernel Lemma)**.** *If* $(G = (V, E), (k, r))$ *is reduced under Reduction Rules 1–6 (with Rule 3a replacing Rule 3) and* $|V| > k + k(k + r)+kr(k+r)+k^2$*, then* $(G, (k, r))$ *is a* NO*-instance for* EXACT WEIGHTED DELETION TO CHOSEN DEGREE SUBGRAPH*.*

Proof. We begin with the following claims:

Claim 4.4. There are no independent clean regions of size greater than k.

As each vertex has weight at least 1, a clean region of size greater than k must have weight greater than k, thus if one remained, the graph would not be reduced under Reduction Rule 3a.

Claim 4.5. There are at most k^2 vertices in independent clean regions.

By Reduction Rule 3a, there are at most $\lfloor k/i \rfloor$ independent clean regions of weight i. The largest size of any such region is k (by claim 4.4). Thus the total size is $\sum_{i=1}^{k} \lfloor k/i \rfloor \, i \leq \sum_{i=1}^{k} (k/i)i = \sum_{i=1}^{k} k = k^2$.

The proof of the Kernel Lemma now follows as before, simply with the new claims taken into account. □

The new reduction rule can clearly be enacted in polynomial time, we need only greedily keep the first few independent clean regions we find of each weight, which can be accomplished at the start of the algorithm in linear time.

Theorem 4.7. EXACT WEIGHTED DELETION TO CHOSEN DEGREE SUBGRAPH *is fixed-parameter tractable for parameter* (k, r)*.*

As before, we also gain the following result as a special case:

Corollary 4.8. Exact Deletion to Regular Subgraph *is fixed-parameter tractable for parameter* (k, r).

5 Conclusion

We have answered Moser and Thilikos's open question, and shown that Vertex Deletion to Regular Subgraph is $W[1]$-hard when parameterized by the number k of vertex deletions. The problem remains hard when we extend the operations available to include edge deletion and/or edge addition. The generalized version of the problem Weighted Deletion (Edit) to Chosen Degree Subgraph is also $W[1]$-hard when parameterized by the number k of deletions.

If we include r as an additional parameter however, all the problems examined are fixed-parameter tractable, most notable being that Weighted Deletion to Chosen Degree Subgraph is fixed-parameter tractable under such a parameterization with a problem kernel of at most $k+k(k+r)+kr(k+r)$ vertices. Deletion to Regular Subgraph is also fixed-parameter tractable with the same kernel, but we also demonstrate a method that allows avoidance of clean region contraction, which may be useful in practice. Similarly the exact versions of the problems remain tractable with the same parameterization.

As our FPT results derive from kernelization, this paper also provides several useful polynomial-time preprocessing algorithms producing bounded problem kernels which can then be solved by any method of choice, such as heuristics or approximations.

References

Abu-Khzam, F. & Fernau, H. (2006), Kernels: Annotated, Proper and Induced, *in* 'International Workshop on Parameterized and Exact Computation 2006 (IWPEC'06)', Lecture Notes in Computer Science, Springer, pp. 264–275.

Cheah, F. & Corneil, D. G. (1990), The Complexity of Regular Subgraph Recognition, 'Discrete Applied Mathematics', **27**, pp. 59–68.

Chvátal, V., Fleischner, H., Sheehan, J. & Thomassen, C. (1979), Three-regular Subgraphs of Four Regular Graphs, 'Journal of Graph Theory', **3**, pp. 371–386.

Downey, R. & Fellows, M. (1997), *Parameterized Complexity*, Springer.

Estivill-Castro, V., Fellows, M., Langston, M. & Rosamond, F. (2005), FPT is P-Time Extremal Structure I, *in* 'Algorithms and Complexity in Durham 2005 (ACiD'05)', Texts in Algorithmics, College Publications, pp. 1–41.

Flum, J. & Grohe, M. (2006), *Parameterized Complexity Theory*, Springer.

Guo, J. & Niedermeier, R. (2007), Invitation to Data Reduction and Problem Kernelization, 'SIGACT News', **38**(1), pp. 31–45.

Fellows, M., Hermelin, D. & Rosamond, F. (2007), On the Fixed-Parameter Intractability and Tractability of Multiple-Interval Graph Problems, Unpublished Result.

Moser, H. & Thilikos, D. (2006), Parameterized Complexity of Finding Regular Induced Subgraphs, *in* 'Algorithms and Complexity in Durham 2006 (ACiD'06)', Texts in Algorithmics, College Publications, pp. 107–118.

Niedermeier, R. (2006), *Invitation to Fixed-Parameter Algorithms*, Oxford University Press.

Plesník, J. (1984), A Note on the Complexity of Finding Regular Subgraphs, 'Discrete Mathematics', **49**, pp. 161–167.

Stewart, I. A. (1994), Deciding Whether a Planar Graph has a Cubic Subgraph is NP-Complete, 'Discrete Mathematics', **126**(1-3), pp. 349-357.

Stewart, I. A. (1996), Finding Regular Subgraphs in Both Arbitrary and Planar Graph, 'Discrete Applied Mathematics', **68**(3), pp. 223-235.

Stewart, I. A. (1997), On Locating Cubic Subgraphs in Bounded-degree Connected Bipartite Graphs, 'Discrete Mathematics', **163**(1-3), pp. 319-324.

Stewart, I. A. (2007), On the Fixed-Parameter Tractability of Parameterized Model-Checking Problems, 'Information Processing Letters', *to appear*.

Well-Covered Graphs and Greedoids

Vadim E. Levit[1,2] **Eugen Mandrescu**[2]

[1] Department of Computer Science and Mathematics
Ariel University Center of Samaria,
Ariel 40700, ISRAEL,
Email: levitv@ariel.ac.il

[2] Department of Computer Science
Holon Institute of Technology,
Holon 58102, ISRAEL,
Email: eugen_m@hit.ac.il

Abstract

G is a *well-covered graph* provided all its maximal stable sets are of the same size (Plummer, 1970). S is a *local maximum stable set* of G, and we denote by $S \in \Psi(G)$, if S is a maximum stable set of the subgraph induced by $S \cup N(S)$, where $N(S)$ is the neighborhood of S.

In 2002 we have proved that $\Psi(G)$ is a greedoid for every forest G. The bipartite graphs and the triangle-free graphs, whose families of local maximum stable sets form greedoids were characterized by Levit and Mandrescu (2003, 2007a).

In this paper we demonstrate that if a graph G has a perfect matching consisting of only pendant edges, then $\Psi(G)$ forms a greedoid on its vertex set. In particular, we infer that $\Psi(G)$ forms a greedoid for every well-covered graph G of girth at least 6, non-isomorphic to C_7.

Keywords: local maximum stable set, greedoid, very well-covered graph, unique perfect matching.

1 Introduction

Throughout this paper $G = (V, E)$ is a simple (i.e., a finite, undirected, and without multiple edges) graph with vertex set $V = V(G)$ and edge set $E = E(G)$. The vertices $x, y \in V(G)$ are called *adjacent* if they are the endpoints of some edge in G, and we write $xy \in E(G)$. We assume also that $xx \notin E(G)$, for every $x \in V(G)$, i.e., G is loopless. If $X \subset V$, then $G[X]$ is the subgraph of G induced 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 $F \subset E(G)$, i.e., $G - F = (V, E - F)$, and we write shortly $G - e$, whenever $F = \{e\}$.

The *neighborhood* of a vertex $v \in V$ is the set $N(v) = \{w : w \in V \text{ and } vw \in E\}$, and $N[v] = \{v\} \cup N(v)$. If $|N(v)| = |\{u\}| = 1$, then v is a *pendant vertex* and vu a *pendant edge* of G. By $\text{pend}(G)$ we mean the set of all pendant vertices of G.

K_n, C_n denote, respectively, the *complete graph* on $n \geq 1$ vertices, and the *chordless cycle* on $n \geq 3$ vertices, i.e., $K_1 = (\{v_1\}, \emptyset)$ and

$$K_n = (\{v_i : 1 \leq i \leq n\}, \{v_i v_j : 1 \leq i < j \leq n\}), n \geq 2,$$

while C_n has

$$V(C_n) = \{v_i : 1 \leq i \leq n\},$$

$$E(C_n) = \{v_i v_{i+1} : 1 \leq i \leq n-1\} \cup \{v_1 v_n\}).$$

A vertex $v \in V(G)$ is called *simplicial* if $G[N[v]]$ is a complete subgraph of G.

We denote the *neighborhood* of some $A \subset V$ 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 $N[A]$, respectively, if no ambiguity.

A *tree* is a cycle-free connected graph, while a *forest* is cycle-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 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 induced by $N[A]$, i.e., $A \in \Omega(G[N[A]])$, (Levit and Mandrescu 2002). Let $\Psi(G)$ stand for the set of all local maximum stable sets of G. Notice that $\Omega(G) \subseteq \Psi(G)$ is true for every graph G.

Clearly, every set $S \subseteq \text{pend}(G)$ belongs to $\Psi(G)$. Nevertheless, there exist local maximum stable sets that do not contain pendant vertices. For instance, $\{e, g\} \in \Psi(G)$, where G is the graph from Figure 1.

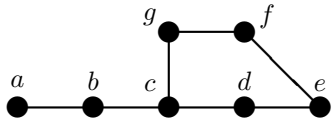

Figure 1: A graph having various local maximum stable sets.

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 $\mu(G)$. A matching is *perfect* if it saturates all the vertices of the graph. Let us recall that G is a *König-Egerváry graph* provided $\alpha(G) + \mu(G) = |V(G)|$. It is known that every bipartite graph is a König-Egerváry graph as well.

A graph G is *well-covered* if every maximal stable set of G is also a maximum stable set, i.e., it belongs to $\Omega(G)$. If, in addition, G has no isolated vertices and $|V(G)| = 2\alpha(G)$, then G is *very well-covered* (Favaron 1982). For instance, the graph depicted in Figure 1 is well-covered, but not very well-covered, while the graph from Figure 2 is very well-covered.

In other words, each stable set of a well-covered graph is contained in a maximum stable set, e.g., the

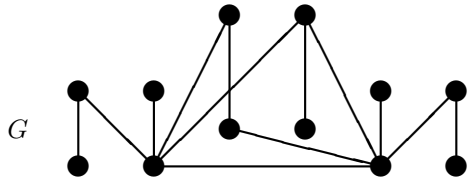

Figure 2: A very well-covered graph whose unique perfect matching has non-pendant edges.

graph H from Figure 3. Since there is no maximum stable set S of G such that $\{b, d\} \subset S$, the graph G in Figure 3 is not well-covered.

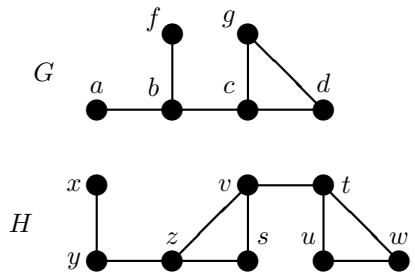

Figure 3: H is well-covered; G is not well-covered.

Well-covered graphs were defined by Plummer (1970). A number of classes of well-covered graphs were completely described; e.g., well-covered bipartite graphs (Ravindra 1977), very well-covered graphs (Favaron 1982), well-covered block graphs and unicyclic graphs (Topp and Volkmann 1990), well-covered graphs of girth ≥ 6 (Finbow, Hartnell and Nowakowski 1993), well-covered cubic graphs (Campbell, Ellingham and Royle 1993), well-covered graphs that contain neither 4- nor 5-cycles (Finbow, Hartnell and Nowakowski 1994), 4-connected claw-free well-covered graphs (Hartnell and Plummer 1996), well-covered simplicial, chordal, and circular arc graphs (Prisner, Topp and Vestergaard 1996), well-covered König-Egerváry graphs (Levit and Mandrescu 1998). A survey on this subject is due to Plummer (1993).

In fact, well-covered graphs are exactly those graphs for which the greedy algorithm constructing maximum stable sets vertex by vertex always yields a maximum stable set, no matter how its greediness makes it to chose vertices of a graph. For general graphs, the problem of finding a maximum stable set, is **NP**-hard.

While, in general, it is co-**NP**-complete to determine if a given graph is well-covered (Chvátal and Slater 1993, Sankaranarayana and Stewart 1992), recognizing weighted well-covered graphs with bounded $\triangle(G)$ can be done in polynomial time (Caro et al. 1998, Zverovich 2004), where $\triangle(G)$ equals the maximum vertex degree of the graph G. Tankus and Tarsi (1996, 1997) showed that claw-free well-covered graphs can be recognized in polynomial time.

It is easy to prove the following.

Proposition 1.1 *Every graph having a perfect matching consisting of pendant edges is very well-covered.*

The converse of Proposition 1.1 is not generally true (e.g., the graph G depicted in Figure 2). Moreover, there are well-covered graphs without perfect matchings, (e.g., K_3). Nevertheless, following Favaron's characterization for very well-covered graphs (i.e., Theorem 1.2), one can assert that "having a perfect matching" is a necessary condition for very well-coveredness.

A matching M in a graph G satisfies *Property P* if for every edge $xy \in M$,

$$N(x) \cap N(y) = \emptyset \text{ and}$$

$$N(x) - \{y\} \text{ is adjacent to all of } N(y) - \{x\}.$$

Theorem 1.2 *For a graph G without isolated vertices the following are equivalent:*
(i) *G is very well-covered;*
(ii) *there is a perfect matching in G that satisfies Property P;*
(iii) *there exists at least one perfect matching in G and every perfect matching in G satisfies Property P.*

By $H \circ K_1$ we mean the graph obtained from H by appending a single pendant edge to each vertex of H. Let us notice that $H \circ K_1$ is very well-covered and $\alpha(H \circ K_1) = |V(H)|$. Moreover, Finbow, Hartnell and Nowakowski (1993) showed (Theorem 1.3) that, under certain conditions, every well-covered graph must be of this form.

Theorem 1.3 *Let G be a connected graph of girth greater than five, which is isomorphic to neither C_7 nor K_1. Then G is well-covered if and only if its pendant edges form a perfect matching, i.e., $G = H \circ K_1$ for some graph H.*

In other words, Theorem 1.3 shows that, apart from K_1 and C_7, connected well-covered graphs of girth ≥ 6 are very well-covered. Consequently, a tree $T \neq K_1$ could be only very well-covered, and this is the case if and only if $T = H \circ K_1$ for some tree H (for additional details, see Ravindra 1977, Favaron 1982, Levit and Mandrescu 1999).

The following theorem concerning maximum stable sets in general graphs, due to Nemhauser and Trotter Jr. (1975), shows that some stable sets can be enlarged to maximum stable sets.

Theorem 1.4 *Every local maximum stable set of a graph is a subset of a maximum stable set.*

The graph W from Figure 1 has the property that every $S \in \Omega(W)$ 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 the equality $\Psi(G) = \Omega(G)$, e.g., $G = C_n$, for $n \geq 4$.

A greedoid (Björner and Ziegler 1992, and Korte et al. 1991) is a set system generalizing the notion of matroid.

Definition 1.5 *A greedoid is a pair $(V, \mathcal{F})$, where $\mathcal{F} \subseteq 2^V$ is a non-empty set system satisfying the following conditions:*
(Accessibility) for every non-empty $X \in \mathcal{F}$ there is an $x \in X$ such that $X - \{x\} \in \mathcal{F}$;
(Exchange) for any $X, Y \in \mathcal{F}, |X| = |Y| + 1$, there is an $x \in X - Y$ such that $Y \cup \{x\} \in \mathcal{F}$.

Recall that a matroid is a set system $(V, \mathcal{F})$ that satisfies both the "exchange property" and the "hereditary property", saying that : if $X \in \mathcal{F}$ and $Y \subseteq X$, then $Y \in \mathcal{F}$. Evidently, any matroid is also a greedoid. It is clear that the family of all stable sets of a graph is a matroid if and only if G is a disjoint union of complete graphs, which means that, necessarily, G must be well-covered of a specific form.

If G is well-covered, $\Psi(G)$ is a matroid if and only if each $S \in \Omega(G)$ consists of only simplicial vertices, because $\Omega(G) \subseteq \Psi(G)$ and every $v \in S$, by hereditary property, satisfies $\{v\} \in \Psi(G)$, i.e., $G[N[v]]$ must be a complete graph.

The notion of matroid was defined by Whitney (1935). Later Edmonds (1971)characterized a matroid as a hereditary set system for which a class of linear optimization problems can be solved by greedy algorithms. Korte and Lovász (1991) introduced the greedoid in an attempt to generalize this characterization to accessibility systems.

It is worth mentioning that if $(V, \mathcal{F})$ is a greedoid and $X \in \mathcal{F}$, $|X| = k \geq 2$, then according to accessibility property, one can build a chain

$$\{x_1\} \subset \{x_1, x_2\} \subset \{x_1, ..., x_3\} \subset ...$$

$$... \subset \{x_1, ..., x_{k-1}\} \subset \{x_1, ..., x_{k-1}, x_k\} = X$$

such that $\{x_1, ..., x_j\} \in \mathcal{F}$, for each $j \in \{1, ..., k-1\}$. Such a chain we call an *accessibility chain* of X.

For example, $\Psi(G_1)$ is a greedoid and

$$\{a\} \subset \{a, b\} \subset \{a, b, c\}$$

is an accessibility chain of $\{a, b, c\} \in \Psi(G_1)$, where G_1 is presented in Figure 4.

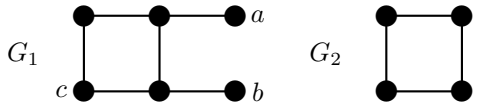

Figure 4: G_1, G_2 are very well-covered graphs, but only for G_1 it is true that every $S \in \Omega(G_1)$ has an accessibility chain.

Levit and Mandrescu (2002) proved the following.

Theorem 1.6 *For every forest T, $\Psi(T)$ is a greedoid on its vertex set.*

The case of bipartite graphs having a unique cycle, whose family of local maximum stable sets forms a greedoid, is studied in Levit and Mandrescu (2001, 2005). The general case of bipartite graphs was treated in Levit and Mandrescu (2003), while for triangle-free graphs we refer the reader to Levit and Mandrescu (2007) for details. Nevertheless, there exist non-bipartite and non-triangle-free graphs whose families of local maximum stable sets form greedoids. The families $\Psi(G_1), \Psi(G_2), \Psi(G_3), \Psi(G_4)$ of the graphs in Figure 5 are greedoids. Let us notice that G_1 is very well-covered and G_3 is well-covered, while G_2, G_4 are not well-covered and also non-triangle-free.

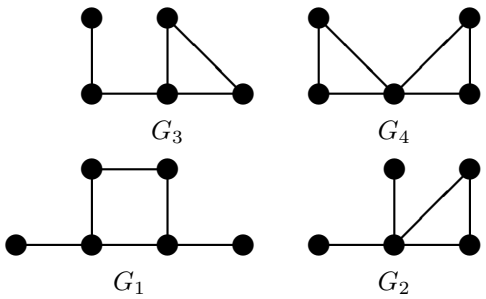

Figure 5: Graphs whose family of local maximum stable sets form greedoids.

In this paper we prove that in a well-covered graph G of girth at least 6, but different from C_7, the family $\Psi(G)$ of local maximum stable sets forms a greedoid on its vertex set.

2 Results

It is easy to see that no maximum stable set of C_7 admits an accessibility chain. The graph G in Figure 6 shows that even if some $S \in \Omega(G)$ admits an accessibility chain, this is not necessarily true for all maximum stable sets. The case of the graph H from Figure 6 is different: each maximum stable set of H has an accessibility chain, and the reason is given in Proposition 2.1.

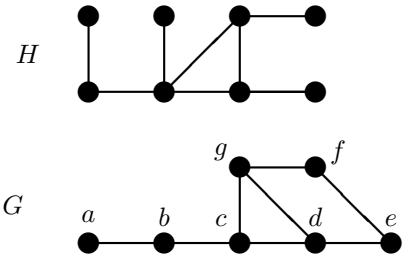

Figure 6: $\{a, c, f\}, \{a, g, e\} \in \Omega(G)$, but only $\{a, c, f\}$ admits an accessibility chain.

Proposition 2.1 *Every maximum stable set of the graph $G = H \circ K_1$ has an accessibility chain.*

Proof. Clearly, $\alpha(G) = n$, where $|V(H)| = n$, and each $S \in \Omega(G)$ satisfies $S \cap \text{pend}(G) \neq \emptyset$.

We prove by induction on n that every $S \in \Omega(G)$ has an accessibility chain.

For $n = 1$, the assertion is clearly true.

For $n = 2$, let $S = \{x_1, x_2\} \in \Omega(G)$. Then at least one of x_1, x_2 is pendant, say x_1. Hence, the chain is $\{x_1\} \subset \{x_1, x_2\} = S$.

Suppose that the assertion is true for $k < n$.

Let $G = (V, E) = H \circ K_1$ be with $|V(H)| = n$, and let $S \in \Omega(G)$.

Since $S \cap \text{pend}(G) \neq \emptyset$, let $a_1 \in S \cap \text{pend}(G)$. If $N(a_1) = \{b_1\}$, then $G - \{a_1, b_1\} = (H - \{b_1\}) \circ K_1$. Hence, we have that

$$S_{n-1} = S - \{a_1\} \in \Omega(G - \{a_1, b_1\}),$$

and by induction hypothesis, there is a chain

$$\{x_1\} \subset \{x_1, x_2\} \subset ... \subset \{x_1, x_2, ..., x_{n-1}\} = S_{n-1}$$

such that $\{x_1, x_2, ..., x_k\} \in \Psi(G - \{a_1, b_1\})$ for each $k \in \{1, ..., n-1\}$. Since $N_G(a_1) = \{b_1\}$, it follows that

$$N_G(\{x_1, x_2, ..., x_k\} \cup \{a_1\}) =$$

$$= N_{G-\{a_1,b_1\}}(\{x_1, x_2, ..., x_k\}) \cup \{b_1\},$$

and therefore $\{x_1, x_2, ..., x_k\} \cup \{a_1\} \in \Psi(G)$, for every $k \in \{1, ..., n-1\}$. Clearly, $\{a_1\} \in \Psi(G)$, and consequently, we obtain the chain:

$$\{a_1\} \subset \{a_1, x_1\} \subset \{a_1, x_1, x_2\} \subset ...$$

$$... \subset \{a_1, x_1, x_2, ..., x_{n-1}\} = \{a_1\} \cup S_{n-1} = S,$$

such that $\{a_1, x_1, x_2, ..., x_k\} \in \Psi(G)$ for every k from $\{1, ..., n-1\}$, i.e., S has an accessibility chain. ■

Let us notice that Proposition 2.1 is not valid for each very well-covered graph; e.g., C_4 is very well-covered, but no $S \in \Omega(C_4)$ has an accessibility chain.

Remark 2.2 *If S consists of only isolated vertices of H, then $S \in \Psi(H \circ K_1)$, because, in this case, $S \subseteq \text{pend}(G)$.*

Remark 2.3 *If S is stable in H and $N_H(S) \neq \emptyset$, then $S \notin \Psi(H \circ K_1)$, because for each $a \in N_H(S)$, the set $\{a\} \cup \{u : u \in \text{pend}(H \circ K_1) \cap N_{H \circ K_1}(S)\}$ is stable in $H \circ K_1$ and larger than S.*

Remark 2.4 *If v is an isolated vertex of the graph H and $S \in \Psi(H \circ K_1)$, such that $S \cap N_{H\circ K_1}[v] = \emptyset$, then $S \cup \{v\} \in \Psi(H \circ K_1)$.*

Lemma 2.5 *If H has no isolated vertices and S is a stable set in $G = H \circ K_1$, then the following assertions are equivalent:*
(i) $S \in \Psi(G)$;
(ii) $S = S_1 \cup S_2$, *where* $\emptyset \neq S_1 \subseteq \text{pend}(G)$ *and* $S_2 \subseteq V(H), N_H(S_2) \subseteq N_G(S_1)$;
(iii) $G[N_G[S]] = H' \circ K_1$, *for some subgraph H' of H, and* $S \in \Omega(H' \circ K_1)$.

Proof. Let us denote:

$$V(H) = \{v_i : 1 \leq i \leq n\},$$

$$V(G) = V(H) \cup \{u_i : 1 \leq i \leq n\},$$

$$E(G) = E(H) \cup \{u_i v_i : 1 \leq i \leq n\}.$$

Notice that

$$\alpha(G) = n, S_0 = \{u_i : 1 \leq i \leq n\} \in \Omega(G)$$

and S_0 =pend(G), since H has no isolated vertices.
(i) $\Longrightarrow$ *(ii)* Assume that $S \in \Psi(G)$.
Let $S_1 = S \cap \text{pend}(G)$ and $S_2 = S \cap V(H)$. Clearly, $S_1 \neq \emptyset$, because S has an accessibility chain.
If $S_2 = \emptyset$, then the assertion is clearly true.
Suppose that $S_2 \neq \emptyset$. If $N_H(S_2) \not\subseteq N_G(S_1)$, then there must be some $v_k \in N_H(S_2)$ such that $u_k \notin S$, i.e., $u_k \notin S_1$. Hence, we get that

$$\{u_k\} \cup (N_G[S] - V(H))$$

is a stable set in $G[N_G[S]]$ larger than S, in contradiction with $S \in \Psi(G)$.
(ii) $\Longrightarrow$ *(iii)* Let $S_3 = \{u_k : v_k \in S_2\}$. Then we infer that

$$G[N_G[S]] = G[S_1 \cup S_3] = H' \circ K_1,$$

for some subgraph H' of H. In addition, we have also that

$$|S| = |S_1| + |S_2| = |S_1| + |S_3| \text{ and } S_1 \cup S_3 \in \Omega(G[S]).$$

Consequently, we deduce that $S \in \Omega(H' \circ K_1)$ as well.
(iii) $\Longrightarrow$ *(i)* As $S \in \Omega(G[N_G[S]])$, it follows, by definition, that $S \in \Psi(G)$. ■
Now we are able to prove the main result of the paper.

Theorem 2.6 *The family $\Psi(H \circ K_1)$ is a greedoid.*

Proof. Let $G = H \circ K_1$ and $S_0 \in \Psi(G)$, i.e., S_0 is a maximum stable set, of size say q, in $H_0 = G[N[S_0]]$.
According to Lemma 2.5, $G[N[S_0]] = H_{S_0} \circ K_1$ for some subgraph H_{S_0} of H, and by Proposition 2.1, we infer that there exists a chain

$$\{x_1\} \subset \{x_1, x_2\} \subset x_1, x_2, x_3\} \subset ...$$

$$... \subset \{x_1, x_2, ..., x_{q-1}\} \subset \{x_1, x_2, ..., x_{q-1}, x_q\} = S_0,$$

such that all $S_k = \{x_1, x_2, ..., x_k\}, 1 \leq k \leq q$, are local maximum stable sets in H_0. Since $N_{H_0}[S_k] = N_G[S_k]$, it results that $S_k \in \Psi(G)$, for any $k \in \{1, ..., q\}$. In other words, $\Psi(G)$ satisfies the accessibility property.
We have to show now that $\Psi(G)$ satisfies also the exchange property.
Let us consider $X, Y \in \Psi(G)$ be such that

$$|Y| = |X| + 1 = m + 1.$$

According to Lemma 2.5*(ii)*, the sets X and Y can be decomposed as follows:

$$X = X_1 \cup X_2 \text{ and } Y = Y_1 \cup Y_2,$$

where X_1, X_2, Y_1, Y_2 satisfy the corresponding conditions, i.e., X_1 and Y_1 are non-empty subsets of pend(G), while X_2, Y_2 are subsets of $V(H)$, such that $N_H(X_2) \subseteq N_G(X_1)$ and $N_H(Y_2) \subseteq N_G(Y_1)$.
Since Y is stable, $X \in \Psi(G)$, and $|X| < |Y|$, it follows that there exists some $y \in Y - X$, such that $y \notin N_G[X]$. In particular, it means that $X \cup \{y\}$ is stable. To check whether $X \cup \{y\} \in \Psi(G)$, we have to analyze the two following cases (see Figure 7).

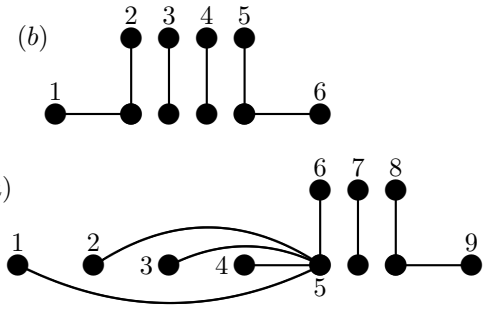

Figure 7: X and Y are local maximum stable sets that illustrate the cases *1* and *2*, respectively. *(a)* $Y = \{1, 2, 3, 4, 6\}, X = \{6, 7, 8, 9\}$, having their upper parts $Y_1 = \{6\} \subset X_1 = \{6, 7, 8\}$; and *(b)* $Y = \{1, 2, 3, 4\}, X = \{4, 5, 6\}$, with their upper parts $Y_1 = \{2, 3, 4\} \not\subseteq X_1 = \{4, 5\}$.

Case 1. $Y_1 \subseteq X_1$.
Firstly, we deduce that $y \in Y_2$. Lemma 2.5*(ii)* implies that $N_H(y) \subseteq N_G(Y_1)$. Since $Y_1 \subseteq X_1$, it follows that $N_G(Y_1) \subseteq N_G(X_1)$. Hence, we get $N_H(y) \subseteq N_G(X_1)$. Therefore, we have that

$$X_1 \subseteq \text{pend}(G),\ X_2 \cup \{y\} \subseteq V(H),$$

and

$$N_H(X_2 \cup \{y\}) = N_H(X_2) \cup N_H(\{y\}) \subseteq N_G(X_1).$$

Consequently, according to Lemma 2.5*(ii)*, we may infer that the stable set $X \cup \{y\}$ is, actually, a maximum local stable set in G.
Case 2. $Y_1 \not\subseteq X_1$.
In this situation, one can choose as y every vertex $z \in Y_1 - X_1$, because clearly, both conditions

$$z \in Y - X \text{ and } X \cup \{z\} \in \Psi(G)$$

are satisfied.
Therefore, $\Psi(G)$ satisfies the exchange property as well.
In conclusion, $\Psi(G)$ is a greedoid on the vertex set of G. ■
Let us notice that $\Psi(C_7)$ is not a greedoid, because every $S \in \Psi(C_7)$ has $|S| \neq 1$.

Corollary 2.7 *Let G be a well-covered graph of girth greater than five, which has no connected components isomorphic to C_7. Then $\Psi(G)$ is a greedoid on the vertex set of G.*

Proof. Firstly, if $G = K_1 = (\{a\}, \emptyset)$, then $\Psi(K_1) = \{\{a\}\}$ and it is clearly a greedoid.
Secondly, if G is a connected well-covered graph of girth ≥ 6, isomorphic to neither C_7 nor K_1, then Theorem 1.3 implies that $G = H \circ K_1$ for some graph H. Further, according to Theorem 2.6, $\Psi(G)$ is a greedoid.

If G is disconnected, and G_i, $i \in \{1, ..., q\}$, are its connected components, then clearly,

$$\Psi(G) = \Psi(G_1) \cup \Psi(G_2) \cup ... \cup \Psi(G_q).$$

and, to complete the proof, one has to take care of every connected component G_i, independently. ■

3 Conclusions

We showed that $\Psi(G)$ is a greedoid on the vertex set of a well-covered graph G, which is well-covered of girth ≥ 6 and non isomorphic to C_7. Since C_5 is well-covered, while $\Psi(C_5)$ is not a greedoid, one can ask to characterize well-covered graphs of girth ≤ 5, whose families of local maximum stable sets form greedoids.

Recently, as proved by Levit and Mandrescu (2007b), each very well-covered graph G of girth ≥ 5 must be of the form $G = H \circ K_1$ for some graph H. Therefore, Corollary 2.7 is true for very well-covered graphs of girth ≥ 5.

References

[1] Björner, A. and Ziegler, G. M. (1992), Introduction to greedoids, in N. White (ed.), Matroid Applications, 284-357, Cambridge University Press.

[2] Campbell, S. R., Ellingham, M. N. and Royle, G. F. (1993), A characterization of well-covered cubic graphs, J. Combin. Math. Combin. Comput. **13** pp. 193-212.

[3] Y. Caro, Y., M. N. Ellingham, M. N. and Ramey, J. E. (1998), Local structure when all maximal independent sets have equal weight, SIAM Journal of Discrete Mathematics **11** pp. 644-654.

[4] Chvátal, V. and Slater, P. J. (1993), *A note on well-covered graphs*, *in* 'Quo Vadis, Graph Theory?', Annals of Discrete Math. **55**, North-Holland, Amsterdam, pp. 179-182.

[5] Edmonds, J. (1971), Matroid and the greedy algorithm, Math. Programming **1**, pp. 127–113.

[6] Favaron, O. (1982), Very well-covered graphs, Discrete Mathematics **42** pp. 177-187.

[7] Finbow, A., Hartnell, B. and Nowakowski, R. J. (1993), A characterization of well-covered graphs of girth 5 or greater, Journal of Combinatorial Theory, Ser B **57** pp. 44-68.

[8] Finbow, A., Hartnell, B. and Nowakowski, R. J. (1994), A characterization of well-covered graphs that contain neither 4- nor 5-cycles, Journal of Graph Theory **18** pp. 713-721.

[9] Hartnell, B., Plummer, M. D. (1996), On 4-connected claw-free well-covered graphs, Discrete Applied Mathematics **64** pp. 57-65.

[10] Hedetniemi, S. T. and Laskar, R. (1984), Connected domination in graphs, in Graph Theory and Combinatorics, Eds. B. Bollobas, Academic Press, London, pp. 209-218.

[11] Korte, B., Lovász, L. and Schrader, R. (1991), *Greedoids*, Springer-Verlag, Berlin.

[12] Levit, V. E. and Mandrescu, E. (1998), Well-covered and König-Egerváry graphs, Congressus Numerantium **130** pp. 209-218.

[13] Levit, V. E. and Mandrescu, E. (1999), Well-covered trees, Congressus Numerantium **139** pp. 101-112.

[14] Levit, V. E. and Mandrescu, E. (2001), Unicycle bipartite graphs with only uniquely restricted maximum matchings, *in* C.S. Calude, M. J. Dinneen and S. Sburlan eds. 'Proceedings of the Third International Conference on Combinatorics, Computability and Logic, (DMTCS'1)', Springer, pp. 151-158.

[15] Levit, V. E. and Mandrescu, E. (2002), A new greedoid: the family of local maximum stable sets of a forest, Discrete Applied Mathematics **124** pp. 91-101.

[16] Levit, V. E. and Mandrescu, E. (2003), Local maximum stable sets in bipartite graphs with uniquely restricted maximum matchings, Discrete Applied Mathematics **132** pp. 163-174.

[17] Levit, V. E. and Mandrescu, E. (2005), Unicycle graphs and uniquely restricted maximum matchings, Electronic Notes in Discrete Mathematics, **22** pp. 261265.

[18] Levit, V. E. and Mandrescu, E. (2007a), Triangle-free graphs with uniquely restricted maximum matchings and their corresponding greedoids, Discrete Applied Mathematics, doi: 10.1016/j.dam.2007.05.039 (in press).

[19] Levit, V. E. and Mandrescu, E. (2007b), Some Structural Properties of Very Well-Covered Graphs, Congressus Numerantium (accepted).

[20] Nemhauser, G. L. and Trotter, E., Jr. (1975), Vertex packings: structural properties and algorithms, Mathematical Programming **8** pp. 232-248.

[21] Plummer, M. D. (1970), Some covering concepts in graphs, Journal of Combinatorial Theory **8** pp. 91-98.

[22] Plummer, M. D. (1993), Well-covered graphs : a survey, Quaestiones Mathematicae **16** pp. 253-287.

[23] E. Prisner, E., Topp, J. and P. D. Vestergaard, P. D. (1996), Well-covered simplicial, chordal, and circular arc graphs, Journal of Graph Theory **21** pp. 113-119.

[24] Ravindra, G. (1977), Well-covered graphs, J. Combin. Inform. System Sci. **2** pp. 20-21.

[25] Sankaranarayana, R., Stewart, L. K. (1992), Complexity results for well-covered graphs, Networks **22** (3) pp. 247-262.

[26] D. Tankus, D. and Tarsi, M. (1996), Well-covered claw-free graphs, Journal of Combinatorial Theory Ser. **B 66** pp. 293-302.

[27] D. Tankus, D. and Tarsi, M. (1997), The structure of well-covered graphs and the complexity of their recognition problems, Journal of Combinatorial Theory Ser. **B 69** pp. 230-233.

[28] Topp, J. and Volkmann, L. (1990), Well-covered and well-dominated block graphs and unicyclic graphs, Mathematica Panonica **1/2** pp. 55-66.

[29] Whitney, H. (1935), On the abstract properties of linear independence, Amer. J. Math. **57** pp. 509-533.

[30] Zverovich, I. E. (2004), Weighted well-covered graphs and complexity questions, Moscow Mathematical Journal **4** pp. 523-528.

On the non-existence of even degree graphs with diameter 2 and defect 2

Mirka Miller[1,2] **Minh H. Nguyen**[3,1] **Guillermo Pineda-Villavicencio**[1,4]

[1] School of Information Technology and Mathematical Sciences
University of Ballarat
P.O.Box 663, Vic 3353, Australia
Emails: m.miller@ballarat.edu.au, gpinedavillavicencio@students.ballarat.edu.au

[2] Department of Mathematics
University of West Bohemia
Pilsen, Czech Republic

[3] Hutchison Managed Service
Ericsson Australia
112-118 Talavera Road, North Ryde, NSW 2113, Australia
Email: minh.n.nguyen@ericsson.com

[4] Department of Computer Science
University of Oriente
Santiago de Cuba, Cuba

Abstract

Using eigenvalue analysis, it was shown by Erdös et al. that, with the exception of C_4, there are no graphs of diameter 2, maximum degree d and d^2 vertices. In this paper, we show that graphs of diameter 2, maximum degree d and d^2-1 vertices do not exist for most values of d, when d is even, and we conjecture that they do not exist for any even d greater than 4.

Keywords: Moore graphs; diameter 2; degree/diameter problem

1 Introduction

There are many famous and difficult graph-theoretical problems that arose over the past four decades from the design of interconnection networks (such as local area networks, parallel computers, switching system architecture in VLSI technology, and many others). Perhaps one of the most prominent problems is the *degree/diameter problem* which is to determine, for each d and k, the largest order $n_{d,k}$ of a graph of maximum degree d and diameter at most k. It is easy to show that $n_{d,k} \leq M_{d,k}$ where $M_{d,k}$ is the *Moore bound*, given by

$$n_{d,k} \leq M_{d,k} = 1 + d + d(d-1) + \ldots + d(d-1)^{k-1}$$

For a survey of the degree/diameter problem, see Miller et al. (2005).

In this paper we concentrate on the case when the diameter is equal to 2. Since a graph of diameter 2 and maximum degree d may have at most d^2+1 vertices, it was asked in (Erdös et al. 1980): Given non-negative integer numbers d and Δ (*defect*), is there a graph of diameter 2 and maximum degree d with $d^2+1-\Delta$ vertices? It was proved in (Hoffman et al. 1960) that if $\Delta = 0$ then there are unique graphs corresponding to $d = 2, 3, 7$ and possibly $d = 57$. The case $\Delta = 1$ was solved by Erdös et al. (Erdös et al. 1980). In this paper we consider the case $\Delta = 2$ and prove that graphs of defect 2 do not exist for most values of degree d in the case when d is even.

We refer to a graph of maximum degree d, diameter $k \geq 2$ and order $M_{d,k} - \Delta$ ($\Delta \geq 1$) as a (d, k, Δ)-*graph*. Let G be a (d, k, Δ)-graph.

Definition 1 Let u be a vertex in G. A vertex v in G is called a *repeat* of u with *multiplicity* $m_v(u)$ ($1 \leq m_v(u) \leq \Delta$) if there are exactly $m_v(u) + 1$ different paths of lengths at most k from u to v.

It is immediate that

Observation 1 *Vertex u is a repeat of v with multiplicity $m_u(v)$ if and only if v is a repeat of u with the same multiplicity.*

A repeat with multiplicity 1 will be called a *single* repeat, a repeat with multiplicity 2 will be called a *double* repeat, a repeat with multiplicity Δ will be called a *maximal* repeat.

We denote by $R_s(u)$ the set of all repeats of a vertex u in G. Taking into account the multiplicities of repeats, we denote by $R_m(u)$ the multiset of all the repeats of a vertex u in G, containing each repeat v of u exactly $m_v(u)$ times.

Let u be a vertex in G. We denote by $N(u)$ the set of all neighbours of u. If A is a multiset of vertices of G, then $N(A)$ denotes the multiset of all the neighbours of the vertices of A. We use $R_m(A)$ to denote the multiset of all the repeats of all vertices in A.

Proposition 1 *If G is regular then, for all $u \in V(G)$,*

$$|R_m(u)| = \sum_{v \in R_s(u)} m_v(u) = \Delta.$$

□

Definition 2 A subset S of $V(G)$ is called a *closed repeat set* if $R_m(S) = S$. A closed repeat set is *minimal* if none of its proper subsets is a closed repeat set.

Definition 3 A *repeat subgraph* H_S of a closed repeat set S of G is a multigraph whose vertex set $V(H_S) = S$ and the number of parallel edges between a vertex u and any of its repeats, say $v \in R_m(u)$, equals the multiplicity $m_v(u)$.

We observe that

Observation 2 *If* $\Delta < 1 + (d-1) + \ldots + (d-1)^{k-1}$ *then* G *is regular.*

It is also true that

Observation 3 *If* G *is regular then the repeat graph* H_G *of* G *is* Δ*-regular.*

Note that instead of writing "a vertex x is adjacent to a vertex y" we write $x \sim y$, and if x is not adjacent to y then we write $x \nsim y$. Unless explicitly shown where necessary, by u_i and u_j $(i \neq j)$ we shall mean two distinct vertices.

2 Structural properties of $(d, 2, 2)$-graphs

In this section we consider graphs of diameter 2 with defect 2. Such graphs do not exist for $d \leq 2$. Let G be a $(d, 2, 2)$-graph for $d \geq 3$. From Observation 2, we have that

Observation 4 *Every* $(d, 2, 2)$*-graph for* $d \geq 3$ *is regular.*

Let us consider repeat configurations in $(d, 2, 2)$-graphs. Let u be a vertex of a $(d, 2, 2)$-graph. Then there are two possibilities:

- u has two single repeats, $r_i(u)$, $i = 1, 2$.
- u has one double (maximal) repeat, $r(u) = r_1(u) = r_2(u)$, with multiplicity 2.

With respect to repeats in G, there are five possible repeat configurations, as depicted in Fig. 1.

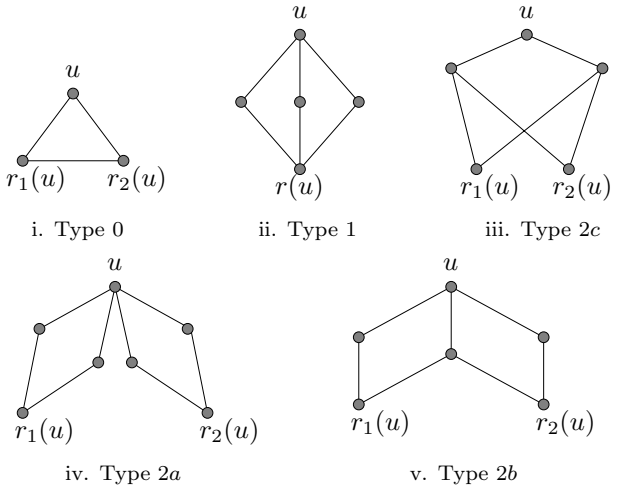

Figure 1: Possible repeat configurations for vertex u in a $(d, 2, 2)$-graph.

We will denote the set of vertices of each type by Type 0, Type 1, Type 2a, Type 2b and Type 2c, as shown in Fig. 1. We denote by n_0, n_1, n_{2a}, n_{2b}, n_{2c} the number of vertices of the corresponding repeat types.

Fig. 2 shows the only known $(d, 2, 2)$-graph (for even d) whose uniqueness is shown in (Broersma et al. 1988).

We observe the following

Observation 5 $n_0 + n_1 + n_{2a} + n_{2b} + n_{2c} = d^2 - 1$.

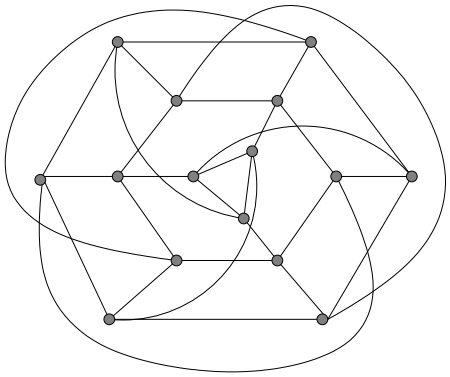

Figure 2: The only known $(d, 2, 2)$-graph for even d.

For the purpose of this paper, we shall consider each pair of parallel edges in H_G as a cycle of length 2.

Observation 6 H_G *is the union of cycles of lengths* ≥ 2*, each cycle a minimal closed repeat set of* G*.*

From now on, each cycle in H_G will be called a *repeat cycle*.

The following structural properties of G were proved in (Nguyen et al. 2007).

Theorem 1 *(Nguyen et al. 2007) In a* $(d, 2, 2)$*-graph* G*, if* d *is even then* $n_0 = 3$ *and* $n_{2b} = d^2 - 4$.

Corollary 1 *(Nguyen et al. 2007)* $n_{2b} \equiv 0 \pmod 2$.

Let the vertices u_0, u_1, u_2 form a triangle in G, denoted by T, and let Υ_{2b} be the subset of all vertices of type $2b$ in $N(u_0) \bigcup N(u_1) \bigcup N(u_2)$. Then Υ_{2b} is a minimal closed repeat set. We shall call Υ_{2b} the *outer repeat cycle* of T in H_G. Note that Υ_{2b} is the set of vertices at distance 1 from T, and $\Upsilon_{2b} \bigcap T = \emptyset$. The number of vertices of Υ_{2b} is $3(d-2)$.

Fig. 3 illustrates a labeled partial structure of G, in the case when d is even, which shows the cycle $u_0u_1u_2$ and its outer repeat cycle. Since Υ_{2b} contains all vertices of type $2b$ and Υ_{2b} is a minimal closed repeat set, by Corollary 1, there exists in H_G another cycle Υ'_{2b}, also of the same size as Υ_{2b}, that is, $3(d-2)$. Note that, in Fig. 3, $u_3 \sim u_{3d-4}$ and $u_{3d-3} \sim u_{9d-16}$. This is because u_3 and u_{9d-16} belong to Υ_{2b} whereas u_{3d-3} and u_{3d-4} belong to Υ'_{2b}.

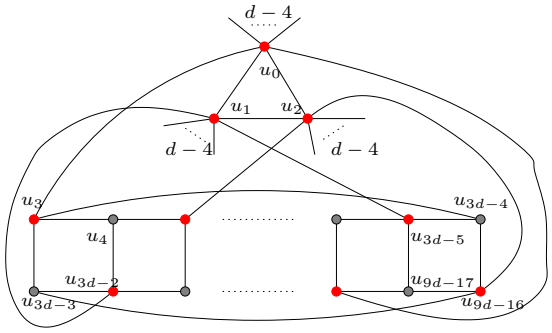

Figure 3: An illustration of the neighbourhood of T in G for even d.

Lemma 1 *(Nguyen et al. 2007) Let* T *be a triangle in* G *and let* Υ_{2b} *be the outer repeat cycle of* T *in* H_G*. Let* C_t *be any repeat cycle in* H_G *of length* $t \geq 4$ *such that there exists in* G *an edge between a vertex on* Υ_{2b} *and a vertex on* C_t*. Then either* $t = \frac{1}{3}|\Upsilon_{2b}|$ *or* $t \equiv 0 \pmod{|\Upsilon_{2b}|}$.

3 On the non-existence of $(d, 2, 2)$-graphs for even d

In this section, we shall prove that for most values of even d, $(d, 2, 2)$-graphs do not exist.

From Theorem 1, it immediately follows that

Corollary 2 *G is not vertex-transitive for even degree d.*

Lemma 2 *For even $d \geq 6$, every cycle, other than the triangle in H_G, has length $3k(d-2)$, for some $k \geq 1$.*

Proof. As demonstrated in Section 2, H_G contains one cycle of length 3 and at least two cycles of length $3(d-2)$. Let $u_1u_2u_3$ be the triangle T of G and let $v_1 \dots v_{3(d-2)}$ be the outer repeat cycle Υ_{2b} of T in H_G such that the repeats of v_j $(1 \leq j \leq 3(d-2))$ are $v_{(j-1)(\bmod (3(d-2)))}$ and $v_{(j+1)(\bmod (3(d-2)))}$. Without loss of generality, let us suppose that $u_1 \sim v_1$ and $u_2 \sim v_2$ in G.

Let the a_i, $i = 1, \dots, b$, be the lengths of the cycles in H_G and let $a_1 = 3$, $a_2 = a_2 = 3(d-2)$ correspond to T, Υ_{2b} and Υ'_{2b}, respectively. Thus, $f = \sum_{i=4}^{b} a_i = (d-2)(d-4)$.

Let C_{a_j} be an arbitrary cycle in H_G $(j \neq 1, 2, 3)$. Then, by Lemma 1, either $a_j = d-2$, or $a_j \equiv 0 \pmod{3(d-2)}$. Suppose that $a_j = d-2$. Denote by $w_1, \dots, w_{d-2}$ the vertices of C_{a_j} such that the repeats of w_k $(1 \leq k \leq d-2)$ are $w_{(k-1)\ (\bmod (d-2))}$ and $w_{(k+1)\ (\bmod (d-2))}$.

We know that the vertices of C_{a_j} must reach the vertices of T through the vertices of Υ_{2b}. Without loss of generality, suppose that $w_1 \sim v_1$ and $w_2 \sim v_2$. However, since $(d-2)$ is not divisible by 3 when d is even, by the Neighbourhood Theorem, u_1 and w_1 would then have at least three common neighbours, namely v_1, v_{d-1} and v_{2d-3}. This is clearly impossible.

Therefore, each a_i $(4 \leq i \leq b)$ must be a multiple of $3(d-2)$. □

Theorem 2 *For even $d \geq 4$, if $d \not\equiv 1 \pmod 3$ then there is no $(d, 2, 2)$-graph.*

Proof. Let b be the number of cycles in H_G. Let a_i, for $i = 1 \dots b$, be the lengths of these cycles, denoting by a_1 the triangle. Then as $\sum_{i=1}^{b} a_i = d^2 - 1$, by Lemma 2, we have that $\sum_{i=2}^{b} a_i = 3(d-2)k = d^2 - 4 = (d-2)(d+2)$. Therefore, $d+2 \equiv 0 \pmod 3$. □

By counting the total number N_5 of 5-cycles in G, we derive some further necessary conditions for the existence of G.

Theorem 3 *For even $d \geq 4$, if $N_5 = \frac{(d-2)(d^4+2d^2-2d-25)}{10}$ is not an integer then there is no $(d, 2, 2)$-graph.*

The results of Theorems 2 and 3 improve the upper bound for the order of $(d, 2, 2)$-graphs so that $n_{d,2} \leq d^2 - 3$ for infinitely many even degrees d. For $d \geq 10$, the first 50 values of d for which G might still exist are shown in Table 1.

We conclude this paper by posing the following

Conjecture 1 *For even $d \geq 6$, $(d, 2, 2)$-graphs do not exist.*

10	22	34	40	52	64	70	82	94	100
112	124	130	142	154	160	172	184	190	202
214	220	232	244	250	262	274	280	292	304
310	322	334	340	352	364	370	382	394	400
412	424	430	442	454	460	472	484	490	502

Table 1: The first 50 values of d for which a $(d, 2, 2)$-graph might still exist for even d.

4 Acknowledgement

We greatfully acknowledge support from the ARC grant DP0450294.

References

Broersma, H.J. & Jagers, A.A. (1988), The unique 4-regular graphs on 14 and 15 vertices with diameter 2, *Ars Combinatoria* **25C**, 55–62.

Erdös, P., Fajtlowicz, S. & Hoffman, A. J. (1980), Maximum degree in graphs of diameter 2, *Networks* **10**, 87–90.

Hoffman, A.J. & Singleton, R.R. (1960), On Moore Graphs with diameters 2 and 3, *IBM J. Res. Dev.* **64**, 15–21.

Miller, M. & Širáň, J. (2005), Moore graphs and beyond: A survey of the degree/diameter problem, *Electronic Journal of Combinatoric* **DS14**, 1–6.

Nguyen, M.H. & Miller, M. (2007), Structural properties of graphs of diameter 2 with defect 2, preprint.

Graph Classes and the Complexity of the Graph Orientation Minimizing the Maximum Weighted Outdegree*

Yuichi Asahiro[1] **Eiji Miyano**[2, †] **Hirotaka Ono**[3]

[1] Department of Social Information Systems, Kyushu Sangyo University,
2-3-1 Matsukadai, Higashi-ku, Fukuoka 813-8503, Japan.
Email: asahiro@is.kyusan-u.ac.jp

[2] Department of Systems Innovation and Informatics, Kyushu Institute of Technology,
680-4 Kawazu, Iizuka, Fukuoka 820-8502, Japan.
Email: miyano@ces.kyutech.ac.jp

[3] Department of Computer Science and Communication Engineering, Kyushu University,
744 Motooka, Nishi-ku, Fukuoka 819-0395, Japan.
Email: ono@csce.kyushu-u.ac.jp

Abstract

Given an undirected graph with edge weights, we are asked to find an orientation, i.e., an assignment of a direction to each edge, so as to minimize the weighted maximum outdegree in the resulted directed graph. The problem is called MMO, and is a restricted variant of the well-known minimum makespan problem. As previous studies, it is shown that MMO is in $\mathcal{P}$ for trees, weak $\mathcal{NP}$-hard for planar bipartite graphs, and strong $\mathcal{NP}$-hard for general graphs. There are still gaps between those graph classes. The objective of this paper is to show tight thresholds of complexity: We show that MMO is (i) in $\mathcal{P}$ for cactuses, (ii) weakly $\mathcal{NP}$-hard for outerplanar graphs, and also (iii) strongly $\mathcal{NP}$-hard for P_4-bipartite graphs. The latter two are minimal superclasses of the former. Also, we show the $\mathcal{NP}$-hardness for the other related graph classes, diamond-free, house-free, series-parallel, bipartite and planar.

Keywords: graph orientation, min-max optimization, $\mathcal{NP}$-hardness, cactus, (outer)planar, (P_4-)bipartite, series-parallel, house-free, diamond-free.

1 Introduction

1.1 Problem and Summary of Results

Let $G = (V, E, w)$ be an undirected and edge weighted graph, where V, E and w denote the set of nodes, the set of edges and a positive integral weight function $w : E \to \mathbb{Z}^+$, respectively. An *orientation* Λ *of the graph* G is a set of an assignment of a direction to each edge $\{u, v\} \in E$, i.e., either (u, v) or (v, u) is contained in Λ. The *weighted outdegree* of u is $\sum_{\substack{\{u,v\}\in E:\\ (u,v)\in\Lambda}} w(\{u, v\})$. In this paper, we consider the problem of finding an orientation such that the maximum weighted outdegree is minimum in the resulted directed graph. We call this problem Minimum Maximum Outdegree (MMO). See Fig. 1 for an example of an edge weighted graph and its orientation in which the maximum weighted outdegree is 3 (optimal).

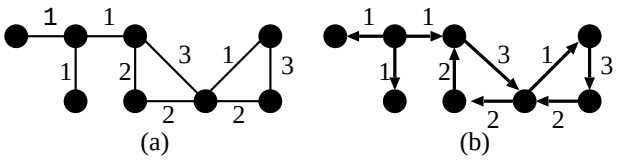

Figure 1: Example of MMO: (a) An edge weighted graph, and (b) an orientation.

MMO has several applications. For example, such orientations can be used in efficient dynamic data structures for graphs that support fast vertex adjacency queries under a series of edge operations (Brodal & Fagerberg 1999). Also, MMO can be considered a variation of *art gallery problems* (e.g., (Chv'atal 1975, O'Rourke 1987)) and *the minimum makespan problem* (e.g., (Lenstra, Shmoys & Tardos 1990)). In particular, we will discuss the minimum makespan problem in the next subsection.

MMO can be solved in polynomial time if all the edge weights are identical (Asahiro, Miyano, Ono, & Zenmyo 2007, Kowalik 2006, Venkateswaran 2004), but it is $\mathcal{NP}$-hard in general (Asahiro, Miyano, Ono, & Zenmyo 2007, Asahiro, Jansson, Miyano, Ono, & Zenmyo 2007). Even with non-identical weights, the problem can be also solved in polynomial time if the input graph is limited to a tree (Asahiro, Miyano, Ono, & Zenmyo 2007), while for planar bipartite graphs it is still (weakly) $\mathcal{NP}$-hard.

As many other studies on the computational complexity, it is valuable to consider the frontier between subproblems we know to be solvable in polynomial time and those we know to be $\mathcal{NP}$-hard. In this paper, we focus on the structure of the input graphs related to the $\mathcal{NP}$-hardness. Fig. 2 shows the current state of knowledge on the complexity of MMO, including the results in this paper. The figure represents that for example, cactus is a superclass of tree at the bottom. As another example, P_4-bipartite is a superclass of bipartite and cactus, but bipartite and cactus are not comparable, and so on. All the reductions to show the $\mathcal{NP}$-hardness are done by simple graphs except outerplanar graphs, which we will explain in a later section. Namely, the weak $\mathcal{NP}$-hardness of series-parallel graphs is proved with simple graphs,

*This work is partially supported by Grant-in-Aid for Scientific Research on Priority Areas 16092222 and 16092223, and by Grant-in-Aid for Young Scientists (B) 17700022, 18700014 and 18700015.

†Currently visiting Dept of Computer Sci and Eng, University of Washington, Seattle, WA 98195-2350, USA.

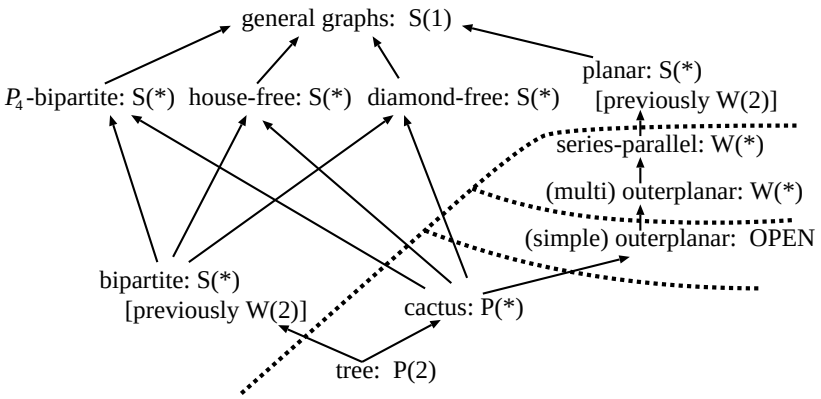

Figure 2: State of knowledge on the complexity of MMO. (W: Weakly $\mathcal{NP}$-hard, S: Strongly $\mathcal{NP}$-hard, (1): The results in (Asahiro, Jansson, Miyano, Ono, & Zenmyo 2007), (2): The results in (Asahiro, Miyano, Ono, & Zenmyo 2007), (*): The results in this paper).

but that of outerplanar graphs is proved with multi graphs, and so the complexity for simple outerplanar graphs is still open. Additionally, we propose a pseudo-polynomial time algorithm for series-parallel graphs, which shows the tightness of our weak $\mathcal{NP}$-hardness result in a sense; MMO for simple outerplanar graphs is either in $\mathcal{P}$ or weakly $\mathcal{NP}$-hard.

1.2 Related Work

As mentioned before, another aspect of the problem MMO is scheduling; MMO is regarded as a special case of *minimum makespan* or *scheduling on unrelated parallel machines* ($R||C_{max}$ in the now-standard notation): Given a set J of jobs, a set M of machines, and the time p_{ij} taken to process job $j \in J$ on machine $i \in M$, its goal is to find a job assignment so as to minimize the makespan, i.e., the maximum processing time of any machine. For an undirected graph, let us regard the nodes as the machines and the edges as the jobs. From the viewpoint of scheduling, MMO has the following two restrictions: (i) Each job must be assigned to exactly one of pre-determined two machines, and (ii) the processing time of each job does not depend on the machines.

In (Lenstra, Shmoys & Tardos 1990), a polynomial time 2-approximation algorithm for the general $R||C_{max}$ and its 3/2 inapproximability are shown. Still there has been gap between these upper and lower bounds; it is one of the well-known open problems(Schuurman & Woeginger 1999). To tackle this kind of situation, it is a natural way to restrict the input as a reasonable subclass: In (Gairing, Lücking, Mavronicolas, & Monien 2004), a polynomial time $2 - 1/k$-approximation algorithm is proposed, under the assumption that the processing times of jobs are integers and k is the maximum among them. Also, (Asahiro, Jansson, Miyano, Ono, & Zenmyo 2007) considers a further restricted problem in which the processing time of each job is either 1 or k, and then proposes a polynomial time $2 - 2/(k+1)$-approximation algorithm for $k \geq 3$, and shows that 3/2 inapproximability still holds for this restricted case even with $k = 2$. In brief summary, the approximation ratios of those algorithms are slightly smaller than two, and the same (3/2) lower bound is shown for the restricted case. However, any tight bound between 3/2 and 2 has not been found for about two decades. The contribution of this paper, from the viewpoint of scheduling, is to make clear what kind of structure of the instances is really difficult to solve.

2 Preliminaries

2.1 Definitions

Let $G = (V, E, w)$ be an edge weighted undirected graph, where V and E are node and edge sets, respectively, and w is a positive integral weight function $w : E \to \mathbb{Z}^+$. $V(G)$ and $E(G)$ also denote the node set and edge set of the graph G, respectively. We denote the undirected edge whose endpoints are u and v where $u < v$ in lexicographic order by $\{u, v\}$, and denote the directed edge (or *arc*) from u toward v by (u, v). An *orientation* Λ *of the graph* G is a set of an assignment of a direction to each edge $\{u, v\} \in E$, i.e., Λ contains exactly either one of (u, v) and (v, u). Also $\Lambda(\{u, v\})$ denotes the direction (u, v) or (v, u) of an edge $\{u, v\}$ in Λ.

For a node v, $d_G(v)$ denotes the *degree* of v in G, i.e., $d_G(v) = |\{\{v, u\} \mid \{v, u\} \in E\}|$. The *weighted outdegree* (or, simply *outdegree*) $d^+_G(\Lambda, v)$ of a node v under an orientation Λ of the graph G is defined as the total weight of outgoing arcs of v, i.e.,

$$d^+_G(\Lambda, v) = \sum_{\{u,v\} \in E:\ (v,u) \in \Lambda} w(\{u, v\}).$$

For simplicity we also use $d(v)$ and $d^+(\Lambda, v)$ instead of $d_G(v)$ and $d^+_G(\Lambda, v)$ if the graph G we are discussing is clear. Then the *cost* of an orientation Λ of a graph G is defined to be $\Delta_\Lambda(G) = \max_{v \in V}\{d^+_G(\Lambda, v)\}$.

A *path* P of length l is denoted by a sequence of nodes such as $P = \langle v_0, v_1, v_2, \ldots, v_l \rangle$. Also a *cycle* C of length l is denoted by $C = \langle v_1, v_2, \ldots, v_l, v_1 \rangle$. In this paper, a *cycle* always refers a simple cycle, namely, for the cycle C, $v_i \neq v_j$ for any i and j. A node in a cycle is a *gate* if it is adjacent to any node that does not belong to the cycle, so that the degree of the gate is at least three.

A graph is a *cactus* if every edge is part of at most one cycle. The definition of the series-parallel graphs is little bit complicated (p.100 of (Gross & Yellen 2004)):

Definition 1 *A* series-parallel graph *with distinguished terminals l and r is denoted (G, l, r) and is defined recursively as follows:*

- *The graph consisting of a single edge $\{v_1, v_2\}$ is a series-parallel graph (G, l, r) with $l = v_1$ and $r = v_2$.*
- *A series operation $(G_1, l_1, r_1) \odot_s (G_2, l_2, r_2)$ forms a series-parallel graph by identifying r_1 with l_2. The terminals of the new graph are l_1 and r_2.*

- *A parallel operation* $(G_1, l_1, r_1) \odot_p (G_2, l_2, r_2)$ *forms a series-parallel graph by identifying* l_1 *with* l_2 *and* r_1 *with* r_2*. The terminals of the new graph are* l_1 *and* r_1*.*
- *A jackknife operation* $(G_1, l_1, r_1) \odot_j (G_2, l_2, r_2)$ *forms a series-parallel graph by identifying* r_1 *with* l_2*; the new terminals are* l_1 *and* r_1*.*

The definitions of graph classes except cactus and series-parallel in this paper, s.t., house-free, diamond-free and so on, can be found in (Brandstädt, BangLe, & Spinrad 1987, Gross & Yellen 2004). We would like to note here that tree, bipartite, cactus, house-free, and diamond-free are obviously recognized in polynomial time (See also, e.g., (Kloks, Kratsch, & Müller 2000)). Also, there are efficient recognition algorithms for outerplanar, series-parallel, and planar that run in linear time (Mitchell 1979, Valdes, Tarjan, & Lawler 1982, Hopcroft, & Tarjan 1974). However, it is $\mathcal{NP}$-hard to recognize P_4-bipartite graphs (Hoàng, & Le 2001).

2.2 Problem S-MMO and Basic Properties

The problem that we consider in this paper is the minimization of the maximum outdegree of a given undirected graph with edge weights. We formally define our problem as follows.

Problem: S-Minimum Maximum Outdegree (S-MMO)

Input: An undirected graph $G = (V, E, w)$, where w is an edge weight function $w : E \to S$.

Output: An orientation Λ that minimizes $\Delta_\Lambda(G)$.

If we have no restriction on the weight function w (just it should be a positive integral function), our problem is $\mathbb{Z}^+$-MMO. In this paper, we mainly consider the problem for the case of $S = \{1, 2, \ldots, k\}$.

Let $\Delta^*(G)$ denote the cost of an optimal orientation OPT_G of the graph G, i.e., $\Delta^*(G) = \Delta_{OPT_G}(G)$. We say a graph orientation algorithm is a σ-approximation algorithm if $\Delta_{ALG}(G)/\Delta^*(G) \leq \sigma$ holds for any graph G, where ALG is an orientation obtained by the algorithm for G. Every orientation has the following trivial lower bound caused by the maximum weight $w_{\max}$ of edges (Asahiro, Miyano, Ono, & Zenmyo 2007): For a graph G and any orientation Λ, $\Delta_\Lambda(G) \geq w_{\max}$, so that $\Delta^*(G) \geq w_{\max}$.

The following property of a cactus is very simple but plays a key role to construct the polynomial time algorithm in the next section.

Proposition 2 *In a cactus* G *in which* $d_G(v) \geq 2$ *for all* $v \in V$*, there always exists a cycle with at most one gate.*

Proof: We prove this proposition by contradiction. Suppose that all cycles have at least two gates. Let C be a cycle of length l, $C = \langle v_1, v_2, \ldots, v_l, v_1 \rangle$.

Without loss of generality, assume that v_1 is a gate, i.e., there exists a node $x_2 \notin V(C)$ adjacent to v_1. Since $d(x_2) \geq 2$ by the assumption, there also exists a node x_3 adjacent to x_2. Similarly, for a node x_i reachable from v_1, there exists a node x_{i+1} adjacent to x_i. Consider a path P starting from v_1, $P = \langle v_1, x_2, \ldots, x_p \rangle$ for $p \geq 2$. If $x_j = v_h$ for some $2 \leq j \leq p$ and $2 \leq h \leq l$, there exists a cycle $C' = \langle v_1, x_2, \cdots, x_j (= v_h), v_{h-1}, \ldots, v_2, v_1 \rangle$. The

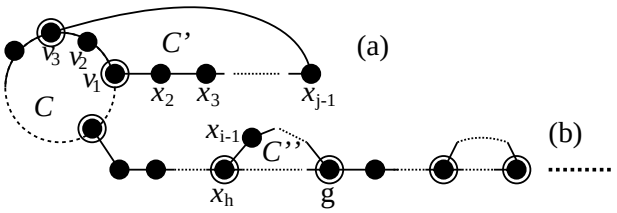

Figure 3: Proof of Proposition 2.

cycles C and C' share the edge $\{v_1, v_2\}$, which contradicts that G is a cactus. (See Fig. 3 (a) in which $h = 3$)

Hence, we assume that such a node x_j does not exist. It turns out to happen $x_i = v_1$ or $x_i = x_h$ for some i and $2 \leq h \leq i - 1$, i.e., $C_1'' = \langle v_1, x_2, \ldots, x_{i-1}, x_i (= v_1) \rangle$ or $C_2'' = \langle x_h, x_{h+1}, \ldots, x_{i-1}, x_i (= x_h) \rangle$, respectively, is a cycle with the gate x_i. Since we assumed that every cycle has at least two gates, there must exist another gate $g \neq v_1$ in C_1'', or $g \in \{x_{h+1}, \ldots, x_{i-1}\}$ in C_2'', respectively. We can replace C by C_1'' or C_2'' and v_1 by g in the above discussion and then continue. However, since the number of the nodes in G is bounded, eventually a contradiction occurs, namely, the cycle C_1'' or C_2'' has only one gate or G is not a cactus. (See Fig. 3 (b)) □

3 Polynomial Time Algorithm for Cactuses

In this section, we present a polynomial time algorithm for cactuses. First we introduce a relaxed version (S, T)-Minimum Maximum Outdegree ((S, T)-MMO) of the original problem S-MMO and show its several propositions in Sec. 3.1. In Sec. 3.2, we describe an algorithm to solve the decision version (S, T)-MMO(K) of (S, T)-MMO. Finally, the proposed polynomial time algorithm to solve (S, T)-MMO will be given in Sec. 3.3.

3.1 Relaxed Problem (S, T)-MMO

We relax S-MMO to a problem whose input graph has node weights as well as edge weights. Before describing the problem formally, we define some notations analogously to those for edge weighted graphs in Sec. 2.1.

Let $G = (V, E, f, w)$ be a node and edge weighted undirected graph, where V and E are node and edge sets, respectively, and f and w are positive integral weight functions $f : V \to \mathbb{Z}^+$ and $w : E \to \mathbb{Z}^+$. The *weighted outdegree* (or, simply *outdegree*) $d_G^+(\Lambda, v)$ of a node v under an orientation Λ of the graph G is modified to the weight of v itself plus the total weight of outgoing arcs of v, i.e.,

$$d_G^+(\Lambda, v) = f(v) + \sum_{\{u,v\} \in E:\ (v,u) \in \Lambda} w(\{u, v\}).$$

Definitions of the others, e.g.,, degree, orientation, cost of an orientation, etc., are the same as before. Then the new problem is defined as follows.

Problem: (S, T)-Minimum Maximum Outdegree ((S, T)-MMO)

Input: An undirected graph $G = (V, E, f, w)$, where f is a node weight function $f : V \to T$, and w is an edge weight function $w : E \to S$.

Output: An orientation Λ that minimizes $\Delta_\Lambda(G)$.

Theorem 3 *Consider a node and edge weighted graph $G = (V, E, f, w)$, an edge weighted graph $G^0 = (V, E, w)$, and a constant c. If $f(v) = c$ for all $v \in V$, then $\Delta_\Lambda(G) = \Delta_\Lambda(G^0) + c$ for any orientation Λ.* □

From the above theorem, we obtain the following corollary in a straightforward way, and so $\mathcal{NP}$-hardness results for S-MMO (by the previous studies and in this paper as well) are directly applied to (S, T)-MMO.

Corollary 4 *For a node and edge weighted graph $G = (V, E, f, w)$, (S, T)-MMO is equivalent to S-MMO, if $f(v) = 0$ for all $v \in V$.* □

For simplicity, we denote (S, c)-MMO to represent $(S, \{c\})$-MMO, that is $f(v) = c$ for all nodes. For a pair of graphs $G = (V, E, f, w)$ and $G' = (V', E', f', w')$, G' is a *subgraph* of G if $V' \subseteq V$, $E' \subseteq E$, and $w'(e) = w(e)$ for all $e \in E'$. A subgraph G' of G is called a *proper subgraph* of G if an additional condition $f'(v) = f(v)$ for all $v \in V'$ is satisfied. Note that in this paper we will only see subgraphs satisfying that $f'(v) \geq f(v)$ for all nodes. Here we extend the definition of the orientation: An orientation Λ of a graph G may contain (u, v) or (v, u) for $\{u, v\} \notin E(G)$. This extension does not affect the value of (out)degrees by definition. When we have to deal with $w(\{u, v\})$ for $\{u, v\} \notin E(G)$, we just consider $w(\{u, v\}) = 0$.

In the following, we state four propositions 5, 6, 7, and 8. These propositions are utilized in order to develop the polynomial time algorithms for cactuses. Proposition 5 shows a relationship between optimal costs for two graphs only node weight functions of which are different. Propositions 6 and 7 are on the optimal costs for proper subgraphs of a graph. Then in Proposition 8, we take a look at the optimal costs for non-proper subgraphs. Since they are not difficult to show, we omit the proofs for these propositions.

Proposition 5 *Consider two graphs $G = (V, E, f, w)$ and $G' = (V, E, f', w)$ such that $f(v) \leq f'(v)$ for all $v \in V$. Then $\Delta^*(G) \leq \Delta^*(G')$ holds.* □

Proposition 6 *For a graph G, its proper subgraph $G' = G - e$ for $e \in E(G)$, and a pair of orientations Λ of G and Λ' of G', s.t., $\Lambda' = \Lambda \setminus \{\Lambda(e)\}$, the following three conditions are satisfied:*

(i) $\Delta_\Lambda(G') = \Delta_{\Lambda'}(G')$,

(ii) $\Delta_\Lambda(G) \geq \Delta_{\Lambda'}(G')$, *and*

(iii) $\Delta^*(G) \geq \Delta^*(G')$. □

Proposition 7 *For a graph G, its proper subgraph $G' = G - v$ for $v \in V(G)$, and a pair of orientations Λ of G and Λ' of G', s.t., $\Lambda' = \Lambda \setminus \{\Lambda(\{v, u\}) \mid \{v, u\} \in E(G)\}$, the following three conditions are satisfied:*

(i) $\Delta_\Lambda(G') = \Delta_{\Lambda'}(G')$,

(ii) $\Delta_\Lambda(G) \geq \Delta_{\Lambda'}(G')$, *and*

(iii) $\Delta^*(G) \geq \Delta^*(G')$. □

Proposition 8 *Consider a graph $G = (V, E, f, w)$ and its edge $e = \{u, v\}$, s.t., $f(u) + w(e) > K$ for a constant K. If $\Delta^*(G) \leq K$, then $(v, u) \in OPT_G$ and also $\Delta^*(G) = \Delta^*(G')$ for the subgraph $G' = (V, E', f', w')$, where $E' = E \setminus \{e\}$, $f'(v) = f(v) + w(e)$, $f'(x) = f(x)$ for all $x \in V' \setminus \{v\}$, and $w'(e) = w(e)$ for all $e \in E'$.* □

3.2 Decision Problem (S, T)-MMO(K)

In this section, we consider a decision version (S, T)-MMO(K) of (S, T)-MMO and present a polynomial time algorithm to solve it, which is the main part of the algorithm to solve $\{1, \ldots, k\}$-MMO for cactuses.

Problem: (S, T)-MMO(K)

Input: An undirected graph $G = (V, E, f, w)$, where f is a node weight function $f : V \to T$, and w is an edge weight function $w : E \to S$.

Question: Is there an orientation Λ such that $\Delta_\Lambda(G) \leq K$?

Remind that any orientation has cost at least the maximum edge weight $w_{\max}$, so it is assumed to be $K \geq w_{\max}$. Again, $(S, 0)$-MMO(K) is considered as a decision version of S-MMO.

We first introduce three procedures `OutAll`, `FixEdge`, and `OrientCycle`, which are used in the proposed algorithm `AlgCactus`. The first procedure `OutAll`(G, Λ, v) (Fig. 4) determines orientations for all edges connecting to a node v, and then remove v and the edges from the (current) graph G. The second procedure `FixEdge`(G, K, Λ, e) (Fig. 5) determines an orientation for an edge e and then remove e from the (current) graph G, which is based on Proposition 8. The last procedure `OrientCycle`(G, Λ, C) (Fig. 6) determines an orientation for a cycle C having at most one gate. Fig. 7 shows a detailed description of the whole algorithm `AlgCactus`. The correctness and time complexity of the algorithm `AlgCactus` are shown in the two lemmas below in this section.

Fig. 8 depicts an example execution of `AlgCactus` for a graph (Fig. 8(a)) with $K = 3$, where nodes and edges drawn by dotted lines are removed and the numbers in boxes represent node weights greater than 0. First, `OutAll` is applied to the node s (Fig. 8(b)), so that there is no node and edge satisfying the condition (1) or (2) in `AlgCactus`. Next, `OrientCycle` is applied to the cycle C (Fig. 8(c)) in which the node t is the gate, and then the edge $\{t, u\}$ is processed by `FixEdge` (Fig. 8(d)). Eventually, we obtain an final (optimal) orientation by applying, say, `OutAll` to the node t of the graph in Fig. 8(d), and then `FixEdge` to the remaining edge.

Lemma 9 *The algorithm* `AlgCactus` *outputs correct answers for (S, T)-MMO(K).*

Proof: Let the final orientation constructed by `AlgCactus` be Λ_f that is constructed regardless of the outputs of `AlgCactus`, 'Yes' or 'No,' for the input graph. Note that orientations for some edges may not be determined in Λ_f when the algorithm outputs 'No.' The algorithm `AlgCactus` determines a part of the orientation Λ_f and constructs a subgraph by removing nodes and edges step by step. We prove that such a constructed subgraph is sufficient to be considered in order to obtain correct answers, i.e., all the nodes removed from the input graph have outdegree at most K under Λ_f, and the optimal cost of such a subgraph is at most that of the input graph.

(**Step 1:** `OutAll`) Let two graphs before and after an application of `OutAll` be G_1 and H_1, respectively. Also Λ_1 denotes the current orientation at the end of Step 1. By definition, `OutAll` does not change the value of $f(u)$ for any node u, and so H_1 is a proper subgraph of G_1. Therefore, $\Delta^*(G_1) \geq \Delta^*(H_1)$ holds from Proposition 7. Since $\Lambda_f \supseteq \Lambda_1$ and orientations for all the edges connecting to the node v that

Procedure `OutAll`(G, Λ, v)

Input: A graph $G = (V, E, f, w)$, a (partial) orientation Λ, and a node $v \in V$.

Step 1: Add (v, u) to Λ for all $\{u, v\} \in E$.

Step 2: Remove the node v and its connecting edges from G.

Figure 4: Procedure `OutAll`

Procedure `FixEdge`(G, K, Λ, e)

Input: A graph $G = (V, E, f, w)$, a constant K, a (partial) orientation Λ and an edge $e = \{u, v\} \in E$.

Step 1: If $f(u) + w(e) > K$, then add (v, u) to Λ, and set $f(v) = f(v) + w(e)$. Otherwise, add (u, v) to Λ, and set $f(u) = f(u) + w(e)$.

Step 2: Remove the edge e from G.

Figure 5: Procedure `FixEdge`

Procedure `OrientCycle`(G, Λ, C)

Input: A graph $G = (V, E, f, w)$, a (partial) orientation Λ, and a cycle $C = \langle v_1, v_2, \ldots, v_l, v_1 \rangle$ having at most one gate.

Step 1: If C has no gate, then

(a): Orient the edges of C in one direction along C, i.e., add (v_1, v_2), (v_2, v_3), $\ldots$, (v_l, v_1) to Λ.

Otherwise, i.e., C has exactly one gate, say, v_1, execute the following:

(b): If $w(\{v_1, v_2\}) < w(\{v_1, v_l\})$, then add (v_1, v_2), (v_2, v_3), $\ldots$, (v_l, v_1) to Λ and set $f(v_1) = f(v_1) + w(\{v_1, v_2\})$.

(c): Otherwise, add (v_1, v_l), (v_l, v_{l-1}), $\ldots$, (v_2, v_1) to Λ and set $f(v_1) = f(v_1) + w(\{v_1, v_l\})$.

Step 2: Remove all the nodes and edges of C except the gate from G.

Figure 6: Procedure `OrientCycle`

Algorithm `AlgCactus`(G, K)

Input: A cactus $G = (V, E, f, w)$ and a constant K.

Output: Yes (and an orientation Λ), or No .

Step 0: Set $\Lambda := \emptyset$, and $G' := G$.

Step 1: If there exists a node $u \in V(G')$, s.t.,

$$f(u) + \sum_{\{u,v\} \in E(G')} w(\{u, v\}) \leq K, \qquad (1)$$

then execute `OutAll`(G', Λ, u).

Step 2: If there exists an edge $e = \{u, v\} \in E(G')$, s.t.,

$$f(u) + w(e) > K, \qquad (2)$$

then execute `FixEdge`(G', K, Λ, e).

Step 3: Repeat Steps 1 and 2, until there is neither a node nor an edge satisfying the conditions (1) or (2).

Step 4: If $f(v) > K$ for some node $v \in V(G')$, then output 'No' and halt.

Step 5: Remove isolated nodes (if exist) from G'. If G' is empty, output 'Yes' (and Λ) and halt.

Step 6: Find a cycle C having at most one gate. Execute `OrientCycle`(G', Λ, C), and then return to Step 1.

Figure 7: Algorithm `AlgCactus`

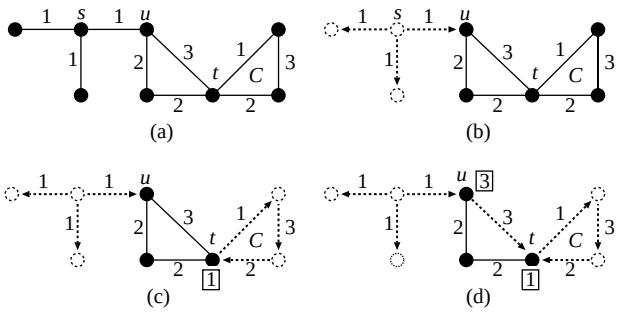

Figure 8: Example execution of `AlgCactus`: (a) Input graph, (b) application of `OutAll` to the node s, (c) application of `OrientCycle` to the cycle C, and (d) application of `FixEdge` to the edge $\{t, u\}$.

is removed by `OutAll` are already determined in Λ_1, $d^+_G(\Lambda_f, v) = d^+_G(\Lambda_1, v) \leq K$.

We can consider other orientations than Λ_1 in relation to the edges connecting to v, for example, one edge $\{v, x\}$ is oriented inward as (x, v). By such an orientation, we obtain a graph H'_1 in which $f(x)$ is equal to $f(x)$ in G_1 plus $w(\{x, v\})$, although $f(x)$ in H_1 equals to that in G_1. The difference between H_1 and H'_1 is only the weight $f(x)$, and from Proposition 5, $\Delta^*(H'_1) \geq \Delta^*(H_1)$ holds. Hence, if $\Delta^*(G_1) \leq K$, then $\Delta^*(H_1) \leq K$ also holds. (It may hold that $\Delta^*(H'_1) > \Delta^*(G_1)$) Therefore, it is sufficient to consider only H_1, to solve (S, T)-MMO(K).

(**Step 2:** `FixEdge`) Let the two graphs at the beginning and the end of Step 2 be $G_2 (= H_1$ above) and H_2, respectively. From Proposition 8, we can see that if $\Delta^*(G_2) \leq K$, then $\Delta^*(G_2) = \Delta^*(H_2)$. Note that no node is removed at Step 2 of `AlgCactus`. Therefore, again it is sufficient to consider only H_2, to solve (S, T)-MMO(K).

(**Step 3:** Repeating Steps 1 and 2) By repeating Steps 1 and 2, we finally obtain a graph H, which is a subgraph of the input graph G. From the above discussions on Steps 1 and 2, we observe that $\Delta^*(G) \geq \Delta^*(H)$. Therefore, since all the nodes al-

ready removed have outdegree at most K under the current orientation and also under the final orientation Λ_f, what we need to do is to consider the optimal cost for H to solve (S,T)-MMO(K).

(**Steps 4 and 5:** Halting criteria) If there exists a node v in H having $f(v) > K$, then it is apparent that $\Delta^*(H) > K$ and so $\Delta^*(G) > K$. Therefore we answer 'No.' The rest of the case is that every node v in H has $f(v) \leq K$. Even if an isolated node is removed, the (current) orientation Λ is not modified at all, and the outdegree of the removed node does not change under Λ_f. If the graph is turned to be empty after removing all the isolated nodes, its optimal cost is trivially zero. Namely, $\Delta^*(H) = \max_{v \in V(H)}\{f(v)\} \leq K$. Also since all orientations for all edges have already determined in Λ, Λ is the final orientation Λ_f. In addition to that the removed nodes at Steps 1 have outdegree at most K under $\Lambda_f (= \Lambda)$ as mentioned above. Therefore we can conclude that the answer is 'Yes.'

(**Step 6:** `OrientCycle`) G_6 and H_6 denote the two graphs at the beginning and the end of Step 6, respectively. All the nodes have degree at least 2 in G_6, because, otherwise a contradiction occurs: All isolated nodes, that is, the nodes having degree 0 are removed in Step. 5. Suppose that there exists a node u in G_6 such that $d_{G_6}(u) = 1$, and let the edge connecting to u be e. Since the degree of u is one, $f(u) + \sum_{\{u,v\} \in E(G_6)} w(\{u,v\}) = f(u) + w(e)$ holds, which means that either of the conditions (1) and (2) is always satisfied. However, this contradicts that the fact that G_6 is obtained after repeatedly applying Steps 1 and 2 until there does not exist such a node (Step 3). From this observation and Proposition 2, there always exists a cycle C having at most one gate. Let a cycle with at most one gate be $C = \langle v_1, \ldots, v_l, v_1 \rangle$ $(l \geq 2)$.

Case (a): C has no gate. In this case, Step. 1(a) of `OrientCycle` is applied to the cycle C. Since there is no node in C satisfying the condition (2), every node in C has outdegree at most K under the orientation determined in Step. 1(a) of `OrientCycle` and also under the final orientation Λ_f. Since C has no gate, C is a maximal connected component, so that C and H_6 does not share any nodes and hence $\Delta^*(G_6) \geq \Delta^*(H_6)$. [End of Case(a)]

Case (b): C has exactly one gate. Let the gate be v_1 without loss of generality, and suppose $w(\{v_1, v_2\}) \leq w(\{v_1, v_l\})$ (The discussion for the case $w(\{v_1, v_2\}) > w(\{v_1, v_l\})$ is similar). In this case, Step. 1(b) of `OrientCycle` is applied to C.

Consider a node $v \in \{v_2, \ldots, v_l\}$. Since v is not a gate, $d(v) = 2$ holds. Let the two edges connecting to v be $e_1 = \{u, v\}$ and $e_2 = \{t, v\}$. For v and e_1, e_2, neither conditions (1) nor (2) does not hold. Hence, if the optimal cost is at most K, we cannot orient e_1 and e_2 as (v, u) and (v, t) at the same time by the condition (1) in order to obtain an orientation whose cost is at most K. Also both of $f(v) + w(e_1)$ and $f(v) + w(e_2)$ are at most K by the condition (2). This situation is true for all the nodes in C except the gate v_1. Therefore, under the orientation Λ_f, the outdegree of every node in C except v_1 is at most K.

There are two other possibilities for the orientation of C in order to construct a final orientation whose cost is at most K: (I) $\Lambda_I \supseteq \{(v_1, v_l), (v_l, v_{l-1}), \ldots, (v_2, v_1)\}$, which is in the reverse direction of that by `OrientCycle`, and (II) $\Lambda_{II} \supseteq \{(v_1, v_2), (v_2, v_3), \ldots, (v_{i-1}, v_i), (v_{i+1}, v_i), (v_{i+2}, v_{i+1}), \ldots, (v_l, v_{l-1}), (v_1, v_l)\}$ for some $i \neq 1$, in which both of the two edges connecting to the gate v_1 in C are oriented outward. By the conditions (1) and (2), another orientation has the cost greater than K. Let $H_6^{(I)}$ and $H_6^{(II)}$ denote the subgraphs that can be obtained by those orientations Λ_I and Λ_{II}, respectively, i.e., by removing the nodes in C except the gate v_1, and increasing $f(v_1)$ by $w(\{v_1, v_l\})$ and $w(\{v_1, v_2\}) + w(\{v_1, v_l\})$, respectively. From Proposition 5, $\Delta^*(H_6^{(I)}) \geq \Delta^*(H_6)$ and $\Delta^*(H_6^{(II)}) \geq \Delta^*(H_6)$ hold, since $f(v_1)$ in H_6 is smaller than those in $H_6^{(I)}$ and $H_6^{(II)}$. Therefore, it is sufficient to consider the graph H_6 to solve (S,T)-MMO(K).

[End of Case (b)]

From the above discussions, by Step 6, the removed nodes have outdegree at most K under Λ_f and for the resulted graph H_6, $\Delta^*(G_6) \geq \Delta^*(H_6)$ holds. In conjunction with the discussions above, the nodes removed so far at Steps 1, 2, 5 and 6 have outdegree at most K under the orientation Λ_f, and also $\Delta^*(G) \geq \Delta^*(H_6)$ holds. Then, in order to solve (S,T)-MMO(K) for the input graph G, what we need to do in the rest is to solve (S,T)-MMO(K) for the graph H_6 by returning to Step 1. □

The following proposition gives the time complexity of the algorithm `AlgCactus`.

Proposition 10 `AlgCactus` *runs in* $O(|E|^2)$ *time.*

Proof: At Steps 1, 2, and 6, orientation of at least one edge is determined. Therefore the total number of processing those steps, and thus Steps 3, 4, and 5 also, are bounded above by $O(|E|)$. Since each step can be done by scanning nodes and edges in $O(|E|)$ time, the total running time is $O(|E|^2)$. □

Although we omit the proof, the running time of `AlgCactus` can be reduced with a careful preprocessing:

Lemma 11 *The algorithm* `AlgCactus` *runs in* $O(|E|)$ *time with preprocessing done in* $O(|V| \log |V|)$ *time.* □

3.3 Polynomial Time Algorithm

In this section, we show that $\{1, \ldots, k\}$-MMO is solvable in polynomial time by proving an upper bound of optimal costs of orientations for cactuses:

Lemma 12 *For any cactus* G, $\Delta^*(G) \leq f_{\max} + 2w_{\max}$ *for* (S,T)-MMO*, where* $f_{\max}$ *and* $w_{\max}$ *are the maximum weights of nodes and edges, respectively.*

Proof: The proof is constructive. First we apply Steps 0 through 5 of `AlgCactus` except for Step 4 to G with $K = f_{\max} + 2w_{\max}$, by which the removed nodes have outdegree at most $f_{\max} + 2w_{\max}$ under the final orientation, and remaining nodes have outdegree 0 under the current orientation at the end of Step 5. Then we modify Step 6 of `AlgCactus` as follows and apply it.

> **Step 6':** Find a cycle $C = \langle v_1, v_2, \ldots, v_l, v_1 \rangle$ having at most one gate.
>
> - If C does not have a gate, add $(v_1, v_2), (v_2, v_3), \ldots, (v_l, v_1)$ to Λ.
> - Otherwise, i.e., C has exactly one gate, say, v_l. Add (v_1, v_l) and (v_1, v_2), (v_2, v_3), …, (v_{l-1}, v_l) to Λ.
>
> Then remove C except the gate and return to Step 1.

By this modified Step 6', we observe that

- Every remaining node v has outdegree $f(v)$ under the current orientation at the end of Step 6'.
- If C has the gate v_l, v_1 has outdegree at most $f(v_1) + 2w_{\max}$ under the final orientation.
- The nodes removed at Step 6' except v_1 and the gate v_l have outdegree at most $f_{\max}+w_{\max}$ under the final orientation.

Repeating the procedures, all the nodes are removed from the graph at last, and all the removed nodes have outdegree at most $f_{\max}+2w_{\max}$ under the final orientation. Therefore, $\Delta^*(G) \leq f_{\max} + 2w_{\max}$ holds. □

Remind that we assume that node and edge weight functions f and w are integral functions in this paper. Therefore, we can obtain optimal orientations for (S,T)-MMO by solving $O(\log(f_{max} + w_{max}))$ times the (S,T)-MMO(K) in a binary search manner on K for $w_{\max} \leq K \leq f_{\max} + 2w_{\max}$ from the above lemma. Based on the Lemma 11, (S,T)-MMO is solvable in polynomial time $O(|V|\log|V|+|E|\log(f_{\max}+w_{\max}))$ for cactuses (Note that the preprocessing for `AlgCactus` has to be done only once). In a straightforward way, we obtain the following theorem for $\{1,\ldots,k\}$-MMO ($\equiv (\{1,\ldots,k\},0)$-MMO):

Theorem 13 *$\{1,\ldots,k\}$-MMO is solvable in polynomial time $O(|V|\log|V| + |E|\log k)$ for cactuses.* □

4 Pseudo-polynomial Time Algorithm for Series-Parallel Graphs

In this section, we describe the main idea of a pseudo-polynomial time algorithm solving $\{1,\ldots,k\}$-MMO for series-parallel graphs. The algorithm is a dynamic programming-based one, which utilizes a decomposition tree(Valdes, Tarjan, & Lawler 1982) defined by the series, parallel and jackknife operations. It is known that determining whether a given graph $G=(V,E)$ is a series-parallel graph can be done in linear time (Wimer & Hedetniemi 1988, Borie, Parker & Tovey 2002). Moreover, we can also obtain a decomposition tree T of G in linear time if G is a series-parallel graph.

For an arbitrary series-parallel graph (G,l,r), where l and r are left and right terminals, respectively, and two values $w_l \in \{0,1,\ldots,w_G(l) \stackrel{\text{def}}{=} \sum_{\{l,u\}\in E(G)} w(\{l,u\})\}$ and $w_r \in \{0,1,\ldots,w_G(r) \stackrel{\text{def}}{=} \sum_{\{r,u\}\in E(G)} w(\{r,u\})\}$, we define

$$WSP(G,l,r,w_l,w_r) = \min_{\Lambda} \max_{v\in V(G)} \left\{ d^+_G(\Lambda,v) \;\middle|\; \begin{array}{l} d^+_G(\Lambda,l)=w_l, \\ d^+_G(\Lambda,r)=w_r \end{array} \right\},$$

where Λ is an orientation for G.

In a decomposition tree, let us assume that a (sub)tree T_a is composed from its subtrees T_b and T_c by an operation series, parallel, or jackknife, where T_a, T_b and T_c correspond to (G_a,l_a,r_a), (G_b,l_b,r_b) and (G_c,l_c,r_c), respectively. Roughly speaking, for series, parallel, and jackknife operations, the following equations (3), (4), and (5) hold, respectively:

$$WSP(G_a,l_a,r_a,w_l,w_r) = \min_{w_b,w_c} \max \left\{ \begin{array}{l} WSP(G_b,l_b,r_b,w_l,w_b), \\ WSP(G_c,l_c,r_c,w_c,w_r), \\ w_b+w_c \end{array} \right\} \quad (3)$$

Algorithm `AlgSP`(G)

Input: A series-parallel graph $G=(V,E,w)$.

Output: $\Delta^*(G)$.

Step 0: Construct a decomposition tree T for G, and let l and r be two terminals of G.

Step 1: For all $w_l = 0,1,\ldots,w_G(l)$ and $w_r = 0,1,\ldots,w_G(r)$, compute $WSP(G,l,r,w_l,w_r)$ in a recursive manner by equations (3), (4) and (5).

Step 2: Output $\min_{w_l,w_r} WSP(G,l,r,w_l,w_r)$.

Figure 9: Algorithm `AlgSP`

$$WSP(G_a,l_a,r_a,w_l,w_r) = \min_{\substack{w_{bl}+w_{cl}=w_l,\\ w_{br}+w_{cr}=w_r}} \max \left\{ \begin{array}{l} WSP(G_b,l_b,r_b,w_{bl},w_{br}), \\ WSP(G_c,l_c,r_c,w_{cl},w_{cr}), \\ w_l, w_r \end{array} \right\} \quad (4)$$

$$WSP(G_a,l_a,r_a,w_l,w_r) = \min_{\substack{w_{br}+w_{cl}=w_r,\\ w_{cr}}} \max \left\{ \begin{array}{l} WSP(G_b,l_b,r_b,w_l,w_{br}), \\ WSP(G_c,l_c,r_c,w_{cl},w_{cr}), \\ w_r(=w_{br}+w_{cl}) \end{array} \right\} \quad (5)$$

The above equations (3), (4) and (5) show a principle of optimality, which yields an algorithm based on the dynamic programming. Fig. 9 shows the algorithm. Now we discuss the time complexity of `AlgSP`. As mentioned above, Step 0 is done in $O(|E|)$ time. In Step 1, we keep $w_G(l)\times w_G(r)$ WSP values for each (G,l,r), and if we have all WSP values for its two children, the evaluation of equations (3), (4) and (5) can be done in $w_{G_b}(r_b)\times w_{G_c}(l_c)$, $w_{G_a}(l_a)\times w_{G_a}(r_a)$ and $w_{G_a}(r_a)\times w_{G_c}(r_c)$ time, respectively. All of these are bounded by $k^2|V|^2$. The number of recursions is at most $|E|$, so this step is done in $O(|E|k^2|V|^2)$. Step 2 can be done also in $O(k^2|V|^2)$ time. Therefore the total running time of `AlgSP` is $O(k^2|E||V|^2)$, which is pseudo-polynomial for the input size. More details will appear in journal version.

Theorem 14 *$\{1,\ldots,k\}$-MMO is solvable in pseudo-polynomial time $O(k^2|E||V|^2)$ for series-parallel graphs.* □

5 $\mathcal{NP}$-hardness

In this section, we show the $\mathcal{NP}$-hardness of $\{1,\ldots,k\}$-MMO for restricted graph classes: outerplanar, series-parallel, planar, bipartite, P_4-bipartite, diamond-free, and house-free. We again note that outerplanar, P_4-bipartite, diamond-free, and house-free are minimal superclasses of cactus (Brandstädt, BangLe, & Spinrad 1987). The following theorem shows the weak $\mathcal{NP}$-hardness of $\{1,\ldots,k\}$-MMO for (multi) outerplanar graphs, but its proof is quite easy.

Theorem 15 *$\{1,\ldots,k\}$-MMO is weakly $\mathcal{NP}$-hard for (multi) outerplanar graphs.*

Proof: The proof is by a polynomial time reduction from the weakly $\mathcal{NP}$-hard problem PARTITION

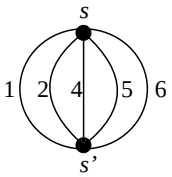

Figure 10: Proof of Theorem 15.

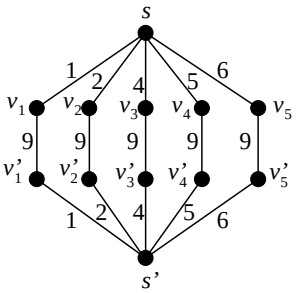

Figure 11: Proof of Theorem 16.

([SP12] on p.223 of (Garey & Johnson 1979)): *Given a set* $S = \{s_1, s_2, \ldots, s_n\}$ *of* n *positive integers, determine if there exists a subset* $S' \subseteq S$ *such that* $\sum_{s_i \in S'} s_i = \sum_{s_i \in S \setminus S'} s_i$.

We construct an edge weighted graph $G = (V, E, w)$ from an instance of PARTITION. Let the instance of PARTITION be $S = \{s_1, s_2, \ldots, s_n\}$. The node set V consists of two nodes, $V = \{s, s'\}$. The edge set E contains n *multiple* edges $e_1, e_2, \ldots, e_n$ connecting between the nodes s and s', where the weight of each edge e_i is equal to s_i, i.e., $w(e_i) = s_i$. The graph G is clearly outerplanar. Let us define $W = \sum_{s_i \in S} s_i/2$. This reduction is obviously done in polynomial time. See Fig. 10 for an example of the case $S = \{1, 2, 4, 5, 6\}$.

We consider that the situation $s_i \in S'$ (or $s_i \notin S'$) corresponds to orient the edge e_i from s to s' (or s' to s) in G. If there is a set $S' \subseteq S$ such that $\sum_{s_i \in S'} s_i = \sum_{s_i \in S \setminus S'} s_i = W$, then both of the outdegrees of s and s' in G is equal to W under the corresponding orientation, which is an optimal orientation. Otherwise, either of them has outdegree greater than W. □

The $\mathcal{NP}$-hardness of $\{1, \ldots, k\}$-MMO for series-parallel graphs is again proved by a reduction from PARTITION. Since the constructed graph in the above proof is also a series-parallel graph, the $\mathcal{NP}$-hardness for series-parallel graphs also holds straightforward. However, the constructed graph in the above proof is a multigraph, and thus, the $\mathcal{NP}$-hardness has been proved only for multigraphs. The objective of the following theorem is to show the $\mathcal{NP}$-hardness for simple graphs; however it is not applicable to outerplanar graphs.

Theorem 16 $\{1, \ldots, k\}$-MMO *is weakly* $\mathcal{NP}$*-hard for series-parallel graphs.*

Proof: From an instance $S = \{s_1, s_2, \ldots, s_n\}$ of PARTITION, we construct an edge weighted graph $G = (V, E, w)$. The node set V is divided into two types: (i) Subset nodes s and s', and (ii) Item nodes v_i and v'_i for each s_i. The total number of nodes is $2n + 2$. Let us define $W = \sum_{s_i \in S} s_i/2$. The edge set E contains $3n$ edges, $\{s, v_i\}$'s, $\{v_i, v'_i\}$'s and $\{v'_i, s'\}$'s for $1 \leq i \leq n$. As for the weights of the edges, $w(\{s, v_i\}) = w(\{v'_i, s'\}) = s_i$, and $w(\{v_i, v'_i\}) = W$ for $1 \leq i \leq n$. This reduction is done in polynomial time. See Fig. 11 for an example of the case $S = \{1, 2, 4, 5, 6\}$ again.

We prove that there is an orientation Λ of G such that $\Delta_\Lambda(G) \leq W$ if and only if there exists a set $S' \subseteq S$ such that $\sum_{s_i \in S'} s_i = W$ in the following.

(If part) Suppose that there exists a subset S' such that $\sum_{s_i \in S'} s_i = W$. Consider the following orientation Λ according to S': If $s_i \in S'$, then (s, v_i), (v_i, v'_i), and (v'_i, s') are in Λ; otherwise (s', v'_i), (v'_i, v_i), and (v_i, s) are in Λ. Under this orientation Λ, $d^+(\Lambda, s) = d^+(\Lambda, s') = W$. Also, if $s_i \in S'$, then $d^+(\Lambda, v_i) = W$ and $d^+(\Lambda, v'_i) = s_i$ hold; otherwise $d^+(\Lambda, v_i) = s_i$ and $d^+(\Lambda, v'_i) = W$ hold. Therefore, since all the nodes have outdegree at most W under Λ, $\Delta_\Lambda(G) \leq W$ holds.

(Only If part) This part is shown by proving that if there exists an orientation Λ of G such that $\Delta_\Lambda(G) \leq W$, there exists a subset $S' \subseteq S$ such that $\sum_{s_i \in S'} s_i = W$. Suppose that such an orientation Λ exists.

Since $w(\{v_i, v'_i\}) = W$ for all i's, we observe that either of v_i and v'_i has outdegree at least W under any orientation. If $(v_i, v'_i) \in \Lambda$, (s, v_i) is also in Λ, because, otherwise $d^+(\Lambda, v_i) > W$ that contradicts the assumption $\Delta_\Lambda(G) \leq W$. Similarly, if $(v'_i, v_i) \in \Lambda$, then $(s', v'_i) \in \Lambda$, too. Let J denote the set of indices i's such that $(s, v_i) \in \Lambda$. The outdegree of s under Λ is $d^+(\Lambda, s) = \sum_{i \in J} w(\{s, v_i\}) = \sum_{i \in J} s_i$. For an index $j \notin J$, the edge $\{s, v_j\}$ is oriented as $(v_j, s) \in \Lambda$, and thus the edge $\{v_j, v'_j\}$ is oriented as (v'_j, v_j), because, otherwise the outdegree of v_j is greater than W. Since $(v'_j, v_j) \in \Lambda$, (s', v'_j) is also in Λ. Then, it holds that $d^+(\Lambda, s') \geq \sum_{i \notin J} w(\{s', v'_i\}) = \sum_{i \notin J} s_i = 2W - d^+(\Lambda, s)$. Since $\Delta_\Lambda(G) \leq W$, both of $d^+(\Lambda, s) \leq W$ and $d^+(\Lambda, s') \leq W$ must hold, by which we have $d^+(\Lambda, s) = d^+(\Lambda, s') = W$ and then $\sum_{i \in J} s_i = W$. □

Now we go to the strong $\mathcal{NP}$-hardness proofs. We show that $\{1, k\}$-MMO for bipartite or planar graphs is $\mathcal{NP}$-hard. Both proofs are based on polynomial time reductions from variants of SAT problem: *Given a set* $U = \{x_1, \ldots, x_n\}$ *of Boolean variables and a CNF formula* $\phi = \bigwedge_{c_i \in C} c_i$*, where* C *is a set of clauses over* U*, determine if there exists a truth assignment for* ϕ.

Before explaining the reduction, we introduce several variants of SAT. At-Most-3SAT(2L) is a restriction of SAT where each clause includes at most three literals and each literal (not variable) appears at most twice in a formula. It can be easily proved that At-Most-3SAT(2L) is $\mathcal{NP}$-complete by using problem [LO1] on p. 259 of (Garey & Johnson 1979).

We call a CNF formula *planar* if graph $G(\phi) = (V, E)$, where $V = U \cup C$ and E contains exactly edges $\{x, c\}$ such that either x or $\overline{x}$ belongs to the clause c for $x \in U$ and $c \in C$. It is known that Planar SAT (or 3SAT), where an input CNF is restricted to be planar, remains $\mathcal{NP}$-complete (Lichtenstein 1982).

We call a CNF formula *monotone* if each clause contains either only negative literals or only positive literals, and it is known that Monotone SAT, where an input CNF is restricted to be monotone, remains $\mathcal{NP}$-complete (Gold 1978) (Also see [LO2] on p.259 of (Garey & Johnson 1979)). Monotone At-Most-3SAT(2L) is a restriction of Monotone SAT where each clause includes at most three literals and each literal (not variable) appears at most twice in a for-

mula. We can see that Monotone At-Most-3SAT(2L) is also $\mathcal{NP}$-complete by the following reduction: Let $c_i = \bigvee_{x_a \in P_i} x_a \vee \bigvee_{x_b \in N_i} \overline{x_b}$ be a clause of an arbitrary At-Most-3SAT(2L) instance ϕ, where P_i (resp., N_i) is the set of positive (resp., negative) literals in c_i. We define a new variable x_{c_i} for each c_i. Then a new monotone formula $\phi' = \bigwedge_{c_i \in C} \left(x_{c_i} \vee \bigvee_{x_a \in P_i} x_a\right) \wedge \left(\overline{x_{c_i}} \vee \bigvee_{x_b \in N_i} \overline{x_b}\right)$ has a truth assignment if and only if ϕ has a truth assignment. Furthermore, ϕ' is still an instance of At-Most-3SAT(2L) because the numbers of appearances of original literals are same as ϕ and new literals x_{c_i}'s and $\overline{x_{c_i}}$'s appear exactly once for each. Hence Monotone At-Most-3SAT(2L) is (strongly) $\mathcal{NP}$-complete.

We first show the strong $\mathcal{NP}$-hardness of $\{1,k\}$-MMO for bipartite graphs.

Theorem 17 *For any integer $k \geq 2$, $\{1,k\}$-MMO is strongly $\mathcal{NP}$-hard for bipartite graphs.*

Sketch of Proof: We only give a polynomial time reduction from Monotone At-Most-3SAT(2L), and omit the proof of its correctness. Suppose that a formula ϕ of Monotone At-Most-3SAT(2L) with n variables $\{x_1, \ldots, x_n\}$ and m clauses $\{c_1, \ldots, c_m\}$ is given. We call a clause *positive* (resp., *negative*) if it contains only positive (resp., negative) literals. For ϕ, we construct a graph G_ϕ including two gadgets that mimic (a) literals and (b) clauses, and also (c) a special gadget. (a) Each literal gadget consists of two nodes labeled by x_i and $\overline{x_i}$ and one edge $\{x_i, \overline{x_i}\}$ between them, corresponding to variable x_i of ϕ. The weight of $\{x_i, \overline{x_i}\}$ is k. (b) Each clause gadget is one node labeled by c_j, corresponding to clause c_j of ϕ. The clause gadget c_j is connected to at most three nodes in the literal gadgets that have the same labels as the literals in the clause c_j, by edges of weight 1. For example, if $c_1 = x \vee y \vee z$ is appeared in ϕ, then node c_1 is connected to nodes x, y and z. (See Fig. 12.) (c) The special gadget is a cycle of $2k$ nodes, say $s_1, s_2, \ldots, s_{2k}$, and $2k$ edges where each edge of the cycle has weight k. If a positive (resp., negative) clause consists of i variable(s), then it is connected to nodes $s_1, s_3, \ldots, s_{2(k-i)-1}$ (resp., $s_2, s_4, \ldots, s_{2(k-i+1)}$) in the special gadget by edges of weight 1. Hence, the degree of every clause node is exactly $k+1$. Note that G_ϕ is bipartite, since nodes associated with positive (resp., negative) clauses are connected only to positive (negative) literal nodes or s_i nodes with odd (resp., even) i in the special gadget, and vice versa. Also, this construction can be done in polynomial time.

For this bipartite G_ϕ, we can show that the following holds: (i) If ϕ is satisfiable, $\Delta^*(G_\phi) \leq k$. (ii) If ϕ is not satisfiable, $\Delta^*(G_\phi) \geq k+1$ (The detailed proof is omitted). □

By the proof of Theorem 17, we obtain the following corollary.

Corollary 18 *Even for bipartite graphs, $\{1,k\}$-MMO has no pseudo-polynomial time algorithm whose approximation ratio is smaller than $1 + 1/k$ unless $\mathcal{P} = \mathcal{NP}$.* □

Since neither the graph house nor diamond is bipartite, a bipartite graph is also a house-free and diamond-free graph. Also a bipartite graph is a P_4-bipartite graph by definition, we obtain the following corollary, too.

Corollary 19 $\{1,k\}$-MMO *is strongly $\mathcal{NP}$-hard for P_4-bipartite, house-free, and diamond-free. Moreover, for these graph classes, $\{1,k\}$-MMO has no pseudo-polynomial time algorithm whose approximation ratio is smaller than $1 + 1/k$ unless $\mathcal{P} = \mathcal{NP}$.* □

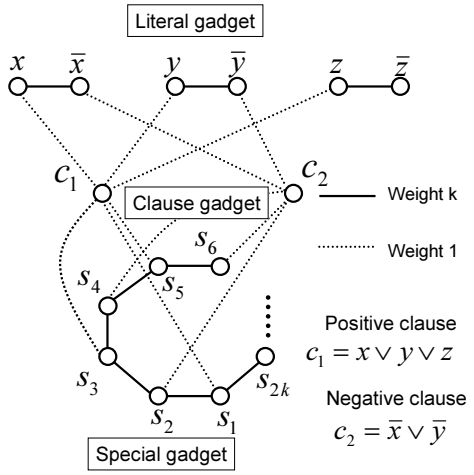

Figure 12: Proof of Theorem 17.

Next we show the strong $\mathcal{NP}$-hardness of $\{1,k\}$-MMO for planar graphs.

Theorem 20 *For any integer $k \geq 2$, $\{1,k\}$-MMO is strongly $\mathcal{NP}$-hard for planar graphs.*

Sketch of Proof: We use a reduction similar to that in Theorem 17. Instead of Monotone At-Most-3SAT, we use the reduction from Planar 3SAT. Again, we only show the reduction and proof is omitted.

Suppose Planar 3SAT instance ϕ and its planar drawing are given. For such an instance, we construct a graph G'_ϕ including gadgets associated with (a) variables and (b) clauses, and (c) special gadgets. (a) A variable gadget of x consists of $3l$ nodes and $3l$ edges, where l is the number of appearances of x in ϕ. For convenience, we assume that variable x appears in clauses $c_1, c_2, \ldots, c_l$, and in the given planar drawing c_i's are drawn in this order (Fig. 13, top). Then we prepare $2l$ nodes labeled by $x^{(i)}$ and $\overline{x^{(i)}}$, and l nodes labeled by $d^{(l)}$. This labeling corresponds to the ordering of c_i's. For these nodes, we put edges $\{x^{(i)}, \overline{x^{(i)}}\}$ with weight k, $\{\overline{x^{(i)}}, d^{(i)}\}$ with weight 1 and $\{d^{(i)}, \overline{x^{(i+1)}}\}$ with weight 1, for $i = 1, 2, \ldots, l$ ($l + 1 \equiv 1$). Note that a variable gadget itself is planar. (b) Each clause gadget is one node labeled by c_j, corresponding to clause c_j of ϕ (same as the proof of Theorem 17). We connect clause gadgets to nodes of variable gadgets as follows: Again, assume that a variable x appears in clauses $c_1, c_2, \ldots, c_l$. In the variable gadget of x, we prepared $2l$ nodes, $x^{(i)}, \overline{x^{(i)}}$ for $i = 1, \ldots, l$, whose numbering corresponds to the index of c_j's. Then, we connect edges according this numbering; if x (resp., $\bar{x}$) appears in c_j, then put edge $\{c_j, x^{(j)}\}$ (resp., $\{c_j, \overline{x^{(j)}}\}$) with weight 1 (Fig. 13, bottom). Since ϕ is 3CNF, c_j is connected to at most three nodes in variable gadgets. (c) A special gadget is a cycle of $k+1$ nodes and $k+1$ edges where each edge of the cycle has weight k. We prepare a special gadget for each clause gadget and for each node $d^{(i)}$ in a variable gadget. If a clause consists of one (two or three, resp.,) variable(s), then it is connected to k (arbitrary $k-1$ or $k-2$, resp.,) nodes in its special gadget by edges of weight 1. For each node $d^{(i)}$, it is connected to $k-1$ nodes in its own special gadget by

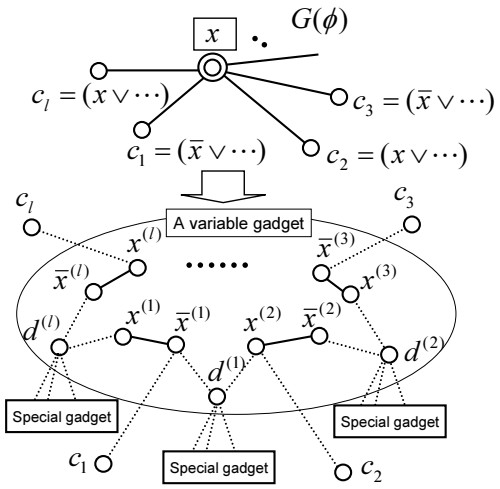

Figure 13: Proof of Theorem 20.

edges of weight 1. Hence, the degree of every clause node or every node $d^{(i)}$ is exactly $k+1$.

Note that G'_ϕ is planar because we can consider G'_ϕ is obtained by replacing each variable node of planar $G(\phi)$ with the corresponding variable gadget, which does not violate its planarity.

We can say that (i) If ϕ is satisfiable, $\Delta^*(G'_\phi) \leq k$. (ii) If ϕ is not satisfiable, $\Delta^*(G'_\phi) \geq k+1$. (The detailed proof is omitted.) □

By the proof of Theorem 20, again we obtain the following corollary.

Corollary 21 *Even for planar graphs, $\{1,k\}$-MMO has no pseudo-polynomial time algorithm whose approximation ratio is smaller than $1+1/k$ unless $\mathcal{P} = \mathcal{NP}$.* □

6 Conclusion

We have discussed about the complexity of MMO for several graph classes. The results are shown in Figure 2. Except others, we would like to note here about outerplanar graphs. In this paper, we show the weak $\mathcal{NP}$-hardness for "multi" outerplanar graphs, however the complexity for "simple" outerplanar graphs is still unknown. Since we have developed a pseudo-polynomial time algorithm for series-parallel graphs, the complexity of MMO for "simple" outerplanar graphs is either $\mathcal{P}$ or weakly $\mathcal{NP}$-hard, which is one of the further research topics.

References

Asahiro, Y., Jansson, J., Miyano, E., Ono, H., & Zenmyo, K. (2007), Approximation algorithms for the graph orientation minimizing the maximum Weighted outdegree *in* 'Proc. 3rd International Conference on Algorithmic Aspects in Information and Management, Lecture Notes in Computer Science', Vol. 4508, pp. 167–177.

Asahiro, Y., Miyano, E., Ono, H., & Zenmyo, K. (2007), 'Graph orientation algorithms to minimize the maximum outdegree', *International Journal of Foundations of Computer Science*, **18**(2), pp. 197–215.

Borie, R., Parker, R., & Tovey, C. (2002), 'Solving problems on recursively constructed graphs', *Technical Report TR-2002-04, Dept. Comp. Sci., University of Alabama.*

Brandstädt, A., BangLe, V., & Spinrad, J. P. (1987), *Graph Classes: A Survey*, SIAM.

Brodal, G. S., & Fagerberg, R. (1999), Dynamic representations of sparse graphs, *in* 'Proc. 6th Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science', Vol. 1663, pp. 342–351.

Chvátal, V. (1975), 'A combinatorial theorem in plane geometry', *J. Combinatorial Theory, series B*, **18**, pp. 39–41.

Gairing, M., Lücking, T., Mavronicolas, M., & Monien, B. (2004), Computing Nash equilibria for scheduling on restricted parallel links, *in* Proc. 36th ACM Symposium on Theory of Computing, pp. 613–622.

Garey, M., & Johnson, D. (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., 1979.

Gold, E. M. (1978), Complexity of automaton identification from given data, *Information and Control*, **37**(3), pp. 302–320.

Gross, J. L., & Yellen, J.(eds) (2004), *Handbook of Graph Theory*, CRC Press.

Hoàng, C.T., & Le, V.B. 'P_4-free colorings and P_4-bipartite graphs', *Discrete Mathematics and Theoretical Computer Science*, **4**, pp. 109–122.

Hopcroft, J.E., & Tarjan, R.E. (1974), 'Efficient planarity testing', *J. ACM*, **21**, pp. 549–568.

Kowalik, L. (2006), Approximation scheme for lowest outdegree orientation and graph density measures, *in* 'Proc. 17th International Symposium on Algorithms and Computation, Lecture Notes in Computer Science', Vol.4288, pp. 557–566.

Kloks, T., Kratsch, D., & Müller, H. (2000), 'Finding and counting small induced subgraphs efficiently,' *Information Processing Letters*, **74**(3-4), pp.115-121

Lenstra, J. K., Shmoys, D. B., & Tardos., É. (1990), 'Approximation algorithms for scheduling unrelated parallel machines', *Mathematical Programming*, **46**(3), 259–271, 1990.

Lichtenstein, D. (1982), 'Planar formulae and their uses', *SIAM Journal on Computing*, **11**(2), pp. 329–343.

Mitchell, S.L. (1979), 'Linear algorithms to recognize outerplanar and maximal outerplanar graphs', *Information Processing Letters*, **9**, pp. 229–232.

O'Rourke, J. (1987), *Art Gallery Theorems and Algorithms*, Oxford University Press.

Schuurman, P., & Woeginger, G. J. (1999), 'Polynomial time approximation algorithms for machine scheduling: Ten open problems,' *J. Scheduling*, **2**, pp. 203–213.

Venkateswaran, V. (2004), Minimizing maximum indegree, *Discrete Applied Mathematics*, **143**(1-3), pp. 374–378.

Valdes, J., Tarjan, R.E., & Lawler, E.L. (1982), 'The recognition of series-parallel digraphs' *SIAM J. Computing*, **11**, pp. 298–313.

Wimer, T.V. & Hedetniemi, S.T. (1988), 'K-terminal recursive families of graphs' *Congressue Numerantium*, **63**, pp. 161–176.

Generating Balanced Parentheses and Binary Trees by Prefix Shifts

Frank Ruskey[1] **Aaron Williams**[2]

Department of Computer Science
University of Victoria,
Victoria BC, V8W 3P6, Canada,
[1] URL: http://www.cs.uvic.ca/~ruskey
[2] Email: haron@uvic.ca

Abstract

We show that the set $\mathbf{B}_n$ of balanced parenthesis strings with n left and n right parentheses can be generated by prefix shifts. If $b_1, b_2, \ldots, b_{2n}$ is a member of $\mathbf{B}_n$, then the k-th prefix shift is the string $b_1, b_k, b_2, \ldots, b_{k-1}, b_{k+1}, \ldots, b_{2n}$. Prefix shift algorithms are also known for combinations, and permutations of a multiset; the combination algorithm appears in fascicles of Knuth vol 4. We show that the algorithm is closely related to the combination algorithm, and like it, has a loopless implementation, and a ranking algorithm that uses $O(n)$ arithmetic operations. Additionally, the algorithm can be directly translated to generate all binary trees by a loopless implementation that makes a constant number of pointer changes for each successively generated tree.

Keywords: Gray codes, Catalan numbers, balanced parentheses, binary trees, combinatorial generation, loopfree algorithm.

1 Introduction

Balanced parenthesis strings are one of the most important of the many discrete structures that are counted by the Catalan numbers, $C_n = \binom{2n}{n}/(n+1)$. The Catalan numbers and the objects counted by them are extensively discussed in Stanley (1999). The online supplement lists 149 distinct discrete structures counted by the Catalan numbers (Stanley (2007)).

Binary trees and ordered trees are also counted by the Catalan numbers; these tree structures are of paramount importance to computer scientists. There is a large number of papers dealing with the fundamental problem of exhaustively listing and ranking binary trees. In this paper we develop an algorithm that has a number of attractive and unique features as compared with existing algorithms.

Let $\mathbf{B}_{t,s}$ be the set of all bitstrings containing t 1s and s 0s and satisfying the constraint that the number of 1s in any prefix is at least as large as the number of 0s. For example, $\mathbf{B}_{3,2} = \{11100, 11010, 11001, 10110, 10101\}$. In particular, $\mathbf{B}_{t,s}$ is empty if $t < s$. Furthermore, if $t = s$ then $\mathbf{B}_{t,s}$ can be thought of as the set of all balanced parenthesis strings by mapping 1 to a left parenthesis and 0 to a right parenthesis. In this case, we sometimes drop the s from the notation; $\mathbf{B}_n = \mathbf{B}_{n,n}$.

If $b_1, b_2, \ldots, b_{t+s}$ is a member of $\mathbf{B}_{t,s}$, then the k-th *prefix shift* is the string $b_1, b_k, b_2, \ldots, b_{k-1}, b_{k+1}, \ldots, b_{t+s}$. Note that the first bit, b_1 is not part of this definition; this is natural since b_1 is always 1. Furthermore, it is impossible to generate $\mathbf{B}_{t,s}$ if b_1 is included in the shifts (e.g., 1^t0^s is the only valid shift of both $1^{t-1}0^s1$ and $1^{t-1}0^{s-1}10$). In order to entice the reader into reading further, below we show the simple iterative rule, whose successive application will generate $\mathbf{B}_{t,s}$ using prefix shifts.

Iterative successor rule: Locate the leftmost 01 and suppose that its 1 is in position k. If the $(k+1)$-st prefix shift is valid (a member of $\mathbf{B}_{t,s}$), then it is the successor; if it is not valid then the k-th prefix shift is the successor.

The only string without a 01 is 1^t0^s, which is the final string. The initial string is $101^{t-1}0^{s-1}$. Applying the rule to $\mathbf{B}_{3,2}$ gives the sequence 10110, 11010, 10101, 11001, 11100.

This is the first paper that considers whether balanced parentheses can be generated by prefix shifts. It is known that $\mathbf{B}_{t,s}$ can be generated by transposing a pair of bits (Ruskey & Proskurowski (1990)), a pair of bits with only 0s in between (Bultena & Ruskey (1998)), or by transposing one or two pairs of adjacent bits (Vajnovszki & Walsh (2006)). In general it is impossible to generate $\mathbf{B}_{t,s}$ by transposing only one pair of adjacent bits (Ruskey & Proskurowski (1990)). Our algorithm will be shown to generate $\mathbf{B}_{t,s}$ by transposing one or two pairs of bits, but those bits are not adjacent in general.

An algorithm for generating combinatorial objects is said to be *loopless* if only a constant amount of computation is used in transforming the current structure into its successor. Loopless algorithms are known for various classes of discrete structures that are counted by the Catalan numbers. See, for example, the papers Roelants (1991), Korsh, LaFolette, & Lipschutz (2003), Vajnovszki (1998), Vajnovszki & Walsh (2006) and Takaoka & Violich (2006).

Binary trees in their conventional representation of a node with two pointers can efficiently be generated by only making a constant number of pointer changes between successive trees (Lucas, Roelants, & Ruskey (1993)). This algorithm can be implemented loopless and is presented in Knuth (2006). The current paper gives the basis for another such algorithm.

The approach taken in this paper was initiated in the papers of Ruskey & Williams (2005, 2008) for generating combinations that are represented by bitstrings in the usual way. There the bitstrings are also generated by prefix shifts. It is remarkable how many of the results of those papers have close analogues with the results of the current paper. The ordering of combinations in (Ruskey & Williams 2005, 2008) was

called cool-lex order because of its close connection with the well-known colex order of combinations. In a similar spirit, we have dubbed our order "CoolCat" order because of its close connections with cool-lex order and with the Catalan numbers.

Relative to a list of objects, the *rank* of a particular object is the position that it occupies in the list, counting from zero.

To summarize, our method has the following properties:

1. Each successive string differs from its predecessor by the rotation of a prefix of the string. Furthermore, the list of strings is circular in the sense that the first and last also differ by a prefix rotation.
2. Each successive string differs from its predecessor by the interchange of one or two pairs of bits.
3. It has a simple recursive description. This description does not involve the reversal of sublist, as is usually the case for Gray codes. The underlying graph is a *directed* graph; that is, if $\mathbf{b}_1$ differs from $\mathbf{b}_2$ by a prefix rotation, then in general it is *not* the case that $\mathbf{b}_2$ differs from $\mathbf{b}_1$ by a prefix rotation.
4. It has a remarkably simple iterative successor rule. This rule was stated above.
5. The iterative successor rule can be implemented as a loopless algorithm. Also, the successor rule can be translated to a loopless algorithm for generating binary trees. No previous listing of balanced parenthesis strings is simultaneously a Gray code for the strings *and* for the corresponding binary trees.
6. It has a ranking algorithm that uses $O(n)$ arithmetic operations. No previous Gray code for balanced parenthesis strings has this property.

2 Generating Binary Trees

To give the reader a flavor of how useful the iterative successor rule is, in this section we translate the rule so that it applies to binary trees, as traditionally implemented on a computer. The result is a loopless algorithm that makes at most 16 pointer updates between successive trees. An implementation of this algorithm is available from the authors.

The standard bijection between $\mathbf{B}_{n,n}$ and extended binary trees with n internal nodes is to associate each internal node with a 1 and each leaf with a 0 and then do a preorder traversal of the tree, ignoring the final leaf. If z is a node in a binary tree, then we use $l(z)$ and $r(z)$ to denote the pointers to the left and right children of z. Unfortunately, we also need to maintain the parent of each internal node; this is denoted $p(z)$.

To update the tree we maintain three pointers: x, the first node that is not on the leftmost path of internal nodes; y, the parent of x; and R, the root of the tree. The assignments below represent parallel executions, so that, for example, $[a, b] \leftarrow [b, a]$ swaps the two values a and b. The algorithm terminates when x becomes nil.

According to the iterative successor rule there are three cases to consider: (a) the string is of the form $1^p0^q11\alpha$, (b) the string is of the form $1^p0^q10\alpha$, with $p > q$, and (c) the string is of the form $1^p0^p10\alpha$. Below we show the updates that are necessary in each of the three cases. Important note: The updates to the parent field are not shown explicitly below, but every time that an update is done to $r(.)$ or $l(.)$, then an update must be done to $p(.)$. E.g., if the update is $r(v) \leftarrow w$, then it should be followed with **if** $w \neq$ nil **then** $p(w) \leftarrow v$.

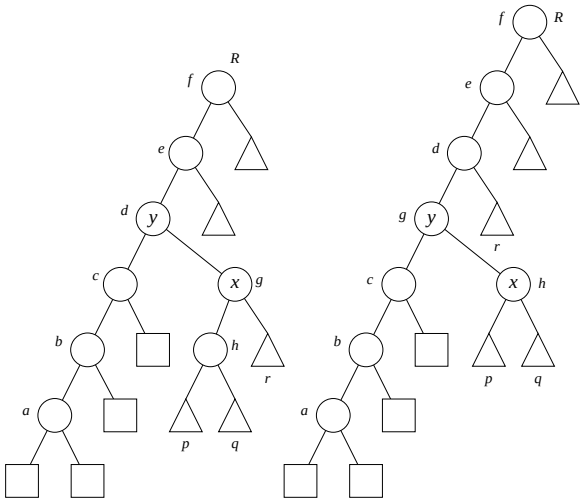

Figure 1: The trees corresponding to 111111000011... → 111111100001.... This is an example of Case (a).

Case (a): The new string is $1^{p+1}0^q1\alpha$. This case occurs when $l(x) \neq$ nil. The corresponding update to the binary tree is

$$[r(y), r(x), l(x), l(y)] \leftarrow [r(x), l(x), l(y), x]$$

$$y \leftarrow x; \quad x \leftarrow r(y);$$

Case (b): The new string is $101^{p-1}0^q1\alpha$. This case occurs when $l(x) =$ nil and $R \neq y$. The corresponding update to the binary tree is

$$[l(p(y)), r(p(y)), l(x), r(x), l(y), r(y)] \leftarrow$$
$$[l(y), x, r(x), r(p(y)), \text{nil}, R];$$

$$[R, x] \leftarrow [y, r(y)];$$

Case (c): the new string is $1^{p+1}0^{p+1}\alpha$. This case occurs when $l(x) =$ nil and $R = y$. The corresponding update to the binary tree is

$$[l(x), r(y)] \leftarrow [y, \text{nil}]; \ [R, y, x] \leftarrow [x, x, r(x)];$$

After this update the algorithm terminates if $x =$ nil (i.e., if α is the empty string).

These three cases are illustrated in Figures 1, 2, and 3. Circles are used for internal nodes, squares are used for leaves, and the triangles represent subtrees whose structure is not specified (but whose preorder order must be preserved).

3 Recursive Structure

In this section we examine the recursive structure of the CoolCat ordering on balanced parenthesis. In particular, we provide two recursive formulae and prove that they produce lists that are identical to those produced by the iterative rule. A corollary to this result is that the iterative rule generates every string in $\mathbf{B}_{t,s}$. For comparison purposes we also provide the recursive structure for co-lexicographic, or colex ordering. We begin this section by giving a formal definition of the iterative rule.

The CoolCat iterative rule maps a binary string $\mathbf{b} \in \mathbf{B}_{t,s}$ to another binary string $\sigma(\mathbf{b}) \in \mathbf{B}_{t,s}$. When $\mathbf{b}$ does not contain any 010 or 011 as a substring then

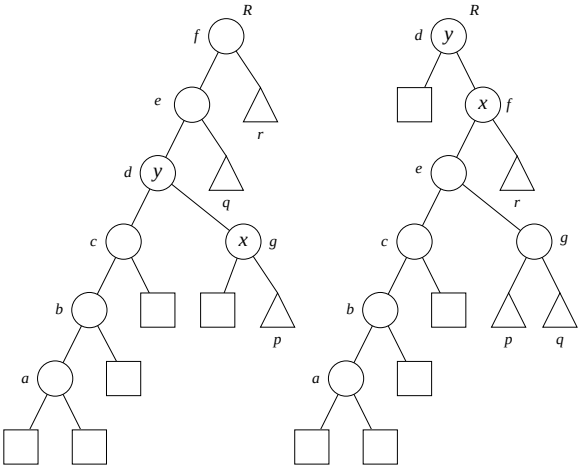

Figure 2: The trees corresponding to 111111000010... → 101111100001.... This is an example of Case (b).

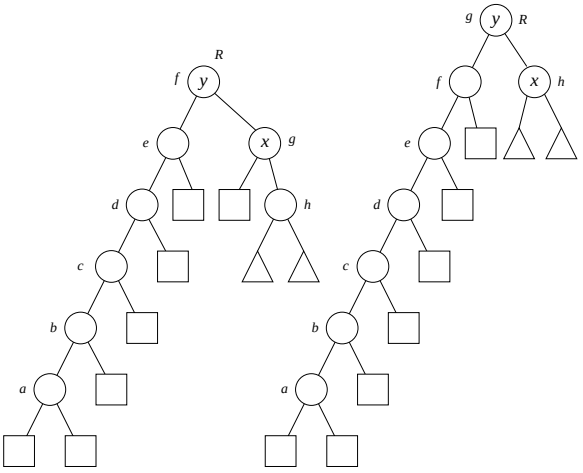

Figure 3: The trees corresponding to 111111000000101... → 111111100000001.... This is an example of Case (c).

it is easiest to define $\sigma(\mathbf{b})$ using the following two special cases, which simply move the rightmost symbol into the second leftmost position.

$$\sigma(\mathbf{b}) = \begin{cases} 101^i0^j & \text{if } \mathbf{b} = 11^i0^j0 \quad (1a) \\ 111^i0^j0 & \text{if } \mathbf{b} = 11^i0^j01 \quad (1b) \end{cases}$$

Otherwise, we can assume that $\mathbf{b} = 11^i00^j1z\mathbf{b}'$ for some symbol z and some (possibly empty) string $\mathbf{b}'$.

$$\sigma(\mathbf{b}) = \begin{cases} 111^i00^jz\mathbf{b}' & \text{if } i = j \quad (2a) \\ 1z1^i00^j1\mathbf{b}' & \text{if } i > j \quad (2b) \end{cases}$$

We inductively let $\sigma^0(\mathbf{b}) = \mathbf{b}$ and $\sigma^k(\mathbf{b}) = \sigma(\sigma^{k-1}(\mathbf{b}))$ for $k > 0$, so that we can define an iterative list $\mathbf{R}_{t,s}$ that uses σ.

$$\mathbf{R}_{t,s} = \mathbf{b},\ \sigma(\mathbf{b}),\ \sigma^2(\mathbf{b}),\ \ldots, \sigma^{k-1}(\mathbf{b}) \quad (3)$$

where $\mathbf{b} = 1^t0^s$ and $k = |\mathbf{B}_{t,s}|$. We'll also find it useful to start the iterative process at the successor of $\mathbf{b}$, and in fact our first recursive structure will equal this secondary listing. Instead of starting the iterative process at the successor of $\mathbf{b}$, this secondary listing can also be seen as the result of applying σ to each string in $\mathbf{R}_{t,s}$.

$$\begin{aligned} \mathbf{S}_{t,s} &= \sigma(\mathbf{b}),\ \sigma^2(\mathbf{b}),\ \ldots,\ \sigma^k(\mathbf{b}) \quad (4) \\ &= \sigma(\mathbf{R}_{t,s}) \quad (5) \end{aligned}$$

To better illustrate our first recursive formula, let us begin by examining the recursive structure of the colex list $\mathbf{L}_{4,4}$ and then comparing it to the CoolCat list $\mathbf{S}_{4,4}$. The term colex refers to the fact that the strings in $\mathbf{B}_{t,s}$ are in increasing lexicographic order when each string is read from right to left. The colex list $\mathbf{L}_{4,4}$ can be built recursively from the smaller lists $\mathbf{L}_{3,i}$ for $0 \le i \le 3$. Each of these lists appears as a column within Figure 4. Notice that in each column the suffixes beginning with 1 are underlined, and all of the strings with a given underlined suffix appear consecutively. In the case of $\mathbf{L}_{4,4}$ (where $t = s$) the suffixes beginning with 1 are 10000, 1000, 100, and 10. Notice that there is no suffix 1 since there is no string in $\mathbf{B}_{4,4}$ with that suffix. However, the suffix 1 does appear in $\mathbf{L}_{3,2}$ (where $t > s$) since there is a string with that suffix in $\mathbf{B}_{3,2}$. Finally, in each case the suffixes are ordered by decreasing number of zeros. In general each of these observations holds true, and it leads to the following recursive formula for $\mathbf{L}_{t,s}$

$$= \begin{cases} \mathbf{L}_{t-1,0}10^s, \mathbf{L}_{t-1,1}10^{s-1}, \ldots, \mathbf{L}_{t-1,s-1}10 & \text{if } t = s \\ \mathbf{L}_{t-1,0}10^s, \mathbf{L}_{t-1,1}10^{s-1}, \ldots, \mathbf{L}_{t-1,s}1 & \text{if } t > s. \end{cases}$$

To compact expressions of this kind we introduce $\prod$ to combine short lists of strings into larger lists, and we restate the recursive formula for $\mathbf{L}_{t,s}$ as follows

$$\mathbf{L}_{t,s} = \begin{cases} \prod_{i=0}^{s-1} \mathbf{L}_{t-1,i}10^{s-i} & \text{if } t = s \quad (6a) \\ \prod_{i=0}^{s} \mathbf{L}_{t-1,i}10^{s-i} & \text{if } t > s. \quad (6b) \end{cases}$$

Now we turn our attention to the recursive structure of $\mathbf{W}_{4,4}$ that is illustrated in Figure 5. As in colex the suffixes beginning with 1 are underlined and the strings with a given underlined suffix appear consecutively within each list. However, in this case the suffixes beginning with 1 are ordered by decreasing

$\mathbf{L}_{3,0}$	$\mathbf{L}_{3,1}$	$\mathbf{L}_{3,2}$	$\mathbf{L}_{3,3}$	$\mathbf{L}_{4,4}$
111	1110	11100	111000	11110000
	1101	11010	110100	11101000
	1011	10110	101100	11011000
		11001	110010	10111000
		10101	101010	11100100
				11010100
				10110100
				11001100
				10101100
				11100010
				11010010
				10110010
				11001010
				10101010

Figure 4: The recursive structure of colex.

$\mathbf{W}_{3,0}$	$\mathbf{W}_{3,1}$	$\mathbf{W}_{3,2}$	$\mathbf{W}_{3,3}$	$\mathbf{W}_{4,4}$
111	1011	10110	101100	10111000
	1101	11010	110100	11011000
	1110	10101	101010	11101000
		11001	110010	10110100
		11100	111000	11010100
				10101100
				11001100
				11100100
				10110010
				11010010
				10101010
				11001010
				11100010
				11110000

Figure 5: The first recursive structure of CoolCat.

number of zeros, *except* for the suffix 10^s that appears last instead of first. Of course, there is only a single string in $\mathbf{B}_{t,s}$ that has the suffix 10^s, namely 1^t0^s. Amazingly, the alternate placement of this single string fully captures the difference between the recursive structure of CoolCat and colex. We define the list $\mathbf{W}_{t,s}$ as follows, and we prove that it is equal to $\mathbf{S}_{t,s}$ in Theorem 1

$$\mathbf{W}_{t,s} = \begin{cases} \prod_{i=1}^{s-1} \mathbf{W}_{t-1,i}10^{s-i},\ 1^t0^s & \text{if } t = s \ \text{(7a)} \\ \prod_{i=1}^{s} \mathbf{W}_{t-1,i}10^{s-i},\ 1^t0^s & \text{if } t > s. \text{(7b)} \end{cases}$$

Since the recursive structure of $\mathbf{W}_{t,s}$ is a reordering of the strings in $\mathbf{L}_{t,s}$ we have the following remark.

Remark 1. *$\mathbf{W}_{t,s}$ contains each string in $\mathbf{B}_{t,s}$ exactly once.*

An important step towards proving Theorem 1 is the following lemma, that explicitly identifies the first and last strings that appear in $\mathbf{W}_{t,s}$ when $s > 0$.

Lemma 1. *For $s > 0$*

$$\begin{aligned} first(\mathbf{W}_{t,s}) &= 101^{t-1}0^{s-1} && (8) \\ last(\mathbf{W}_{t,s}) &= 1^t0^s. && (9) \end{aligned}$$

Proof. The value of $last(W_{t,s})$ follows immediately from (7). To determine the value of $first(\mathbf{W}_{t,s})$ we have the following

$$\begin{aligned} first(\mathbf{W}_{t,s}) &= first(\mathbf{W}_{t-1,1})10^{s-1} \\ &= first(\mathbf{W}_{t-2,1})110^{s-1} \\ &= first(\mathbf{W}_{t-3,1})1110^{s-1} \\ &= \ldots \\ &= first(\mathbf{W}_{1,1})1^{t-1}0^{s-1} \\ &= 101^{t-1}0^{s-1}. \end{aligned}$$

□

Now we are in a position to prove the main result of this section.

Theorem 1. $\mathbf{S}_{t,s} = \mathbf{W}_{t,s}$.

Proof. To prove the result we need to show that within $\mathbf{W}_{t,s}$ the first string in each sublist is obtained by applying σ to the last string of the previous sublist. The sublists in $\mathbf{W}_{t,s}$ are slightly different depending on whether $t = s$ (7a) or $t > s$ (7b), so we proceed in two cases. First we prove the result when $t > s$. For the last transition we have

$$\begin{aligned} \sigma(last(\mathbf{W}_{t-1,s}1)) &= \sigma(1^{t-1}0^s1) \\ &= 1^t0^s \end{aligned}$$

which follows from Lemma 1 and the definition of σ (1b). For the remaining transitions we have, for $1 \le i \le s-1$,

$$\begin{aligned} \sigma(last(\mathbf{W}_{t-1,s-i}10^i)) &= \sigma(1^{t-1}0^{s-i}10^i) \\ &= 101^{t-2}0^{s-i}10^{i-1} \\ &= first(\mathbf{W}_{t-1,s-i+1}10^{i-1}) \end{aligned}$$

which follows from Lemma 1 and the definition of σ (2b). In particular, (2b) applies here since $t > s$ and $i \ge 1$ imply that $t - 1 > s - i$.

Next we prove the result when $t = s$. For the last transition we have

$$\begin{aligned} \sigma(last(\mathbf{W}_{t-1,s-1}10)) &= \sigma(1^{t-1}0^{s-1}10) \\ &= 1^t0^s \end{aligned}$$

which follows from Lemma 1 and the definition of σ (2a). In particular, (2a) applies here since $t = s$. For the remaining cases we have, for $1 \le i \le s-2$,

$$\begin{aligned} \sigma(last(\mathbf{W}_{t-1,s-i}10^i)) &= \sigma(1^{t-1}0^{s-i}10^i) \\ &= 101^{t-2}0^{s-i}10^{i-1} \\ &= first(\mathbf{W}_{t-1,s-i+1}10^{i-1} \end{aligned}$$

which follows from Lemma 1 and the definition of σ (2b). In particular, (2b) applies here since $t = s$ and $i \ge 2$ imply that $t - 1 > s - i$. □

Theorem 1 allows us to show that the iterative definition of CoolCat produces lists that are *circular*. That is, in both $\mathbf{R}_{t,s}$ and $\mathbf{S}_{t,s}$, the first string can be obtained by applying σ to the last string. More generally we have the following corollary.

Corollary 1. *For any $\mathbf{b} \in \mathbf{B}_{t,s}$ and $k = |\mathbf{B}_{t,s}|$*

$$\sigma^k(\mathbf{b}) = \mathbf{b}.$$

Proof. We can prove this result by showing that the list $\mathbf{S}_{t,s}$ is circular. This proves the statement of the corollary and also proves that $\mathbf{R}_{t,s}$ is circular by (4)

and (3). We accomplish our goal through the following chain of equalities that reference Theorem 1, Lemma 1, and (1a)

$$\begin{aligned}\sigma(last(\mathbf{S}_{t,s})) &= \sigma(last(\mathbf{W}_{t,s}))\\ &= \sigma(1^t0^s)\\ &= 101^{t-1}0^{s-1}\\ &= first(\mathbf{W}_{t,s})\\ &= first(\mathbf{S}_{t,s}).\end{aligned}$$

□

Theorem 1 also allows us to prove that the iterative definition of CoolCat generates every string in $\mathbf{B}_{t,s}$.

Corollary 2. *$\mathbf{R}_{t,s}$ and $\mathbf{S}_{t,s}$ contain each string in $\mathbf{B}_{t,s}$ exactly once.*

Proof. The result for $\mathbf{S}_{t,s}$ follows from Remark 1 and Theorem 1. The result for $\mathbf{R}_{t,s}$ follows from the fact that

$$\sigma^k(1^t0^s) = 1^t0^s$$

for $k = |\mathbf{B}_{t,s}|$ by Corollary 1, and thus $\mathbf{R}_{t,s}$ is a reordering of $\mathbf{S}_{t,s}$ by (3) and (4). □

Although the recursive definition of $\mathbf{W}_{t,s}$ has its benefits, sometimes it is more convenient to work with a recursive definition that contains fewer terms. For example, in Section 5 we rank the order of the strings within CoolCat utilizing the following definition

$$\mathbf{K}_{t,s} = \begin{cases}\mathbf{K}_{t,s-1}0 & \text{if } t = s\\ \mathbf{K}_{t-1,s}1,\ 1^{t-1}01 & \text{if } s = 1\\ \mathbf{K}_{t,s-1}0,\ \mathbf{K}_{t-1,s}1,\ 1^{t-1}0^s1 & \text{if } 1 < s < t.\end{cases} \tag{10}$$

In Theorem 2 we prove that $\mathbf{K}_{t,s}$ is identical to $\mathbf{W}_{t,s}$ except that it is missing the string 1^t0^s. The proof is involved, so we provide an illustration for each of the three cases of (10) in Figure 6. In each column the overlined and underlined strings denote whether the number of zeros or ones are being recursively decreased, respectively. Strings without an overline or underline are of the form $1^{t-1}0^s1$ and are not involved in the next lower level of recursion, while the strings below the horizontal line are of the form 1^t0^s and represent the unique string that is in $\mathbf{W}_{t,s}$ but is not in $\mathbf{K}_{t,s}$. For the sake of saving space we only produce the columns with a smaller number of zeros, until the number of zeros equals one.

$\mathbf{K}_{3,1}$	$\mathbf{K}_{4,1}$	$\mathbf{K}_{4,2}$	$\mathbf{K}_{4,3}$	$\mathbf{K}_{4,4}$
$10\underline{11}$	$10\underline{111}$	$101\overline{110}$	$1011\overline{100}$	$10111\overline{000}$
1101	$11\underline{011}$	$110\overline{110}$	$1101\overline{100}$	$11011\overline{000}$
	11101	$111\overline{010}$	$1110\overline{100}$	$11101\overline{000}$
		$10110\underline{1}$	$1011\overline{010}$	$10110\overline{100}$
		$11010\underline{1}$	$1101\overline{010}$	$11010\overline{100}$
		$10101\underline{1}$	$1010\overline{110}$	$10101\overline{100}$
		$11001\underline{1}$	$1100\overline{110}$	$11001\overline{100}$
		111001	$1110\overline{010}$	$11100\overline{100}$
			1011001	$10110\overline{010}$
			1101001	$11010\overline{010}$
			1010101	$10101\overline{010}$
			1100101	$11001\overline{010}$
			1110001	$11100\overline{010}$
1110	11110	111100	1111000	11110000

Figure 6: The second recursive structure for CoolCat.

Theorem 2. $\mathbf{W}_{t,s} = \mathbf{K}_{t,s},\ 1^t0^s$.

Proof. We prove the result by a double induction. The first induction will be on the number of zeros, and the second induction will be on the number of ones. For the base case of the first induction we have $s = 1$ and it is easy to verify that

$$\begin{aligned}\mathbf{W}_{t,1} &= \prod_{i=1}^{t} 1^i01^{t-i}\\ &= \prod_{i=1}^{t-1} 1^i01^{t-i},\ 1^t0\\ &= \mathbf{K}_{t,1},\ 1^t0.\end{aligned}$$

Now suppose that $s = k > 1$ and that the theorem holds for all $s < k$. At this point we start our second induction. For the base case of the second induction we have $t = k$. In other words the number of ones is equal to the number of zeros, which is the minimum possible number of ones. We have the following expression for $\mathbf{W}_{k,k}$

$$\begin{aligned}&= \prod_{i=1}^{k-1} \mathbf{W}_{k-1,i}10^{k-i},\ 1^k0^k\\ &= \prod_{i=1}^{k-1} \mathbf{W}_{k-1,i}10^{k-1-i}0,\ 1^k0^{k-1}0\\ &= \left(\prod_{i=1}^{k-1} \mathbf{W}_{k-1,i}10^{k-1-i},\ 1^k0^{k-1}\right)0\\ &= (\mathbf{W}_{k,k-1})0\\ &= (\mathbf{K}_{k,k-1}, 1^k0^{k-1})0\\ &= \mathbf{K}_{k,k-1}0,\ 1^k0^k\\ &= \mathbf{K}_{k,k},\ 1^k0^k.\end{aligned}$$

Now to continue with the second induction we suppose that $t = k + j$, for some $j > 0$, and that the theorem holds for all $t < k + j$. In other words, we are supposing that there are j more ones than zeros, and that the theorem holds when there are fewer than j additional ones. Then we have the following expression for $\mathbf{W}_{k+j,k}$

$$\begin{aligned}&= \prod_{i=1}^{k} \mathbf{W}_{k+j-1,i}10^{k-i},\ 1^{k+j}0^k\\ &= \prod_{i=1}^{k-1} \mathbf{W}_{k+j-1,i}10^{k-i},\ \mathbf{W}_{k+j-1,k}1,\ 1^{k+j}0^k\\ &= \left(\prod_{i=1}^{k-1} \mathbf{W}_{k+j-1,i}10^{k-1-i}\right)0,\ \mathbf{W}_{k+j-1,k}1,\ 1^{k+j}0^k.\end{aligned}$$

The bracketed product has fewer than k zeros and equals $\mathbf{W}_{k+j,k-1}$ except that it is missing $1^{k+j}0^{k-1}$ as its last string. Therefore, by the first induction

$$= \mathbf{K}_{k+j,k-1}0,\ \mathbf{W}_{k+j-1,k}1,\ 1^{k+j}0^k.$$

The second term has fewer than $k+j$ ones. Therefore, by the second induction we continue as follows

$$\begin{aligned}&= \mathbf{K}_{k+j,k-1}0,\ (\mathbf{K}_{k+j-1,k},\ 1^{k+j-1}0^k)1,\ 1^{k+j}0^k\\ &= \mathbf{K}_{k+j,k-1}0,\ \mathbf{K}_{k+j-1,k}1,\ 1^{k+j-1}0^k1,\ 1^{k+j}0^k\\ &= \mathbf{K}_{k+j,k},\ 1^{k+j}0^k.\end{aligned}$$

This completes the inductive case of the second induction, and so the theorem is true for $s = k$ and all $t \geq k$. This completes the inductive case of the first induction, and so the theorem is true for all $s \geq 1$. □

Before closing this section we explicitly state the first and last strings of $\mathbf{R}_{t,s}$ since it will be useful in the next section.

Lemma 2. *For $s > 0$*

$$\begin{aligned} first(\mathbf{R}_{t,s}) &= 1^t0^s \\ last(\mathbf{R}_{t,s}) &= \begin{cases} 1^{t-1}0^{s-1}10 & \text{if } t = s \\ 1^{t-1}0^s1 & \text{if } t > s. \end{cases} \end{aligned}$$

4 Algorithm

In this section we present an algorithm to generate $\mathbf{R}_{t,s}$. That is, we present an algorithm that iteratively visits each successive string in the CoolCat ordering starting with 1^t0^s. The algorithm is remarkably efficient in terms of time and storage. In particular it is loopless in the sense that each successive string is generated in $\mathbf{O}(1)$ time, and it is constant extra-space in the sense that it uses $\mathbf{O}(1)$ storage when excluding the array b that holds the binary string. The array b uses 1-based indexing, so $b[1]$ is the first value in the array. For proposition P, the notation $[\![P]\!]$ means 1 if P is true and 0 if P is false.

As in Section 2, the variable x is used to represent the position of 1 in the leftmost 01. However, the variable y is now used to represent the position of the leftmost 0. The initial values of x and y do not obey this rule, and they are chosen simply for the sake of the first iteration. The initial value $b = 1^t0^s$ is visited by the visit(b) command on line 5, while all other values of b contain a leftmost 01 and are visited on line 21. We say that each iteration starts at the while statement on line 6. During each iteration there are three possible routes through the if statements and these three routes correspond exactly to the cases from Section 2. If $b = 1^p0^q11\alpha$ (case (a)) at the start of an iteration then the outer if statement on line 11 is not entered. If $b = 1^p0^p10\alpha$ (case (c)) then the inner if statement on line 12 is entered. If $b = 1^p0^q10\alpha$ for $p > q$ (case (b)) then the inner else statement is entered. The general idea of the algorithm is to maintain x and y and to use their values, and the values of $b[x]$ and $b[y]$, to determine how b needs to change from one iteration to the next.

CoolCat(t, s)
Require: $t \geq s > 0$
1: $n \leftarrow t + s$
2: $b \leftarrow \text{array}(1^t0^s)$
3: $x \leftarrow t$
4: $y \leftarrow t$
5: visit(b)
6: **while** $x < n - [\![t = s]\!]$ **do**
7: $b[x] \leftarrow 0$
8: $b[y] \leftarrow 1$
9: $x \leftarrow x + 1$
10: $y \leftarrow y + 1$
11: **if** $b[x] = 0$
12: **if** $x = 2y - 2$ { Case (c) }
13: $x \leftarrow x + 1$
14: **else**
15: $b[x] \leftarrow 1$ { Case (b) }
16: $b[2] \leftarrow 0$
17: $x \leftarrow 3$
18: $y \leftarrow 2$
19: **end**
20: **end** { else Case (a) }
21: visit(b)
22: **end**

To prove the correctness of the algorithm we track the values of the three variables from one visit call to the next visit. We let $b_1, b_2, \ldots$ represent the values taken by variable b at each subsequent visit, and we use the same convention for x and y. For example, b_1 will be the first and only value of b visited at line 5, while b_2 will be the first value of b visited at line 21. When y_i is the smallest value where $b_i[y_i] = 0$ then we will say that y_i is *correct*. Likewise when x_i is the smallest value where $b_i[x_i] = 1$ and $x_i > y_i$ then we will say that x_i is *correct*. For convenience we also let $\mathbf{V}_{t,s} = b_1, b_2, \ldots, b_k$ where b_k is the last value of b that is visited before the program terminates. Ultimately we will show that the program does in fact terminate, and that $\mathbf{V}_{t,s} = \mathbf{R}_{t,s}$ (Theorem 3). We refer to the current values of b, x, and y as the current *configuration*. From lines 2-4 we see that $b_1 = 1^t0^s$, $x_1 = t$ and $y_1 = t$, so the initial configuration before entering the while loop is

$$b = 1^t0^s \quad y = t \quad x = t.$$

By Lemma 2, $first(\mathbf{R}_{t,s}) = 1^t0^s$ so b is initialized to the correct value. The program terminates once $x = n - (t = s)$ (line 6), where $(t = s)$ equals one if $t = s$, and zero otherwise. In other words, if $t = s$ then **CoolCat** terminates once $x = n - 1$, and otherwise it terminates once $x = n$. Recall that this condition echoes the two cases of (7). Finally, we point out **CoolCat**'s explicit requirement that $t \geq s > 0$. The next two lemmas will address the first two iterations of the algorithm.

Lemma 3. $\mathbf{V}_{t,s} = \mathbf{R}_{t,s}$ *when* $t \leq 2$.

Proof. It is easy to verify that $\mathbf{V}_{1,1} = 10$, $\mathbf{V}_{2,1} = 110, 101$, and $\mathbf{V}_{2,2} = 1100, 1010$. In the first case the program does not enter the while loop and in the last two cases the program terminates after the while loop's first iteration. □

Lemma 4. *If $t > 2$ then $b_2 = \sigma(b_1)$, $x_2 = 3$, and $y_2 = 2$.*

Proof. When $t > 2$ the program enters the while loop and after lines 7-10 we have the following configuration

$$b = 1^t0^s \quad y = t + 1 \quad x = t + 1.$$

Since $b[x] = b[t + 1] = 0$ the program enters the outer if statement on line 11. Since $t > 2$ it does not enter the inner if statement on line 12 and so lines 15 and 16 are executed to give the following configuration

$$b = 101^t0^s \quad y = t + 1 \quad x = t + 1.$$

After line 17 and line 18 we have the following configuration

$$b = 101^{t-1}0^{s-1} \quad y = 2 \quad x = 3.$$

Since the next line to execute is a visit statement we have $b_2 = 101^{t-1}0^{s-1}$. Therefore, we have proven the result since $b_1 = 1^t0^s$ and $\sigma(1^t0^s) = 101^{t-1}0^{s-1}$ by (1a). □

The next lemma explains how the algorithm terminates (the values for $last(\mathbf{R}_{t,s})$ are recalled from Lemma 2).

Lemma 5. *If $t > 1$, every $b_i \in \mathbf{B}_{t,s}$, and x_i is correct then*

$$last(\mathbf{V}_{t,s}) = last(\mathbf{R}_{t,s}) = \begin{cases} 1^{t-1}0^{s-1}10 & \text{if } t = s \\ 1^{t-1}0^s1 & \text{if } t > s \end{cases}$$

Proof. When $t = s$, the condition on the while loop is $x < n - 1$. If $b_k = 1^{t-1}0^{s-1}10$ and x_k is updated correctly then $x_k = n-1$, so once b_k is visited the program will terminate. Furthermore, by (6a) we have that $x_i < n - 1$ for all $i \neq k$ since by the assumption all $b_i \in \mathbf{B}_{t,s}$.

When $t > s$, the condition on the while loop is $x < n$. If $b_k = 1^{t-1}0^s1$ and x_k is updated correctly then $x_k = n$, so once b_k is visited the program will terminate. Furthermore, by (6b) we have that $x_i < n$ for all $i \neq k$ since by the assumption all $b_i \in \mathbf{B}_{t,s}$. □

Now that the extreme cases of **CoolCat** have been accounted for, we can focus on the general behavior of the algorithm. In particular, 1^t0^s and $1^{t-1}0^s1$ have been dealt with in Lemma 4 and Lemma 5 respectively, so we need only consider the behavior of the algorithm on binary strings that contain a leftmost 01 and at least one additional symbol following it. In other words, we assume that $b = 11^p00^q1z\ldots$ where $z \in \{0, 1\}$. From Section 3 we recall our iterative definition for $\sigma(\mathbf{b})$

$$= \begin{cases} 111^p00^qz\ldots & \text{if } p = q \text{ and } z = 0 \quad (11a) \\ 1z1^p00^q1\ldots & \text{if } p > q \text{ or } z = 1. \quad (11b) \end{cases}$$

Notice that when $z = 1$ then the left side of (11a) and (11b) are identical. Therefore, we can interchange their roles when the condition of $z = 1$ is satisfied. Thus, the conditions in (11a) and (11b) can be equivalently stated as $p = q$ and $p > q$, respectively. In fact, the conditions were originally stated this way in (2a) and (2b); we make the change here since it optimizes the logic of the resulting program. Another way of stating the equivalence is that if $b = 11^p00^p11\ldots$ then it does not matter if we move the $(2p + 3)$rd symbol or the $(2p + 4)$th symbol since both are equal to 1. We now are able to complete this section with three lemmas. The first two lemmas correspond to (11b) (cases (a) and (b) respectively), while the third lemma corresponds to (11a) (case (c)).

Lemma 6. *Suppose $z = 1$, so that $b_i = 11^p00^q11\ldots$. If x_i and y_i are correct, then $b_{i+1} = \sigma(b_i)$ and x_{i+1} and y_{i+1} are correct.*

Proof. From the statement of the lemma, we can assume that the current configuration appears below and the program just satisfied the condition of the while loop

$$b = 11^p00^q11\ldots \quad y = p + 2 \quad x = p + q + 3.$$

After executing lines 7-10 the current configuration becomes

$$b = 11^p10^q01\ldots \quad y = p + 3 \quad x = p + q + 4.$$

Since $b[x] = 1$ the program does not enter the if statement on line 11 and so b_{i+1}, x_{i+1}, and y_{i+1} are equal to their respective values above. From (11b), $\sigma(b_i) = b_{i+1}$. Furthermore, the values of y_{i+1} and x_{i+1} are correct. □

Lemma 7. *Suppose $p > q$ and $z = 0$, so that $b_i = 11^p00^q10\ldots$ with $p > q \geq 0$. If x_i and y_i are correct, then $b_{i+1} = \sigma(b_i)$ and x_{i+1} and y_{i+1} are correct.*

Proof. From the statement of the lemma, we can assume that the current configuration appears below and the program just satisfied the condition of the while loop

$$b = 11^p00^q10\ldots \quad y = p + 2 \quad x = p + q + 3.$$

After executing lines 7-10 the current configuration becomes

$$\begin{aligned} b &= 11^p10^q00\ldots \quad y = p + 3 \quad x = p + q + 4 \\ &= 111^p0^q00\ldots \end{aligned}$$

Since $b[x] = 0$ the program enters the if statement on line 11. Since $x = 2y - 2$ would imply that $p+q+4 = 2p + 4$ and thus $p = q$, then the if statement on line 12 is not entered. After executing lines 15 through 18 the configuration becomes

$$b = 101^p0^q01\ldots \quad y = 2 \quad x = 3.$$

At this point the program makes the next visit in line 21, so b_{i+1}, x_{i+1}, and y_{i+1} are equal to their respective values above. From (11b), $\sigma(b_i) = b_{i+1}$. Furthermore, the value of y_{i+1} is correct. Finally, the value of x_{i+1} is also correct since $p > 0$. □

Lemma 8. *Suppose $p = q$ and $z = 0$, so that $b_i = 11^p00^p10\ldots$. If x_i and y_i are correct, then $b_{i+1} = \sigma(b_i)$ and x_{i+1} and y_{i+1} are correct.*

Proof. From the statement of the lemma, we can assume that the current configuration appears below and the program just satisfied the condition of the while loop

$$b = 11^p00^p10\ldots \quad y = p + 2 \quad x = 2p + 3.$$

After executing lines 7-10 the current configuration becomes

$$b = 11^p10^p00\ldots \quad y = p + 3 \quad x = 2p + 4.$$

Since $b[x] = 0$ the program enters the if statement on line 11. Since $x = 2y - 2$ the program enters the if statement on line 12. After executing line 13 the current configuration becomes

$$b = 11^p10^p00\ldots \quad y = p + 3 \quad x = 2p + 5.$$

At this point the program makes the next visit in line 21, so b_{i+1}, x_{i+1}, and y_{i+1} are equal to their respective values above. From (11a), $\sigma(b_i) = b_{i+1}$. Furthermore, the value of y_{i+1} is correct. However, can we be certain that the value of x_{i+1} is correct? Notice that the explicitly displayed portion of b in the above configuration contains an equal number of 1s and 0s. Hence, the next symbol must be 1, and so the value of x_{i+1} is also correct. □

The result of Lemmas 3-8 is that **CoolCat**(t, s) correctly visits and updates $first(\mathbf{R}_{t,s})$, and then correctly visits and updates every other string in $\mathbf{R}_{t,s}$ up to and including $last(\mathbf{R}_{t,s})$ after which it terminates. Therefore, we have proven the following theorem.

Theorem 3. *For all $t \geq s > 0$, we have $\mathbf{V}_{t,s} = \mathbf{R}_{t,s}$.*

5 Ranking

In this section we develop a ranking algorithm that uses $O(n)$ arithmetic operations. We will need to know the number of elements in $\mathbf{K}_{t,s}$, which we denote by $K_{t,s} = |\mathbf{B}_{t,s}| - 1$. Table 1 shows $K_{t,s}$ for $0 \leq s \leq t \leq 8$.

Theorem 4. *For all $0 \leq s \leq t$,*

$$K_{t,s} + 1 = \frac{t - s + 1}{t + 1}\binom{t + s}{t} = \binom{t + s}{t} - \binom{t + s}{t + 1}.$$

	0	1	2	3	4	5	6	7	8
0	1								
1	1	1							
2	1	2	2						
3	1	3	5	5					
4	1	4	9	14	14				
5	1	5	14	28	42	42			
6	1	6	20	48	90	132	132		
7	1	7	27	75	165	297	429	429	
8	1	8	35	110	275	572	1001	1430	1430

Table 1: The Catalan triangle. The row t, column s entry is $K_{t,s} = \frac{t-s+1}{t+1}\binom{t+s}{t}$.

Proof. These are well-known properties of the "Catalan triangle" (Knuth (2006), Stanley (1999)). □

Let $\mathbf{b} = b_0 b_2 \cdots b_{t+s-1} \in \mathbf{B}_{t,s}$. We use $\rho(\mathbf{b})$ to denote the rank of $\mathbf{b}$ in the list $\mathbf{K}_{t,s}$. Here is a recursive description of the ranking process; it follows directly from (10). Let $\mathbf{b}' = b_0 b_2 \cdots b_{t+s-2}$.

$$\rho(\mathbf{b}) = \begin{cases} \rho(\mathbf{b}') & \text{if } b_{t+s-1} = 0 \\ K_{t,s} - 1 & \text{if } \mathbf{b} = 1^{t-1}0^s1 \\ K_{t-1,s} + \rho(\mathbf{b}') & \text{otherwise.} \end{cases} \quad (12)$$

For example,

$$\begin{aligned} \rho(1010101) &= K_{4,2} + \rho(101010) \\ &= 8 + \rho(10101) \\ &= 8 + K_{3,1} + \rho(1010) \\ &= 8 + 2 + \rho(101) \\ &= 10 + K_{2,1} - 1 \\ &= 10 \end{aligned}$$

Note that (12) ignores trailing 0s; the rank therefore depends only on the positions of the 1s. If $c_1, c_2, \ldots, c_t$ are the positions occupied by the 1s and q is the minimum value for which $c_q > q$, then (12) can be iterated to obtain

$$\rho(c_1 c_2 \ldots c_t) = K_{q,c_q-q} - 1 + \sum_{j=q+1}^{t} K_{j,c_j-j-1}. \quad (13)$$

We now show that there is a nice way to view the ranking process as a walk on a certain integer lattice. Refer to Figure 7. The walk starts at the upper left; each 1 is a vertical step down and each 0 is a horizontal step to the right. The vertical edges are labeled, where the t-th row of vertical edges (counting from 1) gets labeled as follows from left-to-right: (no label), $K_{t,0}, K_{t,1}, \ldots, K_{t,t-1}$. The label furthest to the right in each row is not on an edge. Figure 7 illustrates the path for the bitstring 11100110101100. The square marks the endpoint of the part of the path that ends at the leftmost 01; i.e, the string 111001 in the example bitstring. The rank of the bitstring is obtained by summing the edge labels on the path after the square, adding the edge label on the edge to the right of the one that precedes the square (the circled label in the figure), and then subtracting 1. Thus $\rho(11100110101100) = 4 + 19 + 74 + 109 + 8 - 1 = 213$.

To unrank we reverse the process. We use $\rho_{t,s}^{-1}(m)$ to denote the string $\mathbf{b} \in \mathbf{B}_{t,s}$ whose rank in $\mathbf{K}_{t,s}$ is m. Suppose, for example, that we want the rank 212 bitstring with $t = 8$ and $s = 6$; i.e., $\rho_{(8,6)}^{-1}(212)$. We start where the example path ends. We move to the left so long as the edge labels exceed the remaining rank,

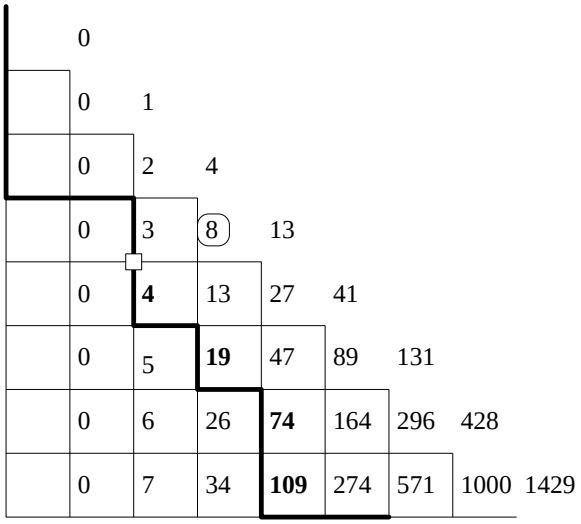

Figure 7: Ranking 11100110101100.

then move up and repeat. Arriving at the old square, we are at an impasse; the remaining rank is 7, so we have yet to encounter the square. So we so up and the rank becomes 4, which is what remains if we make the current location (one move above the old square) the new square. Thus $\rho_{(8,6)}^{-1}(212) = 11001110101100$. We leave it to the reader to turn this description into an algorithm.

What is the running time of the ranking algorithm? Let $n = t+s$. Note that (12) and (13) involve $O(n)$ additions and other operations. We can avoid computing the entire table by only computing the values needed along the path. First compute $K_{t,s}$, which takes $O(n)$ arithmetic operations. Then make use of the following relations which can be checked using Theorem 4:

$$1 + K_{t-1,s} = \frac{(t+1)(t-s)}{(t-s+1)(t+s)}(1 + K_{t,s}) \text{ and}$$

$$1 + K_{t,s-1} = \frac{s(t-s+2)}{(t-s+1)(t+s)}(1 + K_{t,s}).$$

Of course, if many ranking/unranking operations are being performed then it will be better to pre-compute the $K_{t,s}$ table.

6 Final Remarks

For future research, it would be interesting to determine whether the results of this paper can be extended to the natural 0/1 representation of k-ary trees, or to ordered trees with prescribed degree sequence (Zaks & Richards (1979)).

We thank the referees for carefully reading this paper and pointing out a number of typos and places where the exposition could be improved.

References

B. Bultena & F. Ruskey (1998), *An Eades-McKay Algorithm for Well-Formed Parenthesis Strings*, Information Processing Letters, 68, pp. 255–259.

Donald E. Knuth (2005), *The Art of Computer Programming, Volume 4: Generating all Combinations and Partitions*, Fascicle 3, Addison-Wesley, 150 pages.

Donald E. Knuth (2005), *The Art of Computer Programming, Volume 4: Generating all Trees; History of Combinationatorial Generation*, Fascicle 4, Addison-Wesley, 120 pages.

J. Korsh, P. LaFolette, & S. Lipschutz (2003), *Loopless Algorithms and Schröder Trees*, International Journal of Computer Mathematics, 80, pp. 709–725.

J. Lucas, D. Roelants, and F. Ruskey (1993), *On Rotations and the Generation of Binary Trees*, Journal of Algorithms, 15, pp. 343–366.

D. Roelants (1991), *A Loopless Algorithm for Generating Binary Tree Sequences*, Information Processing Letters, 39, pp. 184–194.

F. Ruskey (1979), *Simple combinatorial Gray codes constructed by reversing sublists*, 4th ISAAC (International Symposium on Algorithms and Computation), Lecture Notes in Computer Science, #762, pp. 201–208.

F. Ruskey and A. Proskurowski (1990), *Generating Binary Trees by Transpositions*, Journal of Algorithms, 11, pp. 68–84.

F. Ruskey & A. Williams (2005), *Generating Combinations By Prefix Shifts*, Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings. Lecture Notes in Computer Science **3595**, Springer-Verlag.

F. Ruskey and A. Williams (2008), *The Coolest way to Generate Combinations*, Discrete Mathematics, to appear, 2008.

R.P. Stanley (1999)*Enumerative Combinatorics, vol. 2*, Cambridge University Press, New York/Cambridge, 1999, xii + 581 pages.

R.P. Stanley (2007), *Catalan Addendum*, version of 20 June 2007; 61 pages, `http://www-math.mit.edu/~rstan/ec/`.

T. Takaoka (1999), $O(1)$ *Time Algorithms for Combinatorial Generation by Tree Traversal*, The Computer Journal, vol. 42, no. 5, pp. 400–408.

T. Takaoka & S. Violich (2006), *Combinatorial Generation by Fusing Loopless Algorithms*, In Proc. Twelfth Computing: The Australasian Theory Symposium (CATS2006), Hobart, Australia. CRPIT, 51. Gudmundsson, J. and Jay, B., Eds., ACS. 69–77.

V. Vajnovszki (1998), *On the Loopless Generation of Binary Tree Sequences*, Information Processing Letters, 68, pp. 113–117.

V. Vajnovszki & T. Walsh (2006), *A loopless two-close Gray-code algorithm for listing k-ary Dyck Words*, Journal of Discrete Algorithms, Vol. 4, No. 4, pp. 633–648.

R. Walsh, *A Simple Sequencing And Ranking Method That Works On Almost All Gray Codes*, Unpublished Research Report, Department of Mathematics and Computer Science, UQAM P.O. Box 8888, Station A, Montreal, Quebec, Canada H3C 3P8, 68 pages.

T. R. Walsh (2003), *Generating Gray codes in O(1) worst-case time per word*, Lecture Notes in Computer Science 2731, Proceedings of the 4h International Conference, Discrete Mathematics and Theoretical Computer Science 2003, Dijon, France, July 7-12, 2003, Springer-Verlag, New York, (2003), 73–88.

S. Zaks & D. Richards (1979), *Generating Trees and Other Combinatorial Objects Lexicographically*, SIAM J. Computing, 8, pp. 73–81.

also have applications to isometry groups and mathematical biology (Ganyushkin et al. 1994, Ganyushkin & Tsvirkunov 1994), quadratic forms and phylogenetic analysis (Dress et al. 2001), theory of quaternions (Weston 2001), scalable biological databases (Miranker 2003), sequences homology (Mao et al. 2005) and approximate string matching (Chávez & Navarro 2006), to name a few.

Unfortunately, as we will see in Section 3, it is computationally hard to determine whether one metric spaces is similar to another one under various notions of metric similarity. First, deciding whether two input metric spaces are isometric (Croom 2002) is as hard as the graph isomorphism problem (Papadimitriou 1994), for which no polynomial-time algorithms are known despite extensive research. Second, consider the problem of deciding, on input $L \geq 1$ and finite metric spaces (M, d) and (M, ρ), whether or not these spaces are L-bilipschitz equivalent (Farb & Mosher 1999, David & Semmes 2000). That is, we want to decide whether the metric spaces (M, d) and (M, ρ) exhibit a bijective map between them that preserves distances up to multiplicative factors ranging from $1/L$ to L. We observe that the results of Kenyon, Rabani and Sinclair (Kenyon et al. 2004) imply that it is hard even to approximate the least value of L such that (M, d) and (M, ρ) are L-bilipschitz equivalent. This may be interpreted as saying that it is hard to approximately compute the level of bilipschitz similarity even between finite metric spaces with the same ground set. Given the above hardness results, a randomized approximation algorithm with a reasonable complexity can be an attractive alternative to attack the problem of determining metric similarity or even metric embeddability.

An algorithm in the flavor of property testing (Fischer 2001) is one such alternative. It determines whether a problem instance has a certain property or is ϵ-far (under a certain distance measure) from having such a property, while allowing a small probability of error. In this paper, we seek an algorithm T that, when given as input $\epsilon > 0, L \geq 1$ and given oracle access to finite metric spaces (M, d) and (N, ρ) with $|M| \leq |N|$, has the following two properties. First, T accepts if

$$1/L \cdot d(x, y) \leq \rho(f(x), f(y)) \leq L \cdot d(x, y)$$

holds for some injection $f : M \to N$ and all $(x, y) \in M \times M$, that is, T accepts if (M, d) is L-bilipschitz embeddable into (N, ρ) (Farb & Mosher 1999, David & Semmes 2000, Croom 2002). Second, T rejects with high probability if the above inequality fails on at least an ϵ fraction of pairs $(x, y) \in M \times M$ for every injection $f : M \to N$. Such an algorithm T is called a one-sided tester for bilipschitz embeddability in this paper. Its query complexity is measured in terms of the number of times that it queries the metric spaces, where each query asks for the distance between a pair of points chosen for that query.

We give a one-sided tester for bilipschitz embeddability with query complexity at most $O(\sqrt{\frac{\ln |N|}{\epsilon |M|}}\,(|M|^2 + |N|^2))$. We also show an $\Omega(|N|^{3/2})$ lower bound on the query complexity of any one-sided tester for bilipschitz embeddability even for the special case of finite $|M| = |N|$ and $L = 1$. If (N, ρ) is known in advance, queries need only go to (M, d) and the query complexity is shown to be $O(\frac{|M| \ln |N|}{\epsilon})$. Our results utilize techniques developed by Fischer and Matsliah (Fischer & Matsliah 2006) in an earlier work on testing graph isomorphism.

We also give an extension to the case where the metric space (N, ρ) is known in advance but is not necessarily finite. When (N, ρ) is a totally bounded (Croom 2002) metric space known a priori, we devise an algorithm that in a technical sense tests whether a finite metric space (M, d) is (κ, C) quasi-isometrically embeddable (Ghys & de la Harpe 1991, Farb 1997, Farb & Mosher 1999, 2000) into (N, ρ), for input parameters $\kappa \geq 1$ and $C \geq 0$. The exact statement of this result is given in Section 6.

This paper is organized as follows. Section 2 gives the definitions. Section 3 gives the hardness results, which motivate switching to a property-testing flavored model. Sections 4–5 present upper bound and lower bounds on the query complexity of one-sided testers for bilipschitz embeddability. Section 6 extends the results to testing embeddability of a finite metric space into a totally bounded metric space. Section 7 discusses definitional issues and concludes the paper.

2 Definitions

Let S be an arbitrary set and t be a positive integer. We write S^t for the t-dimensional Cartesian product of S, and any pair $(x, y) \in S \times S$ is understood as an ordered pair unless otherwise specified. A function $d_S : S \times S \to \mathbb{R}$ is a metric on S if for all $x, y, z \in S$, we have $d_S(x, y) \geq 0$, $d_S(x, y) = 0$ if and only if $x = y$, $d_S(x, y) = d_S(y, x)$ and $d_S(x, y) \leq d_S(x, z) + d_S(z, y)$. A metric space is a set (called its ground set) endowed with a metric on it (Rudin 1976).

Let $L \geq 1$, (M, d) be a finite metric space and (N, ρ) be a metric space. We say that (M, d) is L-bilipschitz embeddable into (N, ρ) if there is an injective function $f : M \to N$ satisfying

$$1/L \cdot d(x, y) \leq \rho(f(x), f(y)) \leq L \cdot d(x, y) \qquad (1)$$

for all $(x, y) \in M \times M$ (Apostol 1974, Farb & Mosher 1999, David & Semmes 2000). Clearly, Eq. (1) could also be written equivalently as

$$1/L \cdot \rho(f(x), f(y)) \leq d(x, y) \leq L \cdot \rho(f(x), f(y)).$$

In this paper, we also say that (M, d) is ϵ-far from being L-bilipschitz embeddable into (N, ρ) if, for every injection $f : M \to N$, there are at least $\epsilon |M|^2$ pairs $(x, y) \in M \times M$ violating Eq. (1). Similarly, for $\kappa \geq 1$ and $C \geq 0$, we say that (M, d) is (κ, C) quasi-isometrically embeddable into (N, ρ) if there is a function $f : M \to N$ satisfying

$$1/\kappa \cdot d(x, y) - C \leq \rho(f(x), f(y)) \leq \kappa \cdot d(x, y) + C \qquad (2)$$

for all $(x, y) \in M \times M$ (Ghys & de la Harpe 1991, Farb 1997, Farb & Mosher 1999, 2000). If for every function $f : M \to N$, Eq. (2) fails on at least $\epsilon |M|^2$ pairs $(x, y) \in M \times M$, then (M, d) is said to be ϵ-far from being (κ, C) quasi-isometrically embeddable into (N, ρ).

For finite $|M| = |N|$, we say that (M, d) is L-bilipschitz equivalent to (N, ρ) if (M, d) is L-bilipschitz embeddable into (N, ρ) (Farb & Mosher 1999, David & Semmes 2000). Since for finite $|M| = |N|$, every injection from M to N is a bijection, it is easy to see that (M, d) is L-bilipschitz equivalent to (N, ρ) if and only if Eq. (1) holds for some bijection $f : M \to N$ and all $(x, y) \in M \times M$. Clearly, L-bilipschitz equivalence is a reflexive and symmetric relation between metric spaces. For finite $|M| = |N|$, the minimum value of $L \geq 1$ for which (M, d) and (N, ρ) are L-bilipschitz equivalent can be thought of as a measure on the similarity between (M, d) and (N, ρ). The smaller this value, the more similar the metric spaces are. In the extreme case, (M, d) and (N, ρ) are 1-bilipschitz equivalent if and only if

they are isometric, that is, there exists a distance-preserving bijective map (called an isometry) between them (Croom 2002). For $\epsilon > 0$ and finite $|M| = |N|$, we say that (M, d) and (N, ρ) are ϵ-far from being L-bilipschitz equivalent if (M, d) is ϵ-far from being L-bilipschitz embeddable into (N, ρ). This is the same as saying that Eq. (1) fails on at least an ϵ fraction of pairs $(x, y) \in M \times M$ for every bijection f. If (M, d) and (N, ρ) are ϵ-far from being 1-bilipschitz equivalent, they are said to be ϵ-far from being isometric.

When a metric space is given as an oracle, it means that we can query the oracle for the distance between any pair of points. Given as input $L \geq 1, \epsilon > 0$, positive integers $m \leq n$ and given oracle access to finite metric spaces $(M, d), (N, \rho)$ with $|M| = m$ and $|N| = n$, we are interested in the number of queries (to (M, d) and (N, ρ)) needed to determine whether (M, d) is L-bilipschitz embeddable into (N, ρ) or ϵ-far from being L-bilipschitz embeddable into (N, ρ). In particular, we seek an algorithm T that accepts when (M, d) is L-bilipschitz embeddable into (N, ρ), and rejects with high probability when (M, d) is ϵ-far from being L-bilipschitz embeddable into (N, ρ). Such an algorithm T is said to be a one-sided tester for bilipschitz embeddability in this paper. Similarly, an algorithm is a one-sided tester for bilipschitz equivalence (respectively, isometry) if, when we restrict to finite $|M| = |N|$, it accepts when (M, d) and (N, ρ) are L-bilipschitz equivalent (respectively, isometric) and rejects with high probability when (M, d) and (N, ρ) are ϵ-far from being L-bilipschitz equivalent (respectively, isometric). Finally, a one-sided tester for quasi-isometric embeddability is given as input $\kappa \geq 1, C \geq 0$, positive integers $m \leq n$ and given oracle access to metric spaces (M, d) and (N, ρ) with $|M| = m$ and $|N| = n$. It is required to accept if (M, d) is (κ, C) quasi-isometrically embeddable into (N, ρ) and reject with high probability if (M, d) is ϵ-far from being (κ, C) quasi-isometrically embeddable into (N, ρ).

For $\epsilon > 0$, positive integers $m \leq n$ and a one-sided tester T for bilipschitz embeddability, the query complexity of T with respect to ϵ, m and n is its worst-case number of queries when it is given ϵ, m, n, any $L \geq 1$ and oracle access to any metric spaces (M, d) and (N, ρ) with $|M| = m$ and $|N| = n$. Here the worst case is taken over all $L \geq 1$ and all metric spaces (M, d) and (N, ρ) (of sizes m and n) given as oracles. The query complexity (with respect to ϵ, m and n) of a one-sided tester for isometry is defined similarly except that L is fixed to 1 and m is fixed to equal n.

Let $G_1 = (V, E_1)$ and $G = (V, E_2)$ be undirected simple graphs (West 2001). An isomorphism between G_1 and G_2 is a bijection $\pi : V \to V$ such that for all $x, y \in V$, we have $(x, y) \in E_1$ if and only if $(\pi(x), \pi(y)) \in E_2$ (Papadimitriou 1994). The graph isomorphism problem asks whether two undirected simple graphs exhibit an isomorphism between them (Papadimitriou 1994). For $\epsilon > 0$, we say that G_1 and G_2 are ϵ-far from being isomorphic if for every bijection $\pi : V \to V$, there are at least $\epsilon\binom{|V|}{2}$ unordered pairs $(x, y) \in V \times V$ such that $(x, y) \in E_1$ but $(\pi(x), \pi(y)) \notin E_2$, or $(x, y) \in E_2$ but $(\pi(x), \pi(y)) \notin E_1$ (Fischer & Matsliah 2006).

When an algorithm is given oracle access to an undirected simple graph $G = (V, E)$, it means that the algorithm may query the oracle on any $(x, y) \in V \times V$ and be informed of whether $(x, y) \in E$. A one-sided tester for graph isomorphism receives as input $\epsilon > 0$, a positive integer n and is given oracle access to two undirected simple graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ with $|V| = n$. It must accept if G_1 is isomorphic to G_2 and reject with high probability if G_1 is ϵ-far from being isomorphic to G_2.

3 Hardness

In this section, we show that the problem of deciding whether two input metric spaces are isometric is polynomial-time reducible to and from the graph isomorphism problem, for which no polynomial-time algorithm has been known despite extensive research. Furthermore, we show that it is hard even to approximate the least $L \geq 1$ for which two input finite metric spaces are L-bilipschitz equivalent. In contrast to these hardness results, we will show in the Section 4 that there is an efficient one-sided tester for bilipschitz embeddability.

We state the following theorem. For its proof please refer to Appendix I.

Theorem 1. *The problem of testing whether two input metric spaces with the same finite ground set are isometric is polynomial-time reducible to and from the graph isomorphism problem.*

The problem of approximating the least $L \geq 1$ for which two input finite metric spaces with the same ground set are L-bilipschitz equivalent is even harder, provided that $\mathcal{NP} \neq \mathcal{P}$. This is stated in the following theorem, which is implicit in the work of Kenyon, Rabani and Sinclair (Kenyon et al. 2004) (see Proposition 2.2 and Proposition 2.4 in their paper).

Theorem 2. *((Kenyon et al. 2004)) If there is an algorithm that, on input any two finite metric spaces (M, d) and (M, ρ), outputs a number $L^* \geq 1$ such that (M, d) is L^*-bilipschitz equivalent to (M, ρ) and*

$$L^* < \sqrt{\frac{4}{3}} \cdot \min\{L \geq 1 \mid (M, d) \text{ is } L\text{-bilipschitz equivalent to } (M, \rho)\},$$

then $\mathcal{NP} = \mathcal{P}$.

That is, it is hard to approximate to within a multiplicative $\sqrt{4/3}$ the minimum value of $L \geq 1$ for which two input finite metric spaces are L-bilipschitz equivalent.

4 An upper bound on the query complexity

In this section, we give a one-sided tester for bilipschitz embeddability. Clearly, this also gives one-sided testers for bilipschitz equivalence and isometry. For convenience, we make the following definition.

Definition 1. *Let $L \geq 1$, (M, d) be a finite metric space, (N, ρ) be a metric space and $f : M \to N$ be a function. A quadruple $(x, y, u, v) \in M^2 \times N^2$ refutes f for the L-bilipschitz embeddability of (M, d) into (N, ρ) if $u = f(x), v = f(y)$ but*

$$1/L \cdot d(x, y) \leq \rho(f(x), f(y)) \leq L \cdot d(x, y)$$

fails to hold. A set $S \subseteq M^2 \times N^2$ of quadruples refutes f for the L-bilipschitz embeddability of (M, d) into (N, ρ) if at least one element of S does. When $L, (M, d)$ and (N, ρ) are clear from the context, we may simply say that a quadruple $(x, y, u, v) \in M^2 \times N^2$ or a set of quadruples refutes f without explicitly referring to $L, (M, d)$ and (N, ρ).

The following lemma states that the algorithm TEST-BILIP in Figure 1 is a one-sided tester for bilipschitz embeddability.

Lemma 3. *On input $L \geq 1, \epsilon > 0$, positive integers $m \leq n$ and given oracle access to finite metric spaces (M, d) and (N, ρ) with $|M| = m$ and $|N| = n$,*

if $\frac{\ln n}{\epsilon m} \geq 1/4$ **then**
  Query (M,d) and (N,ρ) for the distances between all pairs of points;
  if (M,d) is L-bilipschitz embeddable into (N,ρ) **then**
    Accept;
  else
    Reject;
  end if
else if $2 \cdot \frac{n}{m}\sqrt{\frac{\ln n}{\epsilon m}} \geq 1$ **then**
  $p_M \leftarrow 1$;
  $p_N \leftarrow 4 \cdot \frac{\ln n}{\epsilon m}$;
else
  $p_M \leftarrow 2 \cdot \frac{n}{m}\sqrt{\frac{\ln n}{\epsilon m}}$;
  $p_N \leftarrow 2 \cdot \frac{m}{n}\sqrt{\frac{\ln n}{\epsilon m}}$
end if
Construct $Q_M \subseteq M \times M$ by choosing each pair in $M \times M$ into Q_M independently with probability p_M;
Construct $Q_N \subseteq N \times N$ by choosing each pair in $N \times N$ into Q_N independently with probability p_N, using random coin tosses independent from those used to construct Q_M;
if $|Q_M| > 1000\, p_M\, m^2$ or $|Q_N| > 1000\, p_N\, n^2$ **then**
  Accept without making any queries;
else
  Query every element of Q_M to (M,d);
  Query every element of Q_N to (N,ρ);
  if all injections from M to N are refuted by $Q_M \times Q_N \subseteq M^2 \times N^2$ **then**
    Reject;
  else
    Accept;
  end if
end if

Figure 1: Algorithm TEST-BILIP. The inputs are $L \geq 1, \epsilon > 0$ and positive integers $m \leq n$. The metric spaces (M,d) and (N,ρ) are given as oracles and satisfy $|M| = m$ and $|N| = n$.

TEST-BILIP *accepts if* (M,d) *is* L*-bilipschitz embeddable into* (N,ρ)*, and rejects with high probability if* (M,d) *is* ϵ*-far from being* L*-bilipschitz embeddable into* (N,ρ)*.*

Proof. If $\frac{\ln n}{\epsilon m} \geq 1/4$, then TEST-BILIP does exhaustive queries and accepts exactly when (M,d) is L-bilipschitz embeddable into (N,ρ). Hence, we may assume that

$$\frac{\ln n}{\epsilon m} < 1/4 \tag{3}$$

in the following.

It is clear that TEST-BILIP accepts whenever (M,d) is L-bilipschitz embeddable into (N,ρ).

Now assume that (M,d) is ϵ-far from being L-bilipschitz embeddable into (N,ρ) and let $f : M \to N$ be an arbitrary injection. Denote by S_f the set of all pairs $(x,y) \in M \times M$ for which the inequality

$$1/L \cdot d(x,y) \leq \rho(f(x), f(y)) \leq L \cdot d(x,y)$$

fails to hold. By assumption we have $|S_f| \geq \epsilon |M|^2 = \epsilon m^2$. For any $(x,y) \in S_f$, the probability, taken over the random coin tosses of TEST-BILIP, that both $(x,y) \in Q_M$ and $(f(x), f(y)) \in Q_N$ is $p_M\, p_N$ (although there are two possible assignments to p_M and p_N by TEST-BILIP). Now write $S_f = \{(x_1,y_1), \ldots, (x_t,y_t)\}$. Since f is injective, the pairs $(f(x_1), f(y_1)), \ldots, (f(x_t), f(y_t))$ are different. Hence, the $2t$ events

$$\begin{gathered}(x_1,y_1) \in Q_M \\ \vdots \\ (x_t,y_t) \in Q_M \\ (f(x_1), f(y_1)) \in Q_N \\ \vdots \\ (f(x_t), f(y_t)) \in Q_N\end{gathered}$$

are independent. From this it is not hard to see that with probability

$$\prod_{(x,y)\in S_f} (1 - p_M\, p_N) \leq (1 - p_M\, p_N)^{\epsilon m^2},$$

none of $(x,y) \in S_f$ satisfies both $(x,y) \in Q_M$ and $(f(x), f(y)) \in Q_N$. Since $Q_M \times Q_N$ refutes f when there is a pair $(x,y) \in S_f$ satisfying both $(x,y) \in Q_M$ and $(f(x), f(y)) \in Q_N$, the probability taken over the random coin tosses of TEST-BILIP that $Q_M \times Q_N$ refutes f is at least

$$1 - (1 - p_M\, p_N)^{\epsilon m^2}.$$

By the union bound and the fact that there are n^m functions from M to N, with probability at least

$$1 - n^m\, (1 - p_M\, p_N)^{\epsilon m^2} \tag{4}$$

over the random coin tosses of TEST-BILIP, every injection from M to N is refuted by $Q_M \times Q_N$.

Now there are two cases to consider. The first is when $2 \cdot \frac{n}{m}\sqrt{\frac{\ln n}{\epsilon m}} \geq 1$. In this case, TEST-BILIP sets $p_M = 1$ and $p_N = 4 \cdot \frac{\ln n}{\epsilon m}$ where $p_N < 1$ is guaranteed by Eq. (3). The second case is when $2 \cdot \frac{n}{m}\sqrt{\frac{\ln n}{\epsilon m}} < 1$. In this case, TEST-BILIP sets $p_M = 2 \cdot \frac{n}{m}\sqrt{\frac{\ln n}{\epsilon m}} < 1$ and $p_N = 2 \cdot \frac{m}{n}\sqrt{\frac{\ln n}{\epsilon m}}$ where $p_N < 1$ is guaranteed by the facts that $p_N = (\frac{m}{n})^2 p_M$ and $m \leq n$. In both cases, we have $p_M\, p_N = 4 \cdot \frac{\ln n}{\epsilon m}$, resulting in Eq. (4) to be

$$\begin{aligned} & 1 - n^m\, (1 - p_M\, p_N)^{\epsilon m^2} \\ = \;& 1 - \exp(m \ln n) \cdot (1 - p_M\, p_N)^{\frac{1}{p_M\, p_N} \cdot p_M\, p_N\, \epsilon m^2} \\ \geq \;& 1 - \exp(m \ln n)\, \exp(-p_M\, p_N\, \epsilon m^2) \\ = \;& 1 - \exp(-3m \ln n). \end{aligned}$$

By the Chernoff bound (Chernoff 1952) and the fact that $m \leq n$, it can be verified that in both the aforementioned cases of setting p_M and p_N, the event $|Q_M| > 1000\, p_M\, m^2$ happens with probability $\exp(-\Omega(n\sqrt{m}))$ over the random coin tosses of TEST-BILIP (in fact, for the case of $p_M = 1$, the probability that $|Q_M| > 1000\, p_M\, m^2$ is zero). Similarly, the event $|Q_N| > 1000\, p_N\, n^2$ happens with probability $\exp(-\Omega(n))$. Finally, if $|Q_M| \leq 1000\, p_M\, m^2, |Q_N| \leq 1000\, p_N\, n^2$ and every injection from M to N is refuted by $Q_M \times Q_N$, then TEST-BILIP clearly rejects. The union bound therefore

shows that TEST-BILIP rejects with probability at least

$$1 - \exp(-3m \ln n) - \exp(-\Omega(n\sqrt{m})) - \exp(-\Omega(n)),$$

which is close to 1 for sufficiently large $n \in \mathbb{N}$. □

We now turn to analyze the query complexity of TEST-BILIP.

Lemma 4. *On input $L \geq 1, \epsilon > 0$, positive integers $m \leq n$ and given oracle access to finite metric spaces (M, d) and (N, ρ) with $|M| = m$ and $|N| = n$, the query complexity (with respect to ϵ, m and n) of* TEST-BILIP *is $O(\sqrt{\frac{\ln n}{\epsilon m}}\,(m^2 + n^2))$.*

Proof. If $\frac{\ln n}{\epsilon m} \geq 1/4$, then TEST-BILIP does exhaustive queries. The query complexity is $m^2 + n^2 = O(\sqrt{\frac{\ln n}{\epsilon m}}\,(m^2 + n^2))$. Hence, we may assume that

$$\frac{\ln n}{\epsilon m} < 1/4 \tag{5}$$

in the following.

The query complexity of TEST-BILIP is at most $1000\,p_M\,m^2 + 1000\,p_N\,n^2 = O(p_M\,m^2 + p_N\,n^2)$. Again, there are two cases to consider. The first is is when

$$2 \cdot \frac{n}{m}\sqrt{\frac{\ln n}{\epsilon m}} \geq 1. \tag{6}$$

In this case, TEST-BILIP sets $p_M = 1$ and $p_N = 4 \cdot \frac{\ln n}{\epsilon m}$. The second is when Eq. (6) does not hold. In this case, TEST-BILIP sets $p_M = 2 \cdot \frac{n}{m}\sqrt{\frac{\ln n}{\epsilon m}} < 1$ and $p_N = 2 \cdot \frac{m}{n}\sqrt{\frac{\ln n}{\epsilon m}}$.

In the first case,

$$\begin{aligned}
& p_M\,m^2 + p_N\,n^2 \\
= \;& m^2 + \frac{\ln n}{\epsilon m} \cdot 4n^2 \\
\overset{\text{Eq. (6)}}{\leq} \;& \frac{\ln n}{\epsilon m} \cdot 4n^2 + \frac{\ln n}{\epsilon m} \cdot 4n^2 \\
\overset{\text{Eq. (5)}}{\leq} \;& \sqrt{\frac{\ln n}{\epsilon m}}\,(4n^2 + 4n^2) \\
= \;& O(\sqrt{\frac{\ln n}{\epsilon m}}\,(m^2 + n^2)).
\end{aligned}$$

In the second case,

$$p_M\,m^2 + p_N\,n^2 = O(mn\sqrt{\frac{\ln n}{\epsilon m}}) = O(\sqrt{\frac{\ln n}{\epsilon m}}\,(m^2 + n^2)).$$

□

Combining Lemmas 3–4, we finally arrive at the the main result for this section.

Theorem 5. TEST-BILIP *is a one-sided tester for bilipschitz embeddability with query complexity $O(\sqrt{\frac{\ln n}{\epsilon m}}\,(m^2 + n^2))$ with respect to any $\epsilon > 0$ and any positive integers $m \leq n$.*

When the space (N, ρ) is not too large, or more specifically when $n = \exp(o(\epsilon m))$, Theorem 5 implies that TEST-BILIP has a query complexity of $o(m^2 + n^2)$ with respect to ϵ, m and n. That is, most distances between pairs need not be queried for one-sided testing of bilipschitz embeddability, provided that the host space is not excessively large.

When (N, ρ) is known in advance, a one-sided tester for bilipschitz embeddability needs only query the other space (M, d). Equivalently, we could consider one-sided testers for bilipschitz embeddability that may still make queries to both metric spaces, while counting only its query complexity concerning (M, d). That is, queries to (N, ρ) are regarded as dummy queries. This gives the following easy extension of Theorem 5, whose sketch of proof is given in Appendix II. But this time we use quasi-isometric embeddability for illustration and to be used later in Section 6.

Theorem 6. *There is a one-sided tester for quasi-isometric embeddability which, on input $\kappa \geq 1, C \geq 0, \epsilon > 0$, positive integers $m \leq n$ and given oracle access to metric spaces (M, d) and (N, ρ) with $|M| = m$ and $|N| = n$, makes $O(\frac{m \ln n}{\epsilon})$ queries to (M, d).*

5 A lower bound on the query complexity

In this section we show a lower bound on the query complexity of any one-sided tester for isometry. This will imply the same lower bound for any one-sided tester of bilipschitz equivalence. For this purpose, we relate the testing of isometry to testing graph isomorphism. The following theorem is due to Fischer and Matsliah (Fischer & Matsliah 2006).

Theorem 7. *((Fischer & Matsliah 2006)) Let $\epsilon \in (0, \frac{1}{100})$ and n be a positive integer. For every one-sided tester T for graph isomorphism, there are undirected simple n-vertex graphs G_1 and G_2 such that given $\epsilon \in (0, \frac{1}{100}), n$ and oracle access to G_1 and G_2, T makes at least $\frac{n^{3/2}}{200}$ queries.*

Using Theorem 7, it is not hard to give the following $\frac{n^{3/2}}{200}$ lower bound on the query complexity of any one-sided tester for isometry.

Theorem 8. *Let $\epsilon \in (0, \frac{1}{200})$ and n be a positive integer. The query complexity of any one-sided tester for isometry is at least $\frac{n^{3/2}}{200}$ with respect to ϵ and n.*

The interested reader is referred to Appendix III for the proof of Theorem 8

6 Embeddability into possibly infinite spaces

So far we have been dealing with the embeddability of a finite metric space into another finite one. In this section, we are interested in testing the embeddability of a finite metric space (M, d) into a totally bounded (Croom 2002) metric space (N, ρ) that is known in advance. Examples of totally bounded metric spaces include all compact metric spaces (Croom 2002), which in turn include all closed and bounded sets in the Euclidean space by the Heine-Borel theorem (Rudin 1976).

Definition 2. *((Croom 2002)) Let (X, d) be any metric space. For $\delta > 0$, a δ-net A_δ of (X, d) is a finite subset of X such that for every point $x \in X$, there is an $y \in A_\delta$ with $d(x, y) < \delta$. If (X, d) has a δ-net for every $\delta > 0$, then (X, d) is totally bounded.*

We are now ready to state our main theorem for this section.

Theorem 9. *Let (N, ρ) be a totally bounded metric space. Assume there is an algorithm that outputs a δ-net A_δ of (N, ρ) on input any $\delta > 0$. Then there is an algorithm T that, on input $\kappa \geq 1, 0 \leq C' < C, \epsilon > 0$, a positive integer m and given oracle access to a metric space (M, d) with $|M| = m$, satisfies the following conditions.*

1. *If (M, d) is (κ, C') quasi-isometrically embeddable into (N, ρ), then T accepts.*
2. *If (M, d) is ϵ-far from being (κ, C) quasi-isometrically embeddable into (N, ρ), then T rejects with high probability.*
3. *T makes $O(\frac{m \ln |A_{(C-C')/2}|}{\epsilon})$ queries to (M, d).*

Proof. Denote by QUASI-ISO the algorithm implied in Theorem 6. The algorithm T first selects a $(C - C')/2$-net $A_{(C-C')/2}$. Then T runs QUASI-ISO on input $\kappa, C, \epsilon, m, |A_{(C-C')/2}|$ and supplies QUASI-ISO with oracle access to (M, d) and $(A_{(C-C')/2}, \rho)$. Clearly, T could satisfy each query of QUASI-ISO by turning the same query to the corresponding metric space. Finally, T accepts if and only if QUASI-ISO accepts. The intuition is that T uses QUASI-ISO to test (M, d) for (κ, C) quasi-isometric embeddability into $(A_{(C-C')/2}, \rho)$.

Now we prove item 1. The premise of item 1 translates to the existence of a function $f : M \to N$ such that

$$1/\kappa \cdot d(x, y) - C' \leq \rho(f(x), f(y)) \leq \kappa \cdot d(x, y) + C' \quad (7)$$

holds for all $(x, y) \in M \times M$. Below we define a function $g : M \to A_{(C-C')/2}$. For each $x \in M$, let $g(x)$ be the point in $A_{(C-C')/2}$ that is closest to $f(x)$, breaking ties arbitrarily. Clearly, we have $\rho(g(x), f(x)) < (C - C')/2$ for each $x \in M$. Therefore,

$$\begin{aligned} & \rho(f(x), f(y)) \\ \leq\ & \rho(f(x), g(x)) + \rho(g(x), g(y)) + \rho(g(y), f(y)) \\ <\ & \rho(g(x), g(y)) + C - C' \end{aligned}$$

for all $x, y \in M$, and in fact $|\rho(f(x), f(y)) - \rho(g(x), g(y))| < C - C'$ for all $x, y \in M$ by a similar argument. This and Eq. (7) give

$$1/\kappa \cdot d(x, y) - C \leq \rho(g(x), g(y)) \leq \kappa \cdot d(x, y) + C$$

for all $x, y \in M$. Therefore, (M, d) is (κ, C) quasi-isometrically embeddable into $(A_{(C-C')/2}, \rho)$ and thus T accepts.

Item 2 is easily justified because its premise trivially implies that (M, d) is ϵ-far from being (κ, C) quasi-isometrically embeddable into $(A_{(C-C')/2}, \rho)$, which results in rejection of T with high probability.

Item 3 is established by directly invoking Theorem 6 and calculating the query complexity. □

We briefly justify the applicability of Theorem 9. It is meant to deal with the case where (M, d) is to be embedded into an already-known (N, ρ). In this case, queries to (N, ρ) can be answered without actually making a query. Since (N, ρ) is known beforehand and since we usually want to embed metric spaces into a host metric space with a simple structure, it is not strange to assume that we can find δ-nets for (N, ρ). For example, if (N, ρ) is a closed ball of radius $R > 0$ in the 3-dimensional Euclidean space, then it is easy to find a δ-net of cardinality $O(R^3/\delta^3)$ for (N, ρ).

7 Concluding remarks

We have defined bilipschitz embeddability and ϵ-farness from bilipschitz embeddability using injective functions. Such a definition is justifiable for the following reasons. First, Eq. (1) could be satisfied for all $(x, y) \in M \times M$ only if $f : M \to N$ is injective. Second and more importantly, one usually defines embeddings between metric spaces using injections, and in fact in many (if not most) areas of mathematics, embeddings are defined using injections (see, e.g., (*Embedding* n.d., Croom 2002, Goodman & O'Rourke 2004, Kenyon et al. 2004)). In contrast, quasi-isometric embeddability is defined via functions that are not necessarily injective (Ghys & de la Harpe 1991, Farb 1997, Farb & Mosher 1999, 2000), as we did in Section 2. We could also define the notions of quasi-isometric embeddability and ϵ-farness from quasi-isometric embeddability using injections by modifying the corresponding definitions in Section 2 to concern only with injections $f : M \to N$. That is, we could define (M, d) to be (κ, C) quasi-isometrically embeddable into (N, ρ) under injections if Eq. (2) holds for some injection $f : M \to N$ and all $(x, y) \in M \times M$. We could also say that (M, d) is ϵ-far from being (κ, C) quasi-isometrically embeddable into (N, ρ) under injections if Eq. (2) fails on at least an ϵ fraction of pairs $(x, y) \in M \times M$ for every injection $f : M \to N$. Theorems 5–6 and 9 can be easily adapted to give the corresponding tests for quasi-isometric embeddability under injections. The proofs are mostly the same except for a few trivial modifications to Definition 1 and algorithm TEST-BILIP. The query complexities remain the same. A minor point is that we have treated pairs selected from a metric space as ordered ones. They could also be treated as unordered since the distance function of any metric space is symmetric. Again, this does not change our results.

Our definition of ϵ-farness from L-bilipschitz embeddability is directly concerned with the least possible (over all injections $f : M \to N$) fraction of pairs $(x, y) \in M \times M$ violating Eq. (1), which is naturally interpreted as the quality of the best possible embedding $f : M \to N$. This seems as intuitively appealing feature of our definition. However, other definitions of ϵ-farness from L-bilipschitz embeddability may also be worth studying. For example, we may adopt one of the following definitions for (M, d) to be ϵ-far from being L-bilipschitz embeddable into (N, ρ).

1. At least an ϵ fraction of (ordered or unordered) pairs $(x, y) \in M \times M$ need to have their d-distance changed to obtain a metric space that is L-bilipschitz embeddable into (N, ρ).
2. Among all (ordered or unordered) pairs in $(M \times M) \cup (N \times N)$, at least an ϵ fraction of them need to have their d-distance or ρ-distance changed so that the modified metric space (M, d) is L-bilipschitz embeddable into the modified metric space (N, ρ).
3. For a reasonable set of edit operations on metric spaces, the least number of edit operations to turn (M, d) into a metric space that is L-bilipschitz embeddable into (N, ρ) is at least $\epsilon|M|^2$ (or $\epsilon|M|$, depending on whichever is more relevant).
4. For a reasonable set of edit operations on metric spaces, the least number of edit operations on (M, d) and (N, ρ) to turn (M, d) into being L-bilipschitz embeddable into (N, ρ) is at least $\epsilon(|M|^2 + |N|^2)$ (or $\epsilon(|M| + |N|)$, depending on whichever is more relevant).

Although in these definitions, farness from L-bilipschitz embeddability may no longer correspond to the quality of the best possible embedding, tests for L-bilipschitz embeddability under these definitions may still be worth studying and may provide new insights.

Appendix I: Proof of Theorem 1

Proof of Theorem 1. We first show the easy reduction from the graph isomorphism problem to the problem of testing isometry between finite metric spaces. Given two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, the reduction outputs two metric spaces (V, d) and (V, ρ) described below. For distinct $x, y \in V$, $d(x, y) = 2$ if $(x, y) \in E_1$ and $d(x, y) = 3$ otherwise. Also, set $d(x, x) = 0$ for each $x \in V$. The metric ρ is defined similarly with E_2 in place of E_1. It is not hard to verify that (V, d) and (V, ρ) are metric spaces and they are isometric if and only if G_1 is isomorphic to G_2.

Now we turn to the other direction of the reduction. Given two finite metric spaces (M, d) and (M, ρ), the reduction computes the sets (not multisets) $\{d(x, y) \mid x, y \in M, x \neq y\}$ and $\{\rho(x, y) \mid x, y \in M, x \neq y\}$. Let $\alpha_1 < \ldots < \alpha_t$ be an enumeration of the first set in strictly increasing order and $\beta_1 < \ldots < \beta_{t'}$ be that of the second. Assume that $t = t'$ and $\alpha_i = \beta_i$ for $1 \leq i \leq t$, for otherwise the reduction just outputs any two non-isomorphic graphs.

The reduction outputs two undirected simple graphs G_1 and G_2 defined below. It begins with G_1 having vertex set M and the empty edge set, and proceeds by adding to G_1 new vertices and new edges. For each pair of distinct $x, y \in M$, denote by $i(x, y, d)$ the unique value of $i \in \{1, \ldots, t\}$ satisfying $d(x, y) = \alpha_i$. The reduction adds $3\,i(x, y, d)$ new vertices $v_{x,y,1}, \ldots, v_{x,y,3\,i(x,y,d)}$ and also adds new edges

$$(x, v_{x,y,1}), \ldots, (x, v_{x,y,3\,i(x,y,d)})$$

and

$$(v_{x,y,1}, y), \ldots, (v_{x,y,3\,i(x,y,d)}, y)$$

to G_1. After adding new vertices and edges as above for each pair of distinct $x, y \in M$, the graph G_1 is finally formed. The graph G_2 is formed similarly with ρ in place of d.

Clearly, if (M, d) is isometric to (M, ρ), then G_1 and G_2 are isomorphic.

Now assume that G_1 is isomorphic to G_2. We are to show that (M, d) is isometric to (M, ρ). The set of vertices of G_1 is $M \cup S_1$ where

$$S_1 = \{v_{x,y,j} \mid x, y \in M, x \neq y, 1 \leq j \leq 3\,i(x, y, d)\}$$

is the set of newly added vertices to G_1. Similarly, the set of vertices of G_2 is denoted $M \cup S_2$ where S_2 is the set of newly added vertices to G_2. We may assume without loss of generality that $|M| \geq 2$. From the way we add edges to G_1 (respectively, G_2), it is not hard to see that every vertex in S_1 (respectively, S_2) has degree exactly two in G_1 (respectively, G_2), and every vertex in M has degree at least 3 in G_1 (respectively, G_2). An isomorphism f from G_1 to G_2 must therefore map M one-to-one and onto to M, and S_1 one-to-one and onto to S_2. Now fix distinct $x, y \in M$ arbitrarily. We are to show that $d(x, y) = \rho(f(x), f(y))$, which implies that f itself (when restricted on M) is an isometry from (M, d) to (M, ρ). That f is an isomorphism implies

$$\begin{aligned} & |\{v \mid (x, v), (v, y) \text{ are edges of } G_1 \text{ and } v \text{ has} \\ & \quad \text{degree exactly 2 in } G_1\}| \\ = \; & |\{u \mid (f(x), u), (u, f(y)) \text{ are edges of } G_2 \text{ and} \\ & \quad u \text{ has degree exactly 2 in } G_2\}|\,. \end{aligned}$$

The fact that S_1 (respectively, S_2) consists of exactly those vertices in G_1 (respectively, G_2) with degree two then implies

$$\begin{aligned} & |\{v \in S_1 \mid (x, v), (v, y) \text{ are edges of } G_1\}| \\ = \; & |\{u \in S_2 \mid (f(x), u), (u, f(y)) \text{ are edges of } G_2\}|\,, \end{aligned}$$

which in turn implies that $d(x, y) = \rho(f(x), f(y))$. □

Appendix II: Proof of Theorem 6

Sketch of proof of Theorem 6. We modify TEST-BILIP slightly to prove the theorem. If $\frac{\ln n}{\epsilon m} \leq 1/4$, the modified TEST-BILIP still does exhaustive queries. Otherwise, TEST-BILIP sets $p_M = 4 \cdot \frac{\ln n}{\epsilon m}$ and $p_N = 1$ (we let TEST-BILIP do exhaustive queries to (N, ρ)). These are different from the original assignments of TEST-BILIP to p_M and p_N. Also modify TEST-BILIP so that after querying Q_M and Q_N to (M, d) and (N, ρ), it rejects if all functions (not necessarily injective) from M to N are refuted by $Q_M \times Q_N$.

Clearly, when $\frac{\ln n}{\epsilon m} \geq 1/4$, the modified TEST-BILIP does exhaustive queries and the query complexity also follows. It is also clear that the modified TEST-BILIP accepts if (M, d) is (κ, C) quasi-isometrically embeddable into (N, ρ).

Now assume that (M, d) is ϵ-far from being (κ, C) quasi-isometrically embeddable into (N, ρ) and $\frac{\ln n}{\epsilon m} < 1/4$. It is clear that the modified assignment of $p_M = 4 \cdot \frac{\ln n}{\epsilon m}$ does not exceed 1. Now fix an arbitrary function $f : M \to N$. Similar to in Lemma 3, we define S_f to be the set of pairs $(x, y) \in M \times M$ violating

$$1/\kappa \cdot d(x, y) - C \leq \rho(f(x), f(y)) \leq \kappa \cdot d(x, y) + C.$$

We have $|S_f| \geq \epsilon m^2$. Since we do exhaustive queries to (N, ρ), this time f can be refuted by $Q_M \times Q_N$ if some pair in S_f is put into Q_M. The probability that $Q_M \times Q_N$ does not refute f is therefore at most

$$(1 - p_M)^{|S_f|} \leq (1 - 4 \cdot \frac{\ln n}{\epsilon m})^{\epsilon m^2}.$$

By the union bound, the probability that every function from M to N is refuted by $Q_M \times Q_N$ is at least $1 - n^m\,(1 - 4 \cdot \frac{\ln n}{\epsilon m})^{\epsilon m^2} = 1 - o(1)$. The probability that $Q_M > 1000\,p_M\,m^2$ is small, and $Q_N > 1000\,p_N\,n^2$ happens with probability zero. Therefore, with high probability $Q_M \times Q_N$ refutes every function from M to N, and the whole Q_M and Q_N are queried to (M, d) and (N, ρ), respectively, resulting in rejection of the modified TEST-BILIP.

The number of queries to (M, d) is at most $1000\,p_M\,m^2$, which is easily verified to obey the desired bound. □

Appendix III: Proof of Theorem 8

Proof of Theorem 8. Let T be a one-sided tester for isometry with query complexity $q(\epsilon, n)$ with respect to ϵ and n. Using T, we develop a one-sided tester T' for graph isomorphism with query complexity at most $q(\epsilon/2, n)$ with respect to ϵ and n. The theorem is then immediate from Theorem 7.

On input ϵ, n and given oracle access to two undirected simple graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ with $|V| = n$, the algorithm T' simulates T on input $n, \epsilon/2$ and provides T with oracle access to two metric spaces (V, d) and (V, ρ) described below. The

metric space (V, d) is defined by $d(x, x) = 0$ for $x \in V$, $d(x, y) = 2$ for $(x, y) \in E_1$ and $d(x, y) = 3$ for distinct $x, y \in V$ with $(x, y) \notin E_1$. The metric space (V, ρ) is defined similarly except that E_1 is replaced by E_2. Whenever T makes a query $(x, y) \in V \times V$ to the metric space (V, d) (respectively, (V, ρ)), T' asks G_1 (respectively, G_2) whether $(x, y) \in E_1$ (respectively $(x, y) \in E_2$) and then computes $d(x, y)$ (respectively, $\rho(x, y)$) to satisfy the query of T. The query complexity of T' is clearly at most $q(\epsilon/2, n)$. Finally, T' accepts (respectively, rejects) if and only if T accepts (respectively, rejects).

It is clear that if G_1 and G_2 are isomorphic, then (V, d) and (V, ρ) are isometric. Hence T and thus T' accepts.

Now assume that G_1 and G_2 are ϵ-far from being isomorphic and let $\pi : V \to V$ be any bijection. There are at least $\epsilon\binom{|V|}{2}$ unordered pairs $(x, y) \in V \times V$ such that either $(x, y) \in E_1$ and $(\pi(x), \pi(y)) \notin E_2$, or $(x, y) \in E_2$ and $(\pi(x), \pi(y)) \notin E_1$, and it is clear that any such pair satisfies $x \neq y$. This implies the existence of at least $2\epsilon\binom{|V|}{2}$ ordered pairs $(x, y) \in V \times V$ with $d(x, y) \neq \rho(\pi(x), \pi(y))$. Since the bijection π is arbitrary, (V, d) and (V, ρ) must be $\frac{2\epsilon\binom{|V|}{2}}{|V|^2} > \epsilon/2$ far from being isometric, resulting in the rejection of T and thus T' with high probability. □

References

Apostol, T. M. (1974), *Mathematical Analysis*, Addison Wesley.

Chávez, E. & Navarro, G. (2006), 'A metric index for approximate string matching', *Theoretical Computer Science* **352**, 266–279.

Chernoff, H. (1952), 'A measure of the asymptotic efficiency of tests of a hypothesis based on the sum of observations', *Annals of Mathematical Statistics* **23**, 493–507.

Croom, F. H. (2002), *Principles of Topology*, 1st edn, Thomson Learning Asia.

David, G. & Semmes, S. (2000), 'Regular mappings between dimensions', *Publicacions Matemàtiques* **44**, 369–417.

Deza, M. & Laurent, M. (1997), *Geometry of Cuts and Metrics*, Vol. 15 of *Algorithms and Combinatorics*, Springer.

Dress, A., Huber, K. T. & Moulton, V. (2001), Metric spaces in pure and applied mathematics, *in* 'Quadratic Forms and Related Topics', pp. 121–139.

Embedding (n.d.), Wikipedia: The Free Encyclopedia. `http://en.wikipedia.org/wiki/Embedding`.

Farb, B. (1997), 'The quasi-isometry classification of lattices in semisimple Lie groups', *Mathematical Research Letters* **4**, 705–717.

Farb, B. & Mosher, L. (1999), 'Quasi-isometric rigidity for the solvable Baumslag-Solitar groups, II', *Inventiones Mathematicae* **137**(3), 613–649.

Farb, B. & Mosher, L. (2000), 'On the asymptotic geometry of abelian-by-cyclic groups', *Acta Mathematica* **184**(2), 145–202.

Fischer, E. (2001), 'The art of uninformed decisions: A primer to property testing', *Bulletin of the European Association for Theoretical Computer Science* **75**, 97–126.

Fischer, E. & Matsliah, A. (2006), Testing graph isomorphism, *in* 'Proceedings of the 17th annual ACM-SIAM Symposium on Discrete Algorithms', pp. 299–308.

Ganyushkin, A. G., Sushchanskii, V. I. & Tsvirkunov, V. V. (1994), 'Computations in isometry groups of finite metric spaces', *Cybernetics and Systems Analysis* **30**(3), 331–347.

Ganyushkin, A. G. & Tsvirkunov, V. V. (1994), 'On classification of finite metric spaces', *Mathematical Notes* **56**(4), 1023–1029.

Ghys, E. & de la Harpe, P. (1991), *Infinite groups as geometric objects (after Gromov)*, Ergodic theory, symbolic dynamics and hyperbolic space, Oxford University Press.

Goodman, J. E. & O'Rourke, J., eds (2004), *Handbook of discrete and computational geometry*, 2nd edn, CRC Press, Inc.

Gupta, A. (2000), Embeddings of Finite Metrics, PhD thesis, University of California, Berkeley.

Indyk, P. (2001), Algorithmic applications of low-distortion geometric embeddings, *in* 'Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science', pp. 10–33.

Johnson, W. B. & Lindenstrauss, J., eds (2003), *Handbook of the Geometry of Banach Spaces*, North Holland.

Kenyon, C., Rabani, Y. & Sinclair, A. (2004), Low distortion maps between point sets, *in* 'Proceedings of the 36th annual ACM Symposium on Theory of Computing', pp. 272–280.

Linial, N. (2002), 'Finite metric spaces — combinatorics, geometry and algorithms', `http://www.cs.huji.ac.il/~nati/PAPERS/icm.ps.gz`.

Mao, R., Xu, W., Singh, N. & Miranker, D. P. (2005), 'An assessment of a metric space database index to support sequence homology', *International Journal on Artificial Intelligence Tools* **14**(5), 867–885.

Matoušek, J. (2002), *Lectures on Discrete Geometry*, Springer-Verlag New York, Inc.

Miranker, D. P. (2003), 'Metric-space indexes as a basis for scalable biological databases', *OMICS: A Journal of Integrative Biology* **7**(1), 57–60.

Papadimitriou, C. H. (1994), *Computational Complexity*, Addison Wesley.

Rudin, W. (1976), *Principles of Mathematical Analysis*, 3rd edn, McGraw-Hill.

West, D. B. (2001), *Introduction to Graph Theory*, 2nd edn, Prentice-Hall.

Weston, J. D. (2001), 'Vectors as quaternions: A corner of linear algebra', *The Mathematical Gazette* **85**(502), 25–35.

Testing Embeddability between Metric Spaces

Ching-Lueh Chang[1] **Yuh-Dauh Lyuu**[2] **Yen-Wu Ti**[3]

[1] Department of Computer Science and Information Engineering
National Taiwan University,
Taipei, Taiwan,
Email: d95007@csie.ntu.edu.tw

[2] Department of Computer Science and Information Engineering
National Taiwan University,
Taipei, Taiwan,
Email: lyuu@csie.ntu.edu.tw

[3] Department of Computer Science and Information Engineering
National Taiwan University,
Taipei, Taiwan,
Email: d91010@csie.ntu.edu.tw

Abstract

Let $L \geq 1, \epsilon > 0$ be real numbers, (M, d) be a finite metric space and (N,ρ) be a metric space (Rudin 1976). The metric space (M, d) is said to be L-bilipschitz embeddable into (N,ρ) if there is an injective function $f : M \to N$ with

$$1/L \cdot d(x, y) \leq \rho(f(x), f(y)) \leq L \cdot d(x, y)$$

for all $x, y \in N$ (Farb & Mosher 1999, David & Semmes 2000, Croom 2002). In this paper, we also say that (M, d) is ϵ-far from being L-bilipschitz embeddable into (N,ρ) if the above inequality fails on at least an ϵ fraction of pairs $(x, y) \in M \times M$ for every injective function $f : M \to N$.

Below, a query to a metric space consists of asking for the distance between a pair of points chosen for that query. We study the number of queries to metric spaces (M, d) and (N,ρ) needed to answer whether (M, d) is L-bilipschitz embeddable into (N,ρ) or ϵ-far from being L-bilipschitz embeddable into (N,ρ). When (M, d) is ϵ-far from being L-bilipschitz embeddable into (N,ρ), we allow an $o(1)$ probability of error (i.e., returning the wrong answer "L-bilipschitz embeddable"). However, we allow no error when (M, d) is L-bilipschitz embeddable into (N,ρ). That is, algorithms with only one-sided errors are considered in this paper. When $|M| \leq |N|$ are finite, we give an upper bound of $O(\sqrt{\frac{\ln |N|}{\epsilon |M|}}\,(|M|^2 + |N|^2))$ on the number of queries for determining with one-sided error whether (M, d) is L-bilipschitz embeddable into (N,ρ) or ϵ-far from being L-bilipschitz embeddable into (N,ρ). For the special case of finite $|M| = |N|$, the above upper bound evaluates to $O(|N|^{3/2}\sqrt{\frac{\ln |N|}{\epsilon}})$. We also prove a lower bound of $\Omega(|N|^{3/2})$ even for the special case when $|M| = |N|$ are finite and $L = 1$, which coincides with testing isometry between finite metric spaces (Croom 2002). For finite $|M| = |N|$, the upper and lower bounds thus match up to a multiplicative factor of at most $\sqrt{\frac{\ln |N|}{\epsilon}}$, which depends only sublogarithmically in $|N|$. We also investigate the case when (N,ρ) is not necessarily finite. Our results are based on techniques developed in an earlier work on testing graph isomorphism (Fischer & Matsliah 2006).

Research supported in part by NSC grant 95-2213-E-002-044.

1 Introduction

The ability to analyze metric spaces is of growing importance across diverse disciplines as huge bodies of data await analysis. In bioinformatics, for example, enormous amounts of data such as DNA sequences and protein sequences are constantly being produced. Efficient algorithms and standard computer programs have been developed over the years for calculating the distances between DNA or protein sequences, and this turns the collection of all known DNA and protein sequences into a huge metric space. As pointed out by Linial (Linial 2002), proper analysis of this space is of great significance to the biological sciences.

The analysis of metric spaces often requires various notions of similarity and embeddability between metric spaces. The philosophy is that, when a metric space is embedded into another metric space such that the original space is similar to the embedded one, understandings of the original space may be achieved through analysis of the embedded one. For example, when a metric space is embedded into the Euclidean plane while roughly preserving the distances between pairs of points, many efficient geometric algorithms that are not available for general metric spaces become applicable and of great help (Goodman & O'Rourke 2004). Another advantage is that embeddings into the Euclidean plane make possible more succinct representations of the original space (Goodman & O'Rourke 2004). In respect of these benefits of metric embedding that preserves similarity between the original and the embedded spaces, it comes without surprise that such embeddings have found tremendous applications in graph theory, combinatorial optimization, learning theory and computational geometry (Deza & Laurent 1997, Gupta 2000, Linial 2002, Matoušek 2002, Johnson & Lindenstrauss 2003, Indyk 2001, Kenyon et al. 2004). Besides, studies on the structures of metric spaces

also have applications to isometry groups and mathematical biology (Ganyushkin et al. 1994, Ganyushkin & Tsvirkunov 1994), quadratic forms and phylogenetic analysis (Dress et al. 2001), theory of quaternions (Weston 2001), scalable biological databases (Miranker 2003), sequences homology (Mao et al. 2005) and approximate string matching (Chávez & Navarro 2006), to name a few.

Unfortunately, as we will see in Section 3, it is computationally hard to determine whether one metric spaces is similar to another one under various notions of metric similarity. First, deciding whether two input metric spaces are isometric (Croom 2002) is as hard as the graph isomorphism problem (Papadimitriou 1994), for which no polynomial-time algorithms are known despite extensive research. Second, consider the problem of deciding, on input $L \geq 1$ and finite metric spaces (M, d) and (M,ρ), whether or not these spaces are L-bilipschitz equivalent (Farb & Mosher 1999, David & Semmes 2000). That is, we want to decide whether the metric spaces (M, d) and (M,ρ) exhibit a bijective map between them that preserves distances up to multiplicative factors ranging from $1/L$ to L. We observe that the results of Kenyon, Rabani and Sinclair (Kenyon et al. 2004) imply that it is hard even to approximate the least value of L such that (M, d) and (M,ρ) are L-bilipschitz equivalent. This may be interpreted as saying that it is hard to approximately compute the level of bilipschitz similarity even between finite metric spaces with the same ground set. Given the above hardness results, a randomized approximation algorithm with a reasonable complexity can be an attractive alternative to attack the problem of determining metric similarity or even metric embeddability.

An algorithm in the flavor of property testing (Fischer 2001) is one such alternative. It determines whether a problem instance has a certain property or is ϵ-far (under a certain distance measure) from having such a property, while allowing a small probability of error. In this paper, we seek an algorithm T that, when given as input $\epsilon > 0, L \geq 1$ and given oracle access to finite metric spaces (M, d) and (N,ρ) with $|M| \leq |N|$, has the following two properties. First, T accepts if

$$1/L \cdot d(x, y) \leq \rho(f(x), f(y)) \leq L \cdot d(x, y)$$

holds for some injection $f : M \to N$ and all $(x, y) \in M \times M$, that is, T accepts if (M, d) is L-bilipschitz embeddable into (N,ρ) (Farb & Mosher 1999, David & Semmes 2000, Croom 2002). Second, T rejects with high probability if the above inequality fails on at least an ϵ fraction of pairs $(x, y) \in M \times M$ for every injection $f : M \to N$. Such an algorithm T is called a one-sided tester for bilipschitz embeddability in this paper. Its query complexity is measured in terms of the number of times that it queries the metric spaces, where each query asks for the distance between a pair of points chosen for that query.

We give a one-sided tester for bilipschitz embeddability with query complexity at most $O(\sqrt{\frac{\ln |N|}{\epsilon |M|}}\,(|M|^2 + |N|^2))$. We also show an $\Omega(\,|N|^{3/2})$ lower bound on the query complexity of any one-sided tester for bilipschitz embeddability even for the special case of finite $|M| = |N|$ and $L = 1$. If (N,ρ) is known in advance, queries need only go to (M, d) and the query complexity is shown to be $O(\frac{|M| \ln |N|}{\epsilon})$. Our results utilize techniques developed by Fischer and Matsliah (Fischer & Matsliah 2006) in an earlier work on testing graph isomorphism.

We also give an extension to the case where the metric space (N,ρ) is known in advance but is not necessarily finite. When (N,ρ) is a totally bounded (Croom 2002) metric space known a priori, we devise an algorithm that in a technical sense tests whether a finite metric space (M, d) is (κ, C) quasi-isometrically embeddable (Ghys & de la Harpe 1991, Farb 1997, Farb & Mosher 1999, 2000) into (N,ρ), for input parameters $\kappa \geq 1$ and $C \geq 0$. The exact statement of this result is given in Section 6.

This paper is organized as follows. Section 2 gives the definitions. Section 3 gives the hardness results, which motivate switching to a property-testing flavored model. Sections 4–5 present upper bound and lower bounds on the query complexity of one-sided testers for bilipschitz embeddability. Section 6 extends the results to testing embeddability of a finite metric space into a totally bounded metric space. Section 7 discusses definitional issues and concludes the paper.

2 Definitions

Let S be an arbitrary set and t be a positive integer. We write S^t for the t-dimensional Cartesian product of S, and any pair $(x, y) \in S \times S$ is understood as an ordered pair unless otherwise specified. A function $d_S : S \times S \to \mathbb{R}$ is a metric on S if for all $x, y, z \in S$, we have $d_S(x, y) \geq 0$, $d_S(x, y) = 0$ if and only if $x = y$, $d_S(x, y) = d_S(y, x)$ and $d_S(x, y) \leq d_S(x, z) + d_S(z, y)$. A metric space is a set (called its ground set) endowed with a metric on it (Rudin 1976).

Let $L \geq 1$, (M, d) be a finite metric space and (N,ρ) be a metric space. We say that (M, d) is L-bilipschitz embeddable into (N,ρ) if there is an injective function $f : M \to N$ satisfying

$$1/L \cdot d(x, y) \leq \rho(f(x), f(y)) \leq L \cdot d(x, y) \qquad (1)$$

for all $(x, y) \in M \times M$ (Apostol 1974, Farb & Mosher 1999, David & Semmes 2000). Clearly, Eq. (1) could also be written equivalently as

$$1/L \cdot \rho(f(x), f(y)) \leq d(x, y) \leq L \cdot \rho(f(x), f(y)).$$

In this paper, we also say that (M, d) is ϵ-far from being L-bilipschitz embeddable into (N,ρ) if, for every injection $f : M \to N$, there are at least $\epsilon|M|^2$ pairs $(x, y) \in M \times M$ violating Eq. (1). Similarly, for $\kappa \geq 1$ and $C \geq 0$, we say that (M, d) is (κ, C) quasi-isometrically embeddable into (N,ρ) if there is a function $f : M \to N$ satisfying

$$1/\kappa \cdot d(x, y) - C \leq \rho(f(x), f(y)) \leq \kappa \cdot d(x, y) + C \qquad (2)$$

for all $(x, y) \in M \times M$ (Ghys & de la Harpe 1991, Farb 1997, Farb & Mosher 1999, 2000). If for every function $f : M \to N$, Eq. (2) fails on at least $\epsilon|M|^2$ pairs $(x, y) \in M \times M$, then (M, d) is said to be ϵ-far from being (κ, C) quasi-isometrically embeddable into (N,ρ).

For finite $|M| = |N|$, we say that (M, d) is L-bilipschitz equivalent to (N,ρ) if (M, d) is L-bilipschitz embeddable into (N,ρ) (Farb & Mosher 1999, David & Semmes 2000). Since for finite $|M| = |N|$, every injection from M to N is a bijection, it is easy to see that (M, d) is L-bilipschitz equivalent to (N,ρ) if and only if Eq. (1) holds for some bijection $f : M \to N$ and all $(x, y) \in M \times M$. Clearly, L-bilipschitz equivalence is a reflexive and symmetric relation between metric spaces. For finite $|M| = |N|$, the minimum value of $L \geq 1$ for which (M, d) and (N,ρ) are L-bilipschitz equivalent can be thought of as a measure on the similarity between (M, d) and (N,ρ). The smaller this value, the more similar the metric spaces are. In the extreme case, (M, d) and (N,ρ) are 1-bilipschitz equivalent if and only if

they are isometric, that is, there exists a distance-preserving bijective map (called an isometry) between them (Croom 2002). For $\epsilon > 0$ and finite $|M| = |N|$, we say that (M, d) and (N,ρ) are ϵ-far from being L-bilipschitz equivalent if (M, d) is ϵ-far from being L-bilipschitz embeddable into (N,ρ). This is the same as saying that Eq. (1) fails on at least an ϵ fraction of pairs $(x, y) \in M \times M$ for every bijection f. If (M, d) and (N,ρ) are ϵ-far from being 1-bilipschitz equivalent, they are said to be ϵ-far from being isometric.

When a metric space is given as an oracle, it means that we can query the oracle for the distance between any pair of points. Given as input $L \geq 1, \epsilon > 0$, positive integers $m \leq n$ and given oracle access to finite metric spaces $(M, d), (N,\rho)$ with $|M| = m$ and $|N| = n$, we are interested in the number of queries (to (M, d) and (N,ρ)) needed to determine whether (M, d) is L-bilipschitz embeddable into (N,ρ) or ϵ-far from being L-bilipschitz embeddable into (N,ρ). In particular, we seek an algorithm T that accepts when (M, d) is L-bilipschitz embeddable into (N,ρ), and rejects with high probability when (M, d) is ϵ-far from being L-bilipschitz embeddable into (N,ρ). Such an algorithm T is said to be a one-sided tester for bilipschitz embeddability in this paper. Similarly, an algorithm is a one-sided tester for bilipschitz equivalence (respectively, isometry) if, when we restrict to finite $|M| = |N|$, it accepts when (M, d) and (N,ρ) are L-bilipschitz equivalent (respectively, isometric) and rejects with high probability when (M, d) and (N,ρ) are ϵ-far from being L-bilipschitz equivalent (respectively, isometric). Finally, a one-sided tester for quasi-isometric embeddability is given as input $\kappa \geq 1, C \geq 0$, positive integers $m \leq n$ and given oracle access to metric spaces (M, d) and (N,ρ) with $|M| = m$ and $|N| = n$. It is required to accept if (M, d) is (κ, C) quasi-isometrically embeddable into (N,ρ) and reject with high probability if (M, d) is ϵ-far from being (κ, C) quasi-isometrically embeddable into (N,ρ).

For $\epsilon > 0$, positive integers $m \leq n$ and a one-sided tester T for bilipschitz embeddability, the query complexity of T with respect to ϵ, m and n is its worst-case number of queries when it is given ϵ, m, n, any $L \geq 1$ and oracle access to any metric spaces (M, d) and (N,ρ) with $|M| = m$ and $|N| = n$. Here the worst case is taken over all $L \geq 1$ and all metric spaces (M, d) and (N,ρ) (of sizes m and n) given as oracles. The query complexity (with respect to ϵ, m and n) of a one-sided tester for isometry is defined similarly except that L is fixed to 1 and m is fixed to equal n.

Let $G_1 = (V, E_1)$ and $G = (V, E_2)$ be undirected simple graphs (West 2001). An isomorphism between G_1 and G_2 is a bijection $\pi : V \to V$ such that for all $x, y \in V$, we have $(x, y) \in E_1$ if and only if $(\pi(x), \pi(y)) \in E_2$ (Papadimitriou 1994). The graph isomorphism problem asks whether two undirected simple graphs exhibit an isomorphism between them (Papadimitriou 1994). For $\epsilon > 0$, we say that G_1 and G_2 are ϵ-far from being isomorphic if for every bijection $\pi : V \to V$, there are at least $\epsilon\binom{|V|}{2}$ unordered pairs $(x, y) \in V \times V$ such that $(x, y) \in E_1$ but $(\pi(x), \pi(y)) \notin E_2$, or $(x, y) \in E_2$ but $(\pi(x), \pi(y)) \notin E_1$ (Fischer & Matsliah 2006).

When an algorithm is given oracle access to an undirected simple graph $G = (V, E)$, it means that the algorithm may query the oracle on any $(x, y) \in V \times V$ and be informed of whether $(x, y) \in E$. A one-sided tester for graph isomorphism receives as input $\epsilon > 0$, a positive integer n and is given oracle access to two undirected simple graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ with $|V| = n$. It must accept if G_1 is isomorphic to G_2 and reject with high probability if G_1 is ϵ-far from being isomorphic to G_2.

3 Hardness

In this section, we show that the problem of deciding whether two input metric spaces are isometric is polynomial-time reducible to and from the graph isomorphism problem, for which no polynomial-time algorithm has been known despite extensive research. Furthermore, we show that it is hard even to approximate the least $L \geq 1$ for which two input finite metric spaces are L-bilipschitz equivalent. In contrast to these hardness results, we will show in the Section 4 that there is an efficient one-sided tester for bilipschitz embeddability.

We state the following theorem. For its proof please refer to Appendix I.

Theorem 1. *The problem of testing whether two input metric spaces with the same finite ground set are isometric is polynomial-time reducible to and from the graph isomorphism problem.*

The problem of approximating the least $L \geq 1$ for which two input finite metric spaces with the same ground set are L-bilipschitz equivalent is even harder, provided that $\mathcal{NP} \neq \mathcal{P}$. This is stated in the following theorem, which is implicit in the work of Kenyon, Rabani and Sinclair (Kenyon et al. 2004) (see Proposition 2.2 and Proposition 2.4 in their paper).

Theorem 2. *((Kenyon et al. 2004)) If there is an algorithm that, on input any two finite metric spaces (M, d) and (M,ρ), outputs a number $L^* \geq 1$ such that (M, d) is L^*-bilipschitz equivalent to (M,ρ) and*

$$L^* < \sqrt{\frac{4}{3}} \cdot \min\{L \geq 1 \mid (M, d) \text{ is } L\text{-bilipschitz equivalent to } (M,\rho)\},$$

then $\mathcal{NP} = \mathcal{P}$.

That is, it is hard to approximate to within a multiplicative $\sqrt{4/3}$ the minimum value of $L \geq 1$ for which two input finite metric spaces are L-bilipschitz equivalent.

4 An upper bound on the query complexity

In this section, we give a one-sided tester for bilipschitz embeddability. Clearly, this also gives one-sided testers for bilipschitz equivalence and isometry. For convenience, we make the following definition.

Definition 1. *Let $L \geq 1$, (M, d) be a finite metric space, (N,ρ) be a metric space and $f : M \to N$ be a function. A quadruple $(x, y, u, v) \in M^2 \times N^2$ refutes f for the L-bilipschitz embeddability of (M, d) into (N,ρ) if $u = f(x), v = f(y)$ but*

$$1/L \cdot d(x, y) \leq \rho(f(x), f(y)) \leq L \cdot d(x, y)$$

fails to hold. A set $S \subseteq M^2 \times N^2$ of quadruples refutes f for the L-bilipschitz embeddability of (M, d) into (N,ρ) if at least one element of S does. When $L, (M, d)$ and (N,ρ) are clear from the context, we may simply say that a quadruple $(x, y, u, v) \in M^2 \times N^2$ or a set of quadruples refutes f without explicitly referring to $L, (M, d)$ and (N,ρ).

The following lemma states that the algorithm TEST-BILIP in Figure 1 is a one-sided tester for bilipschitz embeddability.

Lemma 3. *On input $L \geq 1, \epsilon > 0$, positive integers $m \leq n$ and given oracle access to finite metric spaces (M, d) and (N,ρ) with $|M| = m$ and $|N| = n$,*

if $\frac{\ln n}{\epsilon m} \geq 1/4$ **then**
  Query (M,d) and (N,ρ) for the distances between all pairs of points;
  if (M,d) is L-bilipschitz embeddable into (N,ρ) **then**
    Accept;
  else
    Reject;
  end if
else if $2 \cdot \frac{n}{m}\sqrt{\frac{\ln n}{\epsilon m}} \geq 1$ **then**
  $p_M \leftarrow 1$;
  $p_N \leftarrow 4 \cdot \frac{\ln n}{\epsilon m}$;
else
  $p_M \leftarrow 2 \cdot \frac{n}{m}\sqrt{\frac{\ln n}{\epsilon m}}$;
  $p_N \leftarrow 2 \cdot \frac{m}{n}\sqrt{\frac{\ln n}{\epsilon m}}$
end if
Construct $Q_M \subseteq M \times M$ by choosing each pair in $M \times M$ into Q_M independently with probability p_M;
Construct $Q_N \subseteq N \times N$ by choosing each pair in $N \times N$ into Q_N independently with probability p_N, using random coin tosses independent from those used to construct Q_M;
if $|Q_M| > 1000\, p_M\, m^2$ or $|Q_N| > 1000\, p_N\, n^2$ **then**
  Accept without making any queries;
else
  Query every element of Q_M to (M,d);
  Query every element of Q_N to (N,ρ);
  if all injections from M to N are refuted by $Q_M \times Q_N \subseteq M^2 \times N^2$ **then**
    Reject;
  else
    Accept;
  end if
end if

Figure 1: Algorithm TEST-BILIP. The inputs are $L \geq 1, \epsilon > 0$ and positive integers $m \leq n$. The metric spaces (M,d) and (N,ρ) are given as oracles and satisfy $|M| = m$ and $|N| = n$.

TEST-BILIP *accepts if* (M,d) *is* L*-bilipschitz embeddable into* (N,ρ)*, and rejects with high probability if* (M,d) *is* ϵ*-far from being* L*-bilipschitz embeddable into* (N,ρ).

Proof. If $\frac{\ln n}{\epsilon m} \geq 1/4$, then TEST-BILIP does exhaustive queries and accepts exactly when (M,d) is L-bilipschitz embeddable into (N,ρ). Hence, we may assume that

$$\frac{\ln n}{\epsilon m} < 1/4 \tag{3}$$

in the following.

It is clear that TEST-BILIP accepts whenever (M,d) is L-bilipschitz embeddable into (N,ρ).

Now assume that (M,d) is ϵ-far from being L-bilipschitz embeddable into (N,ρ) and let $f : M \to N$ be an arbitrary injection. Denote by S_f the set of all pairs $(x,y) \in M \times M$ for which the inequality

$$1/L \cdot d(x,y) \leq \rho(f(x), f(y)) \leq L \cdot d(x,y)$$

fails to hold. By assumption we have $|S_f| \geq \epsilon |M|^2 = \epsilon m^2$. For any $(x,y) \in S_f$, the probability, taken over the random coin tosses of TEST-BILIP, that both $(x,y) \in Q_M$ and $(f(x), f(y)) \in Q_N$ is $p_M\, p_N$ (although there are two possible assignments to p_M and p_N by TEST-BILIP). Now write $S_f = \{(x_1,y_1), \ldots, (x_t,y_t)\}$. Since f is injective, the pairs $(f(x_1), f(y_1)), \ldots, (f(x_t), f(y_t))$ are different. Hence, the $2t$ events

$$\begin{gathered}(x_1,y_1) \in Q_M \\ \vdots \\ (x_t,y_t) \in Q_M \\ (f(x_1), f(y_1)) \in Q_N \\ \vdots \\ (f(x_t), f(y_t)) \in Q_N\end{gathered}$$

are independent. From this it is not hard to see that with probability

$$\prod_{(x,y)\in S_f} (1 - p_M\, p_N) \leq (1 - p_M\, p_N)^{\epsilon m^2},$$

none of $(x,y) \in S_f$ satisfies both $(x,y) \in Q_M$ and $(f(x), f(y)) \in Q_N$. Since $Q_M \times Q_N$ refutes f when there is a pair $(x,y) \in S_f$ satisfying both $(x,y) \in Q_M$ and $(f(x), f(y)) \in Q_N$, the probability taken over the random coin tosses of TEST-BILIP that $Q_M \times Q_N$ refutes f is at least

$$1 - (1 - p_M\, p_N)^{\epsilon m^2}.$$

By the union bound and the fact that there are n^m functions from M to N, with probability at least

$$1 - n^m\, (1 - p_M\, p_N)^{\epsilon m^2} \tag{4}$$

over the random coin tosses of TEST-BILIP, every injection from M to N is refuted by $Q_M \times Q_N$.

Now there are two cases to consider. The first is when $2 \cdot \frac{n}{m}\sqrt{\frac{\ln n}{\epsilon m}} \geq 1$. In this case, TEST-BILIP sets $p_M = 1$ and $p_N = 4 \cdot \frac{\ln n}{\epsilon m}$ where $p_N < 1$ is guaranteed by Eq. (3). The second case is when $2 \cdot \frac{n}{m}\sqrt{\frac{\ln n}{\epsilon m}} < 1$. In this case, TEST-BILIP sets $p_M = 2 \cdot \frac{n}{m}\sqrt{\frac{\ln n}{\epsilon m}} < 1$ and $p_N = 2 \cdot \frac{m}{n}\sqrt{\frac{\ln n}{\epsilon m}}$ where $p_N < 1$ is guaranteed by the facts that $p_N = (\frac{m}{n})^2 p_M$ and $m \leq n$. In both cases, we have $p_M\, p_N = 4 \cdot \frac{\ln n}{\epsilon m}$, resulting in Eq. (4) to be

$$\begin{aligned} & 1 - n^m\, (1 - p_M\, p_N)^{\epsilon m^2} \\ = \;& 1 - \exp(m \ln n) \cdot (1 - p_M\, p_N)^{\frac{1}{p_M\, p_N} \cdot p_M\, p_N\, \epsilon m^2} \\ \geq \;& 1 - \exp(m \ln n)\, \exp(-p_M\, p_N\, \epsilon m^2) \\ = \;& 1 - \exp(-3m \ln n). \end{aligned}$$

By the Chernoff bound (Chernoff 1952) and the fact that $m \leq n$, it can be verified that in both the aforementioned cases of setting p_M and p_N, the event $|Q_M| > 1000\, p_M\, m^2$ happens with probability $\exp(-\Omega(n\sqrt{m}))$ over the random coin tosses of TEST-BILIP (in fact, for the case of $p_M = 1$, the probability that $|Q_M| > 1000\, p_M\, m^2$ is zero). Similarly, the event $|Q_N| > 1000\, p_N\, n^2$ happens with probability $\exp(-\Omega(n))$. Finally, if $|Q_M| \leq 1000\, p_M\, m^2, |Q_N| \leq 1000\, p_N\, n^2$ and every injection from M to N is refuted by $Q_M \times Q_N$, then TEST-BILIP clearly rejects. The union bound therefore

shows that TEST-BILIP rejects with probability at least

$$1 - \exp(-3m \ln n) - \exp(-\Omega(n\sqrt{m})) - \exp(-\Omega(n)),$$

which is close to 1 for sufficiently large $n \in \mathbb{N}$. □

We now turn to analyze the query complexity of TEST-BILIP.

Lemma 4. *On input $L \geq 1, \epsilon > 0$, positive integers $m \leq n$ and given oracle access to finite metric spaces (M,d) and (N,ρ) with $|M| = m$ and $|N| = n$, the query complexity (with respect to ϵ, m and n) of* TEST-BILIP *is $O(\sqrt{\frac{\ln n}{\epsilon m}}\,(m^2+n^2))$.*

Proof. If $\frac{\ln n}{\epsilon m} \geq 1/4$, then TEST-BILIP does exhaustive queries. The query complexity is $m^2 + n^2 = O(\sqrt{\frac{\ln n}{\epsilon m}}\,(m^2+n^2))$. Hence, we may assume that

$$\frac{\ln n}{\epsilon m} < 1/4 \tag{5}$$

in the following.

The query complexity of TEST-BILIP is at most $1000\,p_M\,m^2 + 1000\,p_N\,n^2 = O(p_M\,m^2 + p_N\,n^2)$. Again, there are two cases to consider. The first is is when

$$2 \cdot \frac{n}{m}\sqrt{\frac{\ln n}{\epsilon m}} \geq 1. \tag{6}$$

In this case, TEST-BILIP sets $p_M = 1$ and $p_N = 4 \cdot \frac{\ln n}{\epsilon m}$. The second is when Eq. (6) does not hold. In this case, TEST-BILIP sets $p_M = 2 \cdot \frac{n}{m}\sqrt{\frac{\ln n}{\epsilon m}} < 1$ and $p_N = 2 \cdot \frac{m}{n}\sqrt{\frac{\ln n}{\epsilon m}}$.

In the first case,

$$\begin{aligned}
& p_M\,m^2 + p_N\,n^2 \\
= \;& m^2 + \frac{\ln n}{\epsilon m} \cdot 4n^2 \\
\overset{\text{Eq. (6)}}{\leq} \;& \frac{\ln n}{\epsilon m} \cdot 4n^2 + \frac{\ln n}{\epsilon m} \cdot 4n^2 \\
\overset{\text{Eq. (5)}}{\leq} \;& \sqrt{\frac{\ln n}{\epsilon m}}\,(4n^2 + 4n^2) \\
= \;& O(\sqrt{\frac{\ln n}{\epsilon m}}\,(m^2+n^2)).
\end{aligned}$$

In the second case,

$$p_M\,m^2 + p_N\,n^2 = O(mn\sqrt{\frac{\ln n}{\epsilon m}}) = O(\sqrt{\frac{\ln n}{\epsilon m}}\,(m^2+n^2)).$$

□

Combining Lemmas 3–4, we finally arrive at the the main result for this section.

Theorem 5. TEST-BILIP *is a one-sided tester for bilipschitz embeddability with query complexity $O(\sqrt{\frac{\ln n}{\epsilon m}}\,(m^2+n^2))$ with respect to any $\epsilon > 0$ and any positive integers $m \leq n$.*

When the space (N,ρ) is not too large, or more specifically when $n = \exp(o(\epsilon m))$, Theorem 5 implies that TEST-BILIP has a query complexity of $o(m^2+n^2)$ with respect to ϵ, m and n. That is, most distances between pairs need not be queried for one-sided testing of bilipschitz embeddability, provided that the host space is not excessively large.

When (N,ρ) is known in advance, a one-sided tester for bilipschitz embeddability needs only query the other space (M,d). Equivalently, we could consider one-sided testers for bilipschitz embeddability that may still make queries to both metric spaces, while counting only its query complexity concerning (M,d). That is, queries to (N,ρ) are regarded as dummy queries. This gives the following easy extension of Theorem 5, whose sketch of proof is given in Appendix II. But this time we use quasi-isometric embeddability for illustration and to be used later in Section 6.

Theorem 6. *There is a one-sided tester for quasi-isometric embeddability which, on input $\kappa \geq 1, C \geq 0, \epsilon > 0$, positive integers $m \leq n$ and given oracle access to metric spaces (M,d) and (N,ρ) with $|M| = m$ and $|N| = n$, makes $O(\frac{m \ln n}{\epsilon})$ queries to (M,d).*

5 A lower bound on the query complexity

In this section we show a lower bound on the query complexity of any one-sided tester for isometry. This will imply the same lower bound for any one-sided tester of bilipschitz equivalence. For this purpose, we relate the testing of isometry to testing graph isomorphism. The following theorem is due to Fischer and Matsliah (Fischer & Matsliah 2006).

Theorem 7. *((Fischer & Matsliah 2006)) Let $\epsilon \in (0, \frac{1}{100})$ and n be a positive integer. For every one-sided tester T for graph isomorphism, there are undirected simple n-vertex graphs G_1 and G_2 such that given $\epsilon \in (0, \frac{1}{100}), n$ and oracle access to G_1 and G_2, T makes at least $\frac{n^{3/2}}{200}$ queries.*

Using Theorem 7, it is not hard to give the following $\frac{n^{3/2}}{200}$ lower bound on the query complexity of any one-sided tester for isometry.

Theorem 8. *Let $\epsilon \in (0, \frac{1}{200})$ and n be a positive integer. The query complexity of any one-sided tester for isometry is at least $\frac{n^{3/2}}{200}$ with respect to ϵ and n.*

The interested reader is referred to Appendix III for the proof of Theorem 8

6 Embeddability into possibly infinite spaces

So far we have been dealing with the embeddability of a finite metric space into another finite one. In this section, we are interested in testing the embeddability of a finite metric space (M,d) into a totally bounded (Croom 2002) metric space (N,ρ) that is known in advance. Examples of totally bounded metric spaces include all compact metric spaces (Croom 2002), which in turn include all closed and bounded sets in the Euclidean space by the Heine-Borel theorem (Rudin 1976).

Definition 2. *((Croom 2002)) Let (X,d) be any metric space. For $\delta > 0$, a δ-net A_δ of (X,d) is a finite subset of X such that for every point $x \in X$, there is an $y \in A_\delta$ with $d(x,y) < \delta$. If (X,d) has a δ-net for every $\delta > 0$, then (X,d) is totally bounded.*

We are now ready to state our main theorem for this section.

Theorem 9. *Let (N,ρ) be a totally bounded metric space. Assume there is an algorithm that outputs a δ-net A_δ of (N,ρ) on input any $\delta > 0$. Then there is an algorithm T that, on input $\kappa \geq 1, 0 \leq C' < C, \epsilon > 0$, a positive integer m and given oracle access to a metric space (M,d) with $|M| = m$, satisfies the following conditions.*

1. *If (M,d) is (κ,C') quasi-isometrically embeddable into (N,ρ), then T accepts.*
2. *If (M,d) is ϵ-far from being (κ,C) quasi-isometrically embeddable into (N,ρ), then T rejects with high probability.*
3. *T makes $O(\frac{m \ln |A_{(C-C')/2}|}{\epsilon})$ queries to (M,d).*

Proof. Denote by QUASI-ISO the algorithm implied in Theorem 6. The algorithm T first selects a $(C-C')/2$-net $A_{(C-C')/2}$. Then T runs QUASI-ISO on input $\kappa, C, \epsilon, m, |A_{(C-C')/2}|$ and supplies QUASI-ISO with oracle access to (M,d) and $(A_{(C-C')/2},\rho)$. Clearly, T could satisfy each query of QUASI-ISO by turning the same query to the corresponding metric space. Finally, T accepts if and only if QUASI-ISO accepts. The intuition is that T uses QUASI-ISO to test (M,d) for (κ,C) quasi-isometric embeddability into $(A_{(C-C')/2},\rho)$.

Now we prove item 1. The premise of item 1 translates to the existence of a function $f : M \to N$ such that

$$1/\kappa \cdot d(x,y) - C' \leq \rho(f(x),f(y)) \leq \kappa \cdot d(x,y) + C' \quad (7)$$

holds for all $(x,y) \in M \times M$. Below we define a function $g : M \to A_{(C-C')/2}$. For each $x \in M$, let $g(x)$ be the point in $A_{(C-C')/2}$ that is closest to $f(x)$, breaking ties arbitrarily. Clearly, we have $\rho(g(x),f(x)) < (C-C')/2$ for each $x \in M$. Therefore,

$$\begin{aligned} & \rho(f(x),f(y)) \\ \leq\ & \rho(f(x),g(x)) + \rho(g(x),g(y)) + \rho(g(y),f(y)) \\ <\ & \rho(g(x),g(y)) + C - C' \end{aligned}$$

for all $x,y \in M$, and in fact $|\rho(f(x),f(y)) - \rho(g(x),g(y))| < C - C'$ for all $x,y \in M$ by a similar argument. This and Eq. (7) give

$$1/\kappa \cdot d(x,y) - C \leq \rho(g(x),g(y)) \leq \kappa \cdot d(x,y) + C$$

for all $x,y \in M$. Therefore, (M,d) is (κ,C) quasi-isometrically embeddable into $(A_{(C-C')/2},\rho)$ and thus T accepts.

Item 2 is easily justified because its premise trivially implies that (M,d) is ϵ-far from being (κ,C) quasi-isometrically embeddable into $(A_{(C-C')/2},\rho)$, which results in rejection of T with high probability.

Item 3 is established by directly invoking Theorem 6 and calculating the query complexity. □

We briefly justify the applicability of Theorem 9. It is meant to deal with the case where (M,d) is to be embedded into an already-known (N,ρ). In this case, queries to (N,ρ) can be answered without actually making a query. Since (N,ρ) is known beforehand and since we usually want to embed metric spaces into a host metric space with a simple structure, it is not strange to assume that we can find δ-nets for (N,ρ). For example, if (N,ρ) is a closed ball of radius $R > 0$ in the 3-dimensional Euclidean space, then it is easy to find a δ-net of cardinality $O(R^3/\delta^3)$ for (N,ρ).

7 Concluding remarks

We have defined bilipschitz embeddability and ϵ-farness from bilipschitz embeddability using injective functions. Such a definition is justifiable for the following reasons. First, Eq. (1) could be satisfied for all $(x,y) \in M \times M$ only if $f : M \to N$ is injective. Second and more importantly, one usually defines embeddings between metric spaces using injections, and in fact in many (if not most) areas of mathematics, embeddings are defined using injections (see, e.g., (*Embedding* n.d., Croom 2002, Goodman & O'Rourke 2004, Kenyon et al. 2004)). In contrast, quasi-isometric embeddability is defined via functions that are not necessarily injective (Ghys & de la Harpe 1991, Farb 1997, Farb & Mosher 1999, 2000), as we did in Section 2. We could also define the notions of quasi-isometric embeddability and ϵ-farness from quasi-isometric embeddability using injections by modifying the corresponding definitions in Section 2 to concern only with injections $f : M \to N$. That is, we could define (M,d) to be (κ,C) quasi-isometrically embeddable into (N,ρ) under injections if Eq. (2) holds for some injection $f : M \to N$ and all $(x,y) \in M \times M$. We could also say that (M,d) is ϵ-far from being (κ,C) quasi-isometrically embeddable into (N,ρ) under injections if Eq. (2) fails on at least an ϵ fraction of pairs $(x,y) \in M \times M$ for every injection $f : M \to N$. Theorems 5–6 and 9 can be easily adapted to give the corresponding tests for quasi-isometric embeddability under injections. The proofs are mostly the same except for a few trivial modifications to Definition 1 and algorithm TEST-BILIP. The query complexities remain the same. A minor point is that we have treated pairs selected from a metric space as ordered ones. They could also be treated as unordered since the distance function of any metric space is symmetric. Again, this does not change our results.

Our definition of ϵ-farness from L-bilipschitz embeddability is directly concerned with the least possible (over all injections $f : M \to N$) fraction of pairs $(x,y) \in M \times M$ violating Eq. (1), which is naturally interpreted as the quality of the best possible embedding $f : M \to N$. This seems as intuitively appealing feature of our definition. However, other definitions of ϵ-farness from L-bilipschitz embeddability may also be worth studying. For example, we may adopt one of the following definitions for (M,d) to be ϵ-far from being L-bilipschitz embeddable into (N,ρ).

1. At least an ϵ fraction of (ordered or unordered) pairs $(x,y) \in M \times M$ need to have their d-distance changed to obtain a metric space that is L-bilipschitz embeddable into (N,ρ).
2. Among all (ordered or unordered) pairs in $(M \times M) \cup (N \times N)$, at least an ϵ fraction of them need to have their d-distance or ρ-distance changed so that the modified metric space (M,d) is L-bilipschitz embeddable into the modified metric space (N,ρ).
3. For a reasonable set of edit operations on metric spaces, the least number of edit operations to turn (M,d) into a metric space that is L-bilipschitz embeddable into (N,ρ) is at least $\epsilon|M|^2$ (or $\epsilon|M|$, depending on whichever is more relevant).
4. For a reasonable set of edit operations on metric spaces, the least number of edit operations on (M,d) and (N,ρ) to turn (M,d) into being L-bilipschitz embeddable into (N,ρ) is at least $\epsilon(|M|^2 + |N|^2)$ (or $\epsilon(|M| + |N|)$, depending on whichever is more relevant).

Although in these definitions, farness from L-bilipschitz embeddability may no longer correspond to the quality of the best possible embedding, tests for L-bilipschitz embeddability under these definitions may still be worth studying and may provide new insights.

Appendix I: Proof of Theorem 1

Proof of Theorem 1. We first show the easy reduction from the graph isomorphism problem to the problem of testing isometry between finite metric spaces. Given two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, the reduction outputs two metric spaces (V, d) and (V,ρ) described below. For distinct $x, y \in V$, $d(x, y) = 2$ if $(x, y) \in E_1$ and $d(x, y) = 3$ otherwise. Also, set $d(x, x) = 0$ for each $x \in V$. The metric ρ is defined similarly with E_2 in place of E_1. It is not hard to verify that (V, d) and (V,ρ) are metric spaces and they are isometric if and only if G_1 is isomorphic to G_2.

Now we turn to the other direction of the reduction. Given two finite metric spaces (M, d) and (M,ρ), the reduction computes the sets (not multisets) $\{d(x, y) \mid x, y \in M, x \neq y\}$ and $\{\rho(x, y) \mid x, y \in M, x \neq y\}$. Let $\alpha_1 < \ldots < \alpha_t$ be an enumeration of the first set in strictly increasing order and $\beta_1 < \ldots < \beta_{t'}$ be that of the second. Assume that $t = t'$ and $\alpha_i = \beta_i$ for $1 \leq i \leq t$, for otherwise the reduction just outputs any two non-isomorphic graphs.

The reduction outputs two undirected simple graphs G_1 and G_2 defined below. It begins with G_1 having vertex set M and the empty edge set, and proceeds by adding to G_1 new vertices and new edges. For each pair of distinct $x, y \in M$, denote by $i(x, y, d)$ the unique value of $i \in \{1, \ldots, t\}$ satisfying $d(x, y) = \alpha_i$. The reduction adds $3\,i(x, y, d)$ new vertices $v_{x,y,1}, \ldots, v_{x,y,3\,i(x,y,d)}$ and also adds new edges

$$(x, v_{x,y,1}), \ldots, (x, v_{x,y,3\,i(x,y,d)})$$

and

$$(v_{x,y,1}, y), \ldots, (v_{x,y,3\,i(x,y,d)}, y)$$

to G_1. After adding new vertices and edges as above for each pair of distinct $x, y \in M$, the graph G_1 is finally formed. The graph G_2 is formed similarly with ρ in place of d.

Clearly, if (M, d) is isometric to (M,ρ), then G_1 and G_2 are isomorphic.

Now assume that G_1 is isomorphic to G_2. We are to show that (M, d) is isometric to (M,ρ). The set of vertices of G_1 is $M \cup S_1$ where

$$S_1 = \{v_{x,y,j} \mid x, y \in M, x \neq y, 1 \leq j \leq 3\,i(x, y, d)\}$$

is the set of newly added vertices to G_1. Similarly, the set of vertices of G_2 is denoted $M \cup S_2$ where S_2 is the set of newly added vertices to G_2. We may assume without loss of generality that $|M| \geq 2$. From the way we add edges to G_1 (respectively, G_2), it is not hard to see that every vertex in S_1 (respectively, S_2) has degree exactly two in G_1 (respectively, G_2), and every vertex in M has degree at least 3 in G_1 (respectively, G_2). An isomorphism f from G_1 to G_2 must therefore map M one-to-one and onto to M, and S_1 one-to-one and onto to S_2. Now fix distinct $x, y \in M$ arbitrarily. We are to show that $d(x, y) = \rho(f(x), f(y))$, which implies that f itself (when restricted on M) is an isometry from (M, d) to (M,ρ). That f is an isomorphism implies

$$\begin{aligned} & |\{v \mid (x, v), (v, y) \text{ are edges of } G_1 \text{ and } v \text{ has degree exactly 2 in } G_1\}| \\ = \; & |\{u \mid (f(x), u), (u, f(y)) \text{ are edges of } G_2 \text{ and } u \text{ has degree exactly 2 in } G_2\}|\,. \end{aligned}$$

The fact that S_1 (respectively, S_2) consists of exactly those vertices in G_1 (respectively, G_2) with degree two then implies

$$\begin{aligned} & |\{v \in S_1 \mid (x, v), (v, y) \text{ are edges of } G_1\}| \\ = \; & |\{u \in S_2 \mid (f(x), u), (u, f(y)) \text{ are edges of } G_2\}|\,, \end{aligned}$$

which in turn implies that $d(x, y) = \rho(f(x), f(y))$. □

Appendix II: Proof of Theorem 6

Sketch of proof of Theorem 6. We modify TEST-BILIP slightly to prove the theorem. If $\frac{\ln n}{\epsilon m} \leq 1/4$, the modified TEST-BILIP still does exhaustive queries. Otherwise, TEST-BILIP sets $p_M = 4 \cdot \frac{\ln n}{\epsilon m}$ and $p_N = 1$ (we let TEST-BILIP do exhaustive queries to (N,ρ)). These are different from the original assignments of TEST-BILIP to p_M and p_N. Also modify TEST-BILIP so that after querying Q_M and Q_N to (M, d) and (N,ρ), it rejects if all functions (not necessarily injective) from M to N are refuted by $Q_M \times Q_N$.

Clearly, when $\frac{\ln n}{\epsilon m} \geq 1/4$, the modified TEST-BILIP does exhaustive queries and the query complexity also follows. It is also clear that the modified TEST-BILIP accepts if (M, d) is (κ, C) quasi-isometrically embeddable into (N,ρ).

Now assume that (M, d) is ϵ-far from being (κ, C) quasi-isometrically embeddable into (N,ρ) and $\frac{\ln n}{\epsilon m} < 1/4$. It is clear that the modified assignment of $p_M = 4 \cdot \frac{\ln n}{\epsilon m}$ does not exceed 1. Now fix an arbitrary function $f : M \to N$. Similar to in Lemma 3, we define S_f to be the set of pairs $(x, y) \in M \times M$ violating

$$1/\kappa \cdot d(x, y) - C \leq \rho(f(x), f(y)) \leq \kappa \cdot d(x, y) + C.$$

We have $|S_f| \geq \epsilon m^2$. Since we do exhaustive queries to (N,ρ), this time f can be refuted by $Q_M \times Q_N$ if some pair in S_f is put into Q_M. The probability that $Q_M \times Q_N$ does not refute f is therefore at most

$$(1 - p_M)^{|S_f|} \leq (1 - 4 \cdot \frac{\ln n}{\epsilon m})^{\epsilon m^2}.$$

By the union bound, the probability that every function from M to N is refuted by $Q_M \times Q_N$ is at least $1 - n^m\,(1 - 4 \cdot \frac{\ln n}{\epsilon m})^{\epsilon m^2} = 1 - o(1)$. The probability that $Q_M > 1000\,p_M\,m^2$ is small, and $Q_N > 1000\,p_N\,n^2$ happens with probability zero. Therefore, with high probability $Q_M \times Q_N$ refutes every function from M to N, and the whole Q_M and Q_N are queried to (M, d) and (N,ρ), respectively, resulting in rejection of the modified TEST-BILIP.

The number of queries to (M, d) is at most $1000\,p_M\,m^2$, which is easily verified to obey the desired bound. □

Appendix III: Proof of Theorem 8

Proof of Theorem 8. Let T be a one-sided tester for isometry with query complexity $q(\epsilon, n)$ with respect to ϵ and n. Using T, we develop a one-sided tester T' for graph isomorphism with query complexity at most $q(\epsilon/2, n)$ with respect to ϵ and n. The theorem is then immediate from Theorem 7.

On input ϵ, n and given oracle access to two undirected simple graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ with $|V| = n$, the algorithm T' simulates T on input $n,\epsilon/2$ and provides T with oracle access to two metric spaces (V, d) and (V,ρ) described below. The

metric space (V, d) is defined by $d(x, x) = 0$ for $x \in V$, $d(x, y) = 2$ for $(x, y) \in E_1$ and $d(x, y) = 3$ for distinct $x, y \in V$ with $(x, y) \notin E_1$. The metric space (V,ρ) is defined similarly except that E_1 is replaced by E_2. Whenever T makes a query $(x, y) \in V \times V$ to the metric space (V, d) (respectively, (V,ρ)), T' asks G_1 (respectively, G_2) whether $(x, y) \in E_1$ (respectively $(x, y) \in E_2$) and then computes $d(x, y)$ (respectively, $\rho(x, y)$) to satisfy the query of T. The query complexity of T' is clearly at most $q(\epsilon/2, n)$. Finally, T' accepts (respectively, rejects) if and only if T accepts (respectively, rejects).

It is clear that if G_1 and G_2 are isomorphic, then (V, d) and (V,ρ) are isometric. Hence T and thus T' accepts.

Now assume that G_1 and G_2 are ϵ-far from being isomorphic and let $\pi : V \to V$ be any bijection. There are at least $\epsilon\binom{|V|}{2}$ unordered pairs $(x, y) \in V \times V$ such that either $(x, y) \in E_1$ and $(\pi(x), \pi(y)) \notin E_2$, or $(x, y) \in E_2$ and $(\pi(x), \pi(y)) \notin E_1$, and it is clear that any such pair satisfies $x \neq y$. This implies the existence of at least $2\epsilon\binom{|V|}{2}$ ordered pairs $(x, y) \in V \times V$ with $d(x, y) \neq \rho(\pi(x), \pi(y))$. Since the bijection π is arbitrary, (V, d) and (V,ρ) must be $\frac{2\epsilon\binom{|V|}{2}}{|V|^2} > \epsilon/2$ far from being isometric, resulting in the rejection of T and thus T' with high probability. □

References

Apostol, T. M. (1974), *Mathematical Analysis*, Addison Wesley.

Chávez, E. & Navarro, G. (2006), 'A metric index for approximate string matching', *Theoretical Computer Science* **352**, 266–279.

Chernoff, H. (1952), 'A measure of the asymptotic efficiency of tests of a hypothesis based on the sum of observations', *Annals of Mathematical Statistics* **23**, 493–507.

Croom, F. H. (2002), *Principles of Topology*, 1st edn, Thomson Learning Asia.

David, G. & Semmes, S. (2000), 'Regular mappings between dimensions', *Publicacions Matemàtiques* **44**, 369–417.

Deza, M. & Laurent, M. (1997), *Geometry of Cuts and Metrics*, Vol. 15 of *Algorithms and Combinatorics*, Springer.

Dress, A., Huber, K. T. & Moulton, V. (2001), Metric spaces in pure and applied mathematics, *in* 'Quadratic Forms and Related Topics', pp. 121–139.

Embedding (n.d.), Wikipedia: The Free Encyclopedia. `http://en.wikipedia.org/wiki/Embedding`.

Farb, B. (1997), 'The quasi-isometry classification of lattices in semisimple Lie groups', *Mathematical Research Letters* **4**, 705–717.

Farb, B. & Mosher, L. (1999), 'Quasi-isometric rigidity for the solvable Baumslag-Solitar groups, II', *Inventiones Mathematicae* **137**(3), 613–649.

Farb, B. & Mosher, L. (2000), 'On the asymptotic geometry of abelian-by-cyclic groups', *Acta Mathematica* **184**(2), 145–202.

Fischer, E. (2001), 'The art of uninformed decisions: A primer to property testing', *Bulletin of the European Association for Theoretical Computer Science* **75**, 97–126.

Fischer, E. & Matsliah, A. (2006), Testing graph isomorphism, *in* 'Proceedings of the 17th annual ACM-SIAM Symposium on Discrete Algorithms', pp. 299–308.

Ganyushkin, A. G., Sushchanskii, V. I. & Tsvirkunov, V. V. (1994), 'Computations in isometry groups of finite metric spaces', *Cybernetics and Systems Analysis* **30**(3), 331–347.

Ganyushkin, A. G. & Tsvirkunov, V. V. (1994), 'On classification of finite metric spaces', *Mathematical Notes* **56**(4), 1023–1029.

Ghys, E. & de la Harpe, P. (1991), *Infinite groups as geometric objects (after Gromov)*, Ergodic theory, symbolic dynamics and hyperbolic space, Oxford University Press.

Goodman, J. E. & O'Rourke, J., eds (2004), *Handbook of discrete and computational geometry*, 2nd edn, CRC Press, Inc.

Gupta, A. (2000), Embeddings of Finite Metrics, PhD thesis, University of California, Berkeley.

Indyk, P. (2001), Algorithmic applications of low-distortion geometric embeddings, *in* 'Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science', pp. 10–33.

Johnson, W. B. & Lindenstrauss, J., eds (2003), *Handbook of the Geometry of Banach Spaces*, North Holland.

Kenyon, C., Rabani, Y. & Sinclair, A. (2004), Low distortion maps between point sets, *in* 'Proceedings of the 36th annual ACM Symposium on Theory of Computing', pp. 272–280.

Linial, N. (2002), 'Finite metric spaces — combinatorics, geometry and algorithms', `http://www.cs.huji.ac.il/~nati/PAPERS/icm.ps.gz`.

Mao, R., Xu, W., Singh, N. & Miranker, D. P. (2005), 'An assessment of a metric space database index to support sequence homology', *International Journal on Artificial Intelligence Tools* **14**(5), 867–885.

Matoušek, J. (2002), *Lectures on Discrete Geometry*, Springer-Verlag New York, Inc.

Miranker, D. P. (2003), 'Metric-space indexes as a basis for scalable biological databases', *OMICS: A Journal of Integrative Biology* **7**(1), 57–60.

Papadimitriou, C. H. (1994), *Computational Complexity*, Addison Wesley.

Rudin, W. (1976), *Principles of Mathematical Analysis*, 3rd edn, McGraw-Hill.

West, D. B. (2001), *Introduction to Graph Theory*, 2nd edn, Prentice-Hall.

Weston, J. D. (2001), 'Vectors as quaternions: A corner of linear algebra', *The Mathematical Gazette* **85**(502), 25–35.

On the Efficiency of Pollard's Rho Method for Discrete Logarithms

Shi Bai[1] **Richard P. Brent**[2] †

[1] Department of Computer Science,
Australian National University,
Canberra, ACT 0200
Email: shih.bai@gmail.com

[2] Centre for Mathematics and its Applications,
Mathematical Sciences Institute,
Australian National University,
Canberra, ACT 0200
Email: cats@rpbrent.com

Abstract

Pollard's rho method is a randomized algorithm for computing discrete logarithms. It works by defining a pseudo-random sequence and then detecting a match in the sequence. Many improvements have been proposed, while few evaluation results and efficiency suggestions have been reported. This paper is devoted to a detailed study of the efficiency issues in Pollard's rho method. We describe an empirical performance analysis of several widely applied algorithms. This should provide a better combination of algorithms and a good choice of parameters for Pollard's rho method.

Keywords: Pollard's rho method, discrete logarithm, elliptic curve discrete logarithm.

1 Introduction

The discrete logarithm is an analogue of the ordinary logarithm in a finite abelian group. Let H be a finite abelian group with the group operation $\otimes$. G is a cyclic subgroup of H generated by g, denoted as $\langle g \rangle = G$. Then an instance of the discrete logarithm problem (DLP) is stated as follow.

Definition 1.1 (DLP)**.** Given $h, g \in G$ known, DLP is to find the smallest non-negative integer x such that,

$$h = \underbrace{g \otimes g \otimes \cdots \otimes g}_{x \text{ times}}$$

As each element $h \in G$ can be expressed in the form of $h = g \otimes g \otimes \cdots \otimes g$, such x exists and is unique modulo $|G|$. By analogy to the ordinary logarithm, we write $x = \log_g h$. We also simplify the equation $h = g \otimes g \otimes \cdots \otimes g$ by writing $h = g^x$.

The discrete logarithm problem is believed to be hard, without any known efficient algorithm in the general case. Here an efficient algorithm means an algorithm with polynomial bit-complexity. The presumed hardness of DLP is relevant to many cryptosystems and cryptographic protocols such as Diffie-Hellman key exchange protocol (Diffie & Hellman 1976), ElGamal encryption (Gamal 1985), Digital Signature Algorithm (DSA) and Elliptic Curve DSA. Therefore algorithms for computing discrete logarithms are of great academic and practical importance.

Not all discrete logarithm problems are difficult. They may be trivial in some groups. The difficulty of the discrete logarithm problem depends on the representation of the group. Two popular finite groups used for discrete logarithm problems are the multiplicative group $(\mathbb{Z}/p\mathbb{Z})^*$ of integers modulo a prime p and the group of points on an elliptic curve over a finite field, denoted by $E(\mathbb{F}_p)$. In these groups, no polynomial time algorithm for the problem has been reported in the literature.

Pollard's rho method (Pollard 1978) is a randomized algorithm for computing the discrete logarithm. It generates a pseudo-random sequence by an iteration function $Y_{i+1} = f(Y_i)$ in a finite abelian group. Because the order of the group is finite, the sequence will ultimately meet an element that has occurred before. This is called a collision or a match, which can be found by Floyd's collision-detection (cycle-finding) algorithm. Under the assumption that $f : G \to G$ behaves like a truly random mapping, the expected number of evaluations before a match appears is $\sqrt{\pi|G|/2}$, which is fully exponential in the problem size. The space requirement is negligible. In some cases, such as the elliptic curve discrete logarithm problem (ECDLP), Pollard's rho method is the fastest algorithm currently available. Although there exist sub-exponential time algorithms for discrete logarithm problems in the group $(\mathbb{Z}/p\mathbb{Z})^*$ such as the index calculus method (Coppersmith et al. 1986), Pollard's rho method is still of practical interest because of its simplicity and effectiveness for smaller groups. In addition, it does not exploit any special properties of the groups, making it potentially applicable to DLPs in other abelian groups.

The rest of the paper is organized as follows. Section 2 presents a comprehensive analysis of Pollard's rho method and its variants in two aspects: iteration functions and collision-detection algorithms. We also compare the performance of different iteration functions and collision-detection algorithms. In Section 3, we fill some gaps in the previous literature, suggest a good choice of parameters and give an empirical analysis of the performance.

2 Background

In this section we introduce Pollard's rho method and discuss the current status of research on iteration functions and cycle-finding algorithms.

† The work of the second author was supported by the Australian Research Council.

2.1 Pollard's Rho Method

Pollard proposed an elegant algorithm (Pollard 1978) for the discrete logarithm problem based on a Monte Carlo idea and called it the rho method. The rho method works by first defining a sequence of elements that will be periodically recurrent, then looking for a match in the sequence. The match will lead to a solution of the discrete logarithm problem with high probability. The two key ideas involved are the iteration function for generating the sequence and the cycle-finding algorithm for detecting a match.

2.1.1 Pollard's Iteration Function

We first introduce the definition of the iteration function applied in the rho method.

Definition 2.1 (Iteration Function)**.** An iteration function on a set X is a mapping $f : X \to X$.

In Pollard's paper, DLPs in $(\mathbb{Z}/p\mathbb{Z})^*$ are considered where p is a prime. Let g be a generator of the cyclic group $G = (\mathbb{Z}/p\mathbb{Z})^*$. Another element $h \in G$ is given. The discrete logarithm problem is to compute x satisfying $g^x \equiv h \pmod p$. Pollard's iteration function $f_P : G \to G$ is defined as follows,

$$f_P(Y) \equiv \begin{cases} g \cdot Y & (\text{mod } p) \quad Y \in G_1 \\ Y^2 & (\text{mod } p) \quad Y \in G_2 \\ h \cdot Y & (\text{mod } p) \quad Y \in G_3 \end{cases} \tag{2.1}$$

In each iteration of $Y_{i+1} = f_P(Y_i)$, the function uses one of three rules depending on the value of Y_i. The group G is partitioned into three sets G_1, G_2, G_3 with similar sizes, not necessarily subgroups. Each Y_i has the form $g^{a_i}h^{b_i}$. If it happens that $Y_k \equiv Y_j \pmod p$, then $g^{a_k}h^{b_k} \equiv g^{a_j}h^{b_j} \pmod p$. We can often solve the DLP if a_k, a_j, b_k, b_j are known. The sequence (a_i) (and similarly for (b_i)) can be computed using[1],

$$a_{i+1} \equiv \begin{cases} a_i + 1 & (\text{mod } |G|) \quad Y_i \in G_1 \\ 2a_i & (\text{mod } |G|) \quad Y_i \in G_2 \\ a_i & (\text{mod } |G|) \quad Y_i \in G_3 \end{cases} \tag{2.2}$$

Since G is finite, the sequence (Y_i) produced by the iteration function is periodic. Therefore there exist two smallest integers μ and λ ($\mu \geq 0$, $\lambda \geq 1$) such that $Y_k = Y_{k+\lambda}$ for every $k \geqslant \mu$. To analyze the performance of the rho method, we use the following theorem,

Theorem 2.2 (Harris (1960))**.** *Under the assumption that an iteration function $f : G \to G$ behaves like a truly random mapping, the expected values for μ and λ are $\sqrt{\pi|G|/8} \approx 0.63\sqrt{|G|}$. The expected number of evaluations before a match appears is $E(\mu + \lambda) = \sqrt{\pi|G|/2} \approx 1.25\sqrt{|G|}$, provided that all elements are saved, which requires $\sqrt{\pi|G|/2}$ space.*

2.1.2 Reported Performance

Theorem 2.2 makes the assumption of true randomness. However, it has been shown empirically that this assumption does not hold exactly for Pollard's iteration function (Teske 1998). The actual performance is worse than the expected value given in Theorem 2.2. As it is impractical to find the exact value of $\mu + \lambda$ for Pollard's iteration function, a collision-detection algorithm is often applied in practice, needing I iterations. To analyze the performance of the iteration function, we adopt the idea of delay factor $\delta = I/E(\mu + \lambda)$ used in (Teske 1998). The values of δ and I for Pollard's iteration function have been reported and we divide I by δ to get $E(\mu + \lambda)$. The performance is summarized as follows. In groups $(\mathbb{Z}/p\mathbb{Z})^*$, Pollard's iteration function has an average value of $E(\mu+\lambda) \approx 1.37\sqrt{|G|}$. The reported $E(\mu+\lambda)$ for prime order subgroups of $(\mathbb{Z}/p\mathbb{Z})^*$ is $1.55\sqrt{|G|}$ and $1.60\sqrt{|G|}$ for prime order subgroups of of $E(\mathbb{F}_p)$.

2.1.3 Floyd's Cycle-finding Algorithm

In order to minimise the storage requirement, a collision-detection algorithm can be applied with a small penalty in the running time. Collision-detection algorithms do not exploit the group structure and are generic. In Pollard's paper, Floyd's algorithm is applied. It compares each pair of Y_i and Y_{2i} for $i \geqslant 1$. Floyd's algorithm is based on the following fact.

Theorem 2.3 (Knuth (1997))**.** *For a periodic sequence $Y_0, Y_1, Y_2 \cdots$, there exists an $i > 0$ such that $Y_i = Y_{2i}$ and the smallest such i lies in the range $\mu \leqslant i \leqslant \mu + \lambda$.*

Floyd's algorithm uses only a small constant amount of storage. The best running-time requires μ iterations and the worst takes $\mu+\lambda$ iterations. Under the assumption that $f : G \to G$ behaves like a truly random mapping, the expected number of iterations before reaching a match is $\sqrt{\pi^5|G|/288} \approx 1.03\sqrt{|G|}$. In Floyd's algorithm, there are three evaluations and one comparison in each iteration. Hence on average there are $1.03\sqrt{|G|}$ comparisons and $3.09\sqrt{|G|}$ evaluations.

2.2 Advances in Iteration Functions

In this subsection, we consider some recent advances and developments in iteration functions.

2.2.1 Pollard's Generalized Function

We slightly change the rules defining the function. Let $M = g^m$ and $N = h^n$ where m, n are two random elements chosen from $[1, |G|]$, denoted as m, $n \in_R [1, |G|]$. We partition G into 3 sets G_1, G_2, G_3 with similar sizes. Let $f_{PG} : G \to G$ be a mapping,

$$f_{PG}(Y) \equiv \begin{cases} M \cdot Y & (\text{mod } p) \quad Y \in G_1 \\ Y^2 & (\text{mod } p) \quad Y \in G_2 \\ N \cdot Y & (\text{mod } p) \quad Y \in G_3 \end{cases} \tag{2.3}$$

Teske (1998) found that the variance of the performance in Pollard's generalized walk (or iteration function) is smaller than that for Pollard's original function. Therefore this function can be regarded as a controlled version of Pollard's original walk (Teske 1998). The reported $E(\mu + \lambda)$ is $1.62\sqrt{|G|}$ for subgroups of $E(\mathbb{F}_p)$. We cannot find reported results for groups $(\mathbb{Z}/p\mathbb{Z})^*$ and hence we will fill this gap in Section 3.

2.2.2 Teske's Adding-walk

Teske (1998) proposed a better iteration function by applying more arbitrary multipliers. Assume that we are using r partitions (multipliers). We generate $2r$ random numbers,

[1] Initially $Y_0 = 1, a_0 = 0, b_0 = 0$.

$$m_i, n_i \in_R \{1, 2, \cdots |G|\}, \text{ for } i = 1, 2, \cdots, r \quad (2.4)$$

Then we precompute r multipliers $M_1, M_2, \cdots M_r$ where,

$$M_i = g^{m_i} \cdot h^{n_i}, \text{ for } i = 1, 2, \cdots, r \quad (2.5)$$

Define a hash function,

$$v : G \to \{1, 2, \cdots r\} \quad (2.6)$$

This completes the precompute stage. Then the iteration function $f_{TA} : G \to G$ is,

$$f_{TA}(Y) = Y \cdot M_{v(Y)}, \text{ where } v(Y) \in \{1, 2, \cdots r\} \quad (2.7)$$

The indices are updated by,

$$\begin{aligned} a_{i+1} &= a_i + m_{v(Y_i)} \\ b_{i+1} &= b_i + n_{v(Y_i)} \end{aligned} \quad (2.8)$$

Based on the work of Hildebrand (1994), Horwitz & Venkatesan (2002), we have the following theorem to show that the performance of adding-walk is provably good.

Theorem 2.4 (Teske (2001))**.** *Let G be a finite abelian group of prime order. Assume that we work with an r adding-walk together with an independent hash function where $r \geq 16$. Then the average number of iterations before a collision occurs, divided by $\sqrt{|G|}$, is approximately independent of $|G|$. In addition, if $r > 16$ then the average number of iterations is bounded by $1.45\sqrt{|G|}$ when using Teske's modified cycle-finding algorithm.*

The reported $E(\mu + \lambda)$ is $1.29\sqrt{|G|}$ for subgroups of $E(\mathbb{F}_p)$, which is close to the theoretically optimal bound $1.25\sqrt{|G|}$ in Theorem 2.2.

2.2.3 Teske's Mixed-walk

Teske proposed another method named mixed-walk (Teske 1998) which has a similar performance to the adding-walk. It uses a mixture of the adding-walk and some squaring steps, similar to Pollard's iteration function. Assume that we are using r multipliers in the adding-walk and q squaring steps. The pseudo-random function $f_{TM} : G \to G$ is defined as follows,

$$f_{TM}(Y) = \begin{cases} Y \cdot M_{v(Y)} & v(Y) \in \{1, 2, \cdots r\} \\ Y^2 & \text{Otherwise} \end{cases} \quad (2.9)$$

Experimental results show that $r \geq 16$ plus $q/r \approx 0.25$ yields a performance comparable to that of a truly random walk. A mixed-walk of 16 multipliers and 4 squaring steps is reported to have an expected length of $E(\mu + \lambda) \approx 1.3\sqrt{|G|}$.

2.3 Advances in Collision-detection Algorithms

In Floyd's algorithm, some Y_i will be evaluated twice, which is time-consuming. There are faster algorithms. We discuss two of Brent's algorithms (Brent 1980) and a variant (Teske 1998).

2.3.1 Brent's Algorithms

Brent proposed two algorithms (Brent 1980) which are generally 25% faster than Floyd's method. A modified version of them was used in factoring the eighth Fermat number by Brent & Pollard (1981).

Brent's first algorithm (Brent 1980) uses a variable z to keep the values of $Y_{l(i)-1}$ where $l(i) = 2^{\lfloor \log i \rfloor}$. z is compared with Y_i for each iteration and is updated by $z = Y_i$ when $i = 2^x - 1$ for $x = 1, 2, \cdots$ (i is the index of iteration and the base 2 is chosen for ease of implementation). Only one sequence Y_i needs to be computed and the value of z is easily updated. The correctness of this algorithm depends on the following idea.

Theorem 2.5. *For a periodic sequence $Y_0, Y_1, Y_2 \cdots$, there exists an $i > 0$ such that $Y_i = Y_{l(i)-1}$ and $l(i) \leq i < 2l(i)$. The smallest such i is $2^{\lceil \lg \max(\mu+1, \lambda) \rceil} + \lambda - 1$.*

Under the assumption that the iteration function is truly random, an expected number of $1.98\sqrt{|G|}$ iterations for $E(\mu + \lambda)$ is reported (Brent 1980). The number of evaluations is equal to the number of comparisons, and hence the total number of operations is bounded by $3.96\sqrt{|G|}$. If cost of comparisons is insignificant, the algorithm is 30% faster in average than Floyd's algorithm. On the other hand, if comparisons are expensive, the speedup may be compromised.

A second algorithm is given in the same paper (Brent 1980). This algorithm avoids unnecessary comparisons as it is sufficient to compare only when $\frac{3}{2}l(i) \leq i < 2l(i)$. Under the assumption that the iteration function is truly random, the expected number of evaluations is $2.24\sqrt{|G|}$ with an expected number of comparison as $0.88\sqrt{|G|}$. The total number of operations is $3.12\sqrt{|G|}$.

A variation of Brent's algorithms is discussed by Teske (1998). It reduces the number of iterations by using more storage and comparisons. A chain of 8 cells is applied and each cell keeps a triplet (Y_i, a_i, b_i). Initially all the values in cells are Y_0 and is updated according to the following rules. At the i-th iteration, we compare the current value Y_i with previous values in the cells. If they are not equal, we check whether i is greater than 3 times the index of the element in the first cell. If this is true, we put current Y_i into the last cell, remove the element in first cell and then shift the other cells to the previous cell. Under the assumption that the function is truly random, the expected number of iterations is about $1.42\sqrt{|G|}$. For each iteration, there is one evaluation and eight comparisons.

2.4 Summary

We summarize the performance of collision-detection algorithms, making the assumption that the iteration function is truly random. We also compile a table including the performance of iteration functions, which is based on the reported experimental results. In the first table, the columns represent algorithms, number of expected iterations, evaluations and comparisons. In the second table, the columns denote iteration functions, multiplicative groups $(\mathbb{Z}/p\mathbb{Z})^*$, prime order subgroups of $(\mathbb{Z}/p\mathbb{Z})^*$ and prime order subgroups of $E(\mathbb{F}_p)$. $f_{TA[20]}$ denotes Teske's adding-walk with 20 multipliers and $f_{TM[16:4]}$ denotes Teske's mixed-walk with 16 multipliers plus 4 squaring steps. All the data in the table is normalised: $E(\mu + \lambda)$ is divided by $\sqrt{|G|}$.

Remark 2.6. In case where two different experimental results are reported by Teske (1998, 2001), we use first one.

Table 1: Performance of Cycle-finding Algorithms

ALGs	ITERs	EVALs	CMPs
Floyd's	1.03	3.09	1.03
Brent's 1st Alg	1.98	1.98	1.98
Brent's 2nd Alg	2.24	2.24	0.88
Teske's Modified	1.42	1.42	$8 * 1.42$

Table 2: Performance of Iteration Functions

FUNCs	$(\mathbb{Z}/p\mathbb{Z})^*$	$S \leq (\mathbb{Z}/p\mathbb{Z})^*$	$S \leq E(\mathbb{F}_p)$
f_P	1.37	1.55	1.60
f_{PG}	-	-	1.62
$f_{TA[20]}$	-	-	1.29
$f_{TM[16:4]}$	-	-	1.30

3 Experimental Investigation

We can find few comparable results for Pollard's rho method and its variants, except those reported by Teske (1998, 2001). There are some gaps in Table 2. In this section, we fill the gaps in Table 2 by an empirical investigation and give some suggestions on better parameters such as starting values and partitioning methods. In addition, we test Teske's iteration function with a comprehensive set of data and verify Teske's results.

3.1 Description of Experiments

To prepare for the experiments, random prime numbers from 3 digits to 15 digits were chosen to give finite fields $\mathbb{F}_p$. We then considered the groups $(\mathbb{Z}/p\mathbb{Z})^*$ and subgroups of $(\mathbb{Z}/p\mathbb{Z})^*$ with orders from 3 to 13 digits. We also discuss elliptic curve discrete logarithms. Let $E(\mathbb{F}_p)$ be a finite abelian group formed by the points on an elliptic curve. G is a prime order subgroup of $E(\mathbb{F}_p)$ generated by a point P. Then an instance of the elliptic curve discrete logarithm problem (ECDLP) is stated as follows. Here we write the group operation as $\oplus$.

Definition 3.1 (ECDLP)**.** Given P, $Q \in G$ known, ECDLP is to find the smallest non-negative integer x such that,

$$Q = \underbrace{P \oplus P \oplus \cdots \oplus P}_{x \text{ times}}$$

To generate subgroups of $E(\mathbb{F}_p)$, we produce random elliptic curves over $\mathbb{F}_p$ and then compute the order for each group. Due to the Pohlig-Hellman algorithm (Pohlig & Hellman 1978), we concentrate on the subgroups with largest prime orders. A generator for each subgroup is computed. The number of instances of DLPs or ECDLPs computed is given in Table 3. The first column gives the (sub)groups by the number of decimal digits in their order. The second column is the number of DLPs or ECDLPs computed for each row. The third column gives the number of different starting values Y_0 for each instance of DLP or ECDLP.

Our implementaion is based on C++ using the GNU Multiple Precision Arithmetic Library (GMP). We ran the algorithms over Gentoo Linux on a Pentium 2.4GHz platform. The whole computation took more than a month.

Table 3: Instances of DLPs or ECDLPs

DIGITs	#DLPs (ECDLPs)	STs
3 to 8	200	100
9	100	50
10	50	20
11	50	10
12	50	5
13	50	1

3.2 Iteration Functions

We first discuss the performance of different iteration functions without collision-detection algorithms. The whole sequences generated by the iteration functions were stored. Therefore the groups were restricted to be small. 2500 discrete logarithms over groups of 6-7 digits were computed. All the data in Table 4 denotes the values of $E(\mu + \lambda)$ divided by $\sqrt{|G|}$. The results fill the gaps in Table 2.

Table 4: Performance of Iteration Functions

FUNCs	$(\mathbb{Z}/p\mathbb{Z})^*$	$S \leq (\mathbb{Z}/p\mathbb{Z})^*$	$S \leq E(\mathbb{F}_p)$
f_P	1.37	1.55	1.60
f_{PG}	1.41	1.55	1.62
$f_{TA[20]}$	1.28	1.27	1.29
$f_{TM[16:4]}$	1.30	1.30	1.30

We found that Pollard's original iteration function performed worse than the truly random case (Theorem 2.2). In addition, Pollard's generalized iteration funtion is slightly worse than the original function on average. On the other hand, Teske's adding-walk and mixed-walk iteration functions behave better and mimic random walks. We discuss the choice of parameters in the rho method below.

3.3 Starting Values

The value assigned to Y_0 for the iteration function $Y_{i+1} = f(Y_i)$ is called the starting value of the sequence. We can use a fixed value for all DLPs (such as $Y_0 = 1$) or generate a random starting values using powers of g and h. We investigate the potential impacts of different types of starting values, which does not seem to have been done before. The pseudo-random functions are either Pollard's original function or Teske's adding-walk using 20 multipliers. The collision-detection algorithm is Brent's second algorithm. In addition, we adopt the partitioning method used in Pollard's original function[2]. For fixed starting values, we compute 4500 instances for DLPs and ECDLPs. The mean values of results are normalized by $\sqrt{|G|}$ in Table 5.

Table 5: Impact of Initial Values

Groups	Functions	Fixed	Random
$(\mathbb{Z}/p\mathbb{Z})^*$	f_P	2.55	2.50
	$f_{TA[20]}$	2.28	2.28
$(\mathbb{Z}/p\mathbb{Z})^*$ subgroups	f_P	2.95	2.84
	$f_{TA[20]}$	2.27	2.26
$E(\mathbb{F}_p)$ subgroups	f_P	2.88	2.92
	$f_{TA[20]}$	2.28	2.25

Although it seems to lose the advantage of randomness, choosing $Y_0 = 1$ is not significantly worse

[2] Partitioning methods will be discussed in the next part.

than choosing Y_0 at random. However, the variance is smaller in the latter case.

It seems there is no direct way to apply random initial values with Pollard's iteration function (or similarly in Teske's mixed-walk). We may need to store some auxiliary variables and update them. For example in $(\mathbb{Z}/p\mathbb{Z})^*$, we have a collision if $g^i h^j Y_0^x \equiv g^{i'} h^{j'} Y_0^y \pmod{p}$. If Y_0 is not 1, we have to update the powers of Y_0 during the procedure. A random initial value is applicable for Teske's adding-walk function. We assume random starting values in the following sections.

3.4 Partitioning Methods

An important assumption in Theorem 2.4 is that the partitioning method is independent. Here the independence means the performance of the iteration function is not affected by the the properties of the partitioning method. We will consider the potential impact of partitioning methods in this part. As we will see later, the choice may have a strong influence on the performance.

A partitioning method maps values of Y_i into different rules in the iteration function, which behaves like a hash function. In Pollard's iteration function, a partition of size three is used. This is extended to N partitions in Teske's functions. Pollard's partitioning rule is $R = \lceil N \times Y_i/|G| \rceil$ where R is the index of the rule chosen and $|G|$ is the order of the group. This method depends mainly on the high-order bits of Y_i. An alternative, the division method, uses the lower-order bits of Y_i, that is $R = (Y_i \mod N)+1$. Another more complicated method suggested by Teske (2001) is Knuth's multiplicative hash function (Knuth 1981). The principle is as follows. Assume that the partition we want to produce is $v : G \to \{1, \cdots, N\}$ where N denotes the number of partitions. Let A be a rational approximation of the golden ratio $\frac{\sqrt{5}-1}{2}$. Define $u(g) = A \cdot g - \lfloor A \cdot g \rfloor$ where g denotes an element in the group. Then the partitioning method is defined by $v(g) = \lceil u(g) \cdot N \rceil$.

We empirically investigated the impacts of different partitioning algorithms. The pseudo-random functions were either Pollard's original function or Teske's adding-walk with 20 multipliers. The collision-detection algorithms involved include Floyd's algorithm, Brent's algorithms and Teske's algorithm. As there are two iteration functions and four cycle-finding algorithms, we discuss eight combinations of them. For each combination, we index Pollard's partitioning method, the division method and Knuth's method as methods $1, 2, 3$ respectively. The Y-axis denotes the number of iterations divided by $\sqrt{|G|}$. The results in groups $(\mathbb{Z}/p\mathbb{Z})^*$ and elliptic curve subgroups are shown in Figure 1 and Figure 2. Note that the different cycle-finding algorithms have different costs per iteration (see Section 2).

For Pollard's iteration functions in groups $(\mathbb{Z}/p\mathbb{Z})^*$, it is much better to apply the original partitioning proposed by Pollard (1978), which uses high-order bits. The other two methods perform worse in this case. In other cases, such as Pollard's iteration functions in subgroups of $E(\mathbb{F}_p)$, it is slightly better to use the division method.

3.5 Choice of Parameters in Teske's Functions

We have discussed impacts of initial values and performance of different partitioning methods. In this part, we consider how the performance is affected by the parameters in Teske's adding-walk and mixed-walk functions. DLPs in prime order subgroups of

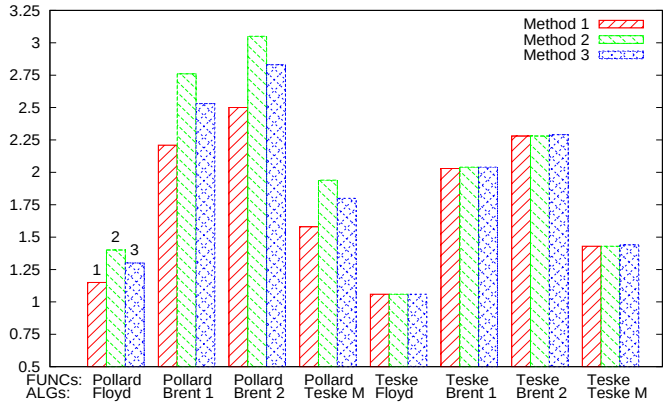

Figure 1: Partitioning Methods in Groups $(\mathbb{Z}/p\mathbb{Z})^*$

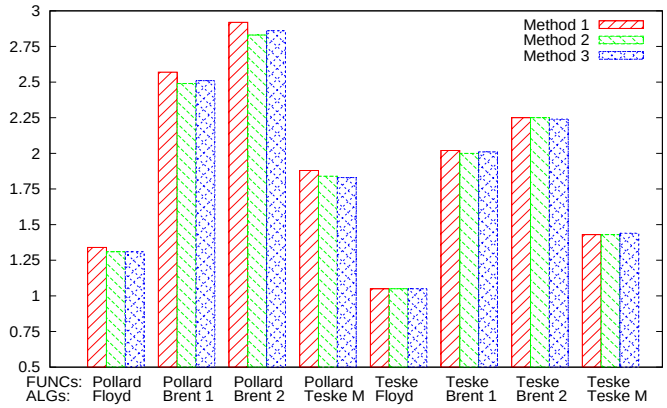

Figure 2: Partitioning Methods in Subgroups of $E(\mathbb{F}_p)$

$(\mathbb{Z}/p\mathbb{Z})^*$ can be considered as analogues of ECDLPs in prime order subgroups of $E(\mathbb{F}_p)$. The discrete logarithm problems considered are defined in groups $(\mathbb{Z}/p\mathbb{Z})^*$ and the largest prime order subgroups of $E(\mathbb{F}_p)$.

3.5.1 Groups $(\mathbb{Z}/p\mathbb{Z})^*$

We discuss the choice of parameters in Teske's function in the groups $(\mathbb{Z}/p\mathbb{Z})^*$. Teske's modified collision-detection algorithm is applied. Theorem 2.4 claims that the number of iterations is bounded by $1.45\sqrt{|G|}$ for Teske's adding-walk with $r \geq 16$ multipliers. The performance of different values of r is plotted in Figure 3. The X-axis denotes the number of multipliers used in adding-walk and the Y-axis denotes the number of iterations divided by $\sqrt{|G|}$.

The empirical results verify Theorem 2.4. We were also able to verify that the performance is generally better using a larger number of partitions. Considering the initialization cost as well, a partition number of 20-60 is a reasonable value. In addition, it has been suggested that mixed-walk with ratios q/r between $1/4$ and $1/2$ with $r \geq 16$ may yield a good performance (Teske 1998). Our experimental results do not support this suggestion. We found that mixed-

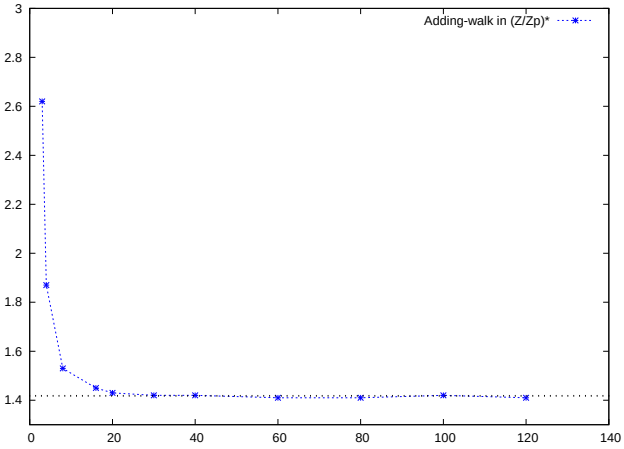

Figure 3: Performance of Adding-walk in Groups $(\mathbb{Z}/p\mathbb{Z})^*$

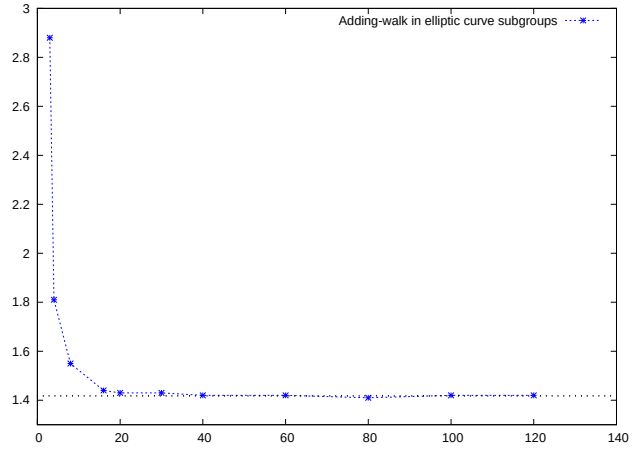

Figure 4: Performance of Adding-walk in Subgroups of $E(\mathbb{F}_p)$

walks with ratios smaller than or equal to 1/4 behave slightly better than those with ratios between 1/4 and 1/2. To illustrate this, the performance for various algorithms is tabulated in Table 6. The columns give the ratios applied, number of multipliers, squaring steps and iterations. As usual, the data in last column is normalized by $\sqrt{|G|}$.

Table 6: Performance of Mixed-walk in Groups $(\mathbb{Z}/p\mathbb{Z})^*$

RATIOs	MULTs	SQRs	ITERs
0.25	16	4	1.46
0.25	20	5	1.45
0.25	40	10	1.44
0.25	60	15	1.44
0.10	20	2	1.45
0.20	20	4	1.47
0.40	20	8	1.50
0.50	20	10	1.51
0.60	20	12	1.53
0.80	20	16	1.57

3.5.2 Prime Order Subgroups of $E(\mathbb{F}_p)$

We discuss the choice of parameters in Teske's function in the subgroups of $E(\mathbb{F}_p)$. Teske's modified collision-detection algorithm is applied. The number of iterations in Figure 4 are bounded by $1.45\sqrt{|G|}$ for Teske's adding-walk function with $r \geq 16$ multipliers. This verifies the effectiveness of Teske's function in the ECDLP case. Similarly a partition number of 20-60 is preferred. The performance of mixed-walk is obtained in the Table 7. For mixed-walk in subgroups of $E(\mathbb{F}_p)$, we arrive a similar result as before. Mixed-walks with ratios q/r smaller or equal to 1/4 with more than 16 multipliers are preferable.

4 Conclusion and Future Work

We discussed efficiency issues regarding Pollard's rho method and its variants for discrete logarithm problems and elliptic curve discrete logarithm problems. We have performed an empirical investigation to fill the current gaps in the literature, suggested better parameters for iteration functions and revisited Teske's adding-walk and mixed-walk functions.

Table 7: Performance of Mixed-walk in Subgroups of $E(\mathbb{F}_p)$

RATIOs	MULTs	SQRs	ITERs
0.25	16	4	1.48
0.25	20	5	1.47
0.25	40	10	1.44
0.25	60	15	1.44
0.10	20	2	1.45
0.20	20	4	1.46
0.40	20	8	1.49
0.50	20	10	1.52
0.60	20	12	1.54
0.80	20	16	1.58

In the previous sections, we have used the assumption that the partitioning method is independent of the iteration function. Finding a way to prove this would be an advance. In addition, the experimental results suggest that Teske's mixed-walk behaves as well as the adding-walk. While the performance of the adding-walk is supported by some theoretical results, we find no easy way to analyze the behavior of the mixed-walk. A potential way to achieve this might be based on the recent work of Miller & Venkatesan (2006) and Kim et al. (2007).

Acknowledgements

We would like to thank the anonymous referees for their helpful comments.

References

Brent, R. P. (1980), 'An improved Monte Carlo factorization algorithm', *BIT* **20**(2), 176–184.

Brent, R. P. & Pollard, J. M. (1981), 'Factorization of the eighth Fermat number', *Mathematics of Computation* **36**, 627–630.

Coppersmith, D., Odlyzko, A. M. & Schroeppel, R. (1986), 'Discrete logarithms in $GF(p)$', *Algorithmica* **1**(1), 1–16.

Diffie, W. & Hellman, M. E. (1976), 'New directions in cryptography', *IEEE Trans. Inform. Theory* **IT-22**, 644–654.

Gamal, T. E. (1985), 'A public key cryptosystem and a signature scheme based on discrete logarithms', *IEEE Trans. Inform. Theory* **31**, 469–472.

Harris, B. (1960), 'Probability Distribution Related to Random Mappings', *Ann. Math. Statist.* **31**, 1045–1062.

Hildebrand, M. (1994), 'Random walks supported on random points of Z/nZ', *Probability Theory and Related Fields* **100**(2), 191–203.

Horwitz, J. & Venkatesan, R. (2002), Random Cayley digraphs and the discrete logarithm, *in* 'Algorithmic Number Theory Symposium V, ANTS-V (LNCS 2369)', pp. 100–114.

Kim, J. H., Montenegro, R. & Tetali, P. (2007), 'A near optimal bound for Pollard's rho to solve discrete log', IEEE Proc. of the Foundations of Computer Science (FOCS), 2007, Providence, RI, to appear.

Knuth, D. E. (1981), *The Art of Computer Programming*, Vol. 3, 2nd edn, Addison-Wesley, Reading, Mass.

Knuth, D. E. (1997), *The Art of Computer Programming*, Vol. 2, 3nd edn, Addison-Wesley, Reading, Mass.

Miller, S. D. & Venkatesan, R. (2006), Spectral analysis of Pollard rho collisions, *in* 'Algorithmic Number Theory Symposium (ANTS VII), LNCS 4076, Springer-Verlag, 573-581'.

Pohlig, S. C. & Hellman, M. E. (1978), 'An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance', *IEEE Trans. Inform. Theory* **IT-24**(1), 106–110.

Pollard, J. M. (1978), 'Monte Carlo methods for index computation mod p', *Mathematics of Computation* **32**, 918–924.

Teske (1998), Speeding up Pollard's rho method for computing discrete logarithms, *in* 'Algorithmic Number Theory Symposium (ANTS IV), LNCS 1423, Springer-Verlag, 541-553'.

Teske, E. (2001), 'On random walks for Pollard's rho method', *Mathematics of Computation* **70**(234), 809–825.

Verifying Michael and Scott's Lock-Free Queue Algorithm using Trace Reduction

Lindsay Groves

School of Mathematics, Statistics and Computer Science,
Victoria University of Wellington,
Wellington, New Zealand
Email: lindsay@mcs.vuw.ac.nz

Abstract

Lock-free algorithms have been developed to avoid various problems associated with using locks to control access to shared data structures. These algorithms are typically more intricate than lock-based algorithms, as they allow more complex interactions between processes, and many published algorithms have turned out to contain errors. There is thus a pressing need for practical techniques for verifying lock-free algorithms and programs that use them.

In this paper we show how Michael and Scott's well known lock-free queue algorithm can be verified using a trace reduction method, based on Lipton's reduction method. Michael and Scott's queue is an interesting case study because, although the basic idea is easy to understand, the actual algorithm is quite subtle, and it demonstrates several way in which the basic reduction method needs to be extended.

Keywords: Concurrency, verification, lock-free, linearisability, reduction

1 Introduction

Increasing use of concurrent software designs has prompted the development of *lock-free* algorithms to implement concurrent data structures to avoid many of the problems associated with the use of locks. Rather than avoid interference using mutual exclusion, lock-free algorithms must behave correctly in the presence of interference, and usually rely on strong synchronisation primitives such as Compare and Swap (CAS). These algorithms tend to be very subtle, and hard to get right; however, proofs of correctness for such algorithms tend to be either so high level as to be unconvincing, or so detailed as to be unenlightening.

In this paper, we consider a slightly simplified version of Michael and Scott's lock-free queue algorithm (Michael & Scott 1998), which is similar to that included in the Java concurrency library. We present a proof that this algorithm is linearisable (Herlihy & Wing 1990), using an extension of the reduction approach proposed by Lipton (Lipton 1975), and further developed by Lamport, Cohen and others (Lamport & Schneider 1989, Cohen & Lamport 1998, Lamport 1990). In this approach, we show that any concurrent execution involving a shared data object, such as a queue, can be transformed into an equivalent execution in which the operations on that object are executed without interruption, and that such uninterrupted executions correctly implement the abstract semantics for the object. This approach allows us to present a proof in sufficient detail to be convincing, highlighting the reasons why the algorithm is correct, without getting lost in a morass of minute detail, and indicating clearly what else needs to be proved to provide a more detailed proof.

We begin in Section 2 by giving an intuitive description of the algorithm and taking a brief look at the code. Then, in Section 3, we discuss our correctness criterion, linearisability, and outline how we prove linearisability using reduction. In Section 4, we present the verification, explaining in more detail how the reduction method is applied and how it is extended in order to verify Michael and Scott's algorithm, and end in Section 5 with some conclusions and comments on related and future work.

2 Michael and Scott's Lock-Free Queue Algorithm

Michael and Scott (Michael & Scott 1998) describe an algorithm which implements a shared queue supporting ENQUEUE and DEQUEUE operations that can be performed concurrently by a finite set of processes. Their algorithm is *lock-free*, which means that no process is ever forced to wait for another process to complete a queue operation. This property precludes the use of traditional synchronisation mechanisms such as locks and semaphores to avoid interference between processes; instead the algorithm is designed to work correctly in the presence of interference, which is detected by using Compare and Swap (CAS) instructions to conditionally update shared locations.

The implementation uses a linked list, with a dummy node at the head, and *Head* and *Tail* pointers. Each node has a *value* field, holding the values stored in the queue in the order they were added to the queue, and a *next* field, linking nodes in the list. Using a dummy node ensures that *Head* and *Tail* are always non-null, which reduces the number of special cases that would otherwise be required; its value is not part of the queue. *Head* always points to the dummy node, and in a quiescent state (i.e. when no operation is in progress) *Tail* points to the last node in the list, as illustrated in Figure 1, which shows an empty queue and a queue containing values *a*, *b* and *c*.

The ENQUEUE and DEQUEUE operations follow a common pattern in which each operation repeatedly attempts to perform its update, succeeding only if the operation is performed without interference. At each attempt, an operation takes a "snapshot" of the part of the global state that it wishes to update, uses this in local computations to prepare a new value, and then uses a CAS to attempt the update. $CAS(loc, old, new)$ atomically compares the contents of the shared location *loc* with the "expected" value, *old*, and if they are the same, *succeeds*, storing the new value, *new*, into the location and returning *true*, and otherwise *fails*, returning *false* and leaving the memory unchanged.

The central problem in designing algorithms based on CAS is to arrange that the shared data structure can be updated atomically using a single CAS operation, with its

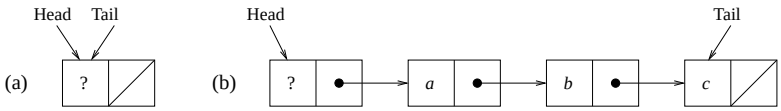

Figure 1: Basic queue representation

test determining when the update is safe. This is easy to do, say, in the case of a shared stack (e.g. (Treiber 1986, Michael & Scott 1998)), where we only need to update a single shared location (the top of stack pointer). But it is not so simple for a queue represented as a linked list. After creating a new node, an ENQUEUE operation must update two shared locations — to make the *next* pointer of the last node point to the new node, and make the *Tail* pointer point to the new node — but we can't perform both updates using a single CAS.

In performing an ENQUEUE, Michael and Scott append the new node using a CAS, and allow other processes to observe the data structure at a point where this update has been performed but the *Tail* pointer has not been advanced. This means that *Tail* may "lag" behind the actual end of the list — this situation is illustrated in Figure 2, which shows a queue containing *a*, *b* and *c*, and one containing just *c*. To avoid other processes being blocked waiting for a process to complete an ENQUEUE, as required for lock-freedom, any process which observes that *Tail* is not pointing to the end of the list attempts to advance *Tail*, effectively completing the operation of the process that performed the append. This ensures that *Tail* never lags behind the end of the list by more than one node.

DEQUEUE is implemented by advancing the *Head* pointer, provided that the queue is not empty, so the node that used to hold the first element of the queue becomes the dummy node. The DEQUEUE operation now has to handle the situation shown in Figure 2(b), where *Head* and *Tail* point to the same node, but the queue is not empty, and in this case checks whether *Tail* is lagging before attempting to perform its update.

The declarations and initialisation are shown in Figure 3, and pseudocode for the ENQUEUE and DEQUEUE operations is given in Figure 4. This code is essentially the same as that given in (Michael & Scott 1998), apart from a few changes in notation and simplifications to make our reasoning easier and more concise. In particular, we assume that a single queue is being implemented, and thus treat *Head* and *Tail* as global variables, encapsulated within a module implementing the queue, rather than as components of a record accessed via a pointer. We also use Algol/Pascal/Ada-like notation for assignment and equality (i.e. $:=$ and $=$, instead of $=$ and $==$), Ada-like parameter modes (**in** and **out**), and assume automatic pointer dereferencing, whereas Michael and Scott use C-like notation.

The most significant difference is that, like the version included in the Java concurrency library (JSR 166), we do not explicitly free popped nodes. This means that heap locations are not reused unless the algorithm is executed on a system with automatic garbage collection (as is the case in Java), and that modification counts are not required.

Looking at the code, some aspects are readily understood as they are similar to a sequential queue implementation. The declarations should be self explanatory, given the previous description of the data structure used, noting just that *new_node*() is assumed to allocate a new node and return a pointer to it.

Lines E1–E3 of ENQUEUE allocate a new node and initialise its fields. Line E9 attempts to append the new node, provided that *Tail* is not lagging. Line E13 attempts to advance *Tail* if it is found to be lagging before appending the new node, and line E17 attempts to advance *Tail* after appending the new node. The operation retries if either of the tests at lines E7 and E8, or the CAS at E9, fails.

Lines D2–D8 of DEQUEUE check to see whether the queue is empty, and if so returns *false*. If not, line D13 attempts to remove the first node from the queue by advancing *Head*, after reading the value to be returned in line D12. Line D10 attempts to advance *Tail* in the special case described above, where *Head* and *Tail* are the same but the queue is not empty.

While this much can be appreciated quite easily, it is not so clear exactly why the various tests are required or why they are ordered as they are: for example, one might consider the effect of deleting the tests at E7 and D7, and the CASes at E17 and D10, or whether DEQUEUE can be modified so as to avoid accessing *Tail*. So, although one can easily understand the basic ideas underlying the algorithm, it is not entirely obvious that the algorithm is correct, nor what changes could be made to the algorithm without affecting its correctness.

3 Proving Linearisability by Trace Reduction

When multiple processes perform concurrent operations on a shared object, we cannot simply define the correctness of these operations in terms of the state of the object before and after a process performs an operation. For example, when a process performs an ENQUEUE operation, there is no guarantee that the values that were in the queue when the enqueue operation began will still be there when the process gets to add its value to the queue, or that the enqueued value will still be in the queue when the enqueue operation is completed.

The standard safety property for concurrent data structures is *linearisability* (Herlihy & Wing 1990), which requires that each operation on the shared data structure appears to occur instantaneously at some point (called its *linearisation point*) between its invocation and its response, and that the effect of the operation be correct with respect to the state immediately before and after this point. The requirement that an operation's linearisation point be between its invocation and its response ensures that the order of non-concurrent operations is preserved; i.e. if an operation op_1 is completed before another operation op_2 begins, then the linearisation point for op_1 must precede that for op_2.

This condition is sometimes called *atomicity* (e.g. (Lynch 1996, Hesselink 2002)), however, we use that term to refer to the weaker requirement, that an operation appear to occur instantaneously, with no reference to the semantics of an abstract operation being implemented, as in (Lamport & Schneider 1989, Flanagan & Qadeer 2003, Sasturkar et al. 2005).

More precisely, a shared object *O* is *linearisable* if, for every execution of a concurrent system involving *O*, there is an "equivalent" legal sequential history, in which the order of non-concurrent operations is preserved. A *history* (or *trace*) is a sequence of invocations and responses occurring in an execution, and is *sequential* if every response is immediately preceded by an invocation of the same operation by the same process, and *legal* if each invocation-response pair is correct with respect to the abstract semantics for the object. Two histories are *equivalent* if they contain the same sequence of invocations and responses. Thus, we can prove that an implementation of a shared object is linearisable by showing, for any concurrent execution, how to construct an equivalent legal sequential history which respects the abstract semantics for that object and preserves the order of non-concurrent operations.

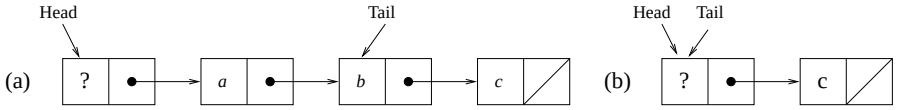

Figure 2: Queue representation with *Tail* lagging

```
type pointer = pointer to node
type node = (value: data_type, next: pointer)
var Head: pointer
var Tail: pointer

initialisation:
    Head := new_node()
    Head.next := null
    Tail := Head
```

Figure 3: Declarations and initialisation

In previous work (Doherty et al. 2004, Colvin et al. 2005, Colvin & Groves 2005, Colvin et al. 2006), we have proved linearisability of several lock-free algorithms using simulation between two labelled transition systems, one modelling the abstract specification and one modelling the implementation.[1] In the simulation approach, we step through an arbitrary execution of the concrete (implementation) model, considering all possible actions that could be taken at each step, and show how to construct a corresponding execution of the abstract (specification) model, using a simulation relation to show that the concrete and abstract states are related appropriately at each step. The simulation relation typically requires that each process is performing the same operation (if any) in both models, that the data structure in the implementation represents the same abstract value as in the abstract model, and that local variables in each process have "appropriate" values.

A well known complication with simulation proofs is that while we are often able to construct the required abstract execution by stepping forwards through the concrete execution (which is called *forward* or *downward* simulation), it is sometimes necessary to instead step backwards (which is called *backward* or *upward* simulation) or to use a combination of both (Jifeng et al. 1986, Lynch & Vaandrager 1995). Although it is widely believed that backward simulation is rarely required in practice (for example, backward simulation rules for data refinement were not defined for Z until 1997 (Stepney et al. 1998), or for B until 2003 (Dunne 2003)), backward simulation turns out to be required frequently in verifying lock-free algorithms, and several of our verifications, including our verification of a version of Michael and Scott's queue (Doherty et al. 2004), have used backward simulation, usually in conjunction with forward simulation.

While the simulation approach has proved to be effective in these verifications, and to be amenable to mechanisation (using PVS), this approach has several drawbacks:

- Translating the algorithm and specification into a transition system formalism obscures the algorithmic structure of the algorithm being verified, so it is often hard to see how verification conditions relate to the algorithm.
- Many of the verification conditions are a consequence of the formalism, rather than the algorithm (e.g. ones to do with program counters).
- The verification has to deal with both the basic operation of the data structure being implemented and the effects of concurrency, whereas it may be more convenient to separate these (this can be avoided by using an extra simulation step, but it is debatable whether this is worthwhile given the overhead introduced).
- Each abstract operation always consists of three steps (an invocation, an internal action corresponding to the linearisation point, and a response), so most steps of the implementation are internal steps, and most of the proof effort is in proving various invariants about the shared and local variables.

The net effect is that the proof requires so much detail that it is hard to identify the essential arguments on which the correctness of the algorithms relies, and it is hard to present such a proof in a way that conveys the important insights into why the algorithm is correct without getting bogged down in the details.

The approach we take in this paper is an attempt to present a proof which is both more concise and more illuminating than our earlier simulation proofs, while also being sufficiently formal to be compelling. Instead of constructing an equivalent legal sequential history by translating one action of the implementation at a time, as in the simulation approach, we first construct an equivalent sequential execution, in which each operation o the abstract object is executed without interruption, by translating an entire operation of the abstract object at a time, and then show that this sequential execution correctly implements the abstract operation. This approach is based on the reduction approach described initially by Lipton (Lipton 1975), and further developed by Lamport, Cohen, and others (Doeppner, Jr. 1977, Lamport & Schneider 1989, Lamport 1990, Cohen & Lamport 1998, Cohen 2000) for synchronisation based on mutual exclusion. We have shown how this approach can be extended to handle lock-free algorithms (Groves 2007*a*) and used it in a constructive way to derive an implementation of a scalable concurrent stack implementation (Groves & Colvin 2006*a*). The rest of this section outlines the basic ideas of reduction, and its use in proving linearisability. We will explain the approach in more detail as we work through the verification in Section 4.

Given a system in which a finite set of processes, *PROC*, operate on a shared queue, our aim, as indicated above is to show that any concurrent execution α of the system is equivalent to an execution α' in which every operation on the queue is performed without interruption, and that the effect of such an uninterrupted execution correctly implements the abstract queue semantics. The key to the reduction approach is therefore to show that the actions in a concurrent execution can be rearranged so that the steps of each operation are executed contiguously. By repeating this transformation for each operation in α, we can then produce an execution in which every operation is executed without interruption.

Suppose that α is an execution of the form $\beta_0\, a_1\, \beta_1\, \ldots\, a_n\, \beta_n$, where $a_1, \ldots, a_n$ are the atomic actions comprising an execution of a queue operation *op* by process *p*, $\beta_0 \ldots \beta_n$ are sequences of actions, and $\beta_1 \ldots \beta_{n-1}$ contain no *p*-actions. We wish to show that there is an "equivalent" execution $\alpha' = \beta_0\, \ldots\, \beta_{k-1}\, a_1\, \ldots\, a_n\, \beta_k\, \ldots\, \beta_n$ in which the atomic steps of *op* are executed contiguously. Thus, we must be able to show that it makes no difference if actions $a_1, \ldots, a_{k-1}$

[1] The transition systems we used were a simplified form of Input/Output Automata (IOAs) (Lynch & Vaandrager 1995), which were convenient because simulation between IOAs is defined in terms of trace inclusion rather than in terms of states, but most other labelled transition system formalisms could be used instead.

```
ENQUEUE(in value: data_type)
E1:  node := new_node()
E2:  node.value := value
E3:  node.next := null
E4:  loop
E5:    tail := Tail
E6:    next := tail.next
E7:    if tail = Tail then
E8:      if next = null then
E9:        if CAS(tail.next, next, node) then
E10:         break
E11:       endif
E12:     else
E13:       CAS(Tail, tail, next)
E14:     endif
E15:   endif
E16: endloop
E17: CAS(Tail, tail, node)
```

```
DEQUEUE(out pvalue: data_type): boolean
D1:  loop
D2:    head := Head
D3:    tail := Tail
D4:    next := head.next
D5:    if head = Head then
D6:      if head = tail then
D7:        if next = null then
D8:          return false
D9:        endif
D10:       CAS(Tail, tail, next)
D11:     else
D12:       pvalue := next.value
D13:       if CAS(Head, head, next) then
D14          break
D15:       endif
D16:     endif
D17:   endif
D18: endloop
D19: return true
```

Figure 4: Queue operations

are executed after the relevant β_i (i.e. for $1 \leq i < k$ and $i \leq j < k$, a_i can be executed after b_j), and actions $a_{k+1}, \ldots, a_n$ are executed before the relevant β_i (i.e. for $k \leq i \leq n$ and $k < j < i$, a_i can be executed before b_j). Since all of the actions of *op* are grouped at the position of a_k, any operation that begins before a_1 (ends after a_n) in α also begins before a_1 (ends after a_n) in α'. Thus, a_k can be taken as the linearisation point for *op*, and the order of non-current operations is preserved, as required for linearisability, so we don't need to explicitly model invocations and responses.

To define the idea of rearranging the steps in an execution more precisely, we write $\sigma \xrightarrow{a} \tau$ to mean that execution of action *a* may take the system from state σ to state τ. For a sequence of actions, $\alpha = a_1 \cdots a_n$, we write $\sigma \xrightarrow{\alpha} \tau$ to mean that there is a sequence of states $\rho_0, \cdots, \rho_n$ such that $\rho_0 = \sigma$, $\rho_n = \tau$, and $\rho_{i-1} \xrightarrow{a_i} \rho_i$, for all $i \in 1 \,..\, n$. For sequences of actions, α and β, we write $\alpha \leq \beta$ to mean that for any states σ and τ, $\sigma \xrightarrow{\alpha} \tau$ implies $\sigma \xrightarrow{\beta} \tau$. An action *a* is *enabled* in a state σ if there exists a state τ such that $\sigma \xrightarrow{a} \tau$.

If $ab \leq ba$, we say that *a right commutes* with *b*, and *b left commutes* with *a*. If *a* right commutes and left commutes with *b*, we just say *a commutes* with *b*, and write $ab = ba$. We can show that for sequences of actions, $\alpha = a_1 \ldots a_m$ and $\beta = b_1 \ldots b_n$, $\alpha\beta \leq \beta\alpha$ if $a_i b_j \leq b_j a_i$ for all $i \in 1 \,..\, m$ and $j \in 1 \,..\, n$.

Given a system with actions *ACT*, an action *a* is called a *right mover* if it right commutes with every action of every other process (i.e. $a_p b_q \leq b_q a_p$ for all $b \in ACT$ and $p \neq q$), a *left mover* if it left commutes with every action of every other process (i.e. $b_q a_p \leq a_p b_q$ for all $b \in ACT$ and $p \neq q$), and a *both mover* if it is both a right mover and a left mover (i.e. $a_p b_q = b_q a_p$ for all $b \in ACT$ and $p \neq q$).

In showing that actions move in particular ways, we appeal to some standard properties of independent operations. For example:

- An action that only accesses local variables or heap locations accessed via a unique pointer in a local variable is a both mover.
- An action that reads a shared variable commutes with any action that does not assign to that variable.
- An action that assigns to a shared variable commutes with any action that does not refer to that variable.

As presented above, the equivalent sequential execution is obtained by rearranging the actions of the concurrent execution. It has been shown (Wang & Stoller 2005) that this is not sufficient to verify some lock-free algorithms. However, the technique can be extended to extend its applicability (Groves 2007*a*). Firstly, we can use the outcome of CAS and other tests to infer properties of the interleaved actions of other processes. Secondly, we observe that while the steps in the sequential execution need to be steps that could be taken by the implementation when executed without interruption, they do not have to be the same steps as in the concurrent execution. In many lock-free algorithms we need to be able to delete actions. We will see in Section 4 that to verify Michael and Scott's queue algorithm, we also need to be able to modify actions so as to assign an action to a different process.

4 Verification

We now consider how the version of Michael and Scott's algorithm presented in Section 2 can be verified using the trace reduction approach described in Section 3. As outlined earlier, our aim is to show that any concurrent execution can be transformed into one in which the atomic steps of each queue operation are executed contiguously, and that when executed without interruption these operations correctly implement the abstract queue semantics. Here, we focus on the former; the latter involves a straightforward data refinement proof, which we present elsewhere (Groves 2007*b*).

We will regard each assignment, test and CAS as an atomic action, which is reasonable since they all access at most one shared variable. We also assume, as in (Michael & Scott 1998) that allocating a new node can be treated as an atomic action. For convenience, the atomic actions and their labels are shown in Figure 5.

4.1 Commutativity properties

The first step in applying the reduction method is to examine the commutativity properties of the atomic actions. From the general properties given at the end of Section 3, and some other general properties, we can see that:

- Actions E8 (*next* = *null*), D6 (*head* = *tail*) and D7 (*next* = *null*) are both movers, as they only involve local variables. Thus, for any action X and distinct processes, p and q, we have:
  $E8_p\, X_q = X_q\, E8_p$, $D6_p\, X_q = X_q\, D6_p$ and $D7_p\, X_q = X_q\, D7_p$
- Actions E2 (*node.value* := *value*) and E3 (*node.next* := *null*) are both movers, since at the point where they are executed, *node* is a unique

E1	node := *new_node*()
E2	node.value := value
E3	node.next := *null*
E5	tail := Tail
E6	next := tail.next
E7	tail = Tail
E8	next = *null*
E9	CAS(tail.next, next, node)
E13	CAS(Tail, tail, next)
E17	CAS(Tail, tail, node)

D2	head := Head
D3	tail := Tail
D4	next := head.next
D5	head = Head
D6	head = tail
D7	next = *null*
D10	CAS(Tail, tail, next)
D12	*pvalue* := next.value
D13	CAS(Head, head, next)

Figure 5: Atomic actions

pointer (*node* is a new node when it is allocated in E1, and cannot be seen by any other process until it is appended to the end of the list by the CAS at E9), and *value* and *null* are local. Thus, for any action X and distinct processes, p and q, we have:

$E2_p\, X_q = X_q\, E2_p$ and $E3_p\, X_q = X_q\, E3_p$

- Actions E5 (*tail* := *Tail*), E7 (*tail* = *Tail*) and D3 (*tail* := *Tail*) commute with any actions that do not alter *Tail*. Thus, for any action X other than E13 or E17 and distinct processes, p and q, we have:

  $E5_p\, X_q = X_q\, E5_p$, $E7_p\, X_q = X_q\, E7_p$ and $D3_p\, X_q = X_q\, D3_p$

- Actions D2 (*head* := *Head*) and D5 (*head* = *Head*) commute with any action that does not alter *Head*. Thus, for any action X other than D13 and distinct processes, p and q, we have:

  $D2_p\, X_q = X_q\, D2_p$ and $D5_p\, X_q = X_q\, D5_p$

- Actions E6 (*next* := *tail.next*) and D4 (*next* := *head.next*) commute with any action that does not alter the *next* field of a node. Thus, for any action X other than E9 and distinct processes, p and q, we have:

  $E6_p\, X_q = X_q\, E6_p$ and $D4_p\, X_q = X_q\, D4_p$

- Action D12 (*pvalue* := *next.value*) commutes with any action that does not alter the *value* field of a node. Since the only action that alters the *value* field of a node is E2, and we have already shown that E2 is a both mover because it updates *value* via a unique pointer, it follows that D12 commutes with all actions. Thus, for any action X and distinct processes, p and q, we have:

  $D12_p\, X_q = X_q\, D12_p$

  Note that we assume that the value of an **out** parameter is not observable to the caller (or any other process) until the queue operation is completed, so in the case of D12, we can treat *pvalue* as being local.

- Action E9 ($CAS(tail.next, next, node)$) commutes with any action that does not access the *next* field of a node. Thus, for any action X other than E6, E9 or D4 and distinct processes, p and q, we have:

  $E9_p\, X_q = X_q\, E9_p$

  Note that E3 is not included in the list of exceptions since we have already shown that E3 is a both mover.

- Actions E13, E17 and D10 ($CAS(Tail, tail, node)$) commute with any action that does not access *Tail*. Thus, for any action X other than E5, E7, E13, E17, D3 or D10 and distinct processes, p and q, we have:

  $E13_p\, X_q = X_q\, E13_p$, $E17_p\, X_q = X_q\, E17_p$ and $D10_p\, X_q = X_q\, D10_p$.

- Action D13 ($CAS(Head, head, next)$) commutes with any action that does not access *Head*. Thus, for any action X other than D2, D5 or D13 and distinct processes, p and q, we have:

  $D13_p\, X_q = X_q\, D13_p$

- Action E1 (*node* := *new_node*()) is a both mover, since it makes no difference what address is allocated provided that it is previously unused. Thus, for any action X and distinct processes, p and q, we have:

  $E1_p\, X_q = X_q\, E1_p$

4.2 Applying reduction to Michael and Scott's algorithm

Next, we consider how to use these properties to rearrange the steps in a concurrent execution to obtain an equivalent sequential execution. Here we find that these properties are not sufficient to show that Michael and Scott's algorithm is atomic — a completed execution of ENQUEUE or DEQUEUE may contain any number of CAS actions, which may be interleaved with CAS actions of other processes, and the above commutativity properties do not allow us to permute the order of CAS actions. We therefore need to perform a more complex transformation than just reordering the steps of a concurrent execution.

In considering how to transform a concurrent execution into a sequential one, we first observe that any completed execution of a queue operation consists of zero or more "failed" iterations of the loop (i.e. ones where the loop does not exit), preceded in the case of ENQUEUE by three initial actions (E1-E3), and followed by one "successful" iteration (i.e. one where the loop does exit). We also observe that in a sequential execution, every operation succeeds the first time it is attempted, so there are no failed iterations, and *Tail* is always updated by the process that appends a node onto the list (at E17), so E13 and D10 are never executed.

We will show how to transform an arbitrary concurrent execution into this form using three transformations, each of which requires an extension to Lipton's basic method. Firstly, we show that "failed" iterations in which *Tail* is not updated can be deleted; secondly, we show that the remaining actions can be rearranged so that the steps of each operation execution are contiguous, except for "successful" iterations in which *Tail* is not updated; and lastly, we show that actions that advance *Tail* can be performed by any process, and in particular by the process that last appended a node, which allows this exception to be addressed. Finally, we consider how to assemble the remaining fragments into complete operations.

4.3 Primitive paths

In describing these transformations, we need to consider different paths that an operation may take through the code. So we break the code into primitive (loop-free) paths and describe how each path is transformed.

We need to identify a set of primitive paths, each comprising a sequence of atomic actions performed by the

same process, such that every execution of a queue operation can be described as a concatenation of primitive paths. We will segment the code so that each primitive path consists of either a sequence of actions performed before the loop (which only occurs in ENQUEUE) or one iteration of the loop (including actions taken after exiting the loop in the case where the loop in DEQUEUE terminates). We further divide loop iterations into four classes which will be handled differently:

- failed iterations in which *Tail* is not updated (i.e. Enq2, Enq3, Enq5, Deq1, Deq3 and Deq5);
- failed iterations in which *Tail* is updated (i.e. Enq4 and Deq2);
- normal successful iterations, which behave as they would in a sequential execution (i.e. Enq7, Deq4 and Deq6); and
- abnormal successful iterations, which do not behave as they would in a sequential execution (i.e. Enq6).

The resulting paths are shown in Figure 6, and are labelled for later reference. In describing execution paths, we use the line numbers shown in Figure 4 to stand for the action on that line, and indicate whether test and CAS actions succeed or fail by appending $^+$ or $^-$, respectively. Where necessary, we indicate the process that performs an operation by adding a process identifier (usually *p* or *q*) as a subscript (these should not be confused with the numerical subscripts used in describing arbitrary actions and action sequences).

It follows from the semantics of our programming constructs that any execution of ENQUEUE consists of the initial segment (Enq1) followed by zero or more failed iterations (Enq2 to Enq5), followed by a single successful iteration (Enq6 or Enq7). Similarly, any execution of DEQUEUE consists of zero or more failed iterations (Deq1, Deq2, Deq3 or Deq5), followed by a single successful iteration (Deq4 or Deq6). Treating the path names as symbols, we can describe the structure of possible executions of ENQUEUE and DEQUEUE with the following regular expressions:

$$\mathit{Enq}1(\mathit{Enq}2 \mid \mathit{Enq}3 \mid \mathit{Enq}4 \mid \mathit{Enq}5)^*(\mathit{Enq}6 \mid \mathit{Enq}7)$$

$$(\mathit{Deq}1 \mid \mathit{Deq}2 \mid \mathit{Deq}3 \mid \mathit{Deq}5)^*(\mathit{Deq}4 \mid \mathit{Deq}6)$$

4.4 Deleting failed iterations that do not advance *Tail*

We first show that any failed iteration that does not advance *Tail* can be deleted. This is easy to see intuitively — an execution in which an operation is attempted unsuccessfully is indistinguishable from one in which the unsuccessful operation was never attempted.

More precisely, let α be an execution which contains a failed iteration that does not advance *Tail* in ENQUEUE (i.e. Enq2, Enq3, Enq5) or in DEQUEUE (i.e. Deq1, Deq3 or Deq5), and let α' be the result of deleting the steps of this failed iteration from α. Then we wish to show that $\alpha \leq \alpha'$.

We will only consider path Enq2 in detail — the arguments for the other failed iterations are similar.

Path Enq2 is: E5, E6, E7$^-$. Let α be an execution containing an execution of Enq2 as part of a completed execution of ENQUEUE by process *p*, say $\alpha_1\, E5_p\, \alpha_2\, E6_p\, \alpha_3\, E7_p^-\, \alpha_4$, where α_2 and α_3 contain no *p*-actions. Removing $E7_p^-$ from this execution does not alter its effect. These occurrences of $E5_p$ and $E6_p$ can then also be removed — since this occurrence of Enq2 is part of a completed ENQUEUE operation, the next two *p*-actions must be E5 and E6, so the values loaded by these occurrences of E5 and E6 will not be referenced again. Thus, we have:

$$\alpha_1\, E5\, \alpha_2\, E6\, \alpha_3\, E7^-\, \alpha_4 \leq \alpha_1\, \alpha_2\, \alpha_3\, \alpha_4$$

This result means that we can ignore all of the failed iterations that do not advance *Tail*, i.e. Enq2, Enq3, Enq5, Deq1, Deq3 and Deq5. Following this transformation, every execution of ENQUEUE or DEQUEUE has the form described by the following regular expressions:

$$\mathit{Enq}1(\mathit{Enq}4)^*(\mathit{Enq}6 \mid \mathit{Enq}7)$$

$$(\mathit{Deq}3)^*(\mathit{Deq}4 \mid \mathit{Deq}6)$$

Notice that the result of this transformation (like the subsequent ones) is a valid execution of the algorithm.

This transformation can be generalised to show that any failed iteration which has no observable effect can be deleted. Such iterations are called "pure" in (Freund & Qadeer 2005), where a similar approach is used in a static analysis technique for determining atomicity.

4.5 Reducing primitive paths

We now consider the remaining basic paths and attempt to show how a concurrent execution, in which the atomic actions of that path may be interleaved with actions of other processes, can be transformed into one in which the atomic steps of that path are executed without interruption. This uses the basic reduction method, augmented with a more detailed analysis of paths containing CASes, and succeeds for all of the remaining paths except failed iterations that update *Tail* (i.e. Enq4 and Deq2), which are considered further in Section 4.6.

The linearisation point for a completed ENQUEUE is the successful CAS at E9, so we want to move everything before that to the right (or delete it), and everything after (i.e. the CAS at E17) to the left. Similarly, the linearisation point for a completed DEQUEUE returning *true* is the successful CAS at D13, and for a DEQUEUE returning *false* is D3, so we want to move everything before that to the right (or delete it).

The important points can be illustrated by considering five cases; the other cases are detailed in (Groves 2007*b*).

4.5.1 Pre-loop path in ENQUEUE

Path Enq1 is E1, E2, E3, where we have:

E1	node := *new_node*()
E2	node.value := value
E3	node.next := *null*

Let α be an execution containing an execution of Enq1 by process *p*, say $\alpha = \alpha_1\, E1_p\, \alpha_2\, E2_p\, \alpha_3\, E3_p\, \alpha_4$, where α_2 and α_3 contain no *p*-actions. We have shown that E1, E2 and E3 are both-movers, so these actions can be moved right over α_2 and α_3 as required. Thus, we have:

$$\alpha_1\, E1_p\, \alpha_2\, E2_p\, \alpha_3\, E3_p\, \alpha_4 \leq \alpha_1\, \alpha_2\, \alpha_3\, E1_p\, E2_p\, E3_p\, \alpha_4$$

4.5.2 Normal successful iteration in ENQUEUE

Path Enq7 is E5, E6, E7$^+$, E8$^+$, E9$^+$, E17$^+$, where we have:

E5	tail := Tail
E6	next := tail.next
E7$^+$	tail = Tail
E8$^+$	next = *null*
E9$^+$	CAS(tail.next, next, node)$^+$
E17$^+$	CAS(Tail, tail, node)$^+$

Let α be an execution containing an execution of Enq7 by process *p*, say $\alpha = \alpha_1\, E5_p\, \alpha_2\, E6_p\, \alpha_3\, E7_p^+\, \alpha_4\, E8_p^+\, \alpha_5\, E9_p^+\, \alpha_6\, E17_p^+\, \alpha_7$, where α_2 to α_6 contain no *p*-actions.

Enq1	E1, E2, E3	Pre-loop
Enq2	E5, E6, $E7^-$	Failed iteration, not updating *Tail*
Enq3	E5, E6, $E7^+$, $E8^-$, $E13^-$	Failed iteration, not updating *Tail*
Enq4	E5, E6, $E7^+$, $E8^-$, $E13^+$	Failed iteration, updating *Tail*
Enq5	E5, E6, $E7^+$, $E8^+$, $E9^-$	Failed iteration, not updating *Tail*
Enq6	E5, E6, $E7^+$, $E8^+$, $E9^+$, $E17^-$	Normal successful iteration
Enq7	E5, E6, $E7^+$, $E8^+$, $E9^+$, $E17^+$	Abnormal successful iteration
Deq1	D2-D4, $D5^-$	Failed iteration, not updating *Tail*
Deq2	D2-D4, D5+, $D6^+$, $D7^-$, $D10^+$	Failed iteration, updating *Tail*
Deq3	D2-D4, D5+, $D6^+$, $D7^-$, $D10^-$	Failed iteration, not updating *Tail*
Deq4	D2-D4, $D5^+$, $D6^+$, $D7^+$	Normal successful iteration
Deq5	D2-D4, $D5^+$, $D6^-$, D12, $D13^-$	Failed iteration, not updating *Tail*
Deq6	D2-D4, $D5^+$, $D6^-$, D12, $D13^+$	Normal successful iteration

Figure 6: Basic paths for ENQUEUE and DEQUEUE

We can move E8 because it only involves local variables, but the other actions require move careful consideration.

Since E17 succeeds, we know that *Tail* has the same value at E17 as it had at E5; however, we can go further and infer that *Tail* is not modified by α_2 to α_6. To see why, we observe that *Tail* can only be modified by a successful CAS at E13, E17 or D10, and that the last such CAS must set *Tail* to *tail*. We can show that this is impossible, by showing that the program maintains the invariant property that the list contains no cycles and *new_node*() always returns a new node, so advancing *Tail* cannot cause it to return to a previous value. This is called the "ABA freedom property", and holds because we assume that memory is not recycled. It follow that E5 and E7 can move right over α_2 to α_5 as required, and E17 can move left over α_6.

Similarly, since E9 succeed, we can infer that *tail.next* is not modified by α_3 to α_5, since this can only be done by a successful CAS at E9, which always sets *tail.next* to a new node previously allocated at E1 which no other process can see. It follows that E6 can move right over α_3 to α_5.

Thus, we move E5 to E8 right and E17 left to the position of E9, so the steps of Enq7 are contiguous; so we have:

$$\alpha_1\, E5_p\, \alpha_2\, E6_p\, \alpha_3\, E7_p^+\, \alpha_4\, E8_p^+\, \alpha_5\, E9_p^+\, \alpha_6\, E17_p^+\, \alpha_7 \;\leq\; \alpha_1\, \alpha_2\, \alpha_3\, \alpha_4\, \alpha_5\, E5_p\, E6_p\, E7_p^+\, E8_p^+\, E9_p^+\, E17_p^+\, \alpha_6\, \alpha_7$$

Path Deq6 is handled in essentially the same way.

4.5.3 Failed iteration in ENQUEUE advancing *Tail*

We consider path Enq4, i.e. E5, E6, $E7^+$, $E8^-$, $E13^+$, where we have:

E5	tail := Tail
E6	next := tail.next
$E7^+$	tail = Tail
$E8^-$	next $\neq$ *null*
$E13^+$	CAS(Tail, tail, next)$^+$

Let α be an execution containing an execution of Enq4 by process *p*, say $\alpha = \alpha_1\, E5_p\, \alpha_2\, E6_p\, \alpha_3\, E7_p^+\, \alpha_4\, E8_p^-\, \alpha_5\, E13^+\, \alpha_6$, where α_2 to α_5 contain no *p*-actions. We will show that actions E5–E8 can be moved right to the position of E13. We know that E8 is a both mover, since if only involve local variables.

We can also treat E6 as a right mover, since we can show that for any node *n*, *n.next* is only ever assigned twice: once at E3 when it is set to *null*, and once at E9 when it is set to a non-*null* value (note that E9 can only be executed when *next* = *null*). Thus, since we know from $E8^+$ that *tail.next* was not *null* when it was read at E6, it cannot be assigned again.

Finally, since the CAS at E13 succeeds, we can infer from the ABA freedom property that *Tail* is not assigned by α_2 to α_5. Thus, we have:

$$\alpha_1\, E5_p\, \alpha_2\, E6_p\, \alpha_3\, E7_p^+\, \alpha_4\, E8_p^-\, \alpha_5\, E13_p^+\, \alpha_6 \;\leq\; \alpha_1\, \alpha_2\, \alpha_3\, \alpha_4\, \alpha_5\, E5_p\, E6_p\, E7_p^+\, E8_p^-\, E13_p^+\, \alpha_6$$

Path Deq2 is handled in essentially the same way.

4.5.4 Normal successful iteration in DEQUEUE

Path Deq4 is D2-D4, $D5^+$, $D6^+$, $D7^+$, where we have:

D2	head := Head
D3	tail := Tail
D4	next := head.next
$D5^+$	head = Head
$D6^+$	head = tail
$D7^+$	next = *null*

Let α be an execution containing an execution of Deq4 by process *p*, say $\alpha = \alpha_1\, D2_p\, \alpha_2\, D3_p\, \alpha_3\, D4_p\, \alpha_4\, D5_p^+\, \alpha_5\, D6_p^+\, \alpha_6\, D7_p^+\, \alpha_7$, where α_2 to α_6 contain no *p*-actions. We can infer from the tests, and the ABA freedom property, that *Head* is not modified by α_2 to α_4 and *head.next* is not modified by α_3, but we don't know anything about whether *Tail* is changed. We therefore move D2 right over α_2, and D4 to D7 left over α_3 to α_7 as required. Thus, we have:

$$\alpha_1\, D2_p\, \alpha_2\, D3_p\, \alpha_3\, D4_p\, \alpha_4\, D5_p^+\, \alpha_5\, D6_p^+\, \alpha_6\, D7_p^+\, \alpha_7 \;\leq\; \alpha_1\, \alpha_2\, D2_p\, D3_p\, D4_p\, D5_p^+\, D6_p^+\, D7_p^+\, \alpha_3\, \alpha_4\, \alpha_5\, \alpha_6\, \alpha_7$$

4.5.5 Abnormal successful iteration

Path Enq6 is E5, E6, $E7^+$, $E8^+$, $E9^+$, $E17^-$, where we have:

E5	tail := Tail
E6	next := tail.next
$E7^+$	tail = Tail
$E8^+$	next = *null*
$E9^+$	CAS(tail.next, next, node)$^+$
$E17^-$	CAS(Tail, tail, node)$^-$

Let α be an execution containing an execution of Enq6 by process *p*, say $\alpha = \alpha_1\, E5_p\, \alpha_2\, E6_p\, \alpha_3\, E7_p^+\, \alpha_4\, E8_p^+\, \alpha_5\, E9_p^+\, \alpha_6\, E17_p^-\, \alpha_7$, where α_2 to α_6 contain no *p*-actions. E8 which is a both mover, since it only involves local variables. Since E7 succeeds, we know that *Tail* is not modified by α_2 or α_3, and since E9 succeeds, we know that α_3 to α_5 do not modify *tail.next*. Thus, there are various ways in which we can move E5 to E9 so that they are contiguous.

However, the fact that E17 fails means that *Tail* is changed some where in α_4 to α_6. Therefore, we can't move E5 to E9 right over α_6, not can we move E17 left over α_6. So at this stage, we cannot rearrange an execution of Enq6 to make its steps contiguous.

4.6 Reassigning *Tail* advance steps

We have now reduced all of the primitive paths so that their steps are contiguous, with the exception of Enq6, where the CAS at E17 fails because *Tail* has been updated by another process. We will address this case by showing that there is an equivalent execution in which the process that appends a node also updates *Tail* (i.e. all E17 actions succeed), which means that paths Enq4, Enq6 and Deq2 never occur.

Let α be an execution which contains a failed E17 action performed by process *p*. The fact that this action fails means that some another process, say *q*, has updated *Tail*, with a successful CAS at E13 or D10, as part of an Enq4 or Deq2 path, since *p* performed its successful CAS at E9. If more than one process has updated *Tail* since *p* performed its successful CAS at E9, we chose *q* to be the first such process. If *q* performs an E13 action, α is of the form $\alpha_1\, E9_p^+\, \alpha_2\, E13_q^+\, \alpha_3\, E17_p^-\, \alpha_4$, where α_2 does not contain any E10 or D10 action (note that α_2 also cannot contain a successful E17 action). We can now construct an equivalent execution α' in which *p* performs a successful E17 action at the point where *q* performed its successful E13 in α, and *q* performs an unsuccessful E13 action at the point where *p* performed its unsuccessful E17 action in α. Thus, we have:

$$\alpha_1\, E9_p^+\, \alpha_2\, E13_q^+\, \alpha_3\, E17_p^-\, \alpha_4 \leq \alpha_1\, E9_p^+\, \alpha_2\, E17_p^+\, \alpha_3\, E13_q^-\, \alpha_4$$

The case where *q* performs a D10 action is symmetrical, giving:

$$\alpha_1\, E9_p^+\, \alpha_2\, D10_q^+\, \alpha_3\, E17_p^-\, \alpha_4 \leq \alpha_1\, E9_p^+\, \alpha_2\, E17_p^+\, \alpha_3\, D10_q^-\, \alpha_4$$

The key observation here is that it doesn't matter what process performs a step that advances *Tail*. By assigning this step to the process which performed the closest preceding E9, we ensure that the resulting execution can be generated by the queue algorithm.

The effect of this transformation is to either swap an occurrence of Enq6 and an occurrence of Enq4 for an occurrence of Enq7 and an occurrence of Enq3, or swap an occurrence of Enq6 and an occurrence of Deq2 for an occurrence of Enq7 and an occurrence of Deq2. The result is that all Enq6 paths become Enq7 paths, which can now be reduced as shown in Section 4.5.2, and all Enq4 and Deq2 paths become Enq3 and Deq3 paths, respectively, which can now be deleted as shown in Section 4.4.

4.7 Assembling the remaining fragments

Following the above transformation, every execution of ENQUEUE or DEQUEUE has the form shown by the following regular expressions:

$$\mathit{Enq}1\ \mathit{Enq}6$$

$$\mathit{Deq}4 \mid \mathit{Deq}6$$

Finally, we observe that since all of the steps in Enq1 are both-movers, these steps can be moved right over any steps that occur between the executions of Enq1 and Enq6 by the same process. Thus, provided α_2 contains no *p* actions, we have:

$$\alpha_1\, \mathit{Enq}_p\, \alpha_2\, \mathit{Enq}6_p\, \alpha_3 \leq \alpha_1\, \alpha_2\, \mathit{Enq}_p\, \mathit{Enq}6_p\, \alpha_3$$

With a little simplification, it follows that ENQUEUE is equivalent to:

```
node := new_node()
node.value := value
node.next := null
tail.next := node
Tail := node
```

and DEQUEUE is equivalent to:

```
if Head = Tail then
  Head.next = null
  return false
else
  pvalue := Head.next.value
  Head := Head.next
  return true
```

It is easy to see that these correctly implement the queue operations with the chosen data representation.

5 Conclusions

We have shown how a version of Michael and Scott's lock-free queue can be proved to be linearisable, using a reduction method based on that of Lipton, Lamport, Cohen, and others. This approach separates reasoning about the concurrent and non-concurrent aspects of the algorithm, and addresses the concurrent part by focusing on the interactions between actions performed by different processes. This allows us to explain why the algorithm is correct in a way that is more compelling that a higher level proof, and provides more insight than a simulation proof, since it highlights properties (such as ABA freedom, unique pointers and fields not changing) on which the correctness of the algorithm relies. Some of these properties can be easily checked by inspection, or verified more rigorously using static analysis techniques; others require more sophisticated verification using model checking or theorem proving. Moreover, similar supporting properties are required in the verification of other lock-free algorithms.

Our trace reduction method extends Lipton's reduction method in several ways: we used a form of conditional reduction, where reductions depend on the outcomes of tests and CASes; we allow loop iterations that have no effect to be deleted (this is called *purity* in (Freund & Qadeer 2005)); we also allow certain actions to be assigned to other processes. This can be done because these actions could in fact be performed by any process, and would be required in verifying other algorithms using similar "helper" mechanisms, such as Shann et al's array-based queue (Shann et al. 2000) and Ladan-Mozes and Shavit's optimistic queue (Ladan-Mozes & Shavit 2004). In other work (Groves & Colvin 2006*b*) we have shown that algorithms such as the scalable stack described in (Hendler et al. 2004), where the linearisation point for one operation may be a step of another process, can be handled by by reducing two operations simultaneously.

We have simplified the original algorithm by assuming that storage is never recycled (or that the implementation language provides automatic garbage collection), which allows us to justify the ABA Freedom assumption. To justify this assumption while recycling storage, Michael and Scott add version numbers to pointer variables, which are incremented every time a pointer is modified. This can be introduced in our context as a further data refinement, but is only strictly correct if version numbers are unbounded. An alternative approach which avoids this problem is described in (Herlihy et al. 2002).

Michael and Scott (Michael & Scott 1998) gave a brief proof of some safety properties, but they were not sufficient to ensure linearisability. Yahav and Sagiv (Yahav & Sagiv 2003) describe an approach to verifying Michael and Scott's safety properties using model checking, but

their analysis appears to be very limited as they don't appear to have run the system with both ENQUEUEs and DEQUEUEs being performed.

Wang and Stoller (Wang & Stoller 2005) describe a static analysis technique for checking atomicity, and apply it to a variant of Michael and Scott's algorithm which avoids the ABA problem by using the less widely available Linked Load/Store Conditional instructions instead of CAS. However, their variant also avoids the main problem addressed in the paper by using a separate process to update *Tail*, which destroys the lock-freedom of the algorithm (since if that process dies the entire system will deadlock).

Doherty et al (Doherty et al. 2004) describe a fully mechanical proof of a variant of Michael and Scott's which is intended to reduce contention in the DEQUEUE operation by testing $next = null$ at D6, to determine whether the queue is empty, rather than $head = tail$, and only reading *Tail* if this test succeeds. This optimisation was discovered while attempting to prove the original algorithm. In our context, this modification would simplify the reasoning about path Deq2. This verification uses simulation between Input/Output Automata (Lynch 1996, Lynch & Vaandrager 1995), and requires a combination of forward and backward simulation to handle DEQUEUE on an empty queue since at the time *Tail* is read it is not known whether the algorithm will return *false*. Our proof requires no special treatment for this case.

Abrial and Cansell (Abrial & Cansell 2005) describe a constructive verification of a variant of Michael and Scott's algorithm using Event-B. They prove a variant of linearisability in which they require the linearisation point to be the last step taken by an operation, and delete line E17 from the algorithm so that *Tail* is always advanced by the next operation that notices *Tail* lagging, at E9 or D10. They also introduce an additional test in DEQUEUE, which requires *Tail* to be read again, before returning *false*. This is precisely the case that required a backward simulation in the verification in (Doherty et al. 2004), and this modification appears to have been required to avoid the need for backward simulation.

It would require a straightforward modification of our proof to show that the variants of Michael and Scott's algorithm described by (Wang & Stoller 2005), (Doherty et al. 2004) and (Abrial & Cansell 2005) are correct, and that the handling of DEQUEUE on an empty queue can be further simplified so that it never needs to access *Tail*.

Our future work will include mechanising our reduction proofs using PVS, and applying the approach to more sophisticated algorithms, such as the optimistic queue described in (Ladan-Mozes & Shavit 2004) and the scalable queue described in (Moir et al. 2005), to see whether other extensions are required and what other properties are required to justify its application.

Acknowledgements We are grateful to Sun Microsystems Laboratories for financial support, and to Rob Colvin and Mark Moir for helpful discussions relating to this work.

References

Abrial, J.-R. & Cansell, D. (2005), 'Formal construction of a non-blocking concurrent queue algorithm', *Journal of Universal Computer Science* **11**(5), 744–770.

Cohen, E. (2000), Separation and reduction, *in* 'Proc. 5th International Conference on Mathematics of Program Construction (MPC)', Springer-Verlag, London, UK, pp. 45–59.

Cohen, E. & Lamport, L. (1998), Reduction in TLA, *in* 'International Conference on Concurrency Theory (CONCUR)', pp. 317–331.

Colvin, R., Doherty, S. & Groves, L. (2005), Verifying concurrent data structures by simulation, *in* E. Boiten & J. Derrick, eds, 'Proc. Refinement Workshop (REFINE 2005)', Vol. 137(2) of *Electronic Notes in Theoretical Computer Science*, Elsevier, Guildford, UK, pp. 93–110.

Colvin, R. & Groves, L. (2005), Formal verification of an array-based nonblocking queue, *in* 'Proc. International Conference on Engineering of Complex Computer Systems (ICECCS)', ACM Press, New York, NY, USA, pp. 92–101.

Colvin, R., Groves, L., Luchangco, V. & Moir, M. (2006), Formal verification of a lazy concurrent list-based set algorithm, *in* T. Ball & R. B. Jones, eds, 'Proc. 18th International Conference on Computer Aided Verification (CAV)', Vol. 4144 of *Lecture Notes in Computer Science*, Springer, pp. 475–488.

Doeppner, Jr., T. W. (1977), Parallel program correctness through refinement, *in* 'Proc. 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)', ACM Press, pp. 155–169.

Doherty, S., Groves, L., Luchangco, V. & Moir, M. (2004), Formal verification of a practical lock-free queue algorithm., *in* D. de Frutos-Escrig & M. Núñez, eds, 'Formal Techniques for Networked and Distributed Systems (FORTE)', Vol. 3235 of *Lecture Notes in Computer Science*, Springer, pp. 97–114.

Dunne, S. (2003), Introducing backward refinement into B, *in* D. Bert, J. P. Bowen, S. King & M. A. Waldén, eds, 'ZB', Vol. 2651 of *Lecture Notes in Computer Science*, Springer, pp. 178–196.

Flanagan, C. & Qadeer, S. (2003), A type and effect system for atomicity, *in* 'Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation', pp. 338–349.

Freund, S. N. & Qadeer, S. (2005), 'Exploiting purity for atomicity', *IEEE Trans. Softw. Eng.* **31**(4), 275–291.

Groves, L. (2007*a*), Reasoning about nonblocking concurrency using reduction, *in* 'Proc. 12th Twelfth IEEE Int. Conf. on Engineering of Complex Computer Systems (ICECCS 2007)', Auckland, New Zealand, pp. 107–116.

Groves, L. (2007*b*), Verifying Michael and Scott's lock-free queue algorithm using trace reduction — the details, Technical report, Victoria University of Wellington. (To appear).

Groves, L. & Colvin, R. (2006*a*), Derivation of a scalable lock-free stack algorithm, *in* 'International Refinement Workshop (Refine 2006)', Electronic Notes in Theoretical Computer Science, Elsevier.

Groves, L. & Colvin, R. (2006*b*), Derivation of a scalable lock-free stack algorithm, *in* 'International Refinement Workshop (Refine 2006)', Electronic Notes in Theoretical Computer Science, Elsevier.

Hendler, D., Shavit, N. & Yerushalmi, L. (2004), A scalable lock-free stack algorithm, *in* 'SPAA 2004: Proceedings of the Sixteenth Annual ACM Symposium on Parallel Algorithms, June 27-30, 2004, Barcelona, Spain', pp. 206–215.

Herlihy, M., Luchangco, V. & Moir, M. (2002), The repeat offender problem: A mechanism for supporting dynamic-sized, lock-free data structures, *in* '16th International Conference on Distributed Computing (DISC 2002)', Vol. 2508 of *Lecture Notes in Computer Science*, Toulouse, France, pp. 339–353.

Herlihy, M. P. & Wing, J. M. (1990), 'Linearizability: a correctness condition for concurrent objects', *TOPLAS* **12**(3), 463–492.

Hesselink, W. H. (2002), 'An assertional criterion for atomicity', *Acta Informatica* **28**(5), 343–366.

Jifeng, H., Hoare, C. & Sanders, J. (1986), Data refinement refined, *in* 'ESOP 86', Vol. 213 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 187–196.

Ladan-Mozes, E. & Shavit, N. (2004), An optimistic approach to lock-free fifo queues, *in* 'Proc. of the 18th International Conference on Distributed Computing', pp. 117–131.

Lamport, L. (1990), 'A theorem on atomicity in distributed algorithms', *Distributed Computing* **4**(2), 59–68.

Lamport, L. & Schneider, F. B. (1989), Pretending atomicity, Technical Report TR89-1005, DEC, SRC.

Lipton, R. J. (1975), 'Reduction: a method of proving properties of parallel programs', *Communications of the ACM* **18**(12), 717–721.

Lynch, N. A. (1996), *Distributed Algorithms*, Morgan Kaufmann.

Lynch, N. A. & Vaandrager, F. W. (1995), 'Forward and backward simulations – Part I: Untimed systems.', *Information and Computation* **121**(2), 214–233.

Michael, M. & Scott, M. (1998), 'Nonblocking algorithms and preemption safe locking on multiprogrammed shared memory multiprocessors', *Journal of Parallel and Distributed Computing* **51**(1), 1–26.

Moir, M., Nussbaum, D., Shalev, O. & Shavit, N. (2005), Using elimination to implement scalable and lock-free fifo queues, *in* 'Proc. 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2005)', ACM Press, Las Vegas, Nevada, USA, pp. 253–262.

Sasturkar, A., Agarwal, R., Wang, L. & Stoller, S. D. (2005), Automated type-based analysis of data races and atomicity, *in* 'PPoPP '05: Proceedings of the tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming', ACM Press, New York, NY, USA, pp. 83–94.

Shann, C.-H., Huang, T.-L. & Chen, C. (2000), A practical nonblocking queue algorithm using compare-and-swap, *in* 'Seventh International Conference on Parallel and Distributed Systems (ICPADS'00)', pp. 470–475.

Stepney, S., Cooper, D. & Woodcock, J. (1998), More powerful Z data refinement: pushing the state of the art in industrial refinement, *in* J. P. Bowen, A. Fett & M. G. Hinchey, eds, 'The Z Formal Specification Notation, 11th International Conference of Z Users, Berlin, Germany, September 1998', Vol. 1493 of *LNCS*, Springer, pp. 284–307.

Treiber, R. K. (1986), Systems Programming: Coping with Parallelism. RJ5118, Technical report, IBM Almaden Research Center.

Wang, L. & Stoller, S. D. (2005), Static analysis of atomicity for programs with non-blocking synchronization, *in* 'PPoPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming', ACM Press, New York, NY, USA, pp. 61–71.

Yahav, E. & Sagiv, M. (2003), Automatically verifying concurrent queue algorithms, *in* B. Cook, S. Stoller & W. Visser, eds, 'Electronic Notes in Theoretical Computer Science', Vol. 89, Elsevier.

Author Index

Allender, Eric, 3
Asahiro, Yuichi, 97
Asquith, Matthew, 49

Bai, Shi, 125
Brent, Richard P., 125
Bunder, Martin, 7

Chang, Ching-Lueh, 117

Gor, Ajay S., 63
Groves, Lindsay, 133
Gudmundsson, Joachim, 49

Harland, James, iii
Huang, Xiaowei, 15
Huston, Samuel, 39

Jiao, Li, 15

Levit, Vadim, 87
Lu, Weiming, 15
Lyuu, Yuh-Dauh, 117

Mandrescu, Eugen, 87
Manyem, Prabhu, iii
Mathieson, Luke, 79
Merrick, Damian, 49
Miller, Mirka, 93
Miyano, Eiji, 97
Morozova, Elena, 57
Mujuni, Egbert, 75

Nguyen, Minh H., 93

Ohrimenko, Olga, 27
Ono, Hirotaka, 97

Pineda-Villavicencio, Guillermo, 93
Puchinger, Jakob, 39

Rosamond, Frances, 75
Ruskey, Frank, 107

Samer, Marko, 67
Shah, Nita H., 63
Stuckey, Peter, 27, 39
Szeider, Stefan, 67, 79

Ti, Yen-Wu, 117

Wee, Hui, 63
Williams, Aaron, 107

Recent Volumes in the CRPIT Series

ISSN 1445-1336

Listed below are some of the latest volumes published in the ACS Series *Conferences in Research and Practice in Information Technology*. The full text of most papers (in either PDF or Postscript format) is available at the series website `http://crpit.com`.

Volume 67 - Conceptual Modelling 2007
Edited by John F. Roddick, *Flinders University* and Annika Hinze, *University of Waikato, New Zealand.* January, 2007. 978-1-920682-48-4.
Contains the proceedings of the Fourth Asia-Pacific Conference on Conceptual Modelling (APCCM2007), Ballarat, Victoria, Australia, January 2007.

Volume 68 - ACSW Frontiers 2007
Edited by Ljiljana Brankovic, *University of Newcastle*, Paul Coddington, *University of Adelaide*, John F. Roddick, *Flinders University*, Chris Steketee, *University of South Australia*, Jim Warren, *the University of Auckland*, and Andrew Wendelborn, *University of Adelaide.* January, 2007. 978-1-920682-49-1.
Contains the proceedings of the ACSW Workshops - The Australasian Information Security Workshop: Privacy Enhancing Systems (AISW), the Australasian Symposium on Grid Computing and Research (AUSGRID), and the Australasian Workshop on Health Knowledge Management and Discovery (HKMD), Ballarat, Victoria, Australia, January 2007.

Volume 69 - Safety Critical Systems and Software 2006
Edited by Tony Cant, *Defence Science and Technology Organisation, Australia.* February, 2007. 978-1-920682-50-7.
Contains the proceedings of the 11th Australian Conference on Safety Critical Systems and Software, August 2006, Melbourne, Australia.

Volume 70 - Data Mining and Analytics 2007
Edited by Peter Christen, Paul Kennedy, Jiuyong Li, Inna Kolyshkina and Graham Williams. December, 2007. 978-1-920682-51-4.
Contains the proceedings of the 6th Australasian Data Mining Conference (AusDM 2007), Gold Coast, Australia. December 2007.

Volume 72 - Advances in Ontologies 2006
Edited by Mehmet Orgun *Macquarie University* and Thomas Meyer, *National ICT Australia, Sydney.* December, 2006. 978-1-920682-53-8.
Contains the proceedings of the Australasian Ontology Workshop (AOW 2006), Hobart, Australia, December 2006.

Volume 73 - Intelligent Systems for Bioinformatics 2006
Edited by Mikael Boden and Timothy Bailey *University of Queensland.* December, 2006. 978-1-920682-54-5.
Contains the proceedings of the AI 2006 Workshop on Intelligent Systems for Bioinformatics (WISB-2006), Hobart, Australia, December 2006.

Volume 74 - Computer Science 2008
Edited by Gillian Dobbie, *University of Auckland, New Zealand* and Bernard Mans *Macquarie University.* January, 2008. 978-1-920682-55-2.
Contains the proceedings of the Thirty-First Australasian Computer Science Conference (ACSC2008), Wollongong, NSW, Australia, January 2008.

Volume 75 - Database Technologies 2008
Edited by Alan Fekete, *University of Sydney* and Xuemin Lin, *University of New South Wales.* January, 2008. 978-1-920682-56-9.
Contains the proceedings of the Nineteenth Australasian Database Conference (ADC2008), Wollongong, NSW, Australia, January 2008.

Volume 76 - User Interfaces 2008
Edited by Beryl Plimmer and Gerald Weber *University of Auckland.* January, 2008. 978-1-920682-57-6.
Contains the proceedings of the Ninth Australasian User Interface Conference (AUIC2008), Wollongong, NSW, Australia, January 2008.

Volume 77 - Theory of Computing 2008
Edited by James Harland, *RMIT University* and Prabhu Manyem, *University of Ballarat.* January, 2008. 978-1-920682-58-3.
Contains the proceedings of the Fourteenth Computing: The Australasian Theory Symposium (CATS2008), Wollongong, NSW, Australia, January 2008.

Volume 78 - Computing Education 2008
Edited by Simon, *University of Newcastle* and Margaret Hamilton, *RMIT University.* January, 2008. 978-1-920682-59-0.
Contains the proceedings of the Tenth Australasian Computing Education Conference (ACE2008), Wollongong, NSW, Australia, January 2008.

Volume 79 - Conceptual Modelling 2008
Edited by Annika Hinze, *University of Waikato, New Zealand* and Markus Kirchberg, *Massey University, New Zealand.* January, 2008. 978-1-920682-60-6.
Contains the proceedings of the Fifth Asia-Pacific Conference on Conceptual Modelling (APCCM2008), Wollongong, NSW, Australia, January 2008.

Volume 80 - Health Data and Knowledge Management 2008
Edited by James R. Warren, Ping Yu, John Yearwood and Jon D. Patrick. January, 2008. 978-1-920682-61-3.
Contains the proceedings of the Australasian Workshop on Health Data and Knowledge Management (HDKM 2008), Wollongong, NSW, Australia, January 2008.

Volume 81 - Information Security 2008
Edited by Ljiljana Brankovic, *University of Newcastle* and Mirka Miller, *University of Ballarat.* January, 2008. 978-1-920682-62-0.
Contains the proceedings of the Australasian Information Security Conference (AISC 2008), Wollongong, NSW, Australia, January 2008.

Volume 82 - Grid Computing and e-Research
Edited by Wayne Kelly and Paul Roe *QUT.* January, 2008. 978-1-920682-63-7.
Contains the proceedings of the Australasian Workshop on Grid Computing and e-Research (AusGrid 2008), Wollongong, NSW, Australia, January 2008.

Volume 83 - Challenges in Conceptual Modelling
Edited by John Grundy, *University of Auckland, New Zealand*, Sven Hartmann, *Massey University, New Zealand*, Alberto H.F. Laender, *UFMG, Brazil*, Leszek Maciaszek, *Macquarie University, Australia* and John F. Roddick, *Flinders University, Australia.* December, 2007. 978-1-920682-64-4.
Contains the tutorials, posters, panels and industrial contributions to the 26th International Conference on Conceptual Modeling - ER 2007.

Volume 84 - Artificial Intelligence and Data Mining 2007
Edited by Kok-Leong Ong, *Deakin University, Australia*, Wenyuan Li, *University of Texas at Dallas, USA* and Junbin Gao, *Charles Sturt University, Australia.* December, 2007. 978-1-920682-65-1.
Contains the proceedings of the 2nd International Workshop on Integrating AI and Data Mining (AIDM 2007), Gold Coast, Australia. December 2007.

Volume 86 - Safety Critical Systems and Software 2007
Edited by Tony Cant, *Defence Science and Technology Organisation, Australia.* December, 2007. 978-1-920682-67-5.
Contains the proceedings of the 12th Australian Conference on Safety Critical Systems and Software, August 2006, Adelaide, Australia.